Machine Learning Advances for Practical Problems in Computer Vision

Philip T. Jackson

A thesis presented for the degree of
Doctor of Philosophy at Durham University

School of Computer Science
Durham University
United Kingdom

5th October 2020

Machine Learning Advances for Practical Problems in Computer Vision

Philip T. Jackson

Submitted for the degree of Doctor of Philosophy
April 2020

Convolutional neural networks (CNN) have become the de facto standard for computer vision tasks, due to their unparalleled performance and versatility. Although deep learning removes the need for extensive hand engineered features for every task, real world applications of CNNs still often require considerable engineering effort to produce usable results. In this thesis, we explore solutions to problems that arise in practical applications of CNNs.

We address a rarely acknowledged weakness of CNN object detectors: the tendency to emit many excess detection boxes per object, which must be pruned by non maximum suppression (NMS). This practice relies on the assumption that highly overlapping boxes are excess, which is problematic when objects are occluding overlapping detections are actually required. Therefore we propose a novel loss function that incentivises a CNN to emit exactly one detection per object, making NMS unnecessary.

Another common problem when deploying a CNN in the real world is domain shift - CNNs can be surprisingly vulnerable to sometimes quite subtle differences between the images they encounter at deployment and those they are trained on. We investigate the role that texture plays in domain shift, and propose a novel data augmentation technique using style transfer to train CNNs that are more robust against shifts in texture. We demonstrate that this technique results in better domain transfer on several datasets, without requiring any domain specific knowledge.

In collaboration with AstraZeneca, we develop an embedding space for cellular images collected in a high throughput imaging screen as part of a drug discovery project. This uses a combination of techniques to embed the images in 2D space such that similar images are nearby, for the purpose of visualization and data exploration. The images are also clustered automatically, splitting the

large dataset into a smaller number of clusters that display a common phenotype. This allows biologists to quickly triage the high throughput screen, selecting a small subset of promising phenotypes for further investigation.

Finally, we investigate an unusual form of domain bias that manifested in a real-world visual binary classification project for counterfeit detection. We confirm that CNNs are able to "cheat" the task by exploiting a strong correlation between class label and the specific camera that acquired the image, and show that this reliably occurs when the correlation is present. We also investigate the question of how exactly the CNN is able to infer camera type from image pixels, given that this is impossible to the human eye.

The contributions in this thesis are of practical value to deep learning practitioners working on a variety of problems in the field of computer vision.

Declaration

The work in this thesis is based on research carried out within the Innovative Computing Group at the Department of Computer Science, Durham University, UK. No part of this thesis has been submitted elsewhere for any other degree or qualification, and it is all the author's work unless referenced to the contrary below.

Note on Publications Included in this Thesis

At the time of submission, four chapters of this thesis are heavily based on papers submitted for publication or published in conferences:

Chapter 3 Jackson, P. T. & Obara, B. *Avoiding Over-Detection: Towards Combined Object Detection and Counting* in *International Conference on Artificial Intelligence and Soft Computing* (2017), 75–85

Chapter 4 Jackson, P. T., Atapour-Abarghouei, A., Bonner, S., Breckon, T. & Obara, B. Style Augmentation: Data Augmentation via Style Randomization. *CVPR DeepVision* (2018)

Chapter 5 Jackson, P. T., Wang, Y., Knight, S., Chen, H., Dorval, T., Brown, M., Bendtsen, C. & Obara, B. *Phenotypic profiling of high throughput imaging screens with generic deep convolutional features* in *2019 16th International Conference on Machine Vision Applications (MVA)* (2019), 1–4

Chapter 6 Jackson, P., Bonner, S., Jia, N., Holder, C., Stonehouse, J. & Obara, B. Camera Bias in a Fine Grained Classification Task. *CVPR DeepVision.* In review (2020)

These chapters are presented mostly as submitted, although referencing and notation has been altered and cross-referencing added for consistency throughout this thesis. The majority of the text is verbatim, however some stylistic changes have been made for consistency and some of the text has been extended to explain or discuss certain points in more detail. The text of all four chapters is the author's own, with some input and corrections from co-authors at the time of paper submission.

Note on Publications Not Included in this Thesis

As well as the above papers, the following works have been published during the period of research for this thesis, however, these publications do not fit into the narrative of this thesis and have not been included in the text.

- Willcocks, C. G., Jackson, P. T., Nelson, C. J. & Obara, B. Extracting 3D parametric curves from 2D images of helical objects. *IEEE transactions on pattern analysis and machine intelligence* **39,** 1757–1769 (2016)
- Nasrulloh, A. V., Willcocks, C. G., Jackson, P. T., Geenen, C., Habib, M. S., Steel, D. H. & Obara, B. Multi-scale segmentation and surface fitting for measuring 3-D macular holes. *IEEE transactions on medical imaging* **37,** 580–589 (2017)
- Willcocks, C. G., Jackson, P. T., Nelson, C. J., Nasrulloh, A. V. & Obara, B. Interactive GPU active contours for segmenting inhomogeneous objects. *Journal of Real-Time Image Processing* **16,** 2305–2318 (2019)
- Alharbi, S. S., Willcocks, C. G., Jackson, P. T., Alhasson, H. F. & Obara, B. Sequential graph-based extraction of curvilinear structures. *Signal, Image and Video Processing* **13,** 941–949 (2019)
- Nelson, C. J., Jackson, P. T. & Obara, B. *Combining mathematical morphology and the Hilbert transform for fully automatic nuclei detection in fluorescence microscopy* in *International Symposium on Mathematical Morphology and Its Applications to Signal and Image Processing* (2019), 532–543
- Bonner, S., Atapour-Abarghouei, A., Jackson, P. T., Brennan, J., Kureshi, I., Theodoropoulos, G., McGough, A. S. & Obara, B. Temporal Neighbourhood Aggregation: Predicting Future Links in Temporal Graphs via Recurrent Variational Graph Convolutions. *arXiv preprint arXiv:1908.08402* (2019)

Acknowledgements

Philip T. Jackson was sponsored by the Engineering and Physical Sciences Research Council, UK (Reference 1647095) for all the research presented in this thesis.

Chapter 5 The cellular image dataset in this chapter was provided by AstraZeneca, and the individuals key to its creation are acknowledged as co-authors in the corresponding paper.

Chapter 6 The shampoo bottle images in this chapter were provided by Proctor & Gamble.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Deep Convolutional Neural Networks (CNN) trained by gradient descent have revolutionised computer vision in recent years [11]. For the first time ever, the methods of machine vision have become reliable and powerful enough that they now see widespread and profitable deployment in industry. The last five years have seen an explosion in machine learning investment, industrial and scientific applications, hardware, startups, and education. Tasks that require the interpretation of raw images, such as object classification [12], object localisation [13], object counting [14], image segmentation [15], and monocular depth estimation [16], are all now dominated by neural networks.

These machine learning approaches have discarded many decades of hand-crafted vision algorithms in favour of end-to-end learning, where every step of processing from raw pixels to final output is learned from experience. CNNs use multiple layers of non-linear processing to incrementally transform a raw image into a prediction, where the action of each layer is determined by learned parameters. For each training example, consisting of an input and a correct output, a scalar error value called the loss is computed, reflecting the difference between the predicted and correct answers. This loss is then differentiated with respect to each parameter in each layer, which allows us to nudge the parameters in a direction that will, on average, result in a lower loss next time the same input is encountered. This approach, in combination with huge training datasets and high performance parallel compute hardware, is what has enabled the deep learning revolution [17].

This rapid growth has been driven in part by the emergence of large scale publicly available benchmark datasets [18], the most famous being the ImageNet dataset for image classification [19]. As of January 2020, ImageNet contains over 14 million images, each annotated with a class label for use as a training example. These enormous datasets not only supply the data needed to train deep neural networks, they also serve as a proving ground for new techniques. The winners of annual competitions such as the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) and Pascal Visual Object Classes (Pascal VOC) become recognized as state of the art approaches, and are quickly adopted as the workhorses of computer vision. This approach has served the research community well, providing a focal point where researchers can compare the empirical performance of competing methods on the fundamental problems of computer vision. However, despite remarkable progress having been made in these benchmark tasks, many obstacles still stand in the way of the application of neural networks to real world problems. Overcoming these obstacles is necessary if efforts of machine learning researchers are to translate into meaningful economic impact and improvements in society.

In this thesis, we describe four contributions that pertain to the practical use of convolutional neural networks, which are briefly outlined in the remainder of this introduction. Chapter 2 provides an introduction to neural networks, deep learning and CNNs which is prerequisite for understanding the rest of this thesis. Chapters 3-6 then describe my contributions in detail, followed by concluding remarks in Chapter 7.

1.1 Avoiding Overdetection: Towards Combined Object Detection and Counting

My first contribution pertains to object detection and counting. Object detection is the task of labeling each object in an image with a bounding box and (typically) a class label. An intuitive and easily implementable approach to object counting is to rely on object detection, first using an off-the-shelf detection network to label all the objects in an image and then counting the number of labels. Most deep learning approaches to object detection result in a fixed number of object bounding boxes being produced, regardless of the number of objects in the image. A confidence value is also predicted for each box, representing the network's confidence that the box in fact contains an object. By design, the fixed number of boxes is normally greater than the number of objects present, which results in a surplus of boxes being emitted for each object in the image. The excess boxes are normally then pruned by non-maximum suppression (NMS) [20], an

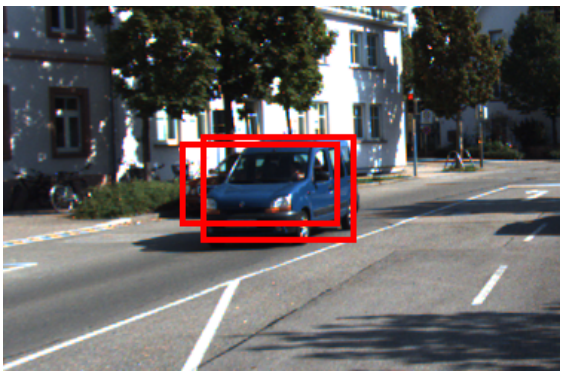


Figure 1.1: Real life contains numerous instances where objects overlap, and yet the non-max suppression algorithm for removing excess detection bounding boxes assumes that strongly overlapping boxes are over-detections of the same object. The two boxes in this example have an intersection-over-union measure of 0.72, which is above the commonly used threshold of 0.7, so one of them would be removed by NMS, even though in this case they are labelling two distinct objects.

iterative algorithm in which boxes which overlap sufficiently with a box of higher confidence are removed. In the majority of cases, this results in one bounding box per object, and so the number of boxes can function as an object count (e.g. [21]). However, in cases where objects are densely clustered or overlapping, NMS may wrongly remove the overlapping boxes (see Figure 1.1). A method which only emits one box per object in the first place would clearly solve this problem and obviate NMS in the process, resulting in a more elegant solution. Explaining why most object detection networks share this weakness is also an interesting research question in itself. My contribution here is an answer to that question, and a novel loss function that trains detection networks to emit one box per object.

1.2 Style Augmentation

Domain bias is a common issue in which a model which attains good performance on inputs similar to those it was trained on fails to perform well on dissimilar inputs. In other words, the model is biased towards the domain in which it was trained. To some extent this is expected, but in the case of computer vision, the differences required to cause domain bias issues can be surprisingly subtle. For example, a model trained on images from a video game can perform poorly on images from the real world, even when the virtual world is highly realistic [22]. Meanwhile, research by Tobin et al. [23] shows that CNNs can generalize from very unrealistic virtual worlds to the real world, if the textures in the virtual world are randomized. This suggests

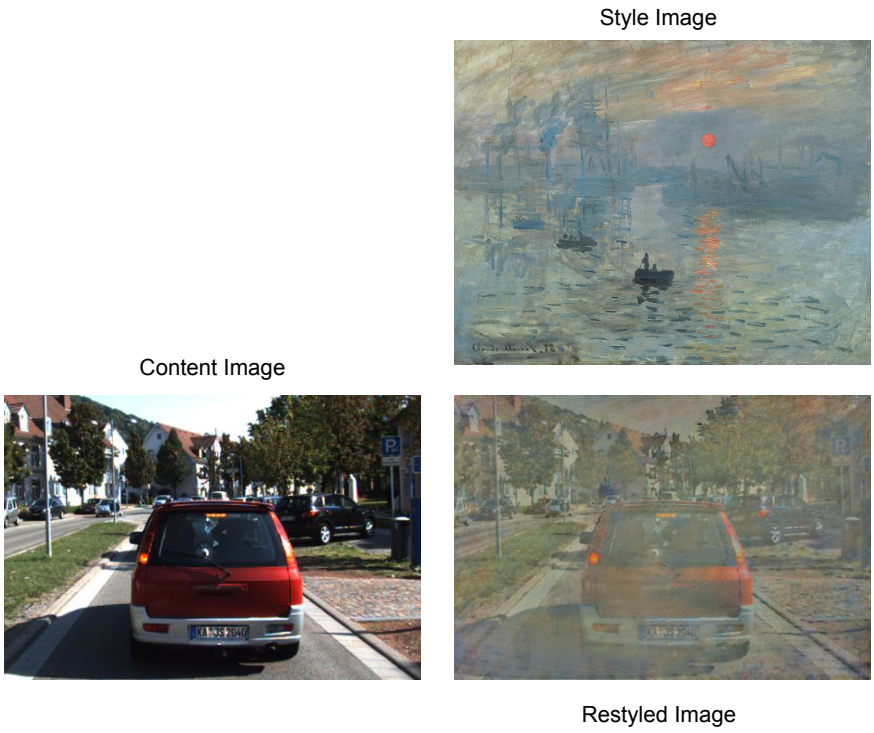


Figure 1.2: The style of Monet transferred to an image from the KITTI dataset. By applying random styles to training images while preserving the labels, we can train networks to be more robust to shifts in texture.

that CNNs are sensitive to texture, and that the key to training a robust model is diversity of texture in the training dataset.

With this in mind, we wondered if it would be possible to improve model robustness by randomizing textures in training images as a post-processing step, without access to the rendering engine that created them. Such a method could randomize textures in ordinary photographic images, potentially increasing the robustness of models trained on any image dataset.

Neural style transfer [24] is a deep learning technique for transferring the artistic style of one image to another, allowing one to apply the style of Monet to a photograph, for example. In practice, the concept of artistic style as it pertains to neural style transfer is very similar to the concept of texture [25]. Style transfer alters the distribution of low-level visual features while

preserving the shape of objects. Therefore, if style transfer could be applied to an image in a randomized manner, rather than copying the style of one image onto another, then we might expect a network trained on such images to rely more heavily on shape than on texture for its judgements. This would encourage good generalization across domains, since a model that can remain accurate when texture is randomized can probably remain accurate when processing slightly different images from another domain.

The practice of randomly pre-processing training images with transforms that do not render the image unidentifiable is called data augmentation, and has been in use since the early days of learned vision (e.g. [26, 27]). It is seen as a way to artificially inflate a small training dataset (e.g. in [28]), and can also be thought of as a way to train a network to be invariant to a certain transform. For example, random horizontal flipping is a common image augmentation because we know that mirroring an image should not affect its label, and Simard et al. [27] inject elastic deformations into handwritten digit images to mimic the distortions caused by the uncontrolled oscillations of human hand muscles. Since the technique discussed in this chapter is also applying randomized label preserving transforms to training images with the intent of teaching invariance to said transforms, it too belongs under the umbrella of data augmentation, hence the name "Style Augmentation". Style Augmentation is designed as a generic data augmentation technique, which could be easily slotted into any machine learning vision process.

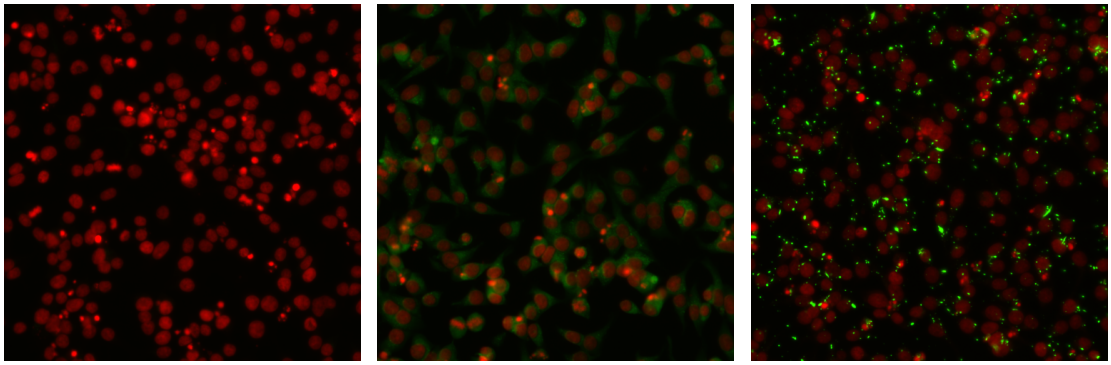
1.3 Phenotypic Profiling of Chemical Clusters with Generic Deep Convolutional Features

Out of all the obstacles to deep learning's practical deployment, its dependency on large quantities of labelled training data is probably the most common. With a few notable exceptions (such as GPT-2 [29]), the state-of-the-art results in most deep learning benchmark tasks have been achieved via supervised learning, in which the target output for each training example is produced manually by a human (e.g. [12, 13, 15]). The ImageNet dataset / ILSVRC challenge is the most prominent example of supervised learning; each of the 14 million images in ImageNet was manually annotated with a class label via the Amazon Mechanical Turk crowdsourcing platform [30]. This is an enormous and expensive undertaking, which most organizations do not have the resources to reproduce for their individual problems.

The contribution in this chapter concerns a project in collaboration with AstraZeneca, in which a large, unlabelled image dataset had to be split into classes. AstraZeneca are a pharmaceutical

firm who use high throughput screening (HTS) techniques for novel drug discovery. HTS uses robotic systems to test interactions between large numbers of novel compounds (typically in the millions) and biological cells in parallel, with the aim of winnowing a field of millions of compounds down to a few promising leads, known as "hits", which will then proceed to further test, preclinical studies, and finally clinical trials. In this case, a set of compounds were applied to human kidney cells and imaged with a wide field fluorescence microscope after an incubation period. These cells were genetically engineered to express both Green Fluorescent Protein bound to Low Density Lipoprotein Receptor (LDLR-GFP), and Induced Degrader Of LDLR (IDOL). Since IDOL breaks down the LDLR-GFP complex, its presence causes the absence of green fluorescence, and anything that inhibits IDOL results in the presence of green fluorescence. The aim of the project was to discover compounds that inhibit IDOL. However, presence of green fluorescence alone is not sufficient - it also matters where within the cell that green signal is localised (e.g. within the cytosol, within the membrane, within the golgi apparatus). The visual manifestations of these differences are complex and highly variable (see Figure 1.3), which makes it difficult to detect them using traditional image processing techniques. The inevitable occurrence of artefacts during image acquisition complicates matters further. Due to the lack of a positive control, the researchers also did not have a clear idea of what a genuine hit looked like; this shifted the emphasis away from binary hit/non-hit classification and towards dataset exploration and class discovery. In a supervised learning project, the network learns from manually generated annotations that explicitly label the category of each image. Is it possible for a CNN to classify the images in this domain specific dataset even though no such labels are available to learn from?

In this chapter we present a pre-clinical screening workflow that exploits a pre-trained CNN to cluster a large cellular image dataset by visual similarity, producing a small number of phenotypic clusters that a domain expert can quickly assess. Grouping similar images together allows biologists to apply their judgement to entire clusters of images at once, rather than assessing images one by one, resulting in much faster pre-clinical screening. ImageNet pre-trained CNNs are known to detect a broad range of low level features [31], and it turns out that these are rich enough to produce distinct responses for cellular images of different phenotypes, even though the network itself was trained to classify images of a very different nature. By passing cellular images through the CNN, feature vectors for the images can be derived from its internal dynamics. These are then grouped by an unsupervised clustering algorithm, and represented graphically via a t-SNE plot, providing an overview of the dataset in which different phenotypes emerge as distinct clusters of points.



(a) (b) (c)

Figure 1.3: Three samples from the high throughput imaging dataset. a) no IDOL inhibition, hence no LDLR-GFP is present, only the cellular nuclei, stained red with Hoechst stain, are visible. b) LDLR-GFP visible as green fluorescence in the cytoplasm c) LDLR-GFP, localised within cellular organelles, visible as bright green dots.

1.4 Camera Bias In Fine Grained Classification: Effects and Mitigations

CNNs (and machine learning methods in general) will exploit any patterns in their training data to minimize their loss functions. When we show a model an input / output training example, we are telling it what the correct output is for that input but not *why* that output is correct. The rules for mapping input to output have to be inferred from many such examples, and the model has no way of knowing if a rule that minimizes loss over the training set will not generalize to the test set. Sometimes this leads to unwanted behaviour, particularly if the model learns to exploit a some subtle idiosyncrasy in the training data, which the engineer may not be aware of.

In this chapter, we investigate an unusual form of domain bias that interfered with another industrial collaboration, this time with Proctor & Gamble. This project aimed to apply standard CNNs with supervised learning to detect counterfeit shampoo bottles, by observing subtle differences in the batch code printed on the underside of the bottle. In this chapter, we prove that correlations between the class label (real or counterfeit) and the type of camera that captured the image are exploited by CNNs, resulting in a model that "cheats" by inferring the class from the camera type. Such models are unable to generalize to domains where the class label does not correlate with camera type. We also perform experiments to determine how CNNs are able to recognize cameras from the images they produce, since the differences between images from

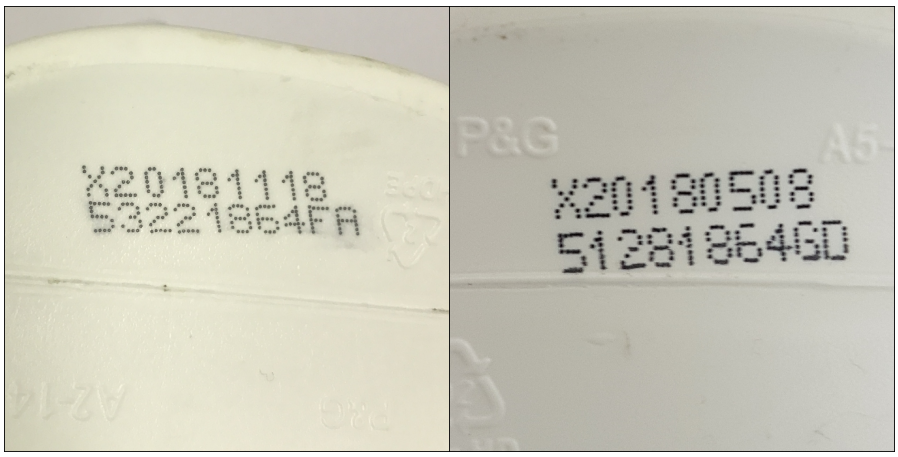


Figure 1.4: Batch codes from a genuine and counterfeit shampoo bottle (left, right). Human domain experts achieve around 60% accuracy on this classification task. With extremely fine-grained classification problems like this, it is often very unclear which features a CNN is exploiting when it achieves high classification accuracy. Answering this question would be useful both for validating the trustworthiness of the model, and for understanding the differences between real and counterfeit bottles.

different cameras are imperceptible to the human eye.

Chapter 2

Deep Neural Networks for Computer Vision

Deep neural networks (DNN), once a fringe sub-field considered by most to be unworkable, experienced a sudden and spectacular rise to prominence in 2012 when a DNN achieved breakthrough performance in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [32]. Alex Krizhevsky's winning entry, which became known as AlexNet, improved on the previous year's classification accuracy by an unprecedented 15% margin [19], establishing DNNs as the dominant technique in computer vision and provoking a massive resurgence of interest in neural networks generally. Since then, DNNs have come to dominate in a number of fields besides computer vision, most notably reinforcement learning [33] and natural language processing [34].

In this chapter, we will provide an introduction to neural networks, their history, key breakthroughs, their structure and the methods by which they are trained. In particular we will focus on convolutional neural networks (CNN), which have become the workhorse of modern computer vision.

2.1 Neural Networks

Artificial neural networks are a biologically inspired model of computation, first proposed by McCulloch and Pitts in 1943 [35]. This paper developed the first model of an artificial neuron

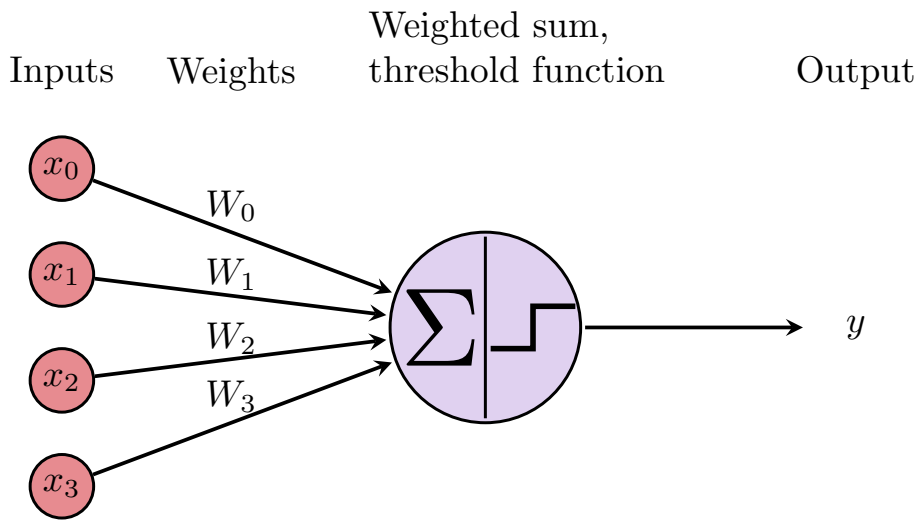


Figure 2.1: A McCulloch and Pitts artificial neuron. It computes a single output value from multiple inputs. If a weighted sum of the inputs exceeds the neuron's threshold value, then the output is 1, otherwise it is 0.

(commonly referred to now as a McCulloch-Pitts (MP) neuron), and analyzed their computational properties.

MP neurons are very simple. Like biological neurons, each MP neuron receives input from several other neurons, and can emit a single output value when sufficiently excited by its inputs. They have weights on their incoming connections, which determine whether those input signals are excitatory or inhibitory; the neuron fires if and only if the weighted sum of its inputs exceeds some threshold value (see Figure 2.1). A neuron that fires is said to be activated, and its output value is often referred to as its activation. It was shown that, despite their enormous simplifications compared to real neurons, networks of MP neurons could implement elementary logic gates such as AND, OR and NOT. This means that networks of artificial neurons are able to represent arbitrary propositional logic formulas. Not only that, but neural networks whose directed connections form cycles are computationally universal, i.e. able to represent any program. Although this model has been tweaked many times since 1943, the basic principles of neural networks were laid down in this seminal work:

- Neurons are discrete computational units with directed, weighted connections between them
- Neurons receive input signals from other neurons, and send output signals to other neurons, according to the connections between them
- Neurons compute weighted sums of their input signals, and their output value is a nonlinear function of this weighted sum
- A neural net contains a subset of neurons called input neurons, who receive no input from other neurons and whose output values are raw input observations
- A neural net contains a subset of neurons called output neurons, who do receive input from other neurons and whose output values form the output of the neural net.

McCulloch-Pitts neurons are very limited. The connection weights are constrained to be either $+1$ or -1, the output of each neuron is either 0 or 1, and most importantly, there is no learning algorithm. Despite their theoretical universality, there was no general algorithm for determining the connections and weights needed to implement a given function or solve a given task. They are also clearly quite ill-suited to representing real valued functions.

As computers became more powerful in the decades that followed, many researchers advanced the study of neural networks in different directions. Among them was Frank Rosenblatt, who in 1958 published the perceptron [36], a simple neural machine originally applied to image recognition. A perceptron has a layer of input neurons, and a layer of output neurons. A perceptron neuron computes a weighted sum of its real valued inputs, using real valued weights, and then passes that weighted sum through a non-linear function to compute its output value. This output is often referred to as the neuron's "activation", and the non-linear function as an "activation function". If we treat the input layer values as a vector $\vec{x}$, and the corresponding weights as another weight vector $\vec{w}$, then a perceptron neuron can be said to perform the following computation:

$$y = f(\vec{x} \cdot \vec{w} + b) \tag{2.1}$$

where f is a continuous activation function in place of the threshold function in MP neurons, and b is a constant bias term. When we have more than one neuron dependent on the same

set of inputs (for example, the "hidden layer" in Figure 2.2), we can treat their activations as a single vector $\vec{y}$, the connection weights between the two layers as a matrix $\mathbf{W}$, and the biases as a vector $\vec{b}$ and describe the action of the entire layer as follows:

$$\vec{y} = f\left(\mathbf{W}\vec{x} + \vec{b}\right) \tag{2.2}$$

where the activation function f is now implicitly applied elementwise across the vector of pre-activation values (also known as *logits*). When f is a threshold function (as in MP neurons), or a continuous version of it such as the logistic function, a perceptron neuron can be understood as a linear classifier - that is, it classifies input vectors based on which side of a hyperplane they lie on. The weight vector is the normal vector of this hyperplane, and the bias determines its distance to the origin.

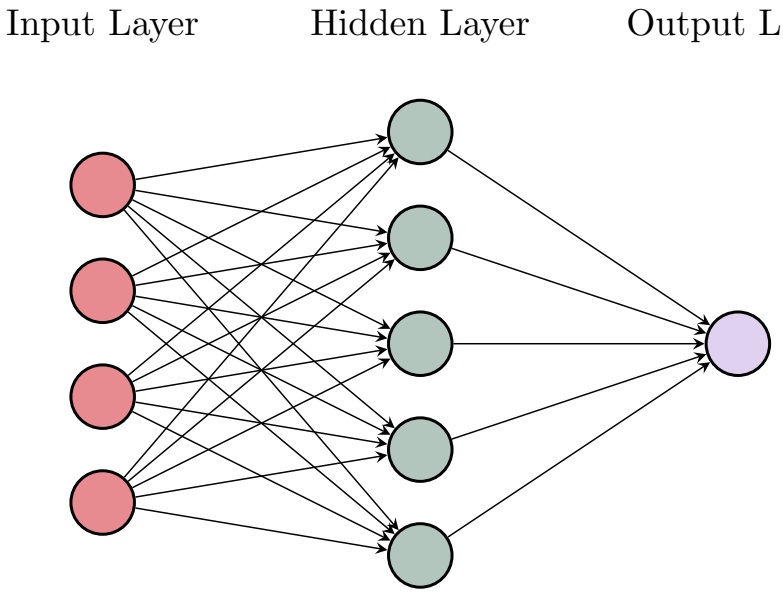


Figure 2.2: A multi-layer perceptron has two (or more) layers of modifiable connections. Even with only one hidden layer, an MLP is theoretically able to approximate any function from the input to output domain to arbitrary accuracy, given sufficient neurons in the hidden layer.

The single layer perceptrons described above are only capable of learning linearly separable patterns, as famously highlighted by Marvin Minsky and Seymour Papert in their 1969 book *Perceptrons* [37]. This relatively trivial observation has been the source of much controversy between connectionist and symbolic AI for several decades [38, 39]. The linear separability limitation can be overcome by inserting an additional "hidden layer" between the input and output layers, creating a multi-layer perceptron (MLP, see Figure 2.2).

MLPs are universal function approximators: with enough neurons in the hidden layer, they are capable of approximating arbitrary functions between finite dimensional vector spaces with arbitrary accuracy [40, 41]. The exact shape of this function depends on the values of the network's inter-neuron connection weights and neuron biases. The architecture of the network (number of layers, number of neurons in each layer, connectivity between layers) therefore defines not one function but a set of functions, parameterized by these weights and biases (which indeed are together referred to as network parameters). To implement a function that does something useful, one must find good values for these parameters.

Unfortunately, manually specifying good values for weights and biases is something human programmers are extremely bad at. The 20th century saw many attempts to address this difficulty using machine learning, including both supervised learning approaches such as the delta rule [42], and biologically inspired unsupervised approaches such as Hebbian learning [43], self-organizing Kohonen maps [44] and Hopfield networks [45]. Eventually, backpropagation of errors emerged as the dominant technique for training neural networks (for a more thorough account see Jurgen Schmidhuber's analysis [46]).

The idea is to make incremental updates to the weights and biases (henceforth referred to as parameters) which, on average, improve the performance of the network at a given task. This requires two things: a precise, quantitative metric of "performance for a given task", and some way of estimating the optimal direction in which to nudge the parameters to increase that performance metric. The former is provided by something called a loss function, and the latter is provided by backpropagation.

2.2 Loss Functions

A loss function is an error function to be minimized, whose formula defines the task to be learnt by the network. For a supervised classification task (e.g. image recognition), this function is nearly always cross entropy, defined as such:

$$H(p,q) = -\sum_{i=1}^{n} p(c_i) log(q(c_i)) \tag{2.3}$$

Cross entropy is a quasi distance metric between the two probability distributions p and q (quasi because $H(p,q) \neq H(q,p)$). In our case, the c_i are possible class labels, n is the number of classes, p is the true probability distribution for a given sample (which we know because the images are pre-annotated), and q is the network's predicted probability distribution. This requires us to interpret the output of our network as a probability distribution, to which end we apply the softmax activation function:

$$y = q(c_i) = \text{softmax}(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}} \tag{2.4}$$

where y is the network's final (post-activation) output and z is the vector of pre-activation logits of the network's output layer, which should contain one neuron per class. Softmax is an unusual activation function in that it is not elementwise: the activation of each unit in the vector depends on the logits of all the others due to the summation in the denominator. This summation normalizes the activations so that they sum to one, as a probability distribution must - an increase in the activation of one neuron will therefore result in a decrease in the activations of the others. Interpreting the output of a neural network as a probability distribution over possible class labels is a key component in training neural networks for discrete classification tasks, as it allows us to define a continuous loss function in which small changes to the real valued output vector produce a non-zero change in the loss. In mathematical terms this means we can define a *differentiable* loss function, which would not be the case if the loss were a function of, say, argmax(z) (because the argmax function is discontinuous and therefore non-differentiable, changing all at once only when the highest value logit is overtaken by another).

Our true probability distribution, p, is likewise denoted by a one-hot vector:

$$p(c_i) = \begin{cases} 1 & \text{if i = t} \\ 0 & \text{otherwise} \end{cases} \tag{2.5}$$

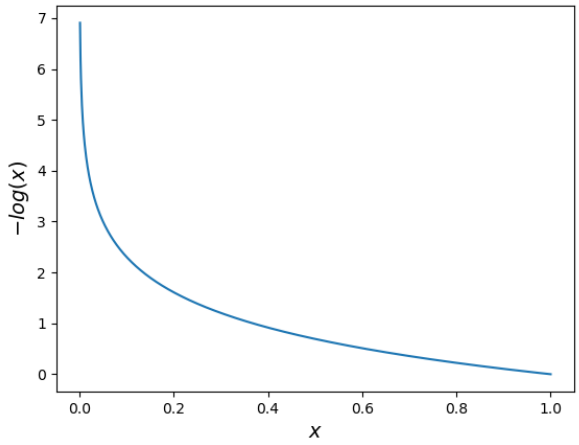


Figure 2.3: Cross entropy loss as a function of the predicted probability of the correct class ($x = \text{softmax}(y_t)$). The steep gradient as x approaches 0 compensates for the low gradient of the softmax with respect to y_t when $\text{softmax}(y_t)$ is low.

where c_t is the target or true class label. With this in mind, it can be seen that Equation 2.3 simplifies to the predicted negative log likelihood of the correct class:

$$H(p, q) = -log(q(c_t)) = -log(y_{c_t}) \tag{2.6}$$

Thus, softmax output defines a continuous output space for a discrete classification problem, and cross entropy loss defines a continuous "wrongness" metric over these outputs - at least for a given instance of the problem. Of course, a correct answer for one particular input does not imply that the network has learned anything useful; to prove that the network has mastered a task, we must achieve correct answers on many diverse instances. Therefore, our true loss function for a classification task is the *average* cross entropy over a large dataset of training examples:

$$\mathcal{L}(X, \theta) = \frac{1}{|X|} \sum_{(x,t) \in X} -log(f(x; \theta)_t) \tag{2.7}$$

Where X is a set of training examples, each consisting of an input x and a class label t, and $f(x; \theta)$ is the network's softmax output, given input x and current parameters θ.

2.3 Backpropagation

Backpropagation is not a neural network specific technique, nor is it a complete learning algorithm per se. Backpropagation is an algorithm for automatically differentiating a composite function through repeated application of the chain rule of differential calculus. The chain rule states that the derivative of the composite of two functions f and g is the product of the derivatives of f and g:

$$(f(g(x)))' = f'(g(x)) \cdot g'(x) \tag{2.8}$$

or, in Leibniz notation:

$$\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx} \tag{2.9}$$

where z is a function of y, which is a function of x. By induction, the chain rule can be extended to function compositions of arbitrary length:

$$\frac{d}{dx} f_1(f_2(f_3(\dots f_n(x)))) = \frac{df_1}{df_2} \cdot \frac{df_2}{df_3} \cdot \frac{df_3}{df_4} \dots \frac{df_n}{dx} \tag{2.10}$$

Since the activation of each layer of a neural network is a function of the previous layer, neural networks are composite functions. To make this clearer, consider the following computational graph, representing the three layer perceptron shown in Figure 2.2.

The edges represent computational dependencies between the various input, output and intermediate values, which are represented as nodes in the graph. The value of a node can only be computed given knowledge of the values of its parent nodes. “Running” a neural network - that is, computing its output y starting from only its input x and parameters w_i, is a process of continually computing the values for nodes whose dependencies are known, until we reach the output node. In deep learning parlance this is known as a *forward pass*, since it follows the edges of the computational graph forward from input to output. Backpropagation, in contrast, is often referred to as a *backward pass*, since it begins at the loss and follows the edges backward to the parameter nodes. In much the same way that the forward pass can only compute the values of

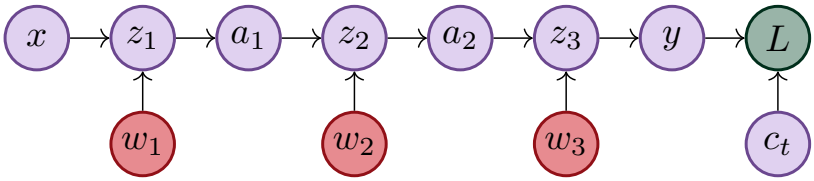


Figure 2.4: A computational graph for the network depicted in Figure 2.2. Nodes denote the input vector x, output vector y, intermediate values z_i, a_i, weights and biases (w_i), class label c_t and loss L; edges denote computational dependencies. z_i denote pre-activation logits, a_i are activations. Leaf nodes x and w_i are nodes whose values are known a priori, everything else must be computed from the values of its parent nodes. Backpropagation begins at the loss (green) and works backwards, terminating at the parameter nodes (red).

nodes whose parents' values are known, the backward pass can only immediately compute the gradient of a node whose child's gradient is known*.

Once backpropagation has computed the gradients of all the network's parameters, we are ready to update those parameters. If we think of the concatenation of all weights and biases as a single vector of parameters θ, then backpropagation yields us $\nabla_\theta L(x, \theta)$, the gradient vector of the loss (for the given input x) with respect to the parameters θ. This vector points in the direction of the steepest rate of increase of loss in parameter space; since we wish to decrease the loss, we therefore take a small step in the opposite direction. So long as this step is small enough that the gradient does not change too much over the distance travelled, this will result in a new parameter vector θ' such that $L(x, \theta') < L(x, \theta)$.

Computational graphs are not just an intuitive and elegant way to visualize a neural network; most modern neural network libraries use computational graphs explicitly to represent neural networks. In TensorFlow [47] for example, a typical program begins by constructing a computational graph representing the desired neural net, which can later to visualized in a web browser (see Figure 2.5).

2.4 Stochastic Gradient Descent

With a well defined objective function and a method for computing its gradient for each of our parameters, neural network training reduces to an optimization problem, which we can solve

* To avoid confusion, it should be mentioned that the "gradient of node i" refers to the gradient of the loss *with respect to* to node i. Although backpropagation can in principle compute the gradient of any node with respect to any upstream node, there is rarely any reason to differentiate anything other than the loss.

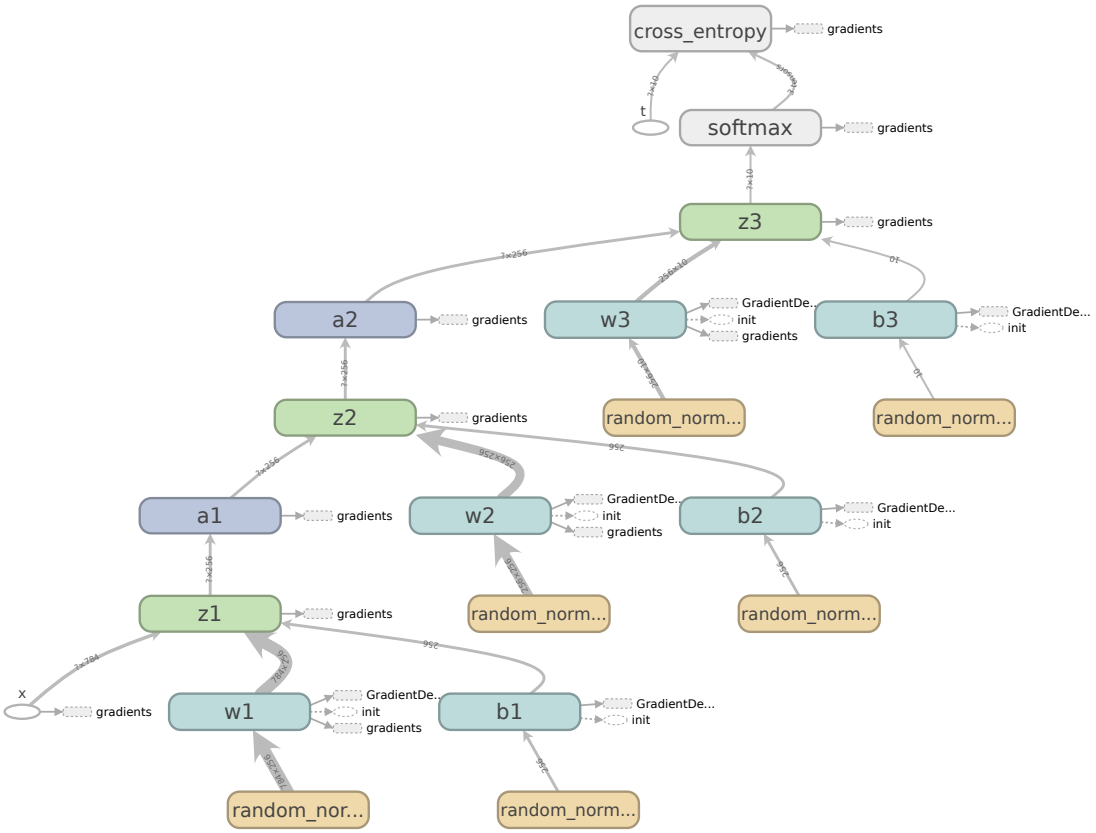


Figure 2.5: A TensorFlow computational graph corresponding to the one in Figure 2.4. Leaf nodes (small white ellipses) are points at which training data is fed into the graph. One is x, the image, while the other is t, the class label, used in the cross entropy loss (which is included in this diagram but not in Figure 2.4). Parameter nodes show a computational dependency on random normal distributions - this is because weights and biases are generally initialized by sampling from normal distributions.

with first-order gradient descent. Gradient descent is a simple, intuitive, well studied technique for finding local minima of scalar functions. Consider a differentiable function $F : \mathbb{R}^n \to \mathbb{R}$. Gradient descent allows us to begin at an arbitrary point $\theta_0 \in \mathbb{R}^n$, and iteratively update this point by shifting it a small amount in the opposite direction to the gradient of F. This results in a series of points that converges on a local minimum, at which the gradient is zero. Formally,

$$\theta_{t+1} = \theta_t - \alpha \nabla_\theta F(\theta_t). \tag{2.11}$$

In the context of machine learning, α is called the *learning rate*, and is generally the most important hyperparameter in any deep learning task. If it is too large then θ_{t+1} may be larger than θ_t and the sequence may diverge, since $\nabla_\theta F(\theta_t)$ tells us only the instantaneous gradient at the point θ_t. Likewise if α is too small, then the sequence will take too long to converge, since each gradient descent step requires another expensive gradient evaluation via backpropagation (see Figure 2.6).

Recall that the loss function we wish to optimise (Equation 2.7) is actually defined as the mean of the losses of individual samples $x \in X$. So far we've only discussed computing and backpropagating the loss for an individual sample, but due to the linearity of expectation, the gradient of the mean loss is just the mean of the gradients of the individual losses:

$$\nabla_\theta \mathbb{E}_{x \in X} \left[L(x, \theta) \right] = \mathbb{E}_{x \in X} \left[\nabla_\theta L(x, \theta) \right] \tag{2.12}$$

In practice, the number of training inputs $|X|$ is often very large, particularly in deep learning settings where copious training data has proved to be quite important. Computing Equation 2.12 in full can therefore be computationally expensive, which in turn limits the rate at which we can perform parameter updates. This problem can be greatly mitigated by computing Equation 2.12 over *minibatches* rather than over the full dataset. A minibatch is a small batch of random samples from X, the size of which is generally limited by the amount of memory available. This random sampling, then, is where the stochasticity of stochastic gradient descent comes from. So long as the minibatch is sampled uniformly at random, then the mean gradient over a minibatch is an unbiased estimate of the gradient over the full dataset (Equation 2.12). The variance of this estimate tends towards zero as the minibatch size tends towards $|X|$, but with diminishing returns. A minibatch much smaller than $|X|$ will typically provide a decent estimate in a fraction of the time required for the full dataset.

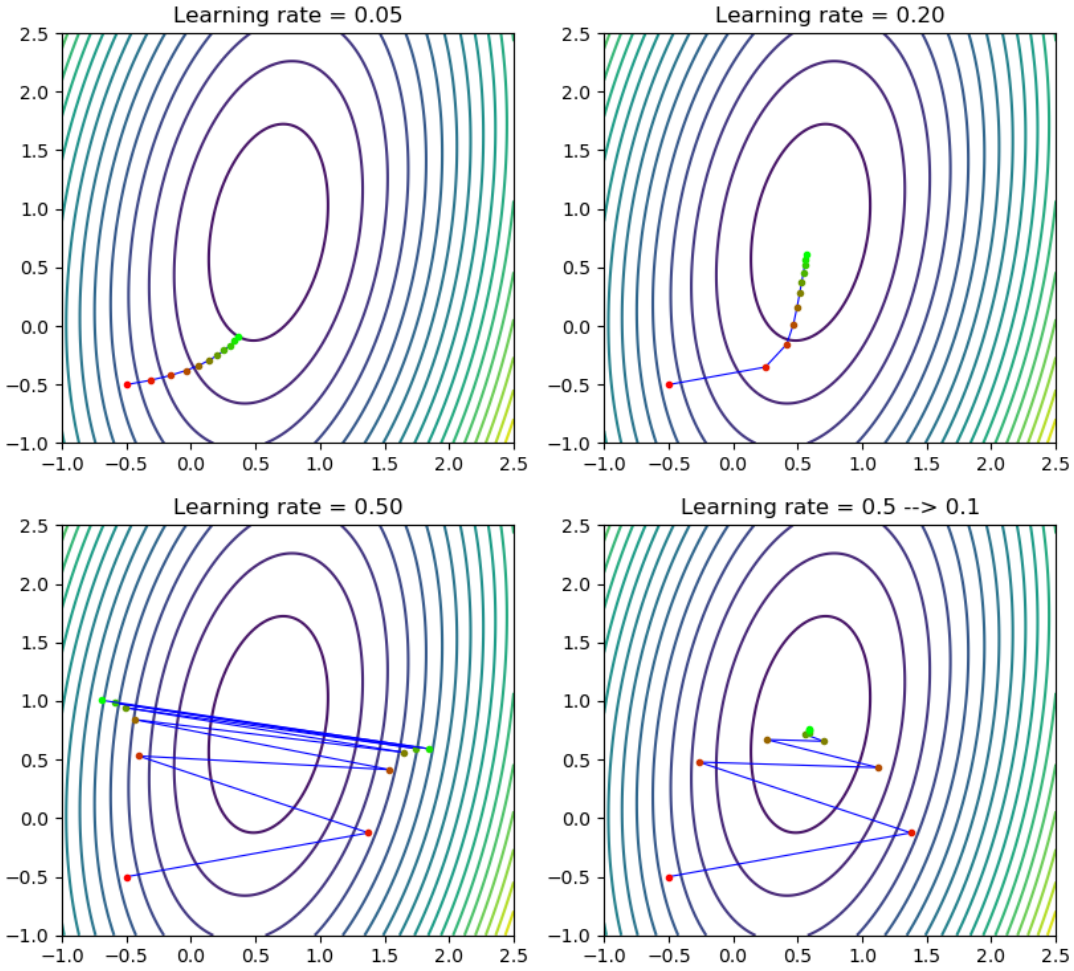


Figure 2.6: Ten steps of gradient descent on a quadratic bowl loss function, with different learning rates. With too low a learning rate, ten steps is not enough to reach the local minimum (upper left), whereas with too high a rate the optimizer diverges (lower left). A good strategy is to start with a high learning rate and reduce it throughout training (lower right). This attains the lowest loss of all, without having to guess a good learning rate (upper right).

2.5 Deep Learning

Although 3-layer MLPs are universal function approximators, this theorem says nothing about the efficiency of such a representation. In particular, we desire models that can represent high dimensional functions with many variations using a tractable number of neurons. Not only are smaller networks less computationally demanding; they also have fewer parameters to train, and thus present a smaller search space in which fewer labelled examples are required to locate a good solution. There are several compelling arguments for why *deep* neural networks, composed of many layers of neurons, might outperform shallow neural networks in these regards [48], particularly for the archetypal AI problem of computer vision.

A sensible sounding plan for image recognition is to build a hierarchy of increasingly complex and abstract visual patterns, beginning with small local features such as edges and culminating in high-level semantic concepts such as "dog". A pattern at one level of such a hierarchy might be composed of several patterns detected by the level below. For example, one layer might detect lines, the next may recognize certain arrangements of lines as forming letters, and the next may recognize sequences of letters as forming words. Similar arguments apply to other cognitive tasks such as sentence parsing.

So intuitive is this hierarchical approach, that it appears regularly in much of the state of the art prior to deep learning (e.g. part-based models [49–51]). Breaking a problem down into a series of sub-problems is a very generic problem solving template, which occurs not only in human engineering but also in biology, for example the Golgi apparatus, whose stacked layers sequentially modify proteins, and in the primate visual cortex [52].

Another compelling argument for hierarchies of features is feature reuse. By exploiting the underlying similarities between related tasks (such as the recognition of different object classes), a model can both represent those computations more compactly and learn them more efficiently, requiring less data [48]. For example, an intermediate feature such as a horizontal edge may be used in the detection of multiple higher level features such as object parts, which themselves may each contribute to the detection of multiple whole objects. Even a binary classifier, which detects only a single type of object, should in theory benefit from feature reuse, because the overall task is composed of many related sub-tasks, computed by individual neurons, each of which shares features from the layer before. Feature reuse can also be seen as analogous to *structured programming* [53], in which code is written once, and encapsulated in functions that can be called many times. Not only does this result in more compact code, it also means that a

single update to a function can improve performance in all tasks that use it. This all suggests that deep architectures could make more efficient use of neurons (or whatever other computational units are used), compared to shallow architectures such as the three-layer perceptron discussed earlier.

Finally, there are some theoretical results proving that certain types of function can be represented more efficiently by deeper networks of computational elements than by shallower ones. For example, Johan Hastad proved that for all k, there exist $k+1$ depth boolean logic circuits with a linear number of gates with respect to input size, which require an exponential number of gates to simulate with a circuit of depth k [54]. Later, Hastad and Goldmann proved a similar result for networks of MP neurons [55]. Likewise, Bengio et al. [56] argue that a number of toy problems can be solved more efficiently by deep architectures than by shallow ones. More rigorous results are derived in [57], showing that a deep network can model a greater number of piecewise linear regions than a shallow network with the same number of hidden neurons.

Deep learning is often contrasted with classifiers based on hand engineered features, which was the dominant machine learning paradigm prior to deep learning (see [19]). Feature engineering levers expert domain knowledge to construct hard-coded functions of the input, whose values are likely to be immediately relevant to the classification task. The learnable part of such models is typically very small, for example a single layer of neurons with learnable parameters, or a nearest neighbour classifier, whose inputs are the hand crafted feature vectors. In machine vision, a typical example of such hand crafted features is Gabor filters. Deep learning replaces hand crafted features with many layers of learnable nonlinear processing, allowing the network to effectively learn its own features, directly from raw input.

2.5.1 The Vanishing Gradient Problem

Despite its promising motivations, deep learning failed to yield competitive results on standard benchmarks for many years. Although they were known to be more expressive than shallow networks, deep neural networks proved to be much harder to train. The main reason for this was eventually identified and became known as the vanishing gradient problem, formalized by Sepp Hochreiter in 2001 [58]. The problem is that gradients attenuate exponentially as they propagate through the network during backpropagation, resulting in very small magnitude gradients in the earliest layers of the network, particularly at the beginning of training (see Figure 2.8). The main culprit turned out to be the saturating activation functions used at the time, namely sigmoid and

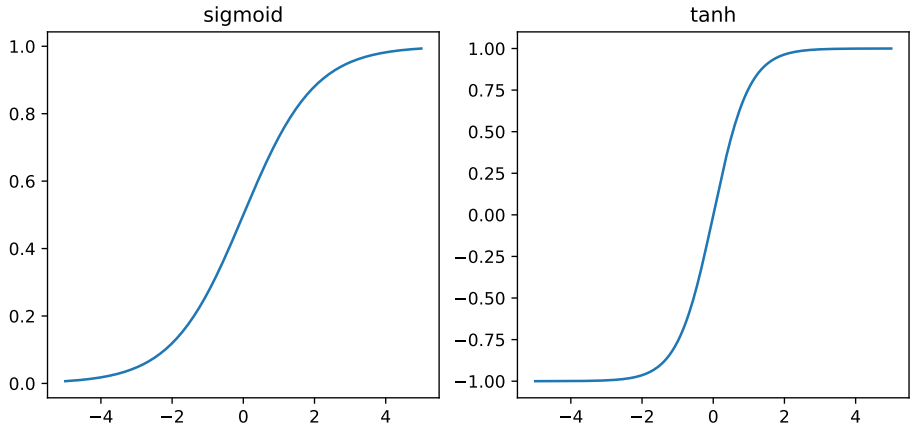


Figure 2.7: Sigmoid (a.k.a. logistic function) and tanh (hyperbolic tangent) are saturating nonlinearities, that is, their gradients are close to zero whenever their input is far from zero. This causes gradients to attenuate exponentially through layers of saturating neurons.

tanh. Saturating functions are bounded, which causes their gradient to approach zero as their input tends to infinity (see Figure 2.7). The gradients of these functions are in fact always less than one, and because the chain rule propagates gradients multiplicatively (see Equation 2.10), this means that gradients are multiplied by something less than one at every layer during the backward pass. An intuitive interpretation is that a small change to a weight in the first layer will produce a smaller effect in the activation of its neuron, which in turn will produce still smaller changes in its own downstream neurons, resulting in a vanishingly small effect on the network's output. It is therefore necessary to use non-saturating activation functions, the de facto standard now being rectified linear function ($\mathrm{ReLU}(x) = \max(0, x)$). By mitigating the vanishing gradient problem, ReLU activation can be considered one of the key enabling technologies in deep learning [17].

2.6 Convolutional Neural Networks

The quest for computer vision has been pursued since long before we had any idea how difficult it would be. We wish to build machines that can perform all the same functions as the human visual system; such a task could alternatively be described as "inverse computer graphics", that is, inferring properties of a 3D world from a 2D projection of it. Vision comprises a broad set of

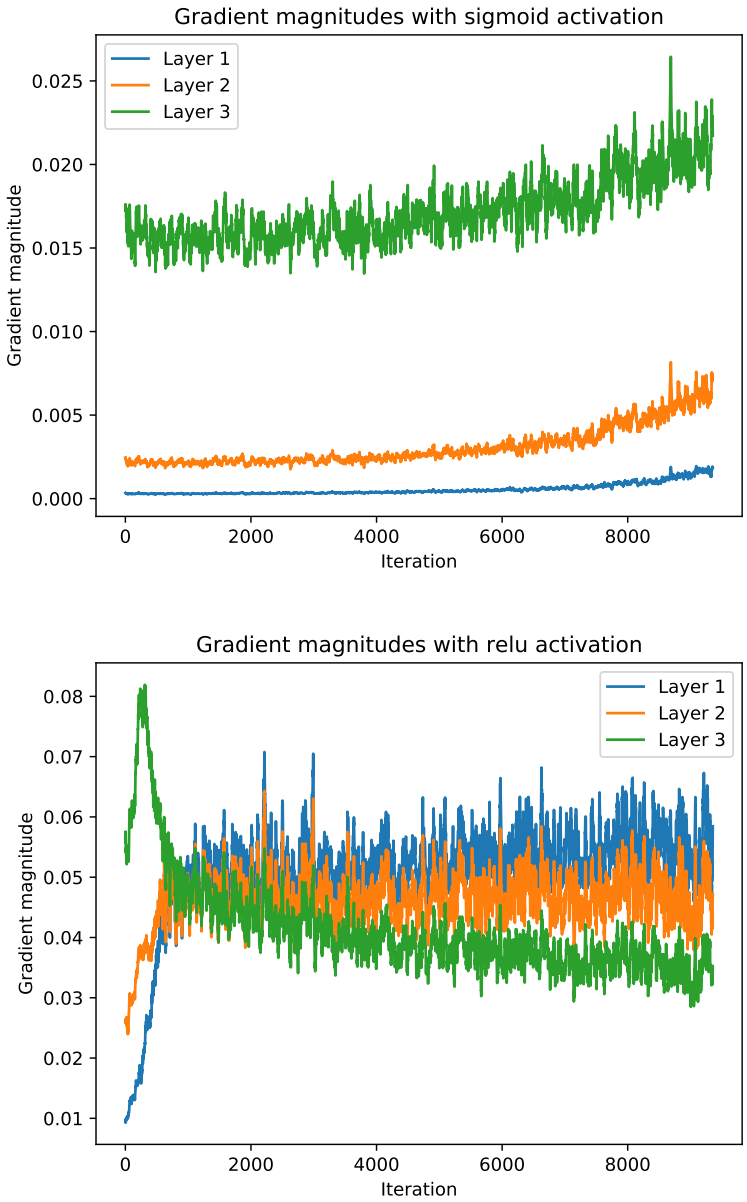


Figure 2.8: Top: Gradient magnitudes of bias vectors for the three hidden layers of a multi-layer perceptron, trained on MNIST (layers numbered from first to last in legend). Because of the saturating sigmoid non-linearity, earlier layers have smaller gradients than deeper layers, with first layer gradients close to zero at the beginning of training (as reported in [59]). Bottom: As above, but with recitified linear activation in place of sigmoid. First layer no longer suffers from vanishing gradients.

overlapping computational tasks, including object recognition, depth perception, estimation of 3D shape, tracking of moving objects, orientation of oneself in a 3D environment, handwritten text recognition, and inferring the properties of unrecognized objects by extrapolation from similar examples. All of this must be done in the presence of sensor noise, inconsistent lighting and shadows, partial occlusion of objects, and distraction by irrelevant stimuli. Prior to 2012, state of the art vision models generally approached this problem by extracting hand engineered feature vectors which were classified using a simple machine learning model. For example, the 2010 and 2011 winners of the ILSVRC image recognition challenge were both hand crafted feature extraction pipelines followed by support vector machines [19]. As discussed in the previous section, hand engineered features are often task-specific and require expert domain knowledge. More importantly, their effectiveness for image classification appears limited, such solutions often proving too brittle to see much real world deployment. If we are unable to hand craft suitable features for vision then it is natural to try ceding this task to machine learning; perhaps a deep neural network could discover more generalizable features by brute gradient descent? Learning machine vision directly from raw pixels poses certain challenges which are absent from simpler machine learning tasks:

- **Complexity of task** - vision tasks typically have very high intra-class variance. The raw pixel values in an image can change completely due to natural variations within a class, e.g. dogs of different breeds, trees of different shapes. Variations in illumination and object pose, if anything, are even more confounding. We therefore might expect that any function mapping pixel values to class labels must be very complex and non-linear, and is thus unlikely to be efficiently represented by shallow models such as three layer perceptrons.
- **High dimensionality** - every component of every pixel is a freely varying input dimension. The volume of the input space is therefore enormous, which makes the training samples very sparse - this is known as the curse of dimensionality.
- **Spatial locality** - unlike an input vector representing, say, a financial transaction, where individual elements have fixed meanings such as “amount of money” or “transaction type”, pixel values have very little meaning by themselves. There is practically nothing to be inferred from the intensity of a single pixel, all the information is contextual - only the pixel’s intensity relative to that of its neighbours matters. There is however a concept of locality and neighbourhood in images, which is absent from other types of input. The order in which the attributes of a financial transaction are represented in an input vector is irrelevant; the input vectors could be permuted in any way without removing any

information or making the task easier or harder (so long as the permutation is consistent). It is safe to assume however that (consistently) shuffling image pixels would make the task much harder (though not impossible, in fact [60]). With images, it actually matters if two input elements are adjacent, because adjacent pixels report the intensity of light incident upon the camera from similar directions in the 3D world. This locality principle is hierarchical, applying not just to pixels but to patterns made of pixels, for example, two circles next to each other might be a pair of eyes, whereas two circles in opposite corners of an image are probably independent.

- **Dimension hopping** - not only are individual pixels meaningless, but the absolute position of the pattern is also meaningless - an edge is an edge no matter where in the image it occurs. The translation of objects across the field of view can cause a pattern with the same meaning to be expressed through totally different input neurons [61]. Some form of translation invariance is needed, so that we don't have to learn to detect an object in one part of the image and then completely re-learn from scratch how to detect it in another part.

Clearly we desire a class of models that are deep enough to efficiently represent complex, high dimensional functions, and whose architecture somehow incorporates this prior knowledge of the spatial relationships between input neurons, the spatial locality of salient patterns, and the irrelevance of absolute position. Convolutional neural networks (CNN) are just that, and their debut in the ILSVRC challenge brought about an unprecedented drop in error rate (see Figure 2.9). They were originally proposed in 1980 [62], and showed promising results in handwritten character recognition in 1998 [26] (effective enough to be deployed reading cheques in commercial banks), but the real breakthrough came when they were successfully scaled up to a much larger computer vision task. The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) is an image classification benchmark on a 1.2 million image dataset spanning 1000 classes. The creators of ILSVRC had been wise to base it around such a large annotated image dataset, for the size of that dataset provided not only a higher bar for contestants to clear but also a unique resource for training machine learning models. It was the ability to absorb the information contained in 1.2 million labelled images into a single model that enabled the breakthrough achieved by CNNs.

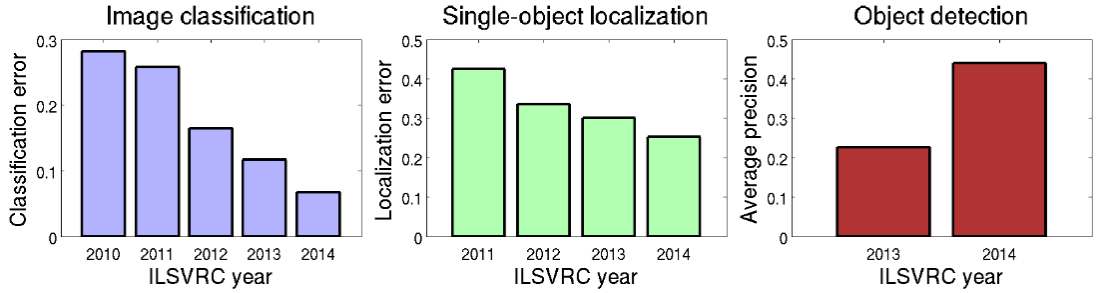


Figure 2.9: ILSVRC results, 2010-2014 [19]. Note the $\sim 33\%$ reduction in classification error in 2012, with the introduction of the first large scale CNN.

2.6.1 Architecture of Convolutional Neural Networks

Convolutional neural architecture is a combination of connectivity structure and weight sharing. In terms of connectivity, neurons are still arranged in layers, but the connectivity between those layers is much sparser. No longer is each neuron connected to every neuron in the previous layer; instead, each neuron is connected only to a small local patch, known as the *receptive field*, in the previous layer (see Figure 2.10, 2.11). For such a thing as a "local patch" to exist, neurons must have some notion of their position and neighbourhood. To this end, convolutional neurons are arranged in grids, so instead of a layer computing a vector of neuron activations as in the MLP case, convolutional layers compute activation matrices. These are commonly referred to as *feature maps*, or *channels*. Furthermore, convolutional layers in general compute multiple feature maps, therefore the outputs of convolutional layers are often referred to as *tensors* (though they are not tensors in the pure mathematical sense - 3D array would be a more accurate term). The input image should also be thought of this way - an RGB image is effectively an input tensor with three channels.

The location of the receptive field that a convolutional neuron connects to is governed by that neuron's position in its own feature map: moving one neuron to the right shifts the input field to the right by some fixed amount, likewise for vertical translations. The size of the receptive field (always rectangular) is called the *kernel size*, and the offset between the input fields of adjacent neurons is called the *stride*. Note that the sparsity only applies to the spatial axes, that is, convolutional neurons connect to all neurons in their receptive field regardless of which channel those neurons belong to. So, if a neuron has a 3×3 receptive field and there are 8 channels in the previous layer, then that neuron will have $3 \times 3 \times 8 = 72$ inputs, each with a learnable connection weight. It is worth noting that in the special case where the kernel is the same size as the input

image, a convolutional layer is equivalent to a fully connected layer. Since there is no room for the kernel to slide around, there can be only one receptive field, and hence no opportunity for parameter reuse, and since the kernel covers the whole image, there is no sparsity.

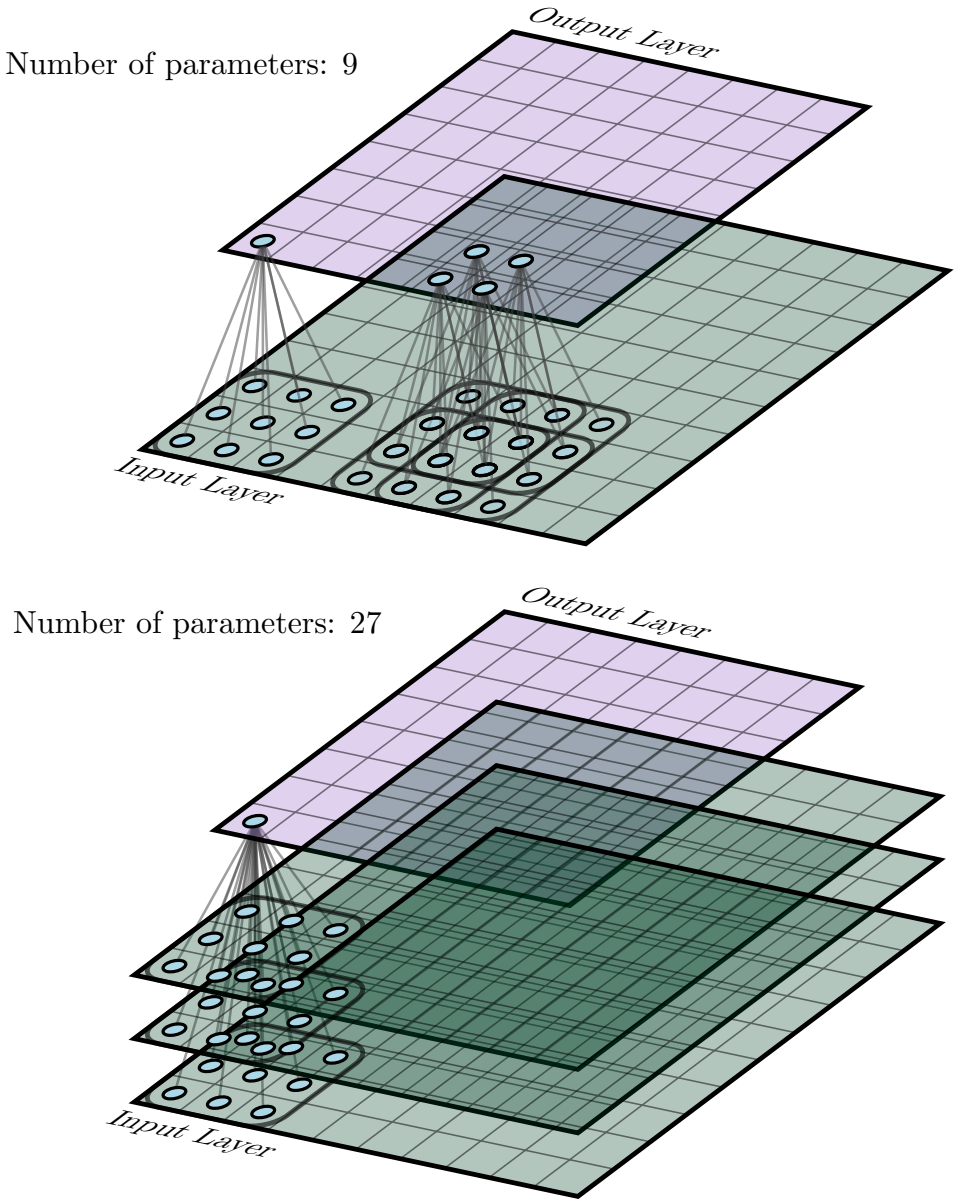
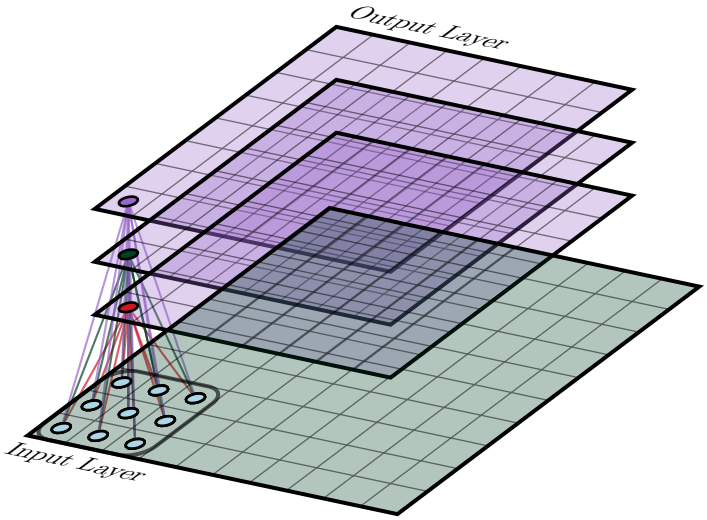


Figure 2.10: Top: convolutional layer with one channel in, one channel out. The kernel size is 3×3 and the stride is 1. All output neurons use the same 9 parameters, so there are 9 parameters in total. An output neuron and its receptive field are highlighted left, along with a group of four neurons right, to illustrate the fact that neighbouring neurons can have overlapping receptive fields. Bottom: convolutional layer with three channels in, one channel out. Since convolution kernels span across channels, each output neuron takes input from 27 neurons. Because parameters are shared across spatial axes but not across channels, we need three times as many parameters now, so there are 27 parameters in total.

Number of parameters: 27



Number of parameters: 81

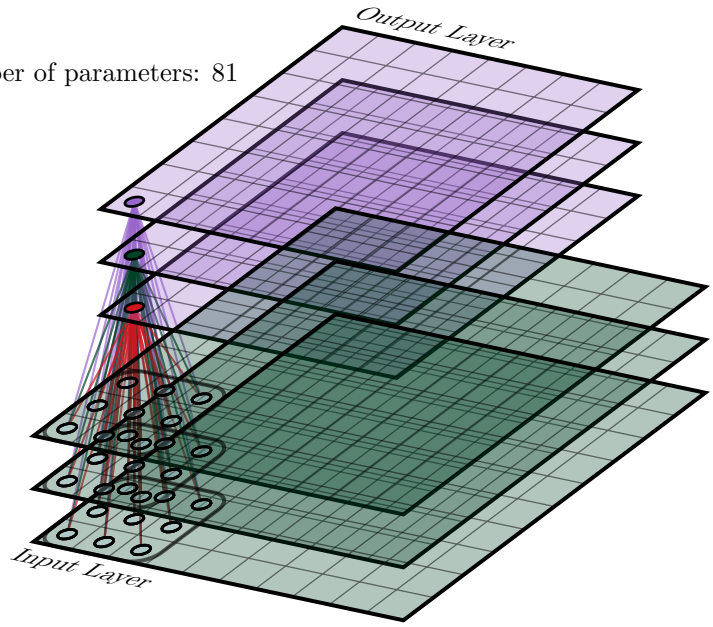


Figure 2.11: Top: convolutional layer with one channel in, three channels out. The three output chanels each have their own 9 parameters, so the total number of parameters is 27. Bottom: convolutional layer with 3 channels in, 3 channels out. Each output channel has 27 parameters, so the total number of parameters is 81. Edges of the same colour connect input channel neurons to the same output channel neuron.

In terms of weight sharing, neurons in the same feature map use the same weights, and the only thing that differs is the location of the receptive field in the previous tensor, as described above. This means that neurons in the same feature map all scan for the same pattern in the input layer, but at different locations. Furthermore, this means that if a neuron learns to detect a certain pattern in one part of the image, the other neurons in the same feature map automatically learn to detect it everywhere else. During training, we pretend that each convolutional neuron had its own weights and compute their gradients via backpropagation as usual, as though we were unaware of weight sharing. This generally results in different weight gradients for different neurons in the same receptive field, even though their weights cannot be independently updated. We remedy this by averaging those gradients together and applying them to that feature map's single underlying set of shared parameters, which is called a *convolution kernel*.

In nearly all cases, CNNs are deep neural networks with many convolutional layers. The earliest notably successful CNN, LeNet [26], had two convolutional layers (followed by two fully connected layers). AlexNet had five, VGG had up to 19 [63], GoogLeNet (also known as Inception) had 22 [12], and Residual CNNs (ResNets) can scale as large as 1,000 layers [64]. These models are all winners of ILSVRC for the years 2012 - 2015, respectively. This clear link between depth empirical accuracy appears to fit well with the hypothesis developed in the previous section, that deeper models can represent more complex functions efficiently. It also conforms with the intuition that deeper models have more learnable parameters and thus a greater capacity for learning.

As one follows the chain of layers through a CNN from input image to output vector, it is generally found that deeper convolutional layers have smaller spatial resolution but greater numbers of feature maps. Layers that reduce spatial resolution are called pooling layers, and generally reduce both the height and width by half, followed by or simultaneous with a channel increase of up to two times. The total size of these intermediate tensors therefore decays exponentially as one moves through a CNN, beginning with a 224×224 RGB image (in the case of AlexNet) and culminating with a vector of class probabilities. This is the reason why stacks of convolutional layers are so often depicted as trapeziums in diagrams, to denote the progressive reduction in representation size. It can be seen as a process of discarding irrelevant information at each step, until the only information remaining is that which correlates most strongly with class label. Pooling layers are convolution-like operations with a stride greater than one, meaning that instead of placing the kernel at every possible position in the input, they skip every nth location (usually $n = 2$). This results in a downsampled output. The convolution-like operation is typically

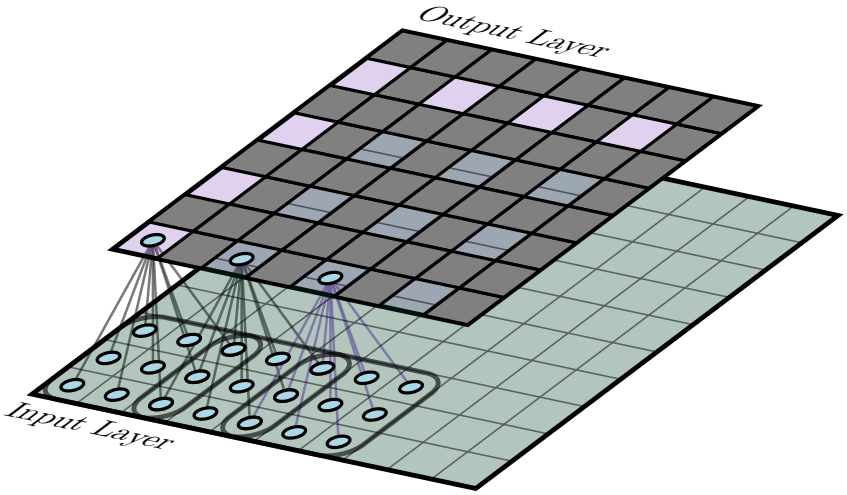


Figure 2.12: 3×3 convolution with a stride of 2. Since every second position is skipped on both axes, the output will now be 4×4 rather than 8×8. The grey squares represent elements that are computed in stride 1 convolutions but not stride 2 and do not represent actual null values in the output; the three output neurons highlighted are adjacent. Note that the receptive fields of these three neurons span a larger area than if the stride had been 1 (as in Figure 2.10, top).

max pooling (take the maximum inside the receptive field rather than a weighted sum), average pooling (used in the final layers of GoogLeNet and ResNets), or just convolution (which was eventually found to work just as well as max pooling [65]). These "convolution-like" operations work by tiling the same operation over a grid of input patches - it is the spacing between these patches, called the stride, that determines the degree of downsampling (see Figure 2.12). Pooling layers not only reduce the size of subsequent layers, allowing us to compute more layers for less time and memory, they also exponentially grow the effective receptive field (EFR) size of subsequent layers, which increases only linearly with regular convolution. By EFR, we mean the input patch that is causally linked to a neuron's activation, which can be found by recursively tracing a neuron's inputs back to the input image. Expanding the EFR is crucial if we wish CNN layers to learn a hierarchy of increasingly large and complex patterns - there can be no dog detecting neuron if no neuron's EFR is large enough to cover a dog.

Where do the words "convolutional" and "kernel" come from? Convolution is a mathematical operation that computes weighted sums of a function f according to a weighting mask g (the kernel) that is translated across the convolved function. The convolution itself (the output of the operation) is a function of the offset by which the kernel is translated. In the one dimensional

continuous case, this can be written:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau) g(\tau - t) d\tau \tag{2.13}$$

where f is the function to be convolved, g is the kernel, τ is a dummy integration variable and t is the offset of the kernel. In CNNs, the operation must be discrete and two dimensional, because the image and kernel are both discrete 2D arrays of pixels. Furthermore, since the kernel has small support (typically 3×3 pixels), we do not need to sum over infinity. Thus,

$$(f * g)(x, y) = \sum_{y'=-h}^{h} \sum_{x'=-w}^{w} f(x', y') g(x' - x, y' - y) \tag{2.14}$$

$$= \sum_{y'=-h}^{h} \sum_{x'=-w}^{w} f(x' + x, y' + y) g(x', y') \tag{2.15}$$

where g is nonzero for $x \in [-w, w], y \in [-h, h]$. Convolution has seen many practical applications in image processing (where they are often called filters), for example, convolving an image with a Sobel filter produces a response image which highlights horizontal or vertical edges. This illustrates how convolution kernels can act as selective detectors for different patterns; generally, parts of the image that match with the kernel will produce high response.

Chapter 3

Avoiding Overdetection: Towards Combined Objected Detection and Counting

Prologue

In Chapter 2, we have provided a basic introduction to deep neural networks and their role to computer vision. This knowledge will be key to understanding the contributions that we develop in this and subsequent chapters. In this chapter, we present a contribution in object detection, the task of localising and classifying multiple objects in an image. Like other vision tasks, object detection is dominated by CNNs. Rather than emitting a single probability distribution for the whole image, detection CNNs are modified to emit the coordinates of multiple object bounding boxes along with (optionally) a class probability distribution for each. They are frequently used for tasks such as autonomous driving, object tracking, automated wildlife observation and object counting.

This chapter focuses on the closely related less frequently studied problem of object counting. Existing object detection frameworks in the deep learning field generally over-detect objects, and use non-maximum suppression (NMS) to filter out excess detections, leaving one bounding box

per object. NMS prunes detection boxes whose overlap is above a certain threshold, leaving only one from each group of overlapping boxes. This works well so long as the ground-truth bounding boxes do not overlap heavily, as would be the case with objects that partially occlude each other, or are packed densely together. This is frequently the case with cell counting in bioimage analysis - a common, labour intensive task which in recent years is increasingly automated by deep learning (e.g. [66, 67]). In these situations, the NMS stage may wrongly remove bounding boxes where cells are genuinely overlapping, resulting in an incorrect cell count. Clearly it would be beneficial, and more elegant, to have a fully end-to-end system that outputs the correct number of objects without requiring a separate NMS stage. This would allow detection networks to be reliably used to simultaneously count and localise objects. Given the ability of CNNs to accomplish difficult vision tasks such as 1000-way image classification, it seems conspicuous that the task of marking each object with a single box is apparently so difficult, no matter how simple the objects were.

In this chapter, we investigate the causes of this difficulty, and propose a modified loss function that is shown to partly overcome it.

Declaration: This chapter is based on the following publication: Jackson, P. T. & Obara, B. *Avoiding Over-Detection: Towards Combined Object Detection and Counting* in *International Conference on Artificial Intelligence and Soft Computing* (2017), 75–85. This chapter is presented largely as accepted, although referencing and notation have been altered and cross-referencing added for consistency across this thesis. Some stylistic changes have been made for consistency. The majority of the text is verbatim, with some minor wording and formatting changes.

3.1 Introduction

Object detection is the task of localising and classifying all objects present in an image [68]. While the field of deep learning has produced many object detection networks with excellent true positive rate, they tend to suffer from low precision, i.e. high false positive rate. Usually the network outputs many bounding boxes per object, and these over-detections are filtered by non-max suppression (NMS) [20], leaving (hopefully) one box per object. NMS is a fixed post-processing step that is not learnt from the data, and typically relies on a user-chosen overlap threshold (for example, 0.7 used in [13]). Furthermore, NMS is unaware of the contents of the boxes it prunes, and so has no way to know if the ground-truth boxes really do overlap.

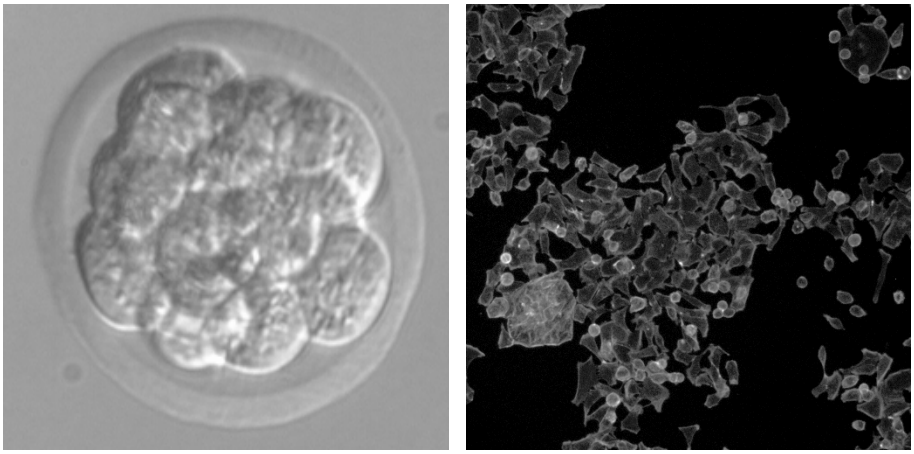
The question arises of how it may be possible to train a deep neural network to output exactly one box per object, without the need for a separate non-learned filtering step. Aside from being more elegant, this approach may have potential for greater accuracy, particularly in the case of detecting many small, densely clustered objects. In these cases, traditional NMS may struggle to tell if two boxes overlap because they are localising the same object or if they are localising different objects which are very close. This is especially true when objects of the same class are not only close but genuinely do overlap. With very high numbers of densely packed objects, another problem may also emerge: because detection networks emit a fixed number of boxes, it may become necessary to coordinate these boxes such that they are properly distributed among the many objects present. Over-detection in these cases may not only raise the false positive rate, but also lower the true positive rate; if there are only enough boxes to detect everything once then over-detecting one object may leave no boxes for another.

Close and overlapping objects occur in pedestrian crowd footage, autonomous vehicle visual feeds, and histological images from biomedical microscopy, such as those in Figure 3.1. In this chapter we choose cell microscopy as a test case, and use the Simulating Microscopy Images with Cell Populations (SIMCEP) [69] system to generate large quantities of synthetic images with perfect ground-truth annotation for training and testing. The simplicity of this benchmark, which can be solved to reasonable accuracy without deep learning [70], allows us to focus solely on the over-detection problem. SIMCEP allows the user to generate artificial cell populations with varying degrees of clustering and overlap, and so makes an excellent testing ground for a dense object detection framework. Using simulated images allows us to generate essentially unlimited quantities of training data, bypassing the scarcity of labelled data that is normally the biggest constraint when training deep networks to solve bio-imaging problems. It is hoped that systems trained on SIMCEP images may still be applicable to real-world histological images via transfer learning. Fluorescence microscopy image analysis often requires objects to be counted as well as localised, so a one-box-per-cell system, which can be seen as combined localisation and counting, would be quite relevant in this field.

3.2 Related Work

3.2.1 Deep learning methods for object detection

Object detection in deep learning is largely dominated by the Region Convolutional Neural Network (R-CNN) family of models. The original R-CNN [72] uses a selective search based method [73] to propose interesting-looking regions, only using the CNN to generate feature



(a) (b)

Figure 3.1: a) mouse embryo, an extreme case of overlapping objects consisting of a ball of around 20 cells. b) Human HT29 colon cancer cells, packed very closely. Both images from the Broad Bioimage Benchmark Collection [71].

vectors for each region and a support vector machine (SVM) approach to then score them for each class. Fast R-CNN [74] is an iteration on this work, speeding the process up mostly by generating all convolutional features for the image in a single pass and pooling sub-sets of them for different region proposals, rather than running each proposed region through the CNN separately. Faster R-CNN [13] improves further by using the same convolutional network for both proposing regions and classifying their contents. This saves computational time and results in slightly more accurate bounding boxes, as well as being a more elegant system. Almost the whole pipeline is performed by the network, only the NMS is done separately.

Faster R-CNN is a fully convolutional network (FCN), so images of arbitrary size can be passed and the feature maps will grow or shrink accordingly. The final convolutional layer outputs feature vectors describing overlapping square regions in the image; these are used by the region proposal network (RPN) to predict a fixed number of bounding boxes per region. The RPN's output tensor consists of multiple "detectors": groups of neurons representing bounding box parameters and confidence levels. Each output box is described relative to a different fixed "anchor" box. The anchors are Faster R-CNN's answer to the problem of expressing an unordered set of boxes with a fixed-size tensor. The loss function must decide at training time which boxes from the RPN are to match with which ground-truth boxes, and which boxes should have high

class probability (i.e. the RPN's confidence that that box contains an object). In practice, all boxes whose anchors overlap sufficiently with a ground-truth box are trained to have high class probability and incur regression loss on their deviation from the ground-truth box. Output boxes whose anchors do not overlap sufficiently with any ground-truth box only incur loss for having high class probability. This can be seen as giving each detector a different "jurisdiction", in which it is responsible for matching any ground-truth box with a certain position, aspect ratio and size. Unfortunately, the anchor boxes must be dense and overlapping to ensure there will be at least one per object, so it is normal for multiple anchor boxes to overlap one object, therefore the loss function will normally train multiple detectors to label the same object.

Another relevant detection framework is YOLO [75], which differs from Faster R-CNN principally in that it is not an FCN. Although this requires that images are resized to a fixed dimension before processing, it also means that the feature maps are of constant size. This allows the final layer to be converted to a fixed-size vector that describes the entire image, in a similar manner to AlexNet [76]. This allows the classifier to make use of global image context, resulting in higher accuracy compared to Faster R-CNN, whose classifier only pools convolutional features from within the proposed bounding boxes. YOLO assigns responsibility to its output boxes in a different way to Faster R-CNN. Unlike Faster R-CNN, the jurisdictions of the detectors are not pre-defined, rather, responsibility for detecting a given ground-truth box is assigned at training time to whichever detector outputs a box with the greatest intersection over union (IoU) with that box. The authors claim this leads to detectors learning to specialise in different sizes, aspect ratios and classes of object.

Although the above methods excel at detecting small numbers of large objects in datasets such as Pascal VOC [77], they are less well tested on large numbers of small objects. In particular, they all tend to over-detect objects, outputting many bounding boxes which must then be pruned by NMS to leave only one box per instance. The problem of learning to count has been explicitly investigated in [78], whose authors show that a network trained only on the multiplicity of a target object type will learn features that are also useful for classification and localisation of said objects. Although the results are encouraging, they do not tackle the problem of coordinating object detectors to output exactly one bounding box per ground-truth object.

3.2.2 Deep learning methods for cell detection

The greatest obstacle in applying deep learning approaches to biomedical image processing is the scarcity of labelled training data. Deep neural networks generally require many thousands of

labelled images to train effectively, but individual problems in biomedicine tend to avail neither thousands of images nor enough trained experts with the time and inclination to label them all. Many proposed methods [79–81] circumvent this problem by using CNNs to perform pixel-wise binary classification. These networks take small image patches as input and output the probability of the central pixel in the patch being part of a target object. Although this is a harder task than whole-image classification, it can yield thousands of training examples per image, since each pixel and its neighbourhood becomes an example in the training set. For example, [82] trains a CNN to identify the central voxels of zebrafish dopaminergic neurons in 3D images. This is part of a larger pipeline, which first uses an SVM to narrow down the set of potential voxels, so that the CNN need not be applied to every possible location in the image. The output probability map is then smoothed and individual cells are detected as local probability maxima. Prentasic et al. [81] use a CNN to detect lipid deposits in retinal images, by classifying the central pixel of 65×65 image patches. Since these deposits are diffuse, amorphous objects, pixel-wise classification is appropriate here and there is no attempt to count the number of deposits present.

Kraus et al. [83] train an FCN to classify histological images at a whole-image level. Although it is only trained with whole-image labels, it is still able to localise individual cells by deriving class probability maps from the final convolutional layers, in a manner inspired by [84] and [85]. FCNs are particularly useful when processing histological images due to their ability to naturally scale to images of arbitrary size, without needing to downsample large images to a fixed size. Litjens et al. [79] train a standard CNN to classify the central pixel of image patches, then convert it to an FCN to perform pixel-wise classification over a whole image in one pass. This has performance benefits over processing patches one-by-one, since computations can be shared among overlapping image patches.

A standard CNN based on the design of Krizhevsky [76] is used to count human embryonic cells in [86]. Since the cells in these images show very high overlap, the act of counting is treated as a classification task and the cells themselves are not localised.

3.3 Method

When attempting to design a network that produces output of variable length, one immediately hits two technical limitations:

- Existing deep learning frameworks process data in "tensors", N-dimensional arrays whose shape is always a hyperrectangle. This includes the output tensor. Deep neural networks generally process images in batches to make optimal use of parallel processing hardware, therefore this output tensor is of rank 3 - a stack of matrices, one for each image, with one row per output box in each matrix. Outputting a different number of boxes for each image in a batch would therefore require these matrices to each have a different number of rows, thus the tensor could no longer be a hyperrectangle.

- In order for the network to learn the correct number of boxes, this number needs to be somehow differentiable. That means the number of boxes produced must vary smoothly with respect to the network parameters; a small parameter change should result in a small improvement in the number of boxes.

These constraints can be satisfied by outputting a fixed number of boxes with confidence scores attached - as is the case in existing detection frameworks. This fixed number is chosen to be reliably greater than the true number of objects, so that there will always be at least enough boxes available to cover them. It is possible for such a format to represent one box per object, by assigning confidence scores such that each object gets exactly one high confidence box that matches its corresponding ground truth box. The question then becomes how to learn such behaviour.

3.3.1 Loss Function

To train a network to behave in such a way, a loss function is required that is minimised if and only if the network outputs exactly one matching box with high confidence for each ground-truth box. This is difficult, because the order in which the boxes are emitted should not matter. Loss functions in supervised learning work by penalising the deviation between a network's output and the desired output, but if a network emits N output boxes per image and an image has M objects, then there are $\frac{N!}{(N-M)!}$ possible correct outputs, corresponding to different orderings of the boxes. This combinatorially large number of possible correct outputs means that it is not obvious which output the network should converge on for a given input. The gradients for a detection task will draw the network's output towards a different pole depending on which of the correct outputs it is closest to, resulting in unstable training dynamics. Loss functions must therefore be carefully designed to minimise this problem.

Faster R-CNN defines an anchor box for each detector, whereby the loss function demands that a box should have high confidence if a ground-truth box falls into its jurisdiction. Similarly, YOLO's loss function assigns responsibility to whichever detector's box has the greatest overlap with the ground truth box, which results in detectors learning their own jurisdictions.

Ideally, we would like the loss function to be minimised no matter which detectors are used to label the objects, so long as there is only one each. To this end, we define a loss function that assigns responsibility for ground-truth boxes based on both the output box parameters (centre coordinates, width and height) and confidence scores. We define a responsibility matrix R, where R_{ij} is the responsibility of detector i for object j, and

$$R_{ij} = \frac{C_i}{D_{ij}} \tag{3.1}$$

$$D_{ij} = (x_i - x_j^*)^2 + (y_i - y_j^*)^2 + (w_i - w_j^*)^2 + (h_i - h_j^*)^2 \tag{3.2}$$

where x, y, w, h are centre coordinates and width and height, normalised to $[0, 1]$ relative to the image dimensions and mean box size, respectively, $*$ denotes ground-truth, and C_i is the confidence of detector i.

At training time, each ground-truth box j selects the detector that is most responsible for it:

$$R_j^* = \arg\max_i R_{ij} \tag{3.3}$$

This chosen detector incurs a regression loss $D_{R_j^*,j}$, causing it to better localise the object for which it was responsible. All detectors also incur regression loss on their confidence, where target confidence C_i^* is 1 if detector i is responsible for an object, and 0 otherwise. The total loss is then:

$$L = \frac{1}{N}\sum_i (C_i - C_i^*)^2 + \frac{1}{M}\sum_j D_{R_j^*,j} \tag{3.4}$$

where M and N are the number of ground-truth objects and detectors, respectively, and $N > M$. This responsibility scheme is similar to that used by [75], but differs in that ours takes into account box confidence, allowing it to penalise over-detection if too many high confidence boxes are emitted.

Using detector confidence to establish responsibility allows the network to choose for itself which detector will be responsible. If detectors $1 - 5$ localise object j, then their regression losses D_{ij}

for $i = 1..5$ will be similar, and so the highest responsibility will go to detector k with highest confidence C_k. This chosen detector will get a target confidence C_k^* of 1 while the others get 0. This reinforces detector k as the detector responsible for that object; next time the same object is seen, C_k will be higher, while others will be lower. This can be seen as a kind of learnt NMS.

We found that if confidence is not used to determine responsibility ($R_{ij} = \frac{1}{D_{ij}}$), the network outputs many boxes per object which all have roughly equal confidence well below 0.5. This is because the network cannot predict which box will be closest to the ground-truth since they are all close, and so cannot predict which should have confidence 1 and which should have 0. Moving all but one box away from the object would be a solution, but this would only produce discontinuous, non-differentiable changes in loss as the responsibility assignment changes suddenly, so the network cannot learn to do this.

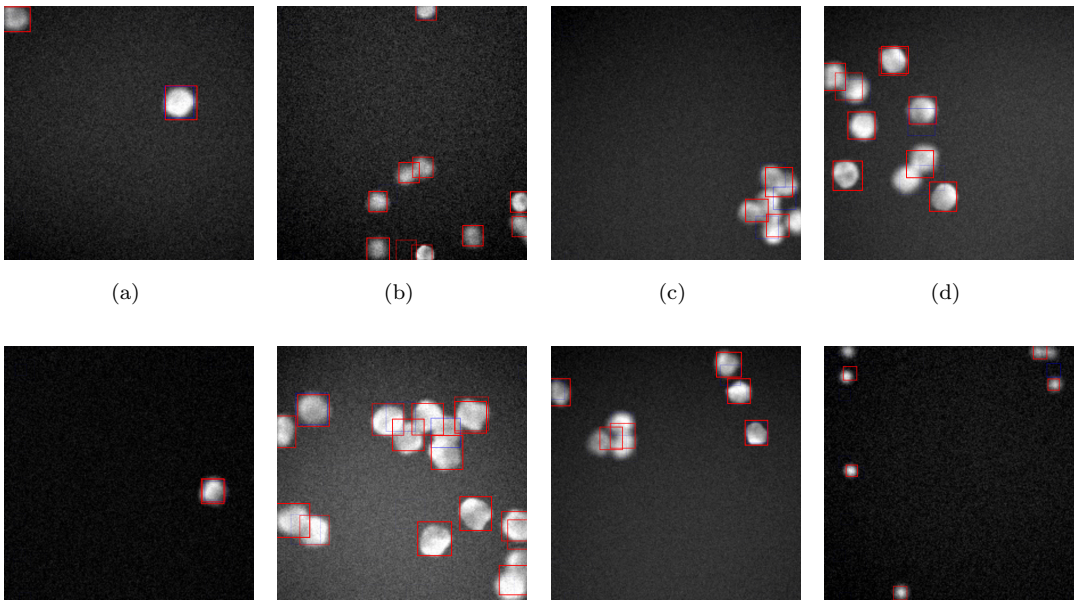
3.3.2 Model Architecture

A recurrent neural network (RNN) would be the obvious choice to minimise the loss function described above. If bounding boxes are emitted sequentially rather than simultaneously, then each one can be dependent on the ones that came before it. In this way, a detector can avoid outputting a high confidence box on an object that has already been detected. Despite this attractiveness though, our best results out of the many architectures trialled came not from an RNN but from an FCN. This architecture is specified in Table 3.1.

Everything from `conv1` to `conv7` is a relatively standard convolution / maxpooling stack, with some slightly unusual features (stride of 2 in `conv6`) which allow the stack to output feature maps whose effective receptive fields in the image overlap by half (effective receptive field size is 64×64 pixels, effective stride is 32×32). This overlap ensures that every object lies fully within at least one neuron's receptive field. `boxes` emits bounding box parameters and `boxes_global` is a custom layer that performs a simple transformation from local coordinate space global image space. `concat` joins the feature maps of `boxes_global` and `conv7`, allowing the remaining three layers to predict confidence scores based on both the boxes themselves and the image features they were predicted from. We observed a modest improvement in performance due to this addition. The final three layers, then, can be seen as a learnt filtering stage that replaces the traditional NMS post-processing. A Theano/Lasagne implementation is available at https://github.com/philipjackson/avoiding-overdetection.

Network Layers		
Name	Type	Parameters
conv1	Convolution	num_filters=32, filter_size=(5,5)
pool1	Maxpool	pool_size=(2,2)
conv2	Convolution	num_filters=48, filter_size=(3,3)
pool2	Maxpool	pool_size=(2,2)
conv3	Convolution	num_filters=64, filter_size=(3,3)
pool3	Maxpool	pool_size=(2,2)
conv4	Convolution	num_filters=86, filter_size=(3,3)
pool4	Maxpool	pool_size=(2,2)
conv5	Convolution	num_filters=128, filter_size=(1,1)
conv6	Convolution	num_filters=128, filter_size=(2,2), stride=(2,2)
conv7	Convolution	num_filters=128, filter_size=(1,1)
boxes	Convolution	num_filters=4*B, filter_size=(1,1), nonlinearity=identity
boxes_global	Coord Transform	
concat	Concatenation	inputs=boxes_global,conv7
filter1	Convolution	num_filters=16*B, filter_size=(3,3)
filter2	Convolution	num_filters=16*B, filter_size=(1,1)
confidence	Convolution	num_filters=B, filter_size=(1,1), nonlinearity=sigmoid

Table 3.1: A specification of our network architecture. Unless otherwise stated, each layer takes the previous layer's output as input. Nonlinearities are leaky rectified linear [87] with $\alpha = 0.1$ unless otherwise stated. B is a hyperparameter denoting the number of detectors per "window" (i.e. position in the final feature map, conv7). $B = 9$ in our experiments.



(a) (b) (c) (d)

(e) (f) (g) (h)

Figure 3.2: A sample of detection results on SIMCEP images. Confidence is represented in the transparency of the boxes; all output boxes with confidence above 0.1 are shown. Instead of post-processing with NMS, we simply take boxes with confidence above 0.5 (shown in red) as positive detections. Boxes with confidence below 0.5 are shown in blue.

3.4 Results

We trained our model on a set of 17000 SIMCEP images using the Adam optimizer [88], and validated against a set of 3000 images. The images were of size 224×224 pixels and contained anywhere from 1 to 15 cells. The parameters of SIMCEP were adjusted to randomise obfuscating features such as blur, Gaussian noise and uneven lighting, and the cells show varying levels of clustering and overlap.

A selection of results is shown in Figure 3.2. We interpret any detection with a confidence above 0.5 as a positive, and so the number of such detections is the network's estimate of the number of cells present. Across our validation set, the root mean square of the deviation of this estimate from the true count was 2.28. Further quantitative results are shown in Table 3.2.

	True Positive Rate	False Positive Rate	F_1-score
Training Set	75.4%	19.2%	0.774
Validation Set	75.3%	19.4%	0.773

Table 3.2: True and false positive rates on training and validation sets. A true positive is counted as any output box with an intersection over union (IoU) above 60% with a ground-truth box, but each ground-truth box can only be paired with a single output box. So if two output boxes cover the same object, then this counts as one true positive and one false positive. Output boxes with less than 60% IoU with any ground-truth box are always false positives.

3.5 Conclusion

For images containing objects whose bounding boxes overlap heavily due to occlusion or dense clustering, NMS cannot reliably remove excess bounding boxes emitted by the network, since ground-truth bounding boxes with identical classes may truly overlap significantly. An end-to-end system that outputs the correct number of boxes without the need for post-processing NMS is therefore preferable. In this paper, we discuss the problem and take some early steps towards solving it, demonstrating a system that can localise densely clustered objects and simultaneously approximate the correct number of boxes. Rather than performing regression directly on the number of objects, we encode this number implicitly in the number of high confidence boxes emitted by the network.

We propose that, unless an alternative output encoding can be found which shows a one-to-one mapping between output values and unordered sets of boxes, supervised learning itself is unsuitable for this task. There are many ways for a network to output the same set of boxes, depending on which box it places on which object (or for an RNN, which order it outputs them in), but supervised learning requires us to arbitrarily choose one of them, and penalise all the others. In this paper we partially solve this problem by choosing which box should have high confidence based partly on the confidence values themselves, however the results are not perfect. We put forward three reasons for this, and suggest how they may be countered by applying reinforcement learning instead of supervised learning.

The first reason is that, because we assign responsibility for an object to the detector with the maximum responsibility for it, our loss function is discontinuous, due to the arg max operation. This is likely to cause problems for supervised learning, which is based on direct optimization of the loss function by gradient descent. Reinforcement learning trains a network to optimize a reward function which may be related to the network's output in a complex, non-differentiable

or even unknown way. In particular, reward functions do not prescribe a target output for every input, and so they completely bypass the problem of choosing which detector should label which object. This makes reward functions a much more natural way to express the goal of one box per object.

The second reason is that in order to teach a network to output the right number of boxes, that number must somehow be made smooth and differentiable, despite the fact that we ultimately want an integral number. To derive this hard number, we currently threshold the confidence levels at 0.5. Not only is this threshold somewhat arbitrary, but worse still, it is effectively a post-processing step that the network itself is unaware of, and indeed cannot be trained to optimize because it is non-differentiable. This too can be solved with reinforcement learning by building the thresholding step into the reward function, because discontinuous reward functions are allowed.

The third reason is that our FCN architecture outputs all the boxes in parallel. This means that each detector is unaware of what the others are doing, so it is difficult for them to coordinate themselves so as to avoid over-detection. Using an RNN that outputs boxes in series would solve this problem, as the output on one time step can be conditioned on that of previous ones. This also fits well with reinforcement learning, since a reward signal can be administered on every time step; this would accelerate training compared to a single overall reward signal per image.

Epilogue

The approach presented in this chapter succeeds in labelling isolated cells with a single box in most cases, which to the best of my knowledge is the first time a feed forward neural network has accomplished such a thing. However, in retrospect, it became clear that there is a fundamental limitation that prevents feed forward networks from solving this task fully. It seems likely that the underlying reason why emitting a single box per object is difficult is that marking objects is fundamentally a serial process, and cannot easily be performed in parallel. Since the detectors in the final layer of this chapter's network compute their activations in parallel, there is no way for them to communicate among each other. The situation is analogous to requiring a group of people to learn to each choose a different object, in such a way that each object will always be selected exactly once - no matter how many objects there are, and where they are placed. They may learn rules over time, each specialising in detecting objects in a particular location or of a particular type or scale, and these rules may prevent two people from labelling the same object,

so long as it only falls into one person's jurisdiction. But this is only reliable if the jurisdictions never overlap, in which case any objects which share the same jurisdiction will be underdetected, since they are only covered by a single detector. Furthermore, there will always be edge cases in which an object straddles the boundary between two jurisdictions, resulting in ambiguity in which person should label it.

Chapter 4

Style Augmentation

Prologue

Another major obstacle limiting the practical applicability of CNNs is the problem of domain bias. In this context, the word "domain" refers to a probability distribution of images, for example, dashcam images from a vehicle driving in the day and at night are a common pair of domains seen in domain bias literature. The term is loosely synonymous with the word "dataset", since any dataset implies a probability distribution from which its images are drawn, and so the two terms are sometimes used interchangeably. Domain bias, then, is the problem of models becoming biased to the domains in which they are trained, performing well on unseen images from that fit the domain on which they are trained but failing to generalize to out-of-domain samples. Domain bias can occur even when the differences between images from the two domains are barely perceptible to the human eye [89]. This brittleness is a recurring problem in deep vision, particularly when deploying trained models in the real world.

Domain bias is a major risk to any project that aims to deploy a trained model in the real world, because any differences between the distribution of images in the training dataset and the real world present a risk of poor generalization. There is an extensive body of literature dealing with the subject of "domain adaptation" (see Section 4.2.1), in which a trained model is modified post-training to improve its accuracy on a specific target domain. These methods are often effective, but suffer from two drawbacks: firstly, they require access to the target

domain in order to perform the adaptation before deployment, which is not always possible. Secondly, domain adaptation is specific to one target domain. It is a patch to improve a model's performance in one area, but does not result in a more generally robust model.

In this chapter, we propose a novel data augmentation method called Style Augmentation, for improving the domain robustness of CNNs during training. Data augmentation is a family of techniques for artificially expanding a training set by creating new samples from existing ones via label preserving transforms. Style Augmentation uses neural style transfer as the label preserving transform. Unlike other data augmentation transforms, style transfer preserves the overall shape and geometry of an image while changing the texture, colour and contrast. By training the model on images with randomized texture and colour, we force it to rely on shape instead. This improves generalization to many domains at once without requiring targeted domain adaptation, because object shape is more consistent across domains than is texture.

Declaration: This chapter is based on the following publication: Jackson, P. T., Atapour-Abarghouei, A., Bonner, S., Breckon, T. & Obara, B. Style Augmentation: Data Augmentation via Style Randomization. *CVPR DeepVision* (2018). This chapter is presented largely as accepted, although referencing and notation have been altered and cross-referencing added for consistency throughout this thesis. Some stylistic changes have been made for consistency. The majority of the text is verbatim, with some minor wording and formatting changes.

4.1 Introduction

Whilst deep neural networks have shown record-breaking performance on complex machine learning tasks over the past few years, exceeding human performance levels in certain cases [11], most deep models heavily rely on large quantities of annotated data for individual tasks, which is often expensive to obtain. A common solution is to augment smaller datasets by creating new training samples from existing ones via label-preserving transformations [90].

Data augmentation imparts prior knowledge to a model by explicitly teaching invariance to possible transforms that preserve semantic content. This is done by applying said transform to the original training data, producing new samples whose labels are known. For example, horizontal flipping is a popular data augmentation technique [28], as it clearly does not change the corresponding class label. The most prevalent forms of image-based data augmentation include geometric distortions such as random cropping, zooming, rotation, flipping, linear intensity

Figure 4.1: Style augmentation applied to an image from the Office dataset [93] (original in top left). Shape is preserved but the style, including texture, color and contrast are randomized.

scaling and elastic deformation. Whilst these are successful at teaching rotation and scale invariance to a model, what of color, texture and complex illumination variations?

Tobin et al. [23] show that it is possible for an object detection model to generalize from graphically rendered virtual environments to the real world, by randomizing color, texture, illumination and other aspects of the virtual scene. It is interesting to note that, rather than making the virtual scene as realistic as possible, they attain good generalization by using an unrealistic but diverse set of random textures. Conversely, Atapour & Breckon [22] train on highly photorealistic synthetic images, but find that the model generalizes poorly to data from the real world. They are able to rectify this by using CycleGAN [91] and fast neural style transfer [92] to transform real world images into the domain of the synthetic images. These results together suggest that deep neural networks can overfit to subtle differences in the distribution of low-level visual features, and that randomizing these aspects at training time may result in better generalization. However, in the typical case where the training images come not from a renderer but from a camera, this randomization must be done via image manipulation, as a form of data augmentation. It is not clear how standard data augmentation techniques could introduce these subtle, complex and ill-defined variations.

Neural style transfer [24] offers the possibility to alter the distribution of low-level visual features in an image whilst preserving semantic content. Exploiting this concept, we propose *Style Augmentation*, a method to use style transfer to augment arbitrary training images, randomizing their color, texture and contrast whilst preserving geometry (see Figure 4.1). Although the original style transfer method was a slow optimization process that was parameterized by a target style image [24], newer approaches require only a single forward pass through a style transfer network, which is parameterized by a style embedding [94]. This is important, because

in order to be effective for data augmentation, style transfer must be both fast and randomized. Since the style transfer algorithm used in our work is parameterized by an $\mathbb{R}^{100}$ embedding vector, we are able to sample that embedding from a multivariate normal distribution, which is faster, more convenient and permits greater diversity than sampling from a finite set of style images.

In addition to standard classification benchmarks, we evaluate our approach on a range of domain adaptation tasks. To the best of our knowledge, this is the first time data augmentation has been tested for domain adaptation. Ordinarily, data augmentation is used to reduce overfitting and improve generalization to unseen images from the same domain, but we reason that domain bias *is* a form of overfitting, and should therefore benefit from the same countermeasures. Data augmentation is not domain adaptation, but it can reduce the need for domain adaptation, by training a model that is more general and robust in the first place. Although this approach may not exceed the performance of domain adaptation to a specific target domain, it has the advantage of improving accuracy on *all* potential target domains before they are even seen, and without requiring separate procedures for each.

In summary, this work explores the possibility of performing data augmentation via style randomization in order to train more robust models that generalize to data from unseen domains more effectively. Our primary contributions can thus be summarized as follows:

- *Style randomization* - We propose a novel and effective method for randomizing the action of a style transfer network to transform any given image such that it contains semantically valid but random styles.
- *Style augmentation* - We utilize the randomized action of the style transfer pipeline to augment image datasets to greatly improve downstream model performance across a range of tasks.
- *Omni-directional domain transfer* - We evaluate the effectiveness of using style augmentation to implicitly improve performance on domain transfer tasks, which ordinarily require adapting a model to a specific target domain post-training.

These contributions are reinforced via detailed experimentation, supported by hyperparameter grid searches, on multiple tasks and model architectures. We open source our PyTorch implementation as a convenient data augmentation package for deep learning practitioners[*].

[*] https://github.com/philipjackson/style-augmentation

4.2 Related Work

4.2.1 Domain Bias

The issue of domain bias or domain shift [95] has long plagued researchers working on the training of discriminative, predictive, and generative models. In short, the problem is that CNNs fail to generalize well to images from domains besides the one they were trained on. For example, a depth estimation model trained on images captured from roads in Florida may fail when deployed on German roads [96], even though the task is the same and even if the training dataset is large. Domain shift can also be caused by subtle differences between distributions, such as variations in camera pose, illumination, lens properties, background and the presence of distractors.

A typical solution to the problem of domain shift is transfer learning, in which a network is pre-trained on a related task with a large dataset and then fine-tuned on the new data [97]. This can reduce the risk of overfitting to the source domain because convolutional features learned on larger datasets are more general [98]. However, transfer learning requires reusing the same architecture as that of the pre-trained network and a careful application of layer freezing and early stopping to prevent the prior knowledge being forgotten during fine-tuning.

Another way of addressing domain shift is domain adaptation, which encompasses a variety of techniques for adapting a model post training to improve its accuracy on a specific target domain. This is often accomplished by minimizing the distance between the source and target feature distributions in some fashion [99–104]. Certain strategies have been proposed to minimize Maximum Mean Discrepancy (MMD), which represents the distance between the domains [99, 105], while others have used adversarial training to find a representation that minimizes the domain discrepancy without compromising source accuracy [100, 101, 103]. Although many adversarial domain adaptation techniques focus on discriminative models, research on generative tasks has also utilized domain transfer [102]. Li et al. [104] propose adaptive batch normalization to reduce the discrepancy between the two domains. More relevant to our work is [22], which employs image style transfer as a means to perform domain adaptation based on [106].

Even though domain adaptation is often effective and can produce impressive results, its functionality is limited in that it only aims to improve generalization to a specific target domain. In contrast, our approach introduces more variation into the source domain by augmenting the data (Section 4.2.3), which can enhance the overall robustness of the model, leading to better generalization to many potential target domains, without first requiring data from them.

4.2.2 Style Transfer

Style transfer refers to a class of image processing algorithms that modify the visual style of an image while preserving its semantic content. In the deep learning literature, these concepts are formalized in terms of deep convolutional features in the seminal work of Gatys et al. [24]. Style is represented as a set of Gram matrices [107] that describe the correlations between low-level convolutional features, while content is represented by the raw values of high level semantic features. Style transfer extracts these representations from a pre-trained loss network (traditionally VGG [108]), and uses them to quantify style and content losses with respect to target style and content images and combines them into a joint objective function. Formally, the content and style losses can be defined as:

$$\mathcal{L}_c = \sum_{i \in \mathcal{C}} \frac{1}{n_i} ||f_i(x) - f_i(c)||_F^2, \tag{4.1}$$

$$\mathcal{L}_s = \sum_{i \in \mathcal{S}} \frac{1}{n_i} ||\mathcal{G}[f_i(x)] - \mathcal{G}[f_i(s)]||_F^2, \tag{4.2}$$

where c, s and x are the content, style and restyled images, f is the loss network, $f_i(x)$ is the activation tensor of layer i after passing x through f, n_i is the number of units in layer i, $\mathcal{C}$ and $\mathcal{S}$ are sets containing the indices of the content and style layers, $\mathcal{G}[f_i(x)]$ denotes the Gram matrix of layer i activations of f, and $||\cdot||_F$ denotes the Frobenius norm.

The Gram matrix of a set of vectors $V = \{v_1, ..., v_n\}$ is a matrix containing the inner products of all pairs of vectors in the set:

$$\mathcal{G}[V]_{ij} = \langle v_i, v_j \rangle. \tag{4.3}$$

The inner product, in this context, is simply the vector dot product. When we take the Gram matrix of an activation tensor $f_i(x)$, we are splitting that rank 3 tensor into its individual feature maps (which are matrices), flattening them into vectors, and computing Equation 4.3 over those vectors. This results in a $c \times c$ Gram matrix, where c is the number of feature maps in $f_i(x)$. For $\mathcal{G}[f_k(x)]_{ij}$ to have a high value, feature maps i and j of layer k must have non-zero values at the same positions, therefore $\mathcal{G}[f_k(x)]$ measures the co-occurrence of features detected by layer k. As shown by Gatys et al. [24], these Gram matrices turn out to correlate remarkably well with subjective impressions of artistic style or texture.

The Frobenius norm, which we use to measure the difference between two Gram matrices, is just the square root of the sum of squared elements, therefore the squared Frobenius norm of an $m \times n$ matrix A is just the sum of squared elements:

$$||A||_F^2 = \sum_{i=1}^{m} \sum_{j=1}^{n} A_{ij}^2. \tag{4.4}$$

The overall objective can then be expressed as:

$$\min_x \mathcal{L}_c(x, c) + \lambda \mathcal{L}_s(x, s), \tag{4.5}$$

where λ is a scalar hyperparameter determining the relative weights of style and content loss.

Originally, this objective was minimized directly by gradient descent in image space [24]. Although the results are impressive, this process is very computationally inefficient, leading to the emergence of alternative approaches that use neural networks to approximate the global minimum of the objective in a single forward pass [92, 109, 110]. These are fully-convolutional networks that are trained to restyle an input image while preserving its content. Although much faster, these networks only learn to apply a single style, and must be re-trained if a different style is required, hence enabling only single-domain rather the multi-domain adaptability proposed here.

Building on the work of Ulyanov et al. [111], and noting that there are many overlapping characteristics between styles (e.g. brushstrokes), Dumoulin et al. [112] train one network to apply up to 32 styles using conditional instance normalization, which sets the mean and standard deviation of each intermediate feature map to different learned values for each style. Ghiasi et al. [94] generalizes this to fully arbitrary style transfer, by using a fine-tuned InceptionV3 network [113] to predict the renormalization parameters from the style image. By training on a large dataset of style and content images, the network is able to generalize to unseen style images. Concurrently, Huang et al. [114] match the mean and variance statistics of a convolutional encoding of the content image with those of the style image, then decode into a restyled image, while Yanai [115] concatenates a learned style embedding onto an early convolutional layer in a style transformer network similar to that of Johnson et al. [92].

In this work, while we utilize the approach presented by Ghiasi et al. [94] as part of our style randomization procedure, any style transfer method capable of dealing with unseen arbitrary

styles can be used as an alternative, with the quality of the results dependent on the efficacy of the style transfer approach.

4.2.3 Data Augmentation

Ever since the work of Krizhevsky et al. [28], data augmentation has been a standard technique for improving the generalization of deep neural networks. Data augmentation artificially inflates a dataset by using label-preserving transforms to derive new examples from the originals. For example, Krizhevsky et al. [28] create ten new samples from each original by cropping in five places and mirroring each crop horizontally. Data augmentation is actually a way of explicitly teaching invariance to whichever transform is used, therefore any transform that mimics intra-class variation is a suitable candidate. For example, the MNIST (handwritten digit) dataset [26] can be augmented using elastic distortions that mimic the variations in pen stroke caused by uncontrollable hand muscle oscillations [27, 116]. Yaeger et al. [90] also use the same technique for balancing class frequencies, by producing augmentations for under-represented classes. Wong et al. [117] compare augmentations in data space versus feature space, finding data augmentations to be superior.

Bouthillier et al. [118] argues that dropout [119] corresponds to a type of data augmentation, and proposes a method for projecting dropout noise back into the input image to create augmented samples. Likewise, Zhong et al. [120] presents random erasing as a data augmentation, in which random rectangular regions of the input image are erased. This is directly analogous to dropout in the input space and is shown to improve robustness to occlusion.

The closest work to ours is that by Geirhos et al. [121], who have recently shown that CNNs trained on ImageNet are more reliant on textures than they are on shape. By training ResNet-50 on a version of ImageNet with randomized textures (a procedure that amounts to performing style augmentation on all images), they are able to force the same network to rely on shape instead of texture. This not only agrees more closely with human behavioural experiments, but also confers unexpected bonuses to detection accuracy when the weights are used in Faster R-CNN, and robustness to many image distortions that did not occur in the training set. Our work corroborates and extends these results by showing an additional benefit in robustness to domain shift, and shows that style randomization can be used as a convenient and effective data augmentation technique.

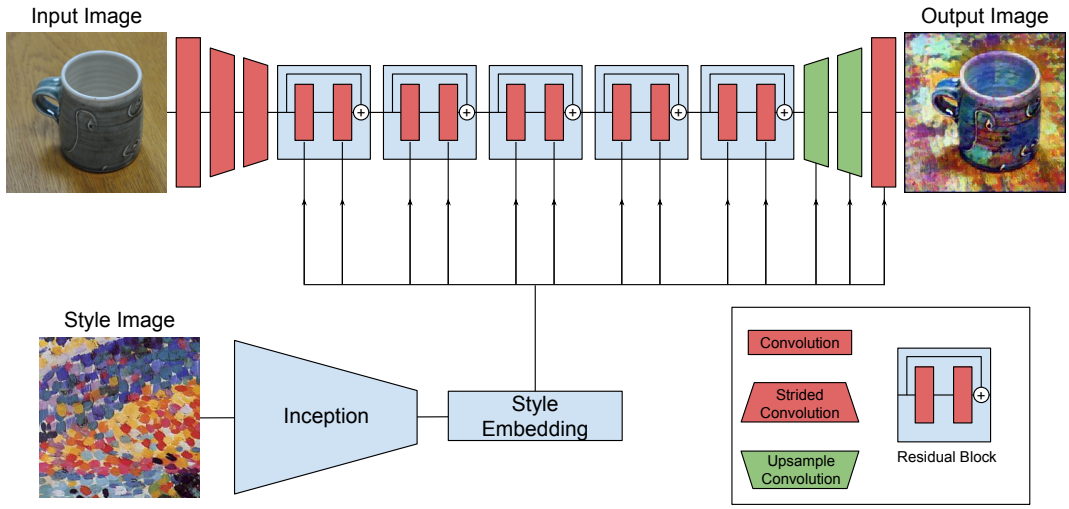


Figure 4.2: Diagram of the arbitrary style transfer pipeline of Ghiasi et al. [94].

4.3 Proposed Approach

For style transfer to be used as a data augmentation technique, we require a single style transfer algorithm that is both fast and capable of applying as broad a range of styles as possible. These requirements narrow our search space considerably, since most approaches are either too inefficient [24] or can only apply a limited number of styles [92, 112]. We chose the approach of Ghiasi et al. [94], for its speed, flexibility, and visually compelling results. A critical part of our data augmentation technique is providing a method for randomizing the action of the style transfer network. In this section we will introduce the style transfer pipeline we utilize and detail our novel randomization procedure.

4.3.1 Style Transfer Pipeline

Our chosen style transfer network (Detailed in Figure 4.2) employs a style predictor network P to observe an arbitrary style image s and output a style embedding $z \in \mathbb{R}^{100}$. The transformer network T must then accept the content image c and the style embedding z as inputs, and produce a restyled image r whose semantic content should be unchanged but whose style / texture should match that of the style image. This style embedding influences the action of the transformer network via conditional instance normalization [112], in which activation channels are shifted and rescaled based on the style embedding. Concretely, if x is a feature map prior to normalization, then the renormalized feature map is as follows:

$$x' = \gamma(\frac{x - \mu}{\sigma}) + \beta, \tag{4.6}$$

where μ and σ are respectively the mean and the standard deviation across the feature map spatial axes, and β and γ are scalars obtained by passing the style embedding through a fully-connected layer. As shown in Figure 4.2, all convolutional layers except for the first three perform conditional instance renormalization. In this way, the restyled image r is conditioned on both the content image c and the style image s:

$$r = T(c, P(s)). \tag{4.7}$$

At training time, we sample style images randomly from the Painter By Numbers (PBN) dataset[†] and content images from ImageNet, as in the original paper [94]. The PBN dataset consists of $79,433$ artistic images; the size and diversity of this dataset is necessary in order for the network to learn a robust mapping that generalizes well to unseen style images, in the same way that large labelled datasets enable classification networks to generalize well.

4.3.2 Randomization Procedure

Randomizing the action of the style transfer pipeline is as simple as randomizing the style embedding that determines the output style. Ordinarily, this embedding is produced by the style predictor network, as a function of the given style image. Rather than feeding randomly chosen style images through the style predictor to produce random style embeddings, it is more computationally efficient to simulate this process by sampling them directly from a probability distribution. However, it is important that this probability distribution closely resembles the distribution of embeddings observed during training. Otherwise, we risk supplying an embedding unlike any that were observed during training, which may produce unpredictable behavior. We use a multivariate normal as our random embedding distribution, the mean and covariance of which are the empirical mean and covariance of the set of all embeddings of PBN images. Qualitatively, we find that this approximation is sufficient to produce diverse yet sensible stylizations (see Figure 4.1).

[†] https://www.kaggle.com/c/painter-by-numbers

Figure 4.3: Output of transformer network with different values for the style interpolation parameter α.

To provide control over the strength of augmentation (see Figure 4.3), the randomly sampled style embedding can be linearly interpolated with the style embedding of the input image, $P(c)$. Passing $P(c)$ instructs the transformer network to change the image style to the style it already has thus leaving it mostly unchanged. In general, our random embedding is therefore a function of the input content image c:

$$z = \alpha\, \mathcal{N}(\mu, \mathbf{\Sigma}) + (1 - \alpha) P(c) \tag{4.8}$$

where P is the style predictor network, α is a hyperparameter controlling style interpolation, and μ, $\mathbf{\Sigma}$ are the mean vector and covariance matrix of the style image embeddings $P(s)$:

$$\mu = \mathbb{E}_s \left[P(s) \right], \tag{4.9}$$

$$\mathbf{\Sigma}_{i,j} = \mathrm{Cov} \left[P(s)_i, P(s)_j \right]. \tag{4.10}$$

4.4 Experimental Results

We evaluate our proposed style augmentation method on three distinct tasks: image classification, cross-domain classification and depth estimation. We present results on the STL-10 classification benchmark [122] (Section 4.4.1), the Office domain transfer benchmark [93] (Section 4.4.2), and the KITTI depth estimation benchmark [96] (Section 4.4.3). We also perform a hyperparameter search to determine the best ratio of unaugmented to augmented training images and the best augmentation strength α (see Eqn. 4.8). In all experiments, we use a learning rate of 10^{-4} and weight decay of 10^{-5}, and we use the Adam optimizer (momentum $\beta_1 = 0.5$, $\beta_2 = 0.999$, initial learning rate of 0.001).

Although we evaluate style augmentation on domain transfer tasks, our results should not be compared directly with those of domain adaptation methods. Domain adaptation uses

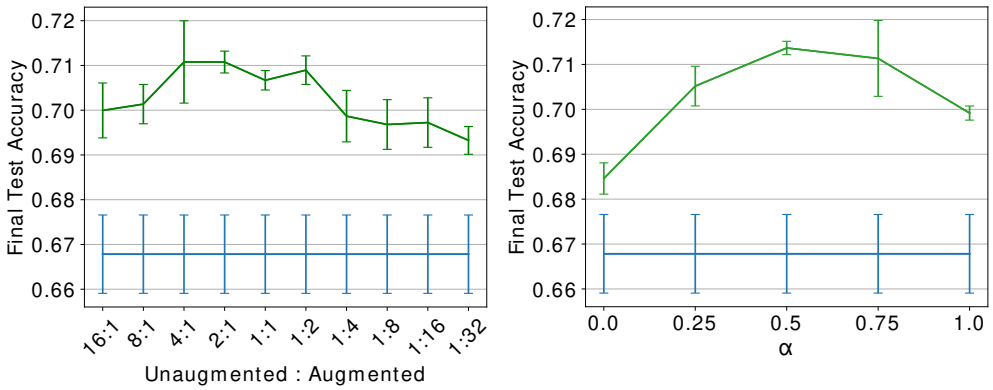


Figure 4.4: Hyperparameter searches on augmentation ratio and style transfer strength (α). Curves are averaged over four experiments; error bars denote one standard deviation. Blue lines depict unaugmented baseline accuracy.

information about a specific target domain to improve performance on that domain. In contrast, data augmentation is domain agnostic, improving generalization to all domains without requiring information about any of them. Therefore we compare our approach against other data augmentation techniques.

4.4.1 Image Classification

We evaluate our style augmentation on the STL-10 dataset [122]. STL-10 consists of 10 classes with only 500 labelled training examples each, a typical case in which data augmentation would be curial since the number of labelled training images is limited.

Prior to the final optimization, we perform a hyperparameter search to determine the optimal values for the ratio of unaugmented to augmented images and the strength of the style transfer, as determined by the interpolation hyperparameter α. We train the InceptionV3 [113] architecture to classify STL-10 images, performing $40,000$ iterations, augmenting the data with style augmentation, and we repeat each experiment four times with different random seeds.

First we test augmentation ratios, interpolating in factors of two from $16:1$ (unaugmented : augmented) to $1:32$. Since we do not know the optimal value of α, we sample it uniformly at random from the interval $[0, 1]$ in these experiments. Figure 4.4 (left) demonstrates the results of this search. We plot the final test accuracy after $40,000$ iterations. A ratio of $2:1$ (corresponding

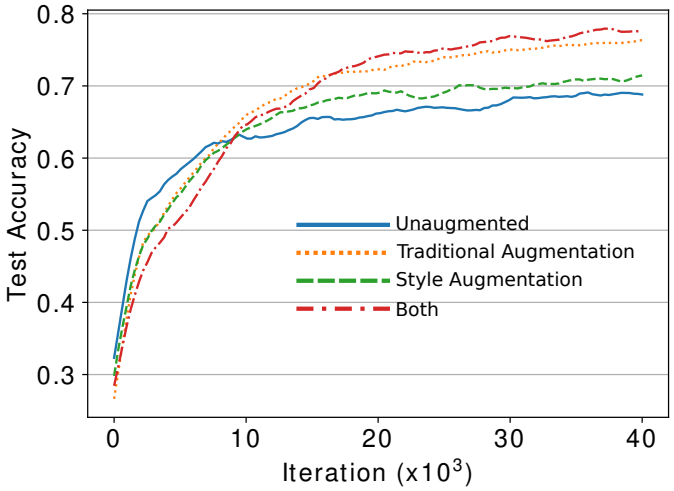


Figure 4.5: Comparing test accuracy curves for a standard classification task on the STL-10 dataset [122].

to an augmentation probability of 0.5) appears to be optimal. Fixing the augmentation ratio at $2:1$, we repeat the experiment for α and find an optimal value of 0.5 (Figure 4.4, right). Style augmentation takes 2.0ms on average per image on a GeForce 1080Ti, which corresponds to a 6% training time increase on this task when the optimal augmentation ratio of $2:1$ is used. If time is critical, the augmentation ratio can be set as low as $16:1$ and still provide a significant accuracy boost, as Figure 4.4 shows.

With suitable hyperparameters determined, we next compare style augmentation against a comprehensive mix of seven traditional augmentation techniques: horizontal flipping, small rotations, zooming (which doubles as random cropping), random erasing [120], shearing, conversion to grayscale and random perturbations of hue, saturation, brightness and contrast. As in the hyperparameter search, we train InceptionV3 [113] to $40,000$ iterations on the $5,000$ labeled images in STL-10. As seen in Figure 4.5, while style augmentation alone leads to faster convergence and better final accuracy versus the unaugmented baseline, in combination with the seven traditional augmentations, it yields an improvement of 8.5%.

Moreover, without using any of the unlabeled data in STL-10 for unsupervised training, we achieve a final test accuracy of 80.8% after $100,000$ iterations of training. This surpasses the reported state of the art [123, 124], using only supervised training with strong data augmentation.

Task	Model	Augmentation Approach			
		None	Trad	Style	Both
$AW \rightarrow D$	InceptionV3	0.789	0.890	0.882	**0.952**
	ResNet18	0.399	0.704	0.495	**0.873**
	ResNet50	0.488	0.778	0.614	**0.922**
	VGG16	0.558	0.830	0.551	**0.870**
$DW \rightarrow A$	InceptionV3	0.183	0.160	0.254	**0.286**
	ResNet18	0.113	0.128	0.147	**0.229**
	ResNet50	0.130	0.156	0.170	**0.244**
	VGG16	0.086	0.149	0.111	**0.243**
$AD \rightarrow W$	InceptionV3	0.695	0.733	0.767	**0.884**
	ResNet18	0.414	0.600	0.424	**0.762**
	ResNet18	0.491	0.676	0.508	**0.825**
	VGG16	0.465	0.679	0.426	**0.752**

Table 4.1: Test accuracies on the Office dataset [93] with A, D and W denoting the *Amazon*, *DSLR* and *Webcam* domains. $DW \rightarrow A$ accuracies are significantly lower for all methods because the Amazon dataset differs significantly from both DSLR and Webcam, featuring objects superimposed on blank white backgrounds instead of photographed in an office setting.

4.4.2 Cross-Domain Classification

To test the effect of our approach on generalization to unseen domains, we apply style augmentation to the Office cross-domain classification dataset [93]. The Office dataset consists of 31 classes and is split into three domains: *Amazon*, *DSLR* and *Webcam*. The classes are typical objects found in office settings, such as staplers, mugs and desk chairs. The *Amazon* domain consists of 2817 images scraped from Amazon product listings, while *DSLR* and *Webcam* contain 498 and 795 images, captured in an office environment with a DSLR camera and webcam, respectively.

We test the effect of style augmentation by training standard classification models on the union of two domains, and testing on the other. We also compare the effects of style augmentation on four different convolutional architectures: InceptionV3 [113], ResNet18 [64], ResNet50 [64] and VGG16 [108]. For each combination of architecture and domain split, we compare test accuracy with no augmentation (None), traditional augmentation (Trad), style augmentation (Style) and the combination of style augmentation and traditional augmentation (Both). Traditional augmentation refers to the same mix of techniques as in Section 4.4.1.

Figure 4.6 shows test accuracy curves for these experiments, and Table 4.1 contains final test accuracies. In certain cases, style augmentation alone (green curve) outperforms all seven

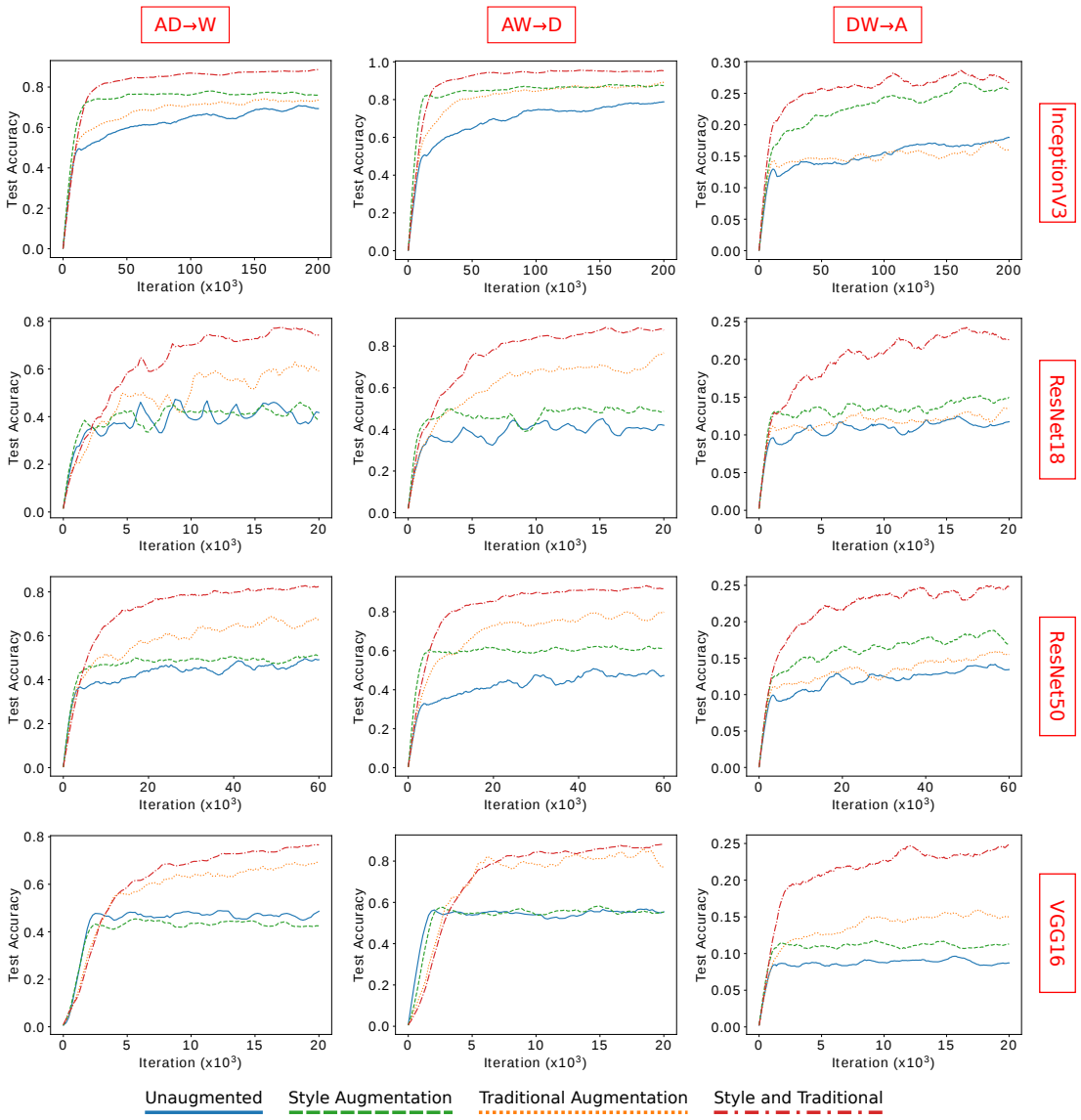


Figure 4.6: Results of the experiments using the Office dataset. Note the consistent superiority of traditional augmentation techniques combined with style augmentation (red curve).

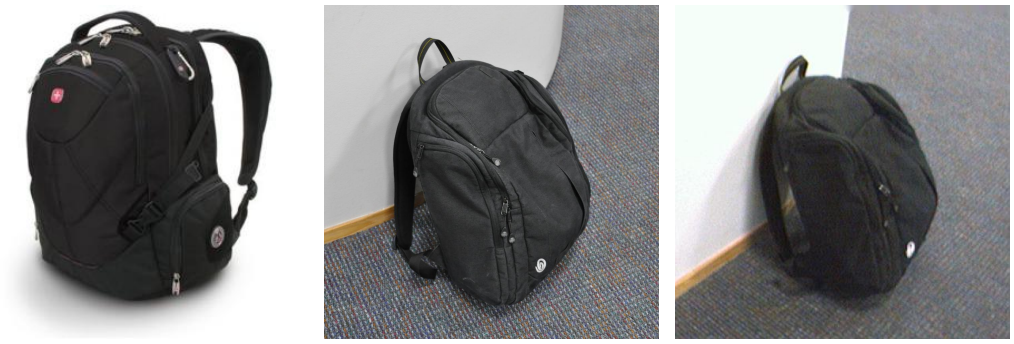
techniques combined (orange curve), particularly when the InceptionV3 architecture [113] is used. This points to the strength of our style augmentation technique and the invariances it can introduce into the model to prevent overfitting.

The largest domain shift occurs when the model is trained on the union of the *DSLR* and *Webcam* domains, and tested on the *Amazon* domain. This is because the *Amazon* images contain single objects against white backgrounds, while the other two domains consist of objects photographed in an office environment (see Figure 4.7). However, as seen in Figure 4.6 (rightmost column), style augmentation consistently improves the test accuracy even though the unaugmented model is barely outperforming random guess work. In all experiments, the combination of style augmentation and traditional techniques achieves the highest final accuracy and fastest convergence (see Figure 4.6).

To confirm that the benefits of style augmentation could not be realized more easily with simple colour space distortions, we ablate against color jitter augmentation, i.e. random perturbations in hue, contrast, saturation and brightness (see Table 4.2). The experiment shows that style augmentation confers accuracy gains at least 4% higher than those resulting from color jitter.

	AD → W	**AW → D**	**DW → A**
Unaugmented	0.684	0.721	0.152
Color Jitter	0.726	0.850	0.185
Style Augmentation	**0.765**	**0.893**	**0.215**

Table 4.2: Comparing style augmentation against color jitter (test accuracies on Office, with InceptionV3). These results demonstrate that the textural shifts induced by Style Augmentation provide accuracy gains beyond what can be achieved with simple colour space perturbations.



(a) Amazon (b) DSLR (c) Webcam

Figure 4.7: Images from the backpack class, from the Amazon, DSLR and Webcam datasets. The absence of any background in Amazon images makes the $DW \rightarrow A$ task more vulnerable to domain bias than $AD \rightarrow W$ or $AW \rightarrow D$, because the Amazon images differ significantly from both DSLR and Webcam images.

4.4.3 Monocular Depth Estimation

Finally, we evaluate our approach within monocular depth estimation - the task of accurately estimating depth information from a single image. The supervised training of a monocular depth estimation model is especially challenging as it requires large quantities of ground truth depth data, which is extremely expensive and difficult to obtain. An increasingly common way to circumvent this problem is to capture synthetic images from virtual environments, which can provide perfect per-pixel depth data for free [22]. However, due to domain shift, a model trained on synthetic imagery may not generalize well to real-world data.

Augmentation	**Error Metrics (lower, better)**				**Accuracy Metrics (higher, better)**		
	Abs. Rel.	Sq. Rel.	RMSE	RMSE log	$\sigma < 1.25$	$\sigma < 1.25^2$	$\sigma < 1.25^3$
None	0.280	0.051	0.135	0.606	0.656	0.862	0.926
Trad	0.266	0.045	0.128	0.527	0.671	0.872	0.936
Style	0.256	**0.040**	**0.123**	0.491	0.696	0.886	0.942
Both	**0.255**	0.041	**0.123**	**0.490**	**0.698**	**0.890**	**0.945**

Table 4.3: Comparing the results of a monocular depth estimation model [22] trained on synthetic data when tested on real-world images from [96].

Using our style augmentation approach, we train a supervised monocular depth estimation network on 65,000 synthetic images captured from the virtual environment of a gaming application [125]. The depth estimation network is a modified U-net with skip connections between every pair of corresponding layers in the encoder and decoder [22] and is trained using a global ℓ_1 loss along with an adversarial loss to guarantee mode selection [126]. By using style augmentation, we hypothesise that the model will learn invariance towards low-level visual features such as texture and illumination, instead of overfitting to them. The model will therefore generalize better to real-world images, where these attributes may differ. Examples of synthetic images with randomized styles are displayed in Figure 4.8.

Quantitative and qualitative evaluations were run using the test split in the KITTI dataset [96]. Similar to our classification experiments, we compare style augmentation against traditional data augmentation techniques. However, since object scale is such a vital cue for depth estimation, any transformations that rescale the image must be ruled out. This eliminates zooming, shearing and random cropping (which requires rescaling to keep the cropped regions a constant size). Random erasing makes no sense in this context since we never estimate the depth to an occluded point. Rotation seems promising, but was empirically found to worsen the results. This leaves

horizontal flipping, conversion to grayscale, and perturbations of hue, saturation, contrast and brightness as our traditional augmentations for depth estimation.

As seen in the numerical results in Table 4.3, models trained with style augmentation generalize better than those trained on traditionally augmented data. These results suggest that style augmentation may be a useful tool in monocular depth estimation, given that most traditional augmentations cannot be used, and the ones that can made little difference. Moreover, qualitative results seen in Figure 4.9 indicate how our augmentation approach can produce sharper output depth with fewer artefacts.

4.5 Discussion

The information imparted to the downstream network by style augmentation, in the form of additional labelled images, is ultimately derived from the pre-trained VGG network which forms the loss function of the transformer network (see Equations 4.1,4.2). Our approach can therefore be interpreted as transferring knowledge from the pre-trained VGG network to the downstream network. By learning to alter style while minimizing the content loss, the transformer network learns to alter images in ways which the content layer (i.e. a high level convolutional layer in pretrained VGG) is invariant to. In this sense, style augmentation transfers representational invariances directly from pretrained VGG to the downstream network.

The case for our style augmentation method is strengthened by the work of Geirhos et al. [121], who recently showed that CNNs trained on ImageNet learn highly texture-dependent representations, at the expense of shape sensitivity. This supports our hypothesis that reliance on texture is a significant cause of domain bias in deep vision models, and heavily suggests style augmentation as a practical tool for combating it.

Style Augmentation appears to provide greater benefit in the presence of domain shift than without it. We observe an absolute test accuracy increase of up to 10% on the Office dataset, which is designed to exhibit domain shift, but only 4% for the STL-10 classification task, which is not (Figure 4.5, Table 4.1). In agreement with Geirhos et al. [121], we found that style augmentation actually worsens accuracy on ImageNet. The texture reliance hypothesis allows this, since texture / label correlations may be present in both the train set and the test set, in which case preventing the model from learning them would lower test accuracy. However, the fact that style augmentation does moderately improve validation accuracy on STL-10 however

suggests that some image classification datasets have stronger correlation between textures and labels than others.

4.6 Conclusion

We have presented style augmentation, a novel approach for image-based data augmentation driven by style transfer. Style augmentation uses a style transfer network to perturb the color and texture of an image, whilst preserving shape and semantic content, with the goal of improving the robustness of any downstream convolutional neural networks. Our experiments demonstrate that our approach yields significant improvements in test accuracy on several computer vision tasks, particularly in the presence of domain shift. The amount of improvement varies across tasks and architectures, but we observe absolute gains in test accuracy of around 4% for a standard image classification task and up to 10% for image classification with domain bias (Figure 4.5, Table 4.1), and $2-4\%$ for monocular depth estimation depending how accuracy is measured (Figure 4.9). This provides evidence that CNNs are heavily reliant on texture, that texture reliance is a significant factor in domain bias, and that style augmentation is viable as a practical tool for deep learning practitioners to mitigate domain bias.

Epilogue

In this chapter we have introduced Style Augmentation, a data augmentation technique that randomizes texture and colour, forcing the network to rely more heavily on object shape and improving generalization to other domains. Unlike domain adaptation methods, Style Augmentation improves the robustness of the model itself, and does not require access to data from any other domain. Its efficacy is proven on both classification and monocular depth estimation tasks, and an ablation study shows that the same effects cannot be replicated with mere colour transforms - the texture randomization is essential. Data augmentation is normally thought of as a technique to improve generalization to unseen images when limited training data is available - Style Augmentation is useful in this regard, but appears to be most useful for combating domain shift.

Domain bias is an important concern in any real world application of machine learning where the deployed model must handle data that may differ systematically from its training data - even if those differences are quite subtle. However, despite the focus on deep learning applicability, this thesis has yet to examine any specific real world problems in detail. In the next chapter,

we discuss an interdisciplinary collaboration we worked on with the pharmaceutical company AstraZeneca.

Figure 4.8: Examples of input monocular synthetic images post style augmentation.

Figure 4.9: Results of unaugmented model (None), style (Style) traditional (None), and complete augmentation (Both) applied to depth estimation on KITTI [96].

Chapter 5

Phenotypic Profiling of Chemical Clusters with Generic Deep Convolutional Features

Prologue

While deep learning has seen many recent applications to pharmaceutical drug discovery, most have focused on predicting biological activity or toxicity directly from chemical structure. However, recent work has shown that cellular imaging assays are a rich source of information about drug activity. This chapter is the outcome of a project undertaken in collaboration with the pharmaceutical firm AstraZeneca, which aimed to develop techniques for analysing a large dataset of cellular images to identify groups of promising chemical compounds. Each image is a two-channel fluorescence microscopy image containing human embryonic kidney cells treated with a different novel chemical substance. The morphological changes induced in the cells by these compounds give some indication as to the compound's biological activity, but by their own admission, the biologists who constructed this dataset are not certain of what to look for. This is therefore a very ill defined problem that cannot be solved simply by training an image classifier, since there are no obvious labels to use as ground-truth.

The approach that we settled on uses pre-trained convolutional image features to assist biochemists in isolating interesting classes of compound for further investigation. The method reduces the dimensionality of raw fluorescence micrograph images, producing an embedding space that groups together images that show similar cellular morphology. Running k-means clustering on this embedding space yields a set of distinct phenotypic clusters, the most subjectively interesting of which can be selected by biochemists for further analysis. This allows us to combine the expert intuition of trained biologists with the image generalization abilities of CNNs, allowing scientists to choose a small number of interesting clusters for further study rather than evaluating each image individually.

Declaration: This chapter is based on the following publication: Jackson, P. T., Wang, Y., Knight, S., Chen, H., Dorval, T., Brown, M., Bendtsen, C. & Obara, B. *Phenotypic profiling of high throughput imaging screens with generic deep convolutional features* in *2019 16th International Conference on Machine Vision Applications (MVA)* (2019), 1–4. This chapter is presented largely as accepted, although referencing and notation have been altered and cross-referencing added for consistency throughout this thesis. Some stylistic changes have been made for consistency. The majority of the text is verbatim, with some minor wording and formatting changes.

5.1 Introduction

Modern drug discovery is a rapidly evolving field with the use of modern AI technology. Nonetheless, it remains expensive and time consuming for the introduction of new medicine ($2.870 billion, >10 years, [127]). Rather than hand-designing and testing novel drugs individually, the modern pharmaceutical approach is to use a library of compounds (often on the order of 2 million compounds), which are filtered by several rounds of complex imaging and biochemical tests. Typically for a high throughput screen (HTS), a single dose experiment is designed to remove the vast majority of irrelevant compounds, reducing the search space by 100 folds.

Following the spectacular rise of deep learning techniques in computer vision, natural language processing and numerous scientific applications in recent years, deep learning has increasingly been applied to the field of chemoinformatics, the application of computational techniques to problems in chemistry and biochemistry [128]. However, most recent work (e.g. [129], [130]) has focused on predicting the biological effects of chemical compounds directly from representations of chemical structure such as SMILES strings [131], or 2D diagrams. Morphological profiling is

a complementary approach that can be used to predict a broad range of biological effects that a chemical may have, such as on gene expression and cell proliferation [132]. In this approach, candidate drugs are applied to human cell cultures and imaged with high throughput fluorescence microscopy; depending on the bioactivity of the drug, this can cause complex morphological changes to occur, yielding clues as to which gene pathways the drug is interacting with. Indeed, Simms et al. [133] show that machine learning methods can predict the outcomes of many other targeted assays using visual features extracted from a single HTI assay.

Despite the rich information provided by morphological profiling, it is sometimes not clear exactly which morphological phenotypes should be screened for. An example of this scenario is a high throughput screen conducted by AstraZeneca, in which novel compounds are screened for inhibition of Inducible Degrader Of the Low density lipoprotein Receptor (IDOL) [134]. In this screen HEK293S human embryonic kidney cells were engineered to express both Low Density Lipoprotein Receptor - Green Fluorescence Protein (LDLR-GFP) and IDOL. Since IDOL degrades LDLR, the presence of green fluorescence is an indicator of IDOL inhibition. However, due to the complex and poorly understood interactions between novel compounds and human cells, presence of the GFP signal is a necessary but not sufficient condition to infer IDOL inhibition - the actual phenotypic appearance of genuine hits is not known at the screening stage. In situations like this, the expert knowledge and intuition of a biologist is required to identify phenotypes that are indicative of genuine hits. Due to the high volume of a HTS screen, this cannot be done manually for each individual image, and due to the unknown nature of the target phenotype, supervised learning and hand-engineered image feature extraction are not applicable here.

In this paper, we propose a novel dimensionality reduction procedure for computing feature vectors for cellular images using a pre-trained convolutional neural network (CNN). The resulting feature vector space can then be partitioned by unsupervised clustering, allowing us to decompose a HTS screen into a small set of visually distinct phenotypes. The expert judgement of biologists can then be applied to whole phenotype clusters rather than individual images, allowing a HTS to be filtered rapidly for promising compounds. The feature vectors can also be embedded in 2D space for visualization using dimensionality reduction techniques such as t-sne [135], providing a visual summary of the phenotypic distribution.

In Section 5.2 we explain our feature extraction pipeline. Section 5.3 describes our clustering procedure and in Section 5.4 we demonstrate and evaluate our approach on a high throughput imaging dataset (IDOL) collected from AstraZeneca.

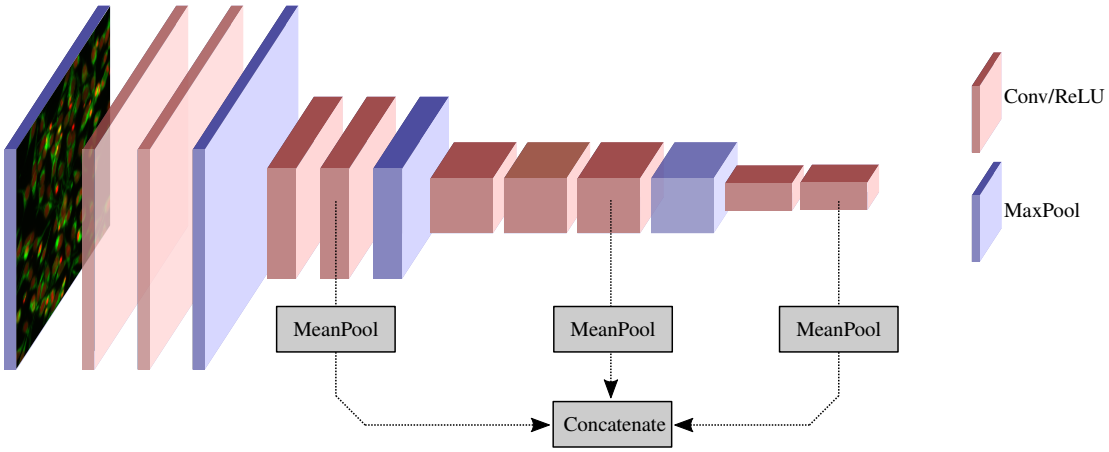


Figure 5.1: Our feature extraction pipeline. We feed our images through a pre-trained VGG16 network, truncated before the fully connected layers, and concatenate the spatial means of three intermediate convolutional layer activations. This yields a vector of multi-scale convolutional features, which we later embed in 2D space via t-sne (see Figure 5.6).

5.2 Feature Extraction

CNNs trained on large datasets such as ImageNet have been found to learn a hierarchy of features, with early layers learning general, task-agnostic features pertaining to texture and shape primitives, and later layers learning more task specific features [31]. Despite the obvious differences between ImageNet images (which are generally photographs) and fluorescence micrographs, the early convolutional layers of a CNN trained on ImageNet are general enough to respond to the differences in shape, colour and texture in our dataset.

Our feature extraction begins by computing feature maps for a histological image, by feeding it through a pre-trained CNN (Figure 5.1). The CNN architecture we use is VGG16 [136], pre-trained on ImageNet. Rather than taking final fully-connected layer activations as our feature vector, we extract our features by mean pooling the (pre-activation) feature maps of three early convolutional layers, and concatenating the means to produce a feature vector of length 1024 (one component for each feature map in the chosen layers). Extracting features from early convolutional layers rather than final fully connected layers has a number of advantages for large images in which the objects of interest are relatively small and homogeneously distributed.

Firstly, high level representations learned by CNNs contain information that is immediately relevant for identifying the classes they are trained to recognize. In the case of ImageNet, these

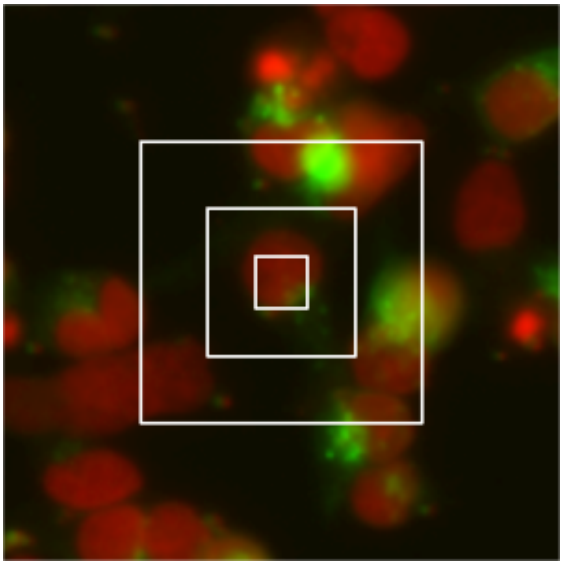


Figure 5.2: Receptive field sizes of our chosen convolutional layers, overlaid on a histological image for reference. By pooling from multiple layers, we can extract information about fine texture, individual cells and small clusters of cells. From the outside in, the white squares show the receptive field sizes of convolutional layers 4, 7 and 9 in VGG16.

are everyday objects such vehicles and animals. High level features are unlikely to be very descriptive for cellular images, which differ substantially from the ImageNet images and training classes, but lower level features are still general enough to capture information about shape, texture and colour.

Because of the fixed weight matrix connecting the first fully connected layer to the last convolutional layer, fully connected layers require the input image to be of a fixed size. By using only the convolutional layers of the network, we avoid the need to downsample our images to 224×224 (for the VGG network), which would discard valuable high frequency information.

Different layers capture information at different scales. Higher level layers have larger receptive fields, and describe patterns of greater size and complexity, but successively discard the higher frequency information captured by lower layers. By extracting features from the fourth, seventh and ninth convolutional layers, we obtain a multi-scale representation (see Figure 5.2).

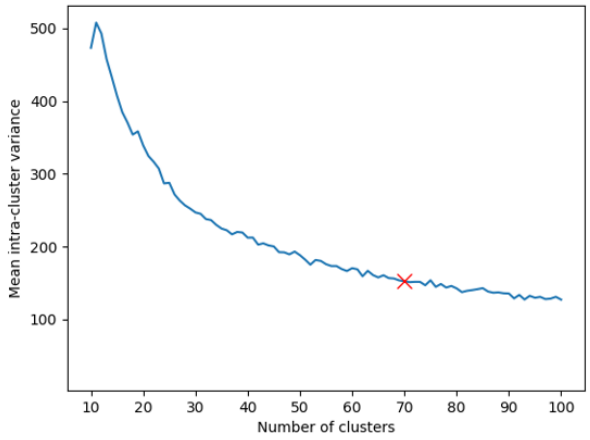


Figure 5.3: Mean intra-cluster variance as a function of the number of clusters, compared with mean intra-well variance. We cap the number of clusters at 70, as diminishing returns are observed past this point.

5.3 Clustering

To discover distinct phenotypes in the dataset, we perform k-means clustering in feature space. The optimal choice for the number of clusters k is a trade-off between making the clusters as homogeneous as possible, and keeping their number low. A lower number of clusters is preferable, since we intend for researchers to filter compounds on a per-cluster basis, allowing for faster filtering when fewer clusters are present. But if there are fewer clusters than there are distinct phenotypes, then individual clusters will contain multiple phenotypes, which defeats the purpose of clustering. Therefore we prefer the lowest number of clusters with the lowest intra-cluster variance. Figure 5.3 shows the mean intra-cluster embedding variance as a function of k; since we observe diminishing returns past $k = 70$, we choose 70 as the optimal number of clusters. This corresponds to the popular "elbow" (or "knee") method for determining the optimal value of k [137].

5.4 Results

The most crucial aspect to validate is that our feature extraction does indeed embed similar images at similar points in the feature space. We can evaluate this quantitatively, by measuring the variance (concretely, mean squared distance to centroid) of different groups of image

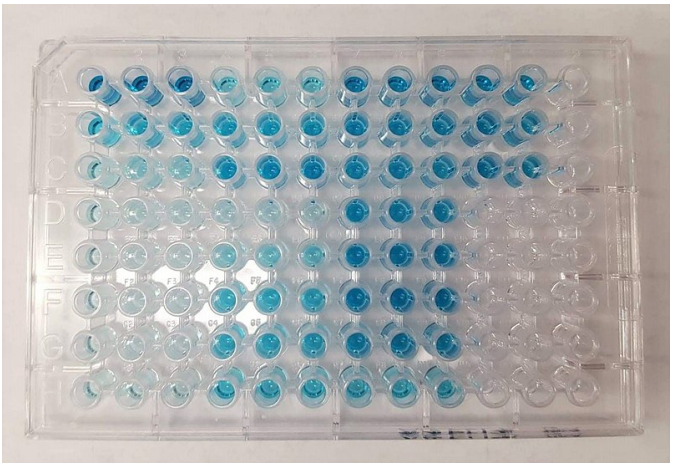


Figure 5.4: An assay plate containing 8×12 wells. In an HTS experiment, each well contains a population of cultured cells and a different compound (often repeating the same compound in varying concentrations). In our dataset, images were acquired from four different locations in each well, which means for each compound we have four images that should contain similar looking cells and thus have similar visual embeddings. Source: Wikimedia Commons (CC0 1.0 License).

embeddings. The compounds tested in this dataset are clustered by AstraZeneca in Extended-Connectivity Fingerprint embedding space [138], resulting in 711 chemical clusters. We would expect images corresponding to compounds from the same chemical cluster to have more tightly clustered feature vectors than the dataset as a whole, because similar compounds may lead to similar morphological changes. As expected, the mean intra-cluster feature variance is 66.5% that of the dataset as a whole. Furthermore, we would expect different images captured from the same well on an assay plate to be clustered more tightly still (see Figure 5.4), because these images all correspond to the same compound. We observe the mean intra-well variance (4 images were captured per well) to be 4.0% that of the full dataset.

To validate the quality of our clusters, we display samples from six phenotypic clusters in Figure 5.5. We see that k-means has identified relatively homogeneous clusters, further validating the quality of our embeddings.

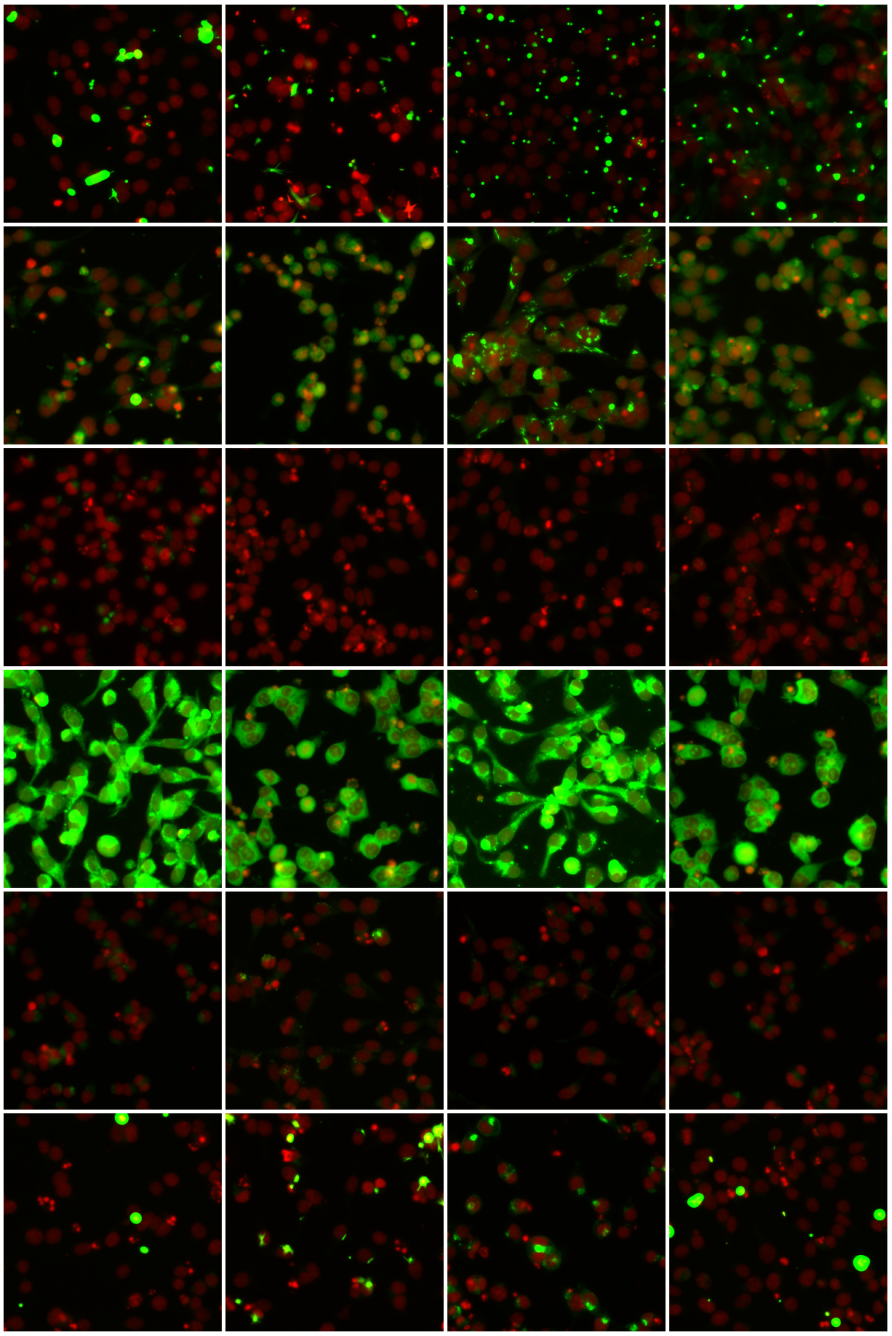


Figure 5.5: Samples from six of the 70 phenotypic clusters detected by k-means. Each row shows a sample from a different cluster. Rows 2 and 4 show genuine GFP expression.

Figure 5.6 shows a t-sne visualization of our entire dataset, with phenotypic clusters detected by k-means labelled by colour and annotated with representative samples from that cluster. To speed up the t-sne process, we used Principal Component Analysis (PCA) to reduce the dimensionality of our features from 1024 to 10. These 10 principal components explain $> 90\%$ of the variance in the embedding space. Clusters 4 and 6 in Figure 5.6 show genuine GFP expression, while the others are judged as uninteresting by biologists at AstraZeneca, and can be discarded, resulting in a 47-fold reduction. Using the full 70 clusters would result in more precise filtering and greater reduction still.

5.5 Conclusion

We have developed a novel workflow for high throughput screening. In this workflow, images were represented as deep learning feature vectors from a pre-trained convolutional neural network. This was followed by the clustering of images with similar image phenotypes. This facilitates scientists to select interesting clusters for downstream screening in an attempt to find hit compounds. Because it uses generic convolutional features extracted from a pre-trained convolutional neural network, our method requires no training and can be applied to any cellular screen dataset without hyperparameter tuning - a significant saving in time. Our visualizations allow chemists to both quickly assess the distribution of cellular morphologies in a high throughput imaging screen, or within a smaller subset of compounds, such as a chemical cluster. Meanwhile, our proposed workflow allows scientists to select interesting/promising phenotypes and quickly retrieve chemical clusters that show a high prevalence of said phenotypes.

Future work could gain further insight into the biological processes at work by investigating the relationship between our morphological embeddings and the ECFP embeddings of the chemicals that produced them, perhaps by predicting ECFP embeddings from morphological embeddings using supervised learning. Our techniques could also be applied to other imaging modalities, such as mass spectrometry imaging, with minimal modification needed; we will follow up with our industrial partner AstraZeneca in these studies.

Epilogue

Although the practical goals of this project were always somewhat ill-defined, it is safe to say that the scientific problem at its heart was representation learning. One way or another, we had to group images together by visual similarity, which generally means mapping them to embedding

vectors such that similar images map to vectors with low Euclidean distance. Perhaps the most interesting part of this project then is the fact that pre-trained convolutional features so greatly outperformed all of the representation learning techniques that we initially tried.

A useful feature of datasets such as this one is that the images are very large relative to the scale of the objects they contain, and those objects are distributed in a largely homogenous way throughout the images. This means that each image contains many examples of the phenotype induced by a certain drug, which we can use as a kind of implicit labelling, for example by splitting the image into 16 sub-images, which can be treated as examples from the same class. We attempted to exploit this by training a CNN to classify which image a small patch came from, hoping that the embeddings represented by the final fully connected layer would be similar for similar images (as is the case when training large networks on ImageNet). Since we had access to hit labels from a previous image-processing screen that had been performed at AstraZeneca, and to chemical cluster labels for each image, we also tried the same technique using these as class labels. We also tried a contrastive loss function, which explicitly trains the network to minimize the embedding difference between patches from the same image while maximizing the difference of embeddings for patches from different images. Other approaches attempted include BiGAN [139] (with and without Wasserstein loss [140]), and autoencoders [141].

None of these approaches produced results as good as the approach outlined in this chapter. Using pre-trained convolutional features unexpectedly outperformed all the other approaches by a considerable margin, producing far more interpretable t-sne plots and higher embedding similarity between images known to contain the same compound. It is counter-intuitive that features learned on the ImageNet dataset, which contains mostly photographs of everyday objects, would be more useful for our purposes than features that were learned from our dataset. As with many things in deep learning, it is hard to say exactly why this is, but we speculate that it has something to do with the scale and difficulty of the ImageNet classification task. ImageNet is a huge dataset of 1.2 million images and 1,000 classes, which requires a network to learn a great diversity of visual features in order to attain good accuracy. By contrast, the learning tasks that we devised may have been deceptively easy for a network to pass. For example, an autoencoder (which is trained to compress its input and then reconstruct that input from the compressed description) can achieve high reconstruction accuracy by placing coloured blobs in the right places, without having to reconstruct (and hence detect in the first place) any fine texture details.

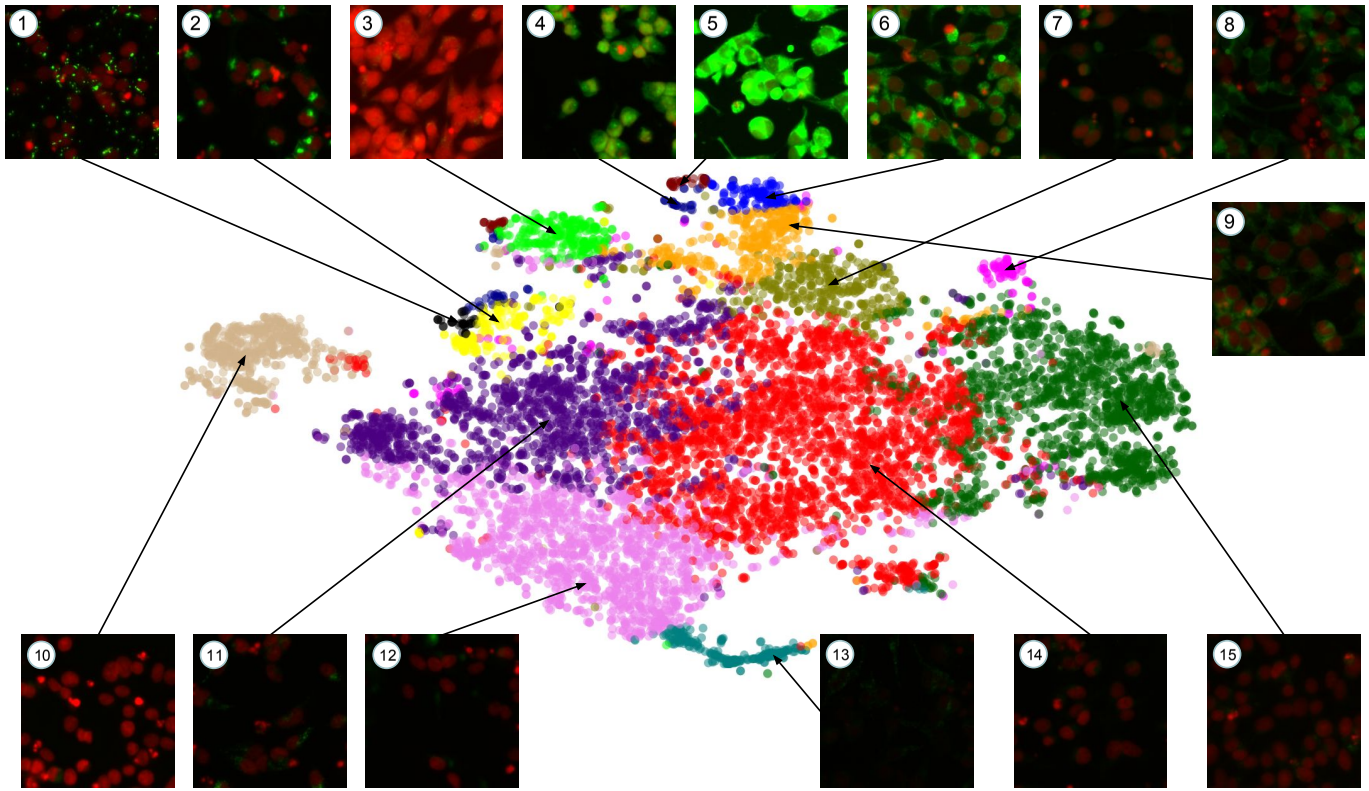


Figure 5.6: A t-sne embedding of our dataset, with colours showing phenotypic clusters discovered by k-means. For visualization purposes, we set $k = 15$ here. This interactive data visualization gives scientists a rapid overview of a large dataset, which would be difficult to obtain by sampling individual images. Furthermore, t-sne makes the phenotypic clusters visible in a way that reveals their relationships with one another, and does not enforce a discrete partitioning of the data as clustering algorithms do.

Chapter 6

Camera Bias in a Fine Grained Classification Problem

Prologue

Chapter 5 detailed an industry led research project to explore a large unlabelled image dataset. In this chapter, we describe a research project that grew out of another industrial collaboration, this time with consumer goods corporation Proctor & Gamble.

We investigate a simple binary image classification task in which correlations exist between the specific camera that acquired a given image and the class label of that image. We show that these correlations can be exploited by CNNs, resulting in a model that "cheats" at the image classification task by recognizing which camera took the image and inferring the class label from the camera. We show that models trained on a dataset with camera / label correlations do not generalize well to images in which those correlations are absent, nor to images from unencountered cameras. Furthermore, we investigate which visual features they are exploiting for camera recognition.

Images from consumer cameras include a number of subtle distortions and artifacts that vary from camera to camera due to design differences and random imperfections in the manufacturing process. These differences are presumably exploited by CNNs to recognize cameras.

Our experiments present evidence that global color statistics, lens deformation and chromatic aberration are not necessary for camera recognition, while high frequency features, which may be introduced by image processing algorithms built into the cameras, are. We also perform adversarial perturbations on images, showing that the perturbations that change a network's classification of an image depend on whether that model was trained on images whose labels correlate with camera type or not.

Declaration: This chapter is based on the following paper, in review at CVPR DeepVision at the time this thesis was submitted: Jackson, P., Bonner, S., Jia, N., Holder, C., Stonehouse, J. & Obara, B. Camera Bias in a Fine Grained Classification Task. *CVPR DeepVision.* In review (2020). The text has been extended slightly, and wording has been altered and cross-referencing added for consistency throughout this thesis. Some stylistic changes have been made for consistency.

6.1 Introduction

Convolutional neural networks (CNN) sometimes learn to satisfy their objective functions in ways we do not intend, typically by exploiting some subtle idiosyncrasy in the training data. For example, in [142] a CNN trained on ImageNet was found to be recognizing chocolate sauce pots by the presence of a spoon, because many of the chocolate sauce pots in the ImageNet dataset are indeed accompanied by a silver spoon. While effective at minimizing the loss function at training time, these clever exploits usually result in the model becoming brittle, as it is relying on characteristics that are specific to the training set and are not representative of the wider world. This tends to manifest as domain bias, whereby the model fails to generalise well to instances from other datasets with different idiosyncrasies.

We investigate a real world applied computer vision problem in which severe domain bias was caused by strong correlations between camera model and class label. Since the training dataset consists of two classes acquired with different cameras, the model learns to predict the class label by recognizing the camera that captured the image. Since the sets of cameras used to acquire the two classes are non intersecting, this is sufficient to achieve perfect training accuracy, whilst learning nothing about the task itself. Our task has characteristics typical of industrial deep vision projects, and we believe the lessons learned will be useful to many deep learning practitioners working on similar projects.

The task itself is to discriminate between shampoo bottles from two different manufacturers, which are distinguished only by very small differences in the printing of a batch code on the underside of the bottle. These differences are caused by different industrial printers being used, are independent of the actual character string that is printed, and are subtle enough that detecting them by eye is difficult even for trained experts. This therefore constitutes a fine grained binary classification problem, in which the intra-class variance is high relative to the inter-class variance.

Fine grained classification is difficult, so one might intuitively expect a model to cheat more often on such tasks, if the correct decision function is more complex compared to a cheating rule. On the other hand, in this instance the exploit of recognizing cameras is also a fine grained classification task, and in general it is not obvious which tasks are "harder" for a CNN to learn. CNNs have been known to cheat by detecting patterns which are barely perceptible to humans, such as chromatic aberration [143]. CycleGAN even cheats its reconstruction loss by inserting steganographic codes into its converted images, which it then uses to reconstruct the originals [144].

In this paper, we closely examine an instance of a model cheating on a real world visual classification task, and attempt to answer the following questions:

1. Is it possible for a CNN to recognize camera types when explicitly trained to do so?
2. Can we prove that the same CNN cheats on the task of manufacturer classification by recognizing camera types?
3. Does the propensity toward cheating depend on model architecture?
4. How exactly does a CNN recognize camera types?

Section 6.2 reviews relevant literature in fine grained classification and overfitting, while Section 6.3 describes out dataset and classification task in detail. Section 6.4 investigates the above questions systematically with a series of experiments, and in Section 6.5 we discuss our findings and draw conclusions.

6.2 Previous Work

Two major branches of literature are relevant to our work: source camera identification from images, and understanding deep neural networks.

6.2.1 Camera / Image Sensor Pattern Identification

Because our work concerns accidental camera detection, a brief review of deliberate camera detection methods is warranted, as it may shed some light on how our model learns to cheat. Many techniques have been developed to trace digital photos back to their camera of origin, primarily by the digital forensics community [145]. Such techniques can be used to detect doctored images or videos, where images or frames from different cameras are spliced together [146, 147]. Most of these methods revolve around extracting a unique sensor noise fingerprint from the image, and matching it against the reference patterns of known cameras. Since sensor noise is a complex phenomenon with multiple sources (e.g. photonic noise, lens imperfections, dust particles, dark currents, non-uniform pixel sensitivity), there are many ways of doing this. Geradts et al. [148] identify cameras by their unique patterns of dead and hot pixels, however not all cameras have dead pixels, and some remove them via post-processing. Kharrazi et al. [149] train an SVM to recognize five different cameras based on hand-engineered feature vectors extracted from images. This approach achieves up to 95% classification accuracy, but this is too low for forensic purposes. Choi et al. [150] take a similar SVM based approach, additionally showing that radial lens distortion is a useful feature for identifying cameras. Unlike noise based approaches, lens distortion can identify models of camera but not individuals. Kurosawa et al. [151] recognizes cameras by dark current noise, which is a small, constant signal emitted by a CCD, varying randomly from pixel to pixel. Although every digital camera has such a noise pattern and it will always be unique, it can only be acquired from dark frames where no light strikes the sensor, and is only a small component of sensor noise. Lukas et al. [152] propose a more robust method that exploits the non-uniform sensitivity to light among sensor pixels, which is a much stronger component and does not require dark frames to measure.

Another feature of consumer cameras that has thwarted a previous deep learning experiment [143] is chromatic aberration, in which different wavelengths of light are refracted by different amounts by the lens. This results in colored fringes around the edges of objects. This too has been used in digital forensics [153].

Recently, CNN-based methods have shown great potential in digital camera identification from images using standard supervised training [154–156], proving that CNNs are indeed able to infer which camera acquired a digital image.

6.2.2 Understanding Deep Convolutional Neural Networks

CNNs are often seen as something of a black box, with no clear consensus as to what information they are using to reach their decisions, how that information is represented internally, or what are the specific roles of their individual components. Attempts to answer these questions can be divided into two strands, feature visualization and attribution.

Feature visualization aims to clarify the function of neurons or channels, by synthesizing images that maximize their activation [157]. Simonyan et al. [84] investigate what patterns CNNs look for in each image class by performing gradient ascent in image space, to maximize the activation of an output class neuron. Yosinski et al. [158] do the same but with better regularization, producing more natural looking images. Mahendran et al. [159] treat intermediate CNN representations as functions which they can invert via gradient ascent in image space. This yields images that the CNN maps to the same representation as the original image, implying that they "look the same" to the CNN. Nguyen et al. [160] find natural looking images that maximally activate feature maps by searching the manifold learned by a generative adversarial network, rather than the full image space. Fong et al. [161] show evidence that far from feature maps learning separate, well defined concepts, the relationship between feature maps and semantic concepts is many-to-many, with each feature map involved in the detection of several concepts and most concepts activating multiple feature maps.

Attribution investigates which parts of an image contribute most to a CNN's decision - often expressed as "where the model is looking". Zeiler and Fergus [85] propose two methods to this end: occlusion mapping, in which the importance of an image patch is measured as the reduction in class probability when it is obscured, and backpropagation of class probability gradients into image pixels. Both of these methods yield saliency maps showing which parts of the image have the greatest effect on the output when changed, corresponding to the notion of how much they contributed to the network's decision. Another popular approach is guided backpropagation [65], which refines the gradient saliency maps of [85] by zeroing out negative gradients at every backpropagation step, so as to focus only on image parts that contribute positively to a particular class. A much faster alternative to occlusion mapping (which must run a forward pass for each

test patch) is class activation mapping (CAM) [162], which uses final layer feature maps as saliency maps, weighted and summed according to the weight of their connection to the class neuron in question. This approach requires that the output layer takes its input directly from mean pooled feature maps (as is the case with GoogLeNet and ResNet but not for networks with fully connected layers such as AlexNet). Selvaraju et al. [163] address this by using mean pooled gradients as a proxy for direct connection weights, allowing feature maps from any layer in any network to be used as saliency maps. Another technique by Fong et al. [142] learns a mask that causes a model to misclassify an image while obscuring the smallest area possible.

6.3 Dataset

Our dataset consists of 3090 RGB images of the undersides of shampoo bottles. These images are of size 1024×1024 and are cropped tightly around the bottle's batch code, which is a two-line alphanumeric serial number printed by a dot matrix printer (see Figure 6.1). The crops are from roughly the same area of the original images, so they should cover mostly the same region of the cameras' sensors. This means they should contain roughly the same sensor pattern noise, up to some random translation. The batch codes of the two manufacturers' products are expected to differ in some potentially very subtle ways, hence the relatively high resolution of our images.

Our images are captured with five different cameras: iPhone, Huawei, Samsung, Redmi and Vivo. In the base dataset these cameras occur at equal frequencies among the two manufacturers, but by excluding certain combinations of camera and label from the dataset, we can introduce correlations between camera type and class label. The images were acquired by five people who each photographed an equal number of images from each class and with each camera, all in the same room, under controlled lighting conditions. By standardizing environmental conditions like so, we can be surer that it is the correlation between camera type and class label that the model exploits, and not some other source of domain bias. The test set is a random 10% of the samples, on which the model is never trained.

6.4 Experiments

To address the questions raised in Section 6.1, we run a series of classification experiments on variations of our dataset with camera/label correlation artificially introduced. All experiments are trained to convergence with a learning rate of 10^{-4}. All accuracy numbers we report are averaged over four runs with different random number generator seeds.

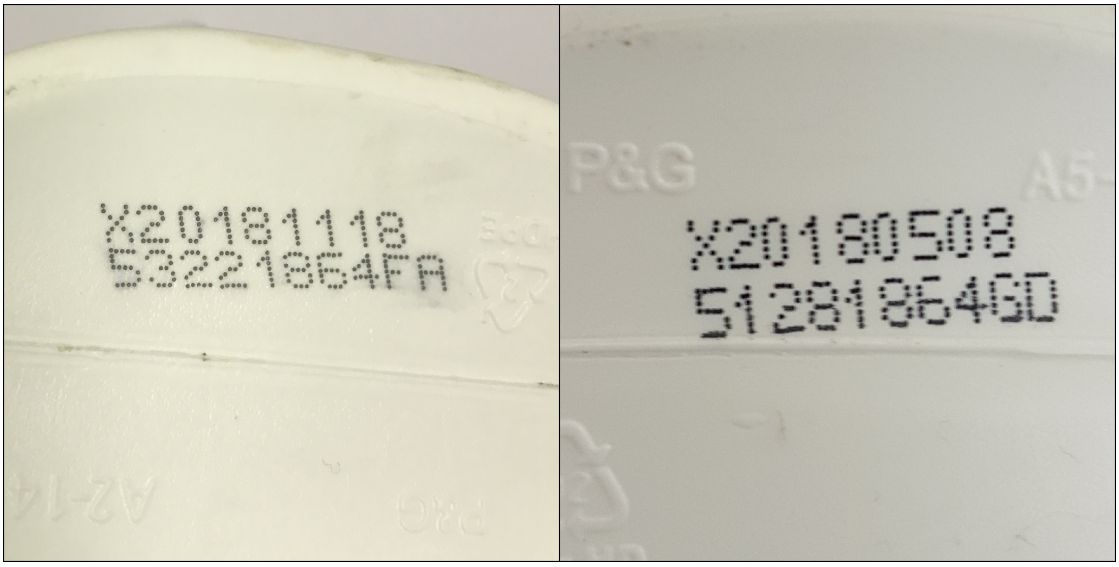


Figure 6.1: A pair of images from our dataset. The left was taken with an iPhone camera, while the right was taken with a Samsung.

6.4.1 Camera Classification

As a basic sanity check, we verify here that state of the art vision models can very easily classify which camera took the image in this dataset. This corroborates the work of [155] [154] for our own datasets and cameras. Table 6.1 shows that very high test accuracies can be achieved on this task across a range of architectures. As Figure 6.2 shows, a pretrained ResNet34 not only achieves high accuracy at camera classification but does so very quickly. Camera recognition is learned faster than manufacturer recognition, suggesting that a model which can minimize its loss by recognizing manufacturers or by cheating by camera recognition will tend toward the latter, as it is somehow easier.

6.4.2 Manufacturer Classification

We investigate our primary task, manufacturer classification, under three settings, which we refer to as Balanced, Partial, and Disjoint. In the Balanced setting, we use the full training set and there are no correlations between camera type and class label. In Partial, we use the same training set but with only iPhone and Samsung cameras included. In Disjoint, we introduce correlations between camera and class label by including only Manufacturer 1 images taken with iPhone or

Model	Test Accuracy
ResNet34	0.999
ResNet101	0.913
InceptionV3	0.998
AlexNet	0.945
VGG16	0.974

Table 6.1: Accuracy on the test set when classifying which camera took an image. All architectures tested show high test accuracy, demonstrating that CNNs can easily learn to recognize which camera took an image. The slightly lower accuracies of AlexNet and VGG16 are probably due to these networks requiring input images to be downsampled to 224×224, whereas the other networks can process arbitrary input sizes and so consume the full 1024×1024 images.

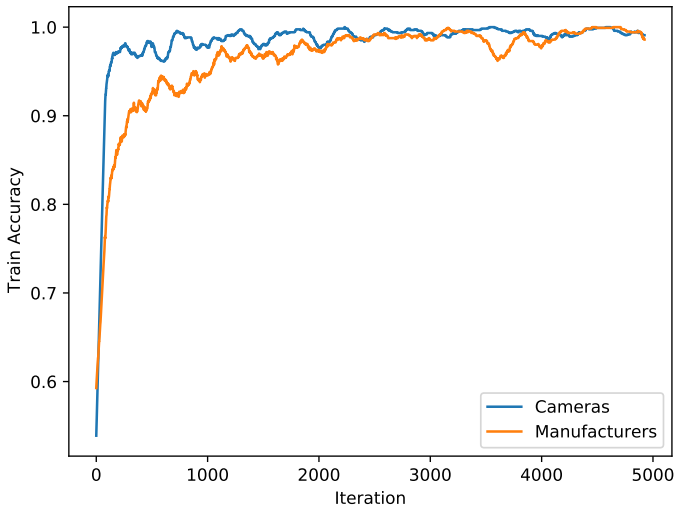


Figure 6.2: A pretrained ResNet34 model learns to recognize manufacturers very quickly, and learns to recognize cameras even faster.

Model	Balanced	Partial	Disjoint
ResNet34	0.974	0.957	0.505
ResNet101	0.969	0.921	0.505
InceptionV3	0.973	0.940	0.518
AlexNet	0.929	0.893	0.573
VGG16	0.979	0.945	0.556

Table 6.2: Manufacturer classification test set accuracy of five models with different training setups.

Samsung cameras, and Manufacturer 2 images taken with Huawei or Redmi cameras. Our test set is the same in all cases, balanced across camera types with no camera/label correlations.

Table 6.2 shows the results of manufacturer classification experiments on these three datasets. It is immediately apparent that while respectable accuracy is achieved when training on the Balanced dataset, an accuracy drop of around 30% occurs when training on Disjoint. In fact, Disjoint accuracy is close to 50%, hardly better than random guessing, which is entirely expected if the model were basing its classifications on camera types, each of which has an equal number of images of each class in the test set.

We can confirm that this drop in accuracy is due to camera bias by observing the model's behavior across camera types in the test set. As Figure 6.3 shows, a ResNet34 trained on the Disjoint dataset predicts Manufacturer 1 exclusively on test images acquired by iPhone or Samsung cameras, and Manufacturer 2 overwhelmingly on Huawei and Redmi images. Similar behavior is observed in the other models.

It is interesting to note that AlexNet and VGG16 both score higher test accuracy after training on Disjoint than their more modern counterparts, ResNet34, ResNet101 and InceptionV3. One possible explanation for this is that AlexNet and VGG16 both use fully connected layers to produce their final output, while the more recent networks are fully convolutional, i.e. consisting entirely of convolutional layers. Fully connected layers have one parameter per input unit and hence require fixed size input, whereas convolutional layers can process arbitrary sized input by using the same convolutional weights at every location in the input. AlexNet and VGG16 therefore require input images to be downsampled to 224×224, whereas the fully convolutional networks receive the full 1024×1024 images. This suggests that camera identification exploits high frequency features (as opposed to geometric distortions caused by lens variations), which

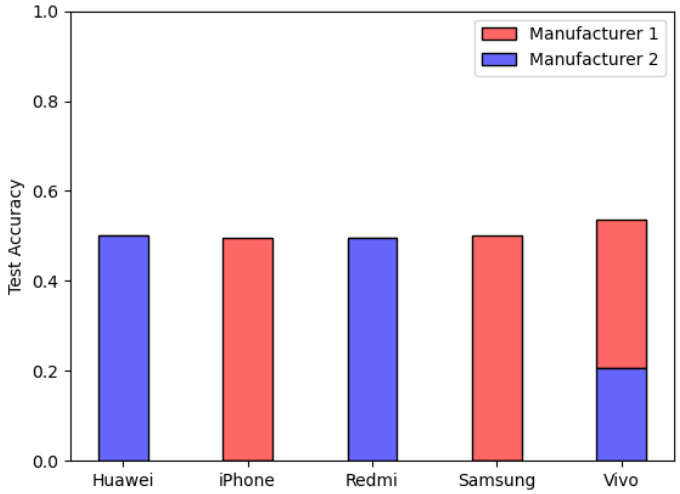


Figure 6.3: Test accuracy plot showing the distribution of predicted labels among correct outputs, for a ResNet34 trained on the Disjoint training set, in which all Manufacturer 1 images are iPhone or Samsung, and all Manufacturer 2 are Huawei or Redmi. For images from iPhone and Samsung cameras the model predicts only Manufacturer 1, while for Huawei and Redmi it predicts only Manufacturer 2, while for the unseen Vivo images it appears to guess randomly, achieving 54% accuracy with a mostly even mix of both classes. Best viewed in color.

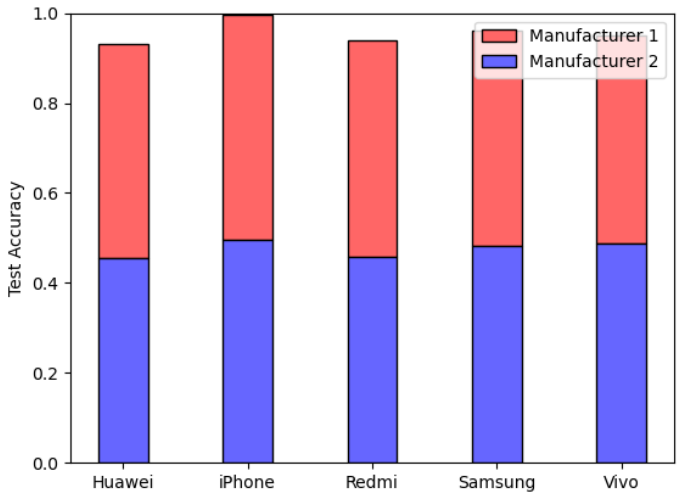


Figure 6.4: Test accuracy plot showing the distribution of predicted bottle manufacturers among correct outputs, for a ResNet34 trained on the Partial training set, in which camera type is uncorrelated with class label but only iPhone and Samsung images are present. Overall accuracy across all camera types is close to that achieved when trained on the full dataset, with little bias in favor of familiar camera types. This implies that in the absence of camera / label correlations, the model learns robust features for manufacturer classification, which generalize well to images from unseen cameras. Best viewed in color.

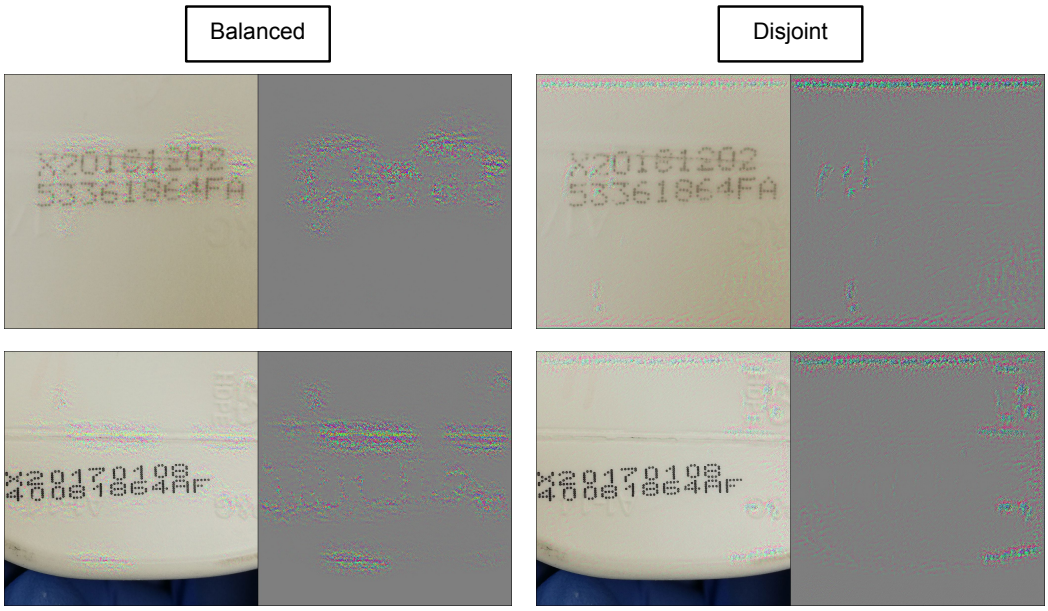


Figure 6.5: Adversarial perturbations applied to two images, classified by a ResNet34 model trained on the Balanced dataset (left) and the Disjoint dataset (right). The left image in each pair shows the input image with the perturbation amplified for visibility and overlaid on top, while the right image shows just the amplified perturbation itself. Strikingly different perturbations to the same image are observed depending on whether the model was trained without camera / label correlations (Balanced) or with them (Disjoint). Best viewed digitally, zoomed in.

are partly destroyed during downsampling, thus preventing AlexNet and VGG16 from exploiting them.

When training on the Partial dataset, where only iPhone and Samsung images are present but no camera/label correlation exists, test accuracy is broadly similar to training on the full (Balanced) dataset. Not only is accuracy high, but as Figure 6.4 shows, the model performs well on the unseen cameras. This implies that in the absence of camera/label correlation, the model learns a robust classification rule that is unaffected by camera type.

6.4.3 Adversarial Attacks on Manufacturer Classifiers

To gain some insight into the effect that camera / label correlations have on a trained model in terms of the patterns it learns to recognize, we perform adversarial attacks on trained models and

visualize the perturbations that flip a trained model's judgement of an image from Manufacturer 2 to 1. Adversarial attacks are small perturbations to input images, imperceptible to the human eye, which nonetheless are sufficient to fool a model into classifying that image as whatever the attacker wishes [164]. They are easily generated by gradient ascent in image space, backpropagating the negative log likelihood of the target label into the image pixels and taking small steps in the direction of the resulting image gradient until the model's prediction favors our target (e.g. see Nguyen et a. [165]). By performing this process using the same image but different models and comparing the resulting image perturbations, we can learn something about how those models differ.

Figure 6.5 shows that strikingly different adversarial perturbations are induced depending one which dataset the model was trained on. Perturbations that fool the Balanced model are focused around the batch code and other visible features of the bottle, such as the plastic seam, whereas those that fool the Disjoint model show a characteristic pink / green banding pattern in flat, featureless areas of the image. A distinct rainbow-like band of perturbation is also visible along the tops of images classified by the Disjoint model; these banding patterns at the tops and in featureless areas of images appear regardless of which input image the attack is performed on.

Adversarial perturbations, when amplified for visibility, usually look like uninterpretable noise bearing little apparent resemblance to the target image class (e.g. Goodfellow et al. [166]), so it is interesting to see so much structure in our case. The appearance of banding patterns in flat regions provides some evidence against chromatic aberration, which should manifest at the edges of objects [167].

6.4.4 Classification of Binary Masks

As discussed in Section 6.2, there is a finite set of image features that may be used to infer the camera from which an image originates. Since most of these features relate to color distribution or high frequency detail (i.e. texture), it seems likely that removal of these features would render camera identification impossible, and hence resolve the domain bias issues. To do this while preserving features that are likely relevant for robust manufacturer detection, we apply local mean thresholding to the images. This yields a binary image that effectively segments the dots of the batch codes while removing all elements of color and texture (see Figure 6.6).

As Table 6.3 shows, training on binary segmented images does not yield usable results on the manufacturer detection or camera classification tasks - in both cases, the test accuracy is close to

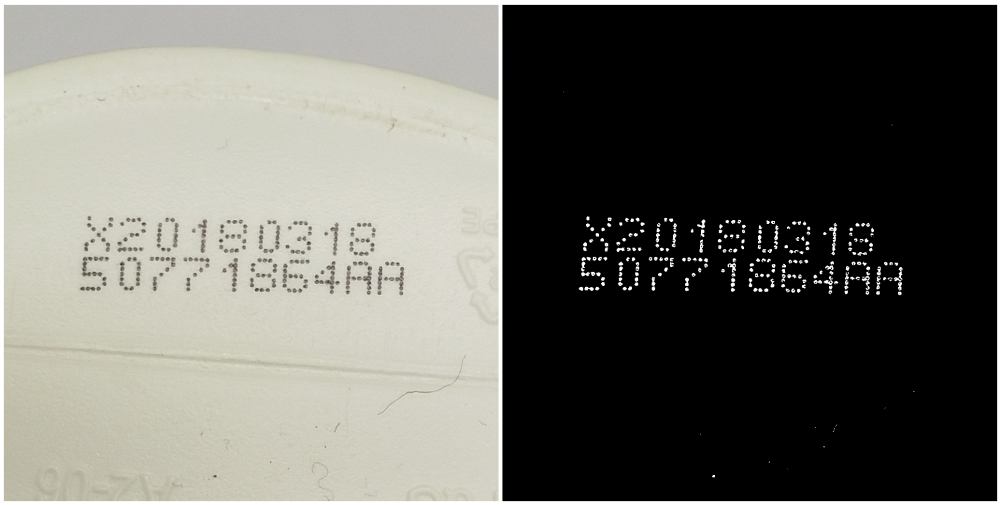


Figure 6.6: A bottle image with local mean thresholding applied, segmenting the batchcode dots. Origin camera classification does not work on such images, indicating that models use something other than the shape and position of the dots to classify cameras.

the level expected of random guessing. As expected, we also observed no significant correlation between manufacturer classification test accuracy and camera type when training on the Disjoint dataset with binary thresholding. This largely rules out lens distortion or other large scale geometric artifacts as the source of camera bias, since these distortions would cause the dots to move and thus be visible in the binary thresholded images.

6.4.5 Classification of Color Jittered Images

One hypothesis is that different cameras have subtly different color correction / white balancing settings, which a CNN could very easy detect and exploit, especially since the images were acquired in laboratory conditions with controlled lighting. We test this hypothesis by randomizing the hue, saturation, contrast and brightness of the images at training time, thus removing any correlation between camera type and global image color statistics. Table 6.4 shows that even with the high level of color randomization used (see Figure 6.7), camera and manufacturer classification test accuracy remains high. Accuracy is somewhat diminished for the AlexNet and VGG16 architectures, which require downsampled images as input, suggesting that these features are still useful when high frequency features are less available. We also train models to recognize cameras from grayscale images, achieving test accuracies roughly identical to those in Table 6.4.

Model	Test Accuracy	
	Manufacturers	Cameras
ResNet34	0.489	0.229
ResNet101	0.530	0.186
InceptionV3	0.525	0.270
Alexnet	0.499	0.214
VGG16	0.616	0.384

Table 6.3: Manufacturer and camera classification accuracy on the test set when trained (and tested) on binary segmented images (see Figure 6.6).

Figure 6.7: Color jitter augmentations applied to a single image (original in top left). Augmenting our training images with basic color distortions removes any correlations that may exist between class label and white balance, saturation, hue.

Model	Test Accuracy	
	Manufacturers	Cameras
ResNet34	0.975	0.992
ResNet101	0.961	0.995
InceptionV3	0.974	0.998
Alexnet	0.923	0.768
VGG16	0.972	0.883

Table 6.4: Manufacturer and camera classification test set accuracy when trained on images with randomized hue, saturation, contrast and brightness. Robust camera classification accuracy implies that image color statistics are not necessary for camera inference.

6.4.6 Classifying Cameras from Small Image Patches

With lens deformation and color statistics ruled out as camera identifying features, we turn our attention towards high frequency features. As discussed in Section 6.2, such features could be introduced by various forms of fixed sensor pattern noise, dust particles stuck to the lens, and image processing / compression algorithms performed automatically by the camera. We investigate the role of high frequency features by training CNNs to classify cameras given only a random 32×32 crop of our original input images (upsampled to 224×224 for Alexnet and VGG16). As Table 6.5 shows, camera identification accuracy remains surprisingly robust even when input is restricted to a 32×32 window. This strongly implies that high frequency features are sufficient for camera identification, and confirms that lens distortion is not required. However, it remains unclear whether these features are localized to certain regions of the image or present uniformly. Figure 6.9 shows an accuracy heatmap, constructed by repeatedly sampling 32×32 crops from our training set and drawing a white square at the location of each correctly classified crop. This shows that classification accuracy is independent of the location of the crop, at least when averaged over the whole dataset. This implies that whatever pattern is being exploited occurs uniformly across the images on average. Figure 6.8 shows how classification accuracy for 32×32 crops varies across five individual images, one from each camera.

6.4.7 Generalizing from Left Field of View to Right

Pixel non-uniformity (PNU) noise, as described in [152], is a high frequency noise fingerprint, manifested as randomly varying sensitivities of individual sensor pixels to light. We would expect such a noise fingerprint to be non-repeating, that is, the noise pattern in one part of an image should be different to that in other parts. If the models are learning to recognize

Model	Test Accuracy
ResNet34	0.665
ResNet101	0.681
InceptionV3	0.948
AlexNet	0.770
VGG16	0.872

Table 6.5: Camera classification test accuracy when trained only on random 32×32 crops of the input data. High accuracy in this regime implies that high frequency features are sufficient for camera classification, and lens deformation (which would not be detectable in a 32×32 region) is not necessary.

Model	Test Accuracy
ResNet34	0.992
ResNet101	0.889
InceptionV3	0.999
AlexNet	0.818
VGG16	0.851

Table 6.6: Camera classification accuracy on right halves of images after training on the left halves. Strong generalization to an unseen area of the training images implies that PNU noise fingerprints of the sort discussed by Lukas et al. [152] are unlikely to be the mechanism by which CNNs are recognizing cameras, because the noise fingerprint on the right side of the images will be different to that on the left side.

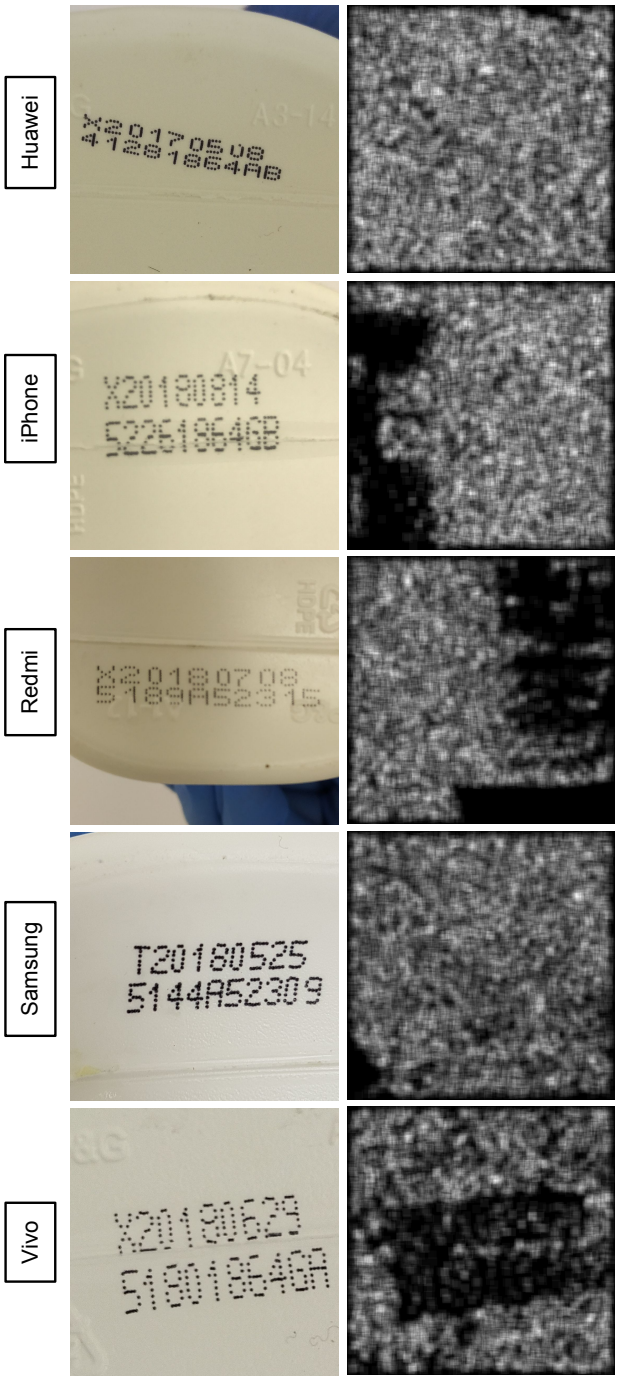


Figure 6.8: Heatmaps representing the camera identification accuracy on 32×32 patches at different locations in single images (white = 100% accuracy, black = 0%). An image from each camera is shown, and the predictions are all from the same ResNet34 checkpoint. The model is able to correctly classify patches from most locations on most images, but some significant dark patches occur.

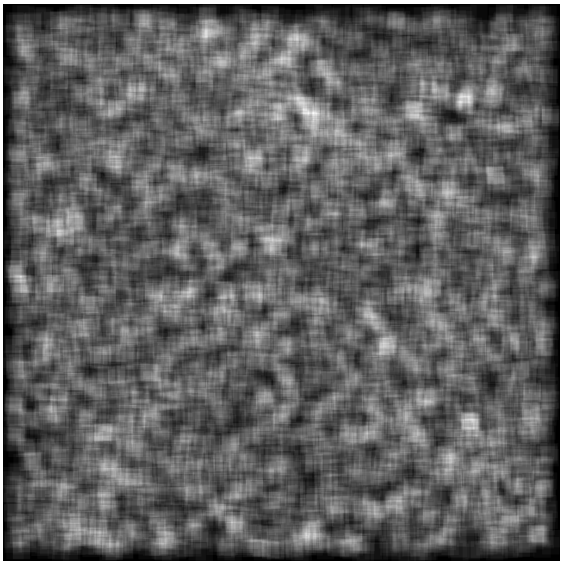


Figure 6.9: Heatmap representing relative camera classification accuracy of 32×32 crops at different locations in the image, averaged across images from the whole dataset. The lack of bias toward any particular part of the image implies that camera predictive patterns are present uniformly across the images.

cameras by recognizing their PNU noise fingerprints, they should therefore be incapable of recognizing cameras from patches of noise fingerprint they have not encountered during training. We therefore test our models' reliance on PNU noise by training them on only the left halves of our Balanced training set images, and testing on the right halves. If they are reliant on PNU noise then generalization to the right halves of images should be poor. As Table 6.6 shows, this is not the case, therefore PNU noise is unlikely to be the primary source of camera identifying information.

6.5 Conclusion

We have shown that CNNs learn to exploit camera / class label correlations in an image classification dataset in which such correlations are present. By recognizing the camera that acquired an image, CNNs are able to infer the class label without learning any features that are relevant to the task (in our case, manufacturer classification), as evidenced by poor generalization to images where the camera / label correlation is broken. We also show that CNNs are capable of learning to infer origin cameras when explicitly trained to do so, corroborating the results of Bondi et al. and Tuama et al. [154, 155]. We test these phenomena across five different CNN architectures and show that the effects are common to all of them, although lesser among AlexNet and VGG16, the two architectures whose inputs must be downsampled to a smaller size due to the use of fully connected layers. We have also performed a number of experiments to gain insight into how CNNs are recognizing cameras, the results of which require some discussion in this section. Section 6.2 outlines a number of potential sources of camera identifying information, and our experiments provide evidence for and against those hypotheses.

A simple explanation for camera bias would be differences in average color statistics among cameras, caused by differences in white balance and color correction settings. This hypothesis is largely ruled out by the fact that CNNs still recognize cameras easily even when hue, saturation, contrast and brightness are randomized (see Figure 6.7, Table 6.4). Another potential explanation was lens distortion; if different cameras have different shaped lenses then there may be slight differences in geometric distortion (e.g. radial lens distortion [150]). This hypothesis too is ruled out, by the fact that CNNs are incapable of inferring cameras from binary segmented images (see Figure 6.6, Table 6.3). Geometric distortions would be visible in the spacing of the dots from the batch codes, which are the only features visible in these images. Chromatic aberration is also unlikely since it should only be visible at the edges of objects, not in flat regions (Figure 6.5,6.8), and should also be undetectable in grayscale images (Section 6.4.5). These results increase the

likelihood that texture, which is absent in segmented images but preserved in color randomized images, plays an important role. High camera recognition accuracy on 32×32 random crops (Table 6.5), including in empty patches of the image where it is hard to imagine what features besides faint, high frequency texture are available (Figure 6.8), increases this likelihood further.

There are two likely sources of camera correlated texture: pixel non-uniformity (PNU) noise, and the camera's on-board image processing, which typically includes algorithms such as kernel filtering, image sharpening and compression (both discussed by Lukas et al. [152]). PNU noise is dominated by a fixed multiplicative noise pattern that is introduced during manufacturing, as such we would expect different noise patterns in different parts of the field of view, as opposed to a repeating pattern. The fact that CNNs trained on the left hand sides of images generalize well to the right hand sides of those images (Table 6.6) therefore implies that PNU noise is not crucial for camera recognition, since they should not be able to recognize unseen noise patterns on the right side of the images. The fact that AlexNet and VGG16 are also able to recognize cameras from downsampled images is also strong evidence against PNU noise, which should be undetectable after downsampling.

By a process of elimination, the most likely explanation therefore seems to be on-camera image processing algorithms. We do not consider these results to be conclusive; a conclusive answer would require full knowledge of the original cameras, which we do not have. Further research is required to ascertain exactly which textural features are exploited by CNNs to recognize cameras.

Epilogue

In this chapter, we have demonstrated that CNNs are able to cheat at image classification tasks by exploiting correlations between camera type and class label. By recognizing the camera which an image originated from, these correlations allow networks to infer class label reliably without ever learning robust features that predict the class accurately independent of camera type. This is a form of domain bias, as discussed in Chapter 4, and to the best of our knowledge this particular source of domain bias has not previously been reported in the literature.

Throughout this thesis, we have focused on practical obstacles to adoption of deep learning, including overdetection of objects (Chapter 3), domain bias (Chapter 4, Chapter 6), and lack of labelled training data (Chapter 5). We hope that the contributions outlined in these chapters will be of use to the applied deep learning community.

Chapter 7

Conclusion

In this thesis, we have developed new machine vision techniques for object detection, data augmentation, and unsupervised clustering of cellular images. We have also presented evidence that CNNs are able to recognize the individual camera that acquired an image, and will use this information to cheat a binary classification task if it correlates with the class label. These contributions are relevant to machine vision practitioners in a variety of practical application settings, and also have some theoretical implications. In particular, the findings from our work on Style Augmentation (Chapter 4) add to a growing pile of evidence ([121, 168, 169]) that CNNs have an inductive bias in favour of learning low level textural features, as opposed to higher level shape features. Likewise, our work on Camera Bias (Chapter 6) shows that CNNs are able to learn subtle, high frequency features that are indiscernible to the human eye very quickly (Figure 6.2).

We have introduced a novel loss function for deep object detection networks, which incentivises the network to emit a single bounding box per object (Chapter 3). This is in contrast to standard object detection approaches in deep learning, in which networks typically emit many boxes per object which must then be pruned by non-maximum suppression. In doing so, we elucidated a fundamental problem in which most supervised learning settings use a loss function that measures the deviation of the network's output from a single correct answer, but the object detection task has numerous correct outputs for any given input. State of the art object detection CNNs solve this dilemma with loss functions that establish jurisdictions for the final layer object

detectors, and require any detector whose jurisdiction contains an object to emit a box - since the jurisdictions are dense and overlapping to ensure complete coverage when multiple objects are present, this generally results in over-detection. Although the proposed loss function does largely eliminate the problem of overdetection, it does so at the cost of lower accuracy and occasional underdetection. We conclude that any attempt by a network which emits object bounding boxes in parallel (as is the case in my work and most contemporary approaches) to avoid overdetection will likely be thwarted by a coordination problem. If detectors (groups of output neurons whose activations encode a bounding box) are to each label a unique object in parallel, then they must do so without knowledge of each others' activations. The behaviour of each detector will always be a function whose output is fixed entirely by the image patch covered by its receptive field, therefore, to avoid overdetection, these functions must be learned such that detectors covering the same or overlapping patches must respond selectively to objects of different size, shape or position within that patch. The precise nature of these learned jurisdictions - the types of attributes they select for, the boundaries between neighbouring jurisdictions, and the mechanisms by which they evolve - remains an open question.

In Chapter 4, we proposed a new form of data augmentation called Style Augmentation. Style Augmentation is intended to improve the robustness of CNNs by randomizing textures in the training set, forcing the network to rely less on texture and more on shape for its predictions. We show that this mitigates the effects of domain bias in object classification and depth estimation tasks, and functions effectively in combination with traditional data augmentations in settings without domain bias. This result corroborates the work of Geirhos et al. [121], who shows that ImageNet trained CNNs are prone to rely on texture and ignore shape, and that this behaviour can be mitigated by de-correlating texture with class label in the training set. This conclusion is further supported by other results in the literature, such as discovery that CNNs constrained to only see local image patches achieve good accuracy on ImageNet [169], and the poor performance of CNNs on sketch-like images which contain shape but no texture [168].

My work on phenotypic profiling (Chapter 5) aimed to speed up the pre-clinical candidate filtering stage in pharmaceutical drug discovery, by clustering candidate compounds by their morphological effects on human kidney cells, as observed in a high throughput imaging screen. This allows biomedical researchers to rapidly explore and gain insight into a dataset of millions of images, as is common in high throughput screening (though the project in this thesis sufficed with only $40,000$ images). By using CNNs to compute a low dimensional embedding vector for each image, such that similar images have low Euclidean distance in the embedding space,

we were able to cluster the images by visual similarity using the k-means algorithm. This effectively classifies the images by phenotype, allowing researchers to accept or discard whole classes at a time, rather than having to consider each image individually. A surprising finding from this work was that using convolutional features derived from a CNN trained on ImageNet turned out to be far more effective than any attempt to learn features from the cellular images themselves. Multiple attempts to train CNNs to produce good embeddings using unsupervised learning objectives, including contrastive loss [170] and adversarial loss (specifically, BiGANs [139]), resulted in far inferior embeddings compared to simply averaging feature map activations from a pre-trained CNN and reducing dimensionality via principal component analysis. This is counter-intuitive, since ImageNet generally consists of photographs of everyday objects and animals, and does not contain anything that looks like a cellular image captured by fluorescent microscopy. The fact that CNN features, taken from early layers in an ImageNet trained network, are general enough to describe cellular images supports the intuition that CNNs learn a hierarchy of increasingly complex features, with the early, low level features being more general and the higher level features being more task specific.

Finally, in Chapter 6 we investigate the causes of heavy domain bias in a real world counterfeit detection task. In a setting where the dataset is acquired by five unique cameras, we show that CNNs are able to infer which individual camera captured an image, and that when the class label correlates with the camera (e.g. all counterfeit products photographed by the same two cameras), they readily exploit this correlation to minimize training loss. This results in a brittle model that fails completely to generalize to any setting where different cameras are used. These results provide further evidence that CNNs have an inductive bias toward learning low level texture like features, at least when they correlate strongly enough with the target output.

7.1 Contributions

In this thesis, we have:

- Created a new loss function for combined object detection and counting, which is shown to mostly eliminate the problem of over-detection, making non maximum suppression unnecessary
- Created Style Augmentation, a data augmentation tool that randomizes texture in training images, and provided an open source implementation

- Found evidence to support the emerging view that CNNs have a strong inductive bias toward learning to use texture for their decisions
- Built an interactive tool for viewing the distribution of cellular morphological phenotypes in a high throughput imaging screen
- Created an embedding function from CNN features pre-trained on ImageNet, which clusters cellular images by visual similarity, enabling researchers to quickly filter large image datasets produced by high throughput screening
- Shown that CNNs are not only able to infer the camera from which an image originated, but will readily exploit this ability to predict class label if it correlates.

7.2 Future Work

Since Chapters 4 and 6 both deal heavily with the intuitive notion of "low level" or "textural" features, it might be valuable to try to solidify and quantify this intuition. What exactly is meant by low level features, and how can we quantify the degree to which a CNN is utilising them in its predictions? Since the distinction between low and high level features corresponds to a distinction in physical scale, with lower level features being smaller in scale and hence visible from smaller image patches, it would seem natural to define a feature's level in terms of which layer in a CNN is detecting it. Lower level (or "earlier") layers in the CNN stack have smaller receptive fields and occur before most of the pooling layers, meaning they observe smaller patches of the image and at higher resolution than later layers, making them apt to detect simpler, lower level features such as edges [11, 17]. Thus, we would expect that in situations where CNNs are heavily utilising low level features to make decisions, this information would be picked up in the earlier convolutional layers, with the later layers extracting relatively less useful information, mostly just passing along information that was extracted by earlier layers. It would be interesting therefore to test this intuition, by attempting to directly observe and measure it in action. This could be accomplished by training standard CNN architectures on custom toy datasets where features of controllable scale are known to be the only discriminative features for the task to be learned, and observing how the internal dynamics of these CNNs depend on the scale of the features we know they are learning.

Bibliography

1. Jackson, P. T. & Obara, B. *Avoiding Over-Detection: Towards Combined Object Detection and Counting* in *International Conference on Artificial Intelligence and Soft Computing* (2017), 75–85 (*pp.* iii, 35).
2. Jackson, P. T., Atapour-Abarghouei, A., Bonner, S., Breckon, T. & Obara, B. Style Augmentation: Data Augmentation via Style Randomization. *CVPR DeepVision* (2018) (*pp.* iii, 49).
3. Jackson, P. T., Wang, Y., Knight, S., Chen, H., Dorval, T., Brown, M., Bendtsen, C. & Obara, B. *Phenotypic profiling of high throughput imaging screens with generic deep convolutional features* in *2019 16th International Conference on Machine Vision Applications (MVA)* (2019), 1–4 (*pp.* iii, 71).
4. Jackson, P., Bonner, S., Jia, N., Holder, C., Stonehouse, J. & Obara, B. Camera Bias in a Fine Grained Classification Task. *CVPR DeepVision.* In review (2020) (*pp.* iii, 82).
5. Willcocks, C. G., Jackson, P. T., Nelson, C. J. & Obara, B. Extracting 3D parametric curves from 2D images of helical objects. *IEEE transactions on pattern analysis and machine intelligence* **39,** 1757–1769 (2016) (*p.* iv).
6. Nasrulloh, A. V., Willcocks, C. G., Jackson, P. T., Geenen, C., Habib, M. S., Steel, D. H. & Obara, B. Multi-scale segmentation and surface fitting for measuring 3-D macular holes. *IEEE transactions on medical imaging* **37,** 580–589 (2017) (*p.* iv).
7. Willcocks, C. G., Jackson, P. T., Nelson, C. J., Nasrulloh, A. V. & Obara, B. Interactive GPU active contours for segmenting inhomogeneous objects. *Journal of Real-Time Image Processing* **16,** 2305–2318 (2019) (*p.* iv).

8. Alharbi, S. S., Willcocks, C. G., Jackson, P. T., Alhasson, H. F. & Obara, B. Sequential graph-based extraction of curvilinear structures. *Signal, Image and Video Processing* **13,** 941–949 (2019) (*p.* iv).

9. Nelson, C. J., Jackson, P. T. & Obara, B. *Combining mathematical morphology and the Hilbert transform for fully automatic nuclei detection in fluorescence microscopy* in *International Symposium on Mathematical Morphology and Its Applications to Signal and Image Processing* (2019), 532–543 (*p.* iv).

10. Bonner, S., Atapour-Abarghouei, A., Jackson, P. T., Brennan, J., Kureshi, I., Theodoropoulos, G., McGough, A. S. & Obara, B. Temporal Neighbourhood Aggregation: Predicting Future Links in Temporal Graphs via Recurrent Variational Graph Convolutions. *arXiv preprint arXiv:1908.08402* (2019) (*p.* iv).

11. LeCun, Y., Bengio, Y. & Hinton, G. Deep learning. *nature* **521,** 436–444 (2015) (*pp.* 1, 49, 105).

12. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V. & Rabinovich, A. *Going deeper with convolutions* in *Proceedings of the IEEE conference on computer vision and pattern recognition* (2015), 1–9 (*pp.* 1, 5, 31).

13. Ren, S., He, K., Girshick, R. & Sun, J. *Faster r-cnn: Towards real-time object detection with region proposal networks* in *Advances in neural information processing systems* (2015), 91–99 (*pp.* 1, 5, 35, 37).

14. Wang, C., Zhang, H., Yang, L., Liu, S. & Cao, X. *Deep people counting in extremely dense crowds* in *Proceedings of the 23rd ACM international conference on Multimedia* (2015), 1299–1302 (*p.* 1).

15. Ronneberger, O., Fischer, P. & Brox, T. *U-net: Convolutional networks for biomedical image segmentation* in *International Conference on Medical image computing and computer-assisted intervention* (2015), 234–241 (*pp.* 1, 5).

16. Laina, I., Rupprecht, C., Belagiannis, V., Tombari, F. & Navab, N. *Deeper depth prediction with fully convolutional residual networks* in *2016 Fourth international conference on 3D vision (3DV)* (2016), 239–248 (*p.* 1).

17. Goodfellow, I., Bengio, Y. & Courville, A. *Deep learning* (MIT press, 2016) (*pp.* 1, 23, 105).

18. Zhang, Q., Zha, L., Lin, J., Tu, D., Li, M., Liang, F., Wu, R. & Lu, X. *A Survey on Deep Learning Benchmarks: Do We Still Need New Ones?* in *International Symposium on Benchmarking, Measuring and Optimization* (2018), 36–49 (*p.* 2).

19. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C. & Fei-Fei, L. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)* **115,** 211–252 (2015) (*pp.* 2, 9, 22, 25, 27).

20. Rothe, R., Guillaumin, M. & Van Gool, L. *Non-Maximum Suppression for Object Detection by Passing Messages Between Windows* in *Asian Conference on Computer Vision* (2014), 290–306 (*pp.* 2, 35).

21. Sam, D. B., Peri, S. V., Kamath, A., Babu, R. V. *et al.* Locate, Size and Count: Accurately Resolving People in Dense Crowds via Detection. *arXiv preprint arXiv:1906.07538* (2019) (*p.* 3).

22. Atapour-Abarghouei, A. & Breckon, T. P. *Real-Time Monocular Depth Estimation using Synthetic Data with Domain Adaptation via Image Style Transfer* in *Conf. Computer Vision Pattern Recognition* (2018), 1–8 (*pp.* 3, 50, 52, 64).

23. Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W. & Abbeel, P. *Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World* in *Int. Conf. Intelligent Robots and Systems* (2017), 23–30 (*pp.* 3, 50).

24. Gatys, L. A., Ecker, A. S. & Bethge, M. *Image Style Transfer using Convolutional Neural Networks* in *IEEE Conf. Computer Vision and Pattern Recognition* (2016), 2414–2423 (*pp.* 4, 50, 53–54, 56).

25. Gatys, L. A., Ecker, A. S. & Bethge, M. Texture and art with deep neural networks. *Current opinion in neurobiology* **46,** 178–186 (2017) (*p.* 4).

26. LeCun, Y., Bottou, L., Bengio, Y. & Haffner, P. Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE* **86,** 2278–2324 (1998) (*pp.* 5, 26, 31, 55).

27. Simard, P., Steinkraus, D. & Platt, J. *Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis* in *Int. Conf. Document Analysis and Recognition* **1** (2003), 958–963 (*pp.* 5, 55).

28. Krizhevsky, A., Sutskever, I. & Hinton, G. E. ImageNet Classification with Deep Convolutional Neural Networks. *ACM Communications* **60,** 84–90 (2017) (*pp.* 5, 49, 55).

29. Radford, A., Wu, J., Child, R., Luan, D., Amodei, D. & Sutskever, I. Language models are unsupervised multitask learners. *OpenAI Blog* **1,** 9 (2019) (*p.* 5).

30. Sorokin, A. & Forsyth, D. *Utility data annotation with amazon mechanical turk* in *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops* (2008), 1–8 (*p.* 5).

31. Yosinski, J., Clune, J., Bengio, Y. & Lipson, H. *How Transferable Are Features in Deep Neural Networks?* in *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2* (MIT Press, Montreal, Canada, 2014), 3320–3328. http://dl.acm.org/citation.cfm?id=2969033.2969197 (*pp.* 6, 73).

32. Krizhevsky, A., Sutskever, I. & Hinton, G. E. *Imagenet classification with deep convolutional neural networks* in *Advances in neural information processing systems* (2012), 1097–1105 (*p.* 9).

33. Li, Y. Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274* (2017) (*p.* 9).

34. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł. & Polosukhin, I. *Attention is all you need* in *Advances in neural information processing systems* (2017), 5998–6008 (*p.* 9).

35. McCulloch, W. S. & Pitts, W. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics* **5,** 115–133 (1943) (*p.* 9).

36. Rosenblatt, F. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review* **65,** 386 (1958) (*p.* 11).

37. Minsky, M. & Papert, S. A. *Perceptrons: An introduction to computational geometry* (MIT press, 2017) (*p.* 13).

38. Olazaran, M. A sociological study of the official history of the perceptrons controversy. *Social Studies of Science* **26,** 611–659 (1996) (*p.* 13).

39. Pollack, J. B. *No harm intended: Marvin L. Minsky and Seymour A. Papert. Perceptrons: An Introduction to Computational Geometry, Expanded Edition. Cambridge, MA: MIT Press, 1988. Pp. 292. $12.50* 1989 (*p.* 13).

40. Hornik, K., Stinchcombe, M. & White, H. Multilayer feedforward networks are universal approximators. *Neural networks* **2,** 359–366 (1989) (*p.* 13).

41. Hornik, K. Approximation capabilities of multilayer feedforward networks. *Neural networks* **4,** 251–257 (1991) (*p.* 13).

42. Widrow, B. & Hoff, M. E. *Adaptive switching circuits* technical report (Stanford Univ Ca Stanford Electronics Labs, 1960) (*p.* 13).

43. Hebb, D. O. *The organization of behavior: A neuropsychological theory* (Psychology Press, 2005) (*p.* 13).

44. Kohonen, T. Self-organized formation of topologically correct feature maps. *Biological cybernetics* **43,** 59–69 (1982) (*p.* 13).

45. Hopfield, J. J. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences* **79,** 2554–2558 (1982) (*p.* 13).

46. *Who Invented Backpropagation?* http://people.idsia.ch/~juergen/who-invented-backpropagation.html. Accessed: 29-04-2019 (*p.* 13).

47. Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M. *et al. Tensorflow: A system for large-scale machine learning* in *12th Symposium on Operating Systems Design and Implementation* (2016), 265–283 (*p.* 17).

48. Bengio, Y. *et al.* Learning deep architectures for AI. *Foundations and trends® in Machine Learning* **2,** 1–127 (2009) (*p.* 21).

49. Niebles, J. C. & Li Fei-Fei. *A Hierarchical Model of Shape and Appearance for Human Action Classification* in *2007 IEEE Conference on Computer Vision and Pattern Recognition* (June 2007), 1–8 (*p.* 21).

50. Sudderth, E. B., Torralba, A., Freeman, W. T. & Willsky, A. S. Describing visual scenes using transformed objects and parts. *International Journal of Computer Vision* **77,** 291–330 (2008) (*p.* 21).

51. Weber, M., Welling, M. & Perona, P. *Unsupervised Learning of Models for Recognition* in. **1842** (June 2000), 18–32 (*p.* 21).

52. Serre, T., Kreiman, G., Kouh, M., Cadieu, C., Knoblich, U. & Poggio, T. A quantitative theory of immediate visual recognition. *Progress in brain research* **165,** 33–56 (2007) (*p.* 21).

53. Utgoff, P. E. & Stracuzzi, D. J. Many-layered learning. *Neural Computation* **14,** 2497–2529 (2002) (*p.* 21).

54. Håstad, J. *Computational limitations of small-depth circuits* (MIT Press, 1987) (*p.* 22).

55. Håstad, J. & Goldmann, M. On the power of small-depth threshold circuits. *Computational Complexity* **1,** 113–129 (1991) (*p.* 22).

56. Bengio, Y., LeCun, Y. *et al.* Scaling learning algorithms towards AI. *Large-scale kernel machines* **34,** 1–41 (2007) (*p.* 22).

57. Pascanu, R., Montufar, G. & Bengio, Y. On the number of response regions of deep feed forward networks with piece-wise linear activations. *arXiv preprint arXiv:1312.6098* (2013) (*p.* 22).

58. Hochreiter, S., Bengio, Y., Frasconi, P., Schmidhuber, J. *et al. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies* 2001 (*p.* 22).

59. Glorot, X. & Bengio, Y. *Understanding the difficulty of training deep feedforward neural networks* in *Proceedings of the thirteenth international conference on artificial intelligence and statistics* (2010), 249–256 (*p.* 24).

60. Li, C., Farkhoor, H., Liu, R. & Yosinski, J. Measuring the intrinsic dimension of objective landscapes. *arXiv preprint arXiv:1804.08838* (2018) (*p.* 26).

61. Hinton, G., Srivastava, N. & Swersky, K. *Neural Networks for Machine Learning Lecture 5a - Why object recognition is difficult* http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec5.pdf (*p.* 26).

62. Fukushima, K. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics* **36,** 193–202 (1980) (*p.* 26).

63. Simonyan, K. & Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014) (*p.* 31).

64. He, K., Zhang, X., Ren, S. & Sun, J. *Deep residual learning for image recognition* in *Proceedings of the IEEE conference on computer vision and pattern recognition* (2016), 770–778 (*pp.* 31, 61).

65. Springenberg, J. T., Dosovitskiy, A., Brox, T. & Riedmiller, M. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806* (2014) (*pp.* 32, 85).

66. Hadi Hosseini, S., Chen, H. & Jablonski, M. M. Automatic detection and counting of retina cell nuclei using deep learning. *arXiv,* arXiv–2002 (2020) (*p.* 35).

67. Chen, T. & Chefd'Hotel, C. *Deep learning based automatic immune cell detection for immunohistochemistry images* in *International workshop on machine learning in medical imaging* (2014), 17–24 (*p.* 35).

68. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M. S., Berg, A. C. & Li, F.-F. *ImageNet Large Scale Visual Recognition Challenge* in. **1409** (2014) (*p.* 35).

69. Lehmussola, A., Ruusuvuori, P., Selinummi, J., Huttunen, H. & Yli-Harja, O. Computational Framework for Simulating Fluorescence Microscope Images With Cell Populations. *IEEE Transactions on Medical Imaging* **26,** 1010–1016 (2007) (*p.* 36).

70. Ruusuvuori, P., Manninen, T. & Huttunen, H. *Image segmentation using sparse logistic regression with spatial prior* in *2012 Proceedings of the 20th European Signal Processing Conference (EUSIPCO)* (Aug. 2012), 2253–2257 (*p.* 36).

71. Ljosa, V., Sokolnicki, K. L. & Carpenter, A. E. Annotated High-Throughput Microscopy Image Sets for Validation. *Nature Methods* **9,** 637 (2012) (*p.* 37).

72. Girshick, R., Donahue, J., Darrell, T. & Malik, J. *Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation* in *IEEE Conference on Computer Vision and Pattern Recognition* (2014), 580–587 (*p.* 36).

73. Uijlings, J. R., van de Sande, K. E., Gevers, T. & Smeulders, A. W. Selective Search for Object Recognition. *International Journal of computer vision* **104,** 154–171 (2013) (*p.* 36).

74. Girshick, R. *Fast R-CNN* in *IEEE International Conference on Computer Vision* (2015), 1440–1448 (*p.* 37).

75. Redmon, J., Divvala, S., Girshick, R. & Farhadi, A. You Only Look Once: Unified, Real-Time Object Detection. *Computing Research Repository* (2015) (*pp.* 38, 41).

76. Krizhevsky, A., Sutskever, I. & Hinton, G. E. ImageNet Classification with Deep Convolutional Neural Networks. *Advances In Neural Information Processing Systems,* 1–9 (2012) (*pp.* 38–39).

77. Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J. & Zisserman, A. *The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results* http://www.pascal-network.org/challenges/VOC (*p.* 38).

78. Seguí, S., Pujol, O. & Vitria, J. *Learning to Count with Deep Object Features* in *IEEE Conference on Computer Vision and Pattern Recognition* (2015), 90–96 (*p.* 38).

79. Litjens, G., Sánchez, C. I., Timofeeva, N., Hermsen, M., Nagtegaal, I., Kovacs, I., Hulsbergen-van de Kaa, C., Bult, P., van Ginneken, B. & van der Laak, J. Deep Learning as a Tool for Increased Accuracy and Efficiency of Histopathological Diagnosis. *Scientific Reports* **6** (2016) (*p.* 39).

80. Ciresan, D., Giusti, A., Gambardella, L. M. & Schmidhuber, J. in *Advances in Neural Information Processing Systems 25* 2843–2851 (Curran Associates, Inc., 2012) (*p.* 39).

81. Prentašić, P. & Lončarić, S. *Detection of Exudates in Fundus Photographs Using Convolutional Neural Networks* in *International Symposium on Image and Signal Processing and Analysis* (2015), 188–192 (*p.* 39).

82. Dong, B., Shao, L., Da Costa, M., Bandmann, O. & Frangi, A. F. *Deep Learning for Automatic Cell Detection in Wide-Field Microscopy Zebrafish Images* in *IEEE International Symposium on Biomedical Imaging* (2015), 772–776 (*p.* 39).

83. Kraus, O. Z., Ba, J. L. & Frey, B. J. Classifying and Segmenting Microscopy Images with Deep Multiple Instance Learning. *Bioinformatics* **32,** 52–59 (2016) (*p.* 39).

84. Simonyan, K., Vedaldi, A. & Zisserman, A. *Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps* in (2013) (*pp.* 39, 85).

85. Zeiler, M. D. & Fergus, R. *Visualizing and Understanding Convolutional Networks* in *European Conference on Computer Vision* (2014), 818–833 (*pp.* 39, 85).

86. Khan, A., Gould, S. & Salzmann, M. *Deep Convolutional Neural Networks for Human Embryonic Cell Counting* in *European Conference on Computer Vision* (2016), 339–348 (*p.* 39).

87. Maas, A. L., Hannun, A. Y. & Ng, A. Y. *Rectifier Nonlinearities Improve Neural Network Acoustic Models* in *International Conference on Machine Learning* **30** (2013) (*p.* 43).

88. Kingma, D. & Ba, J. Adam: A Method for Stochastic Optimization. *International Conference on Learning Representations* (2015) (*p.* 44).

89. Hoffman, J., Wang, D., Yu, F. & Darrell, T. Fcns in the wild: Pixel-level adversarial and constraint-based adaptation. *arXiv preprint arXiv:1612.02649* (2016) (*p.* 48).

90. Yaeger, L. S., Lyon, R. F. & Webb, B. J. in *Advances in Neural Information Processing Systems* 807–816 (1997) (*pp.* 49, 55).

91. Zhu, J.-Y., Park, T., Isola, P. & Efros, A. A. *Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks* in *Int. Conf. Computer Vision* (2017) (*p.* 50).

92. Johnson, J., Alahi, A. & Fei-Fei, L. *Perceptual Losses for Real-Time Style Transfer and Super-Resolution* in *Euro. Conf. Computer Vision* (2016), 694–711 (*pp.* 50, 54, 56).

93. Saenko, K., Kulis, B., Fritz, M. & Darrell, T. *Adapting Visual Category Models to New Domains* in *Euro. Conf. Computer Vision* (2010), 213–226 (*pp.* 50, 58, 61).

94. Ghiasi, G., Lee, H., Kudlur, M., Dumoulin, V. & Shlens, J. *Exploring the structure of a real-time, arbitrary neural artistic stylization network* in *British Machine Vision Conference* (2017) (*pp.* 50, 54, 56–57).

95. Gretton, A., Smola, A., Huang, J., Schmittfull, M., Borgwardt, K. & Schölkopf, B. *Covariate Shift by Kernel Mean Matching* (MIT press, 2008) (*p.* 52).

96. Uhrig, J., Schneider, N., Schneider, L., Franke, U., Brox, T. & Geiger, A. *Sparsity Invariant CNNs* in *Int. Conf. 3D Vision* (2017), 11–20 (*pp.* 52, 58, 64, 69).

97. Shao, L., Zhu, F. & Li, X. Transfer Learning for Visual Categorization: A Survey. *IEEE Trans. Neural Networks and Learning Systems* **26,** 1019–1034. ISSN: 2162-237X (2015) (*p.* 52).

98. Yosinski, J., Clune, J., Bengio, Y. & Lipson, H. in *Advances in Neural Information Processing Systems* 3320–3328 (2014) (*p.* 52).

99. And Yue Cao and Jianmin Wang, M. L. & Jordan, M. I. *Learning Transferable Features with Deep Adaptation Networks* in *Int. Conf. Machine Learning* (2015), 97–105 (*p.* 52).

100. Ghifary, M., Kleijn, W. B., Zhang, M., Balduzzi, D. & Li, W. *Deep Reconstruction-Classification Networks for Unsupervised Domain Adaptation* in *Int. Conf. Computer Vision* (2016), 597–613 (*p.* 52).

101. Hoffman, J., Tzeng, E., Darrell, T. & Saenko, K. in *Domain Adaptation in Computer Vision Applications* 173–187 (2017) (*p.* 52).

102. Donahue, J., Krähenbühl, P. & Darrell, T. *Adversarial Feature Learning* in *Int. Conf. Learning Representations* (2017) (*p.* 52).

103. Tzeng, E., Hoffman, J., Saenko, K. & Darrell, T. *Adversarial Discriminative Domain Adaptation* in *IEEE Conf. Computer Vision and Pattern Recognition* (2017) (*p.* 52).

104. Li, Y., Wang, N., Shi, J., Liu, J. & Hou, X. Revisiting Batch Normalization for Practical Domain Adaptation. *arXiv:1603.04779* (2016) (*p.* 52).

105. Sun, B. & Saenko, K. *Deep CORAL: Correlation Alignment for Deep Domain Adaptation* in *Workshops in Euro. Conf. Computer Vision* (2016), 443–450 (*p.* 52).

106. Li, Y., Wang, N., Liu, J. & Hou, X. *Demystifying Neural Style Transfer* in *Int. Conf. Artificial Intelligence* (2017), 2230–2236 (*p.* 52).

107. Schwerdtfeger, H. *Introduction to Linear Algebra and the Theory of Matrices* (P. Noordhoff, 1950) (*p.* 53).

108. Simonyan, K. & Zisserman, A. *Very Deep Convolutional Networks for Large-Scale Image Recognition* in *Int. Conf. Learning Representations* (2015) (*pp.* 53, 61).

109. Ulyanov, D., Lebedev, V., Vedaldi, A. & Lempitsky, V. S. *Texture Networks: Feed-forward Synthesis of Textures and Stylized Images* in *Int. Conf. Machine Learning* (2016), 1349–1357 (*p.* 54).

110. Chen, T. Q. & Schmidt, M. *Fast Patch-based Style Transfer of Arbitrary Style* in *Workshop in Constructive Machine Learning* (2016) (*p.* 54).

111. Ulyanov, D., Vedaldi, A. & Lempitsky, V. Instance Normalization: The Missing Ingredient for Fast Stylization. *arXiv:1607.08022* (2016) (*p.* 54).

112. Dumoulin, V., Shlens, J. & Kudlur, M. *A Learned Representation for Artistic Style* in *Int. Conf. Learning Representations* (2017) (*pp.* 54, 56).

113. Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J. & Wojna, Z. *Rethinking the Inception Architecture for Computer Vision* in *IEEE Conf. Computer Vision and Pattern Recognition* (2016), 2818–2826 (*pp.* 54, 59–62).

114. Huang, X. & Belongie, S. *Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization* in *Int. Conf. Computer Vision* (2017) (*p.* 54).

115. Yanai, K. *Unseen Style Transfer Based on a Conditional Fast Style Transfer Network* in *Learning Representations Workshops* (2017) (*p.* 54).

116. Ciresan, D. C., Meier, U., Gambardella, L. M. & Schmidhuber, J. Deep Big Simple Neural Nets Excel on Digit Recognition. *Neural Computation* **22,** 3207–3220 (2010) (*p.* 55).

117. Wong, S. C., Gatt, A., Stamatescu, V. & McDonnell, M. D. Understanding Data Augmentation for Classification: When to Warp? *arXiv:1609.08764* (2016) (*p.* 55).

118. Bouthillier, X., Konda, K., Vincent, P. & Memisevic, R. Dropout as Data Augmentation. *arXiv:1506.08700* (2015) (*p.* 55).

119. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. & Salakhutdinov, R. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Machine Learning Research* **15,** 1929–1958 (2014) (*p.* 55).

120. Zhong, Z., Zheng, L., Kang, G., Li, S. & Yang, Y. Random Erasing Data Augmentation. *arXiv:1708.04896* (2017) (*pp.* 55, 60).

121. Geirhos, R., Rubisch, P., Michaelis, C., Bethge, M., Wichmann, F. A. & Brendel, W. ImageNet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness. *arXiv preprint arXiv:1811.12231* (2018) (*pp.* 55, 65, 102–103).

122. Coates, A., Ng, A. & Lee, H. *An Analysis of Single-Layer Networks in Unsupervised Feature Learning* in *Int. Conf. Artificial Intelligence and Statistics* (2011), 215–223 (*pp.* 58–60).

123. Zhao, J., Mathieu, M., Goroshin, R. & LeCun, Y. *Stacked What-Where Auto-encoders* in *Int. Conf. Learning Representations* (2016) (*p.* 60).

124. Thoma, M. *Analysis and Optimization of Convolutional Neural Network Architectures* Master's thesis (Karlsruhe Institute of Technology, 2017) (*p.* 60).

125. Miralles, R. *An Open-Source Development Environment for Self-Driving Vehicles* Master's thesis (Universitat Oberta de Catalunya, 2017) (*p.* 64).

126. Isola, P., Zhu, J.-Y., Zhou, T. & Efros, A. A. *Image-to-Image Translation with Conditional Adversarial Networks* in *IEEE Conf. Computer Vision and Pattern Recognition* (2017) (*p.* 64).

127. DiMasi, J. A., Grabowski, H. G. & Hansen, R. W. Innovation in the pharmaceutical industry: new estimates of R&D costs. *Journal of health economics* (2016) (*p.* 71).

128. Chen, H., Engkvist, O., Wang, Y., Olivecrona, M. & Blaschke, T. The rise of deep learning in drug discovery. *Drug Discovery Today* **23,** 1241–1250. ISSN: 1359-6446. http://www.sciencedirect.com/science/article/pii/S1359644617303598 (2018) (June 2018) (*p.* 71).

129. Bjerrum, E. J. SMILES enumeration as data augmentation for neural network modeling of molecules. *arXiv preprint arXiv:1703.07076* (2017) (*p.* 71).

130. Goh, G. B., Siegel, C., Vishnu, A., Hodas, N. O. & Baker, N. Chemception: a deep neural network with minimal chemistry knowledge matches the performance of expert-developed QSAR/QSPR models. *arXiv preprint arXiv:1706.06689* (2017) (*p.* 71).

131. Weininger, D. SMILES, a chemical language and information system. 1. Introduction to methodology and encoding rules. *Journal of chemical information and computer sciences* **28,** 31–36 (1988) (*p.* 71).

132. Pepperkok, R. & Ellenberg, J. High-throughput fluorescence microscopy for systems biology. *Nature reviews Molecular cell biology* **7,** 690–696 (2006) (*p.* 72).

133. Simm, J., Klambauer, G., Arany, A., Steijaert, M., Wegner, J. K., Gustin, E., Chupakhin, V., Chong, Y. T., Vialard, J., Buijnsters, P., Velter, I., Vapirev, A., Singh, S., Carpenter, A. E., Wuyts, R., Hochreiter, S., Moreau, Y. & Ceulemans, H. Repurposing High-Throughput Image Assays Enables Biological Activity Prediction for Drug Discovery. English. *Cell Chemical Biology* **25,** 611–618.e3. ISSN: 2451-9456, 2451-9448. https://www.cell.com/cell-chemical-biology/abstract/S2451-9456(18)30037-0 (2018) (May 2018) (*p.* 72).

134. Knight, S., Plant, H., McWilliams, L., Murray, D., Dixon-Steele, R., Varghese, A., Harper, P., Ramne, A., McArdle, P., Engberg, S. *et al.* Enabling 1536-well high-throughput cell-based screening through the application of novel centrifugal plate washing. *SLAS DISCOVERY: Advancing Life Sciences R&D,* 732–742 (2017) (*p.* 72).

135. Maaten, L. v. d. & Hinton, G. Visualizing data using t-SNE. *Journal of machine learning research* **9,** 2579–2605 (2008) (*p.* 72).

136. Simonyan, K. & Zisserman, A. *Very Deep Convolutional Networks for Large-Scale Image Recognition* 2014. arXiv: 1409.1556 [cs.CV] (*p.* 73).

137. Satopaa, V., Albrecht, J., Irwin, D. & Raghavan, B. *Finding a" kneedle" in a haystack: Detecting knee points in system behavior* in *2011 31st international conference on distributed computing systems workshops* (2011), 166–171 (*p.* 75).

138. Rogers, D. & Hahn, M. Extended-Connectivity Fingerprints. *Journal of Chemical Information and Modeling* **50,** 742–754 (2010) (*p.* 76).

139. Donahue, J., Krähenbühl, P. & Darrell, T. Adversarial feature learning. *arXiv preprint arXiv:1605.09782* (2016) (*pp.* 79, 104).

140. Arjovsky, M., Chintala, S. & Bottou, L. Wasserstein gan. *arXiv preprint arXiv:1701.07875* (2017) (*p.* 79).

141. Masci, J., Meier, U., Cireşan, D. & Schmidhuber, J. *Stacked convolutional auto-encoders for hierarchical feature extraction* in *International Conference on Artificial Neural Networks* (2011), 52–59 (*p.* 79).

142. Fong, R. C. & Vedaldi, A. *Interpretable explanations of black boxes by meaningful perturbation* in *Proceedings of the IEEE International Conference on Computer Vision* (2017), 3429–3437 (*pp.* 82, 86).

143. Doersch, C., Gupta, A. & Efros, A. A. *Unsupervised visual representation learning by context prediction* in *Proceedings of the IEEE International Conference on Computer Vision* (2015), 1422–1430 (*pp.* 83–84).

144. Chu, C., Zhmoginov, A. & Sandler, M. CycleGAN, a master of steganography. *arXiv preprint arXiv:1712.02950* (2017) (*p.* 83).

145. Fridrich, J. Digital image forensics. *IEEE Signal Processing Magazine* **26,** 26–37 (2009) (*p.* 84).

146. Cozzolino, D., Poggi, G. & Verdoliva, L. *Extracting camera-based fingerprints for video forensics* in *IEEE Conference on Computer Vision and Pattern Recognition Workshops* (2019), 130–137 (*p.* 84).

147. Cozzolino, D. & Verdoliva, L. Noiseprint: a CNN-based camera model fingerprint. *IEEE Transactions on Information Forensics and Security* (2019) (*p.* 84).

148. Geradts, Z. J., Bijhold, J., Kieft, M., Kurosawa, K., Kuroki, K. & Saitoh, N. *Methods for identification of images acquired with digital cameras* in *Enabling technologies for law enforcement and security* **4232** (2001), 505–512 (*p.* 84).

149. Kharrazi, M., Sencar, H. T. & Memon, N. *Blind source camera identification* in *2004 International Conference on Image Processing, 2004. ICIP'04.* **1** (2004), 709–712 (*p.* 84).

150. Choi, K. S., Lam, E. Y. & Wong, K. K. *Source camera identification using footprints from lens aberration* in *Digital photography II* **6069** (2006), 172–179 (*pp.* 84, 100).

151. Kurosawa, K., Kuroki, K. & Saitoh, N. *CCD fingerprint method-identification of a video camera from videotaped images* in *Proceedings 1999 International Conference on Image Processing (Cat. 99CH36348)* **3** (1999), 537–540 (*p.* 84).

152. Lukáš, J., Fridrich, J. & Goljan, M. Digital camera identification from sensor pattern noise. *IEEE Transactions on Information Forensics and Security* **1,** 205–214 (2006) (*pp.* 84, 96–97, 101).

153. Johnson, M. K. & Farid, H. *Exposing digital forgeries through chromatic aberration* in *Proceedings of the 8th workshop on Multimedia and security* (2006), 48–55 (*p.* 84).

154. Tuama, A., Comby, F. & Chaumont, M. *Camera model identification with the use of deep convolutional neural networks* in *IEEE International Workshop on Information Forensics and Security* (2016), 1–6 (*pp.* 85, 87, 100).

155. Bondi, L., Güera, D., Baroffio, L., Bestagini, P., Delp, E. J. & Tubaro, S. A preliminary study on convolutional neural networks for camera model identification. *Electronic Imaging,* 67–76 (2017) (*pp.* 85, 87, 100).

156. Yao, Y., Hu, W., Zhang, W., Wu, T. & Shi, Y.-Q. Distinguishing computer-generated graphics from natural images Based on sensor pattern noise and deep learning. *Sensors* **18,** 1296 (2018) (*p.* 85).

157. Olah, C., Mordvintsev, A. & Schubert, L. Feature visualization. *Distill* **2,** e7 (2017) (*p.* 85).

158. Yosinski, J., Clune, J., Nguyen, A., Fuchs, T. & Lipson, H. Understanding neural networks through deep visualization. *arXiv preprint arXiv:1506.06579* (2015) (*p.* 85).

159. Mahendran, A. & Vedaldi, A. *Understanding deep image representations by inverting them* in *Proceedings of the IEEE conference on computer vision and pattern recognition* (2015), 5188–5196 (*p.* 85).

160. Nguyen, A., Dosovitskiy, A., Yosinski, J., Brox, T. & Clune, J. *Synthesizing the preferred inputs for neurons in neural networks via deep generator networks* in *Advances in Neural Information Processing Systems* (2016), 3387–3395 (*p.* 85).

161. Fong, R. & Vedaldi, A. *Net2vec: Quantifying and explaining how concepts are encoded by filters in deep neural networks* in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2018), 8730–8738 (*p.* 85).

162. Zhou, B., Khosla, A., Lapedriza, A., Oliva, A. & Torralba, A. *Learning deep features for discriminative localization* in *Proceedings of the IEEE conference on computer vision and pattern recognition* (2016), 2921–2929 (*p.* 86).

163. Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D. & Batra, D. *Grad-cam: Visual explanations from deep networks via gradient-based localization* in *Proceedings of the IEEE International Conference on Computer Vision* (2017), 618–626 (*p.* 86).

164. Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I. & Fergus, R. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199* (2013) (*p.* 93).

165. Nguyen, A., Yosinski, J. & Clune, J. *Deep neural networks are easily fooled: High confidence predictions for unrecognizable images* in *Proceedings of the IEEE conference on computer vision and pattern recognition* (2015), 427–436 (*p.* 93).

166. Goodfellow, I. J., Shlens, J. & Szegedy, C. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572* (2014) (*p.* 93).

167. Thibos, L., Bradley, A., Still, D., Zhang, X. & Howarth, P. Theory and measurement of ocular chromatic aberration. *Vision research* **30,** 33–49 (1990) (*p.* 93).

168. Ballester, P. & Araujo, R. M. *On the performance of GoogLeNet and AlexNet applied to sketches* in *Thirtieth AAAI Conference on Artificial Intelligence* (2016) (*pp.* 102–103).

169. Brendel, W. & Bethge, M. Approximating cnns with bag-of-local-features models works surprisingly well on imagenet. *arXiv preprint arXiv:1904.00760* (2019) (*pp.* 102–103).

170. Hadsell, R., Chopra, S. & LeCun, Y. *Dimensionality reduction by learning an invariant mapping* in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)* **2** (2006), 1735–1742 (*p.* 104).