Efficient Galaxy Classification Through Pretraining

by

Jesse Schneider

B.Sc., University of London, 2021

Project Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

in the
Department of Statistics and Actuarial Science
Faculty of Science

SIMON FRASER UNIVERSITY
Summer 2023

Declaration of Committee

Name:	**Jesse Schneider**
Degree:	**Master of Science**
Thesis title:	**Efficient Galaxy Classification Through Pretraining**
Committee:	**Chair:** Liangliang Wang Associate Professor, Statistics and Actuarial Science
	David C. Stenning Supervisor Assistant Professor, Statistics and Actuarial Science
	Lloyd T. Elliott Committee Member Assistant Professor, Statistics and Actuarial Science
	Richard Lockhart Examiner Professor, Statistics and Actuarial Science

Abstract

Deep learning has increasingly been applied to supervised learning tasks in astronomy, such as classifying images of galaxies based on their apparent shape (i.e., galaxy morphology classification) to gain insight regarding the evolution of galaxies. In this work, we examine the effect of pretraining on the performance of the classical AlexNet convolutional neural network (CNN) in classifying images of 14,034 galaxies from the Sloan Digital Sky Survey Data Release 4. Pretraining involves designing and training CNNs on large labeled image datasets unrelated to astronomy, which takes advantage of the vast amounts of such data available compared to the relatively small amount of labeled galaxy images. We show a statistically significant benefit of using pretraining, both in terms of improved overall classification success and reduced computational cost to achieve such performance.

Keywords: convolutional neural networks, machine learning, galaxy morphology, astrostatistics

Acknowledgements

1. Thank you to Professors David Stenning and Lloyd Elliott for their facilitation and guidance in the development of this project.

2. Thank you to the Digital Research Alliance of Canada and the BC DRI Group for their provision of the Cedar system at Simon Fraser University, which was used in the preparation of this project.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Convolutional neural networks (CNNs) are a type of deep learning that is particularly well suited to computer vision tasks (Aggarwal, 2018). Originally inspired by research on the visual cortex by neurophysiologists Hubel and Wiesel in the mid-20^{th} century (Aggarwal, 2018; Hubel & Wiesel, 1959), CNNs have been successfully applied to a variety of computer vision tasks, such as image classification (Krizhevsky, Sutskever, & Hinton, 2012), facial recognition (Taigman, Yang, Ranzato, & Wolf, 2014), and classification of various forms of interstitial lung disease (Li et al., 2014). CNNs have also been applied to tasks outside of computer vision, such as forecasting prices in financial stock markets (Tsantekidis et al., 2017).

As astronomy and astrophysics increasingly rely on large sets of image data, CNNs have increasingly been used to tackle a variety of interesting astronomical and astrophysical problems. These include identifying gravitational lenses (Davies, Serjeant, & Bromley, 2019), identifying contamination in astronomical images, e.g., by cosmic rays and diffraction spikes (Paillassa, M., Bertin, E., & Bouy, H., 2020), and for supernovae detection (Cabrera-Vives, Reyes, Förster, Estévez, & Maureira, 2016). One particular task for which CNNs have proved successful, which will be discussed in more depth below, is galaxy morphology classification as in (Cavanagh, Bekki, & Groves, 2021). We will therefore use galaxy morphology classification to explore how *pretraining* (a type of transfer learning (Ribani & Marengoni, 2019; Tan et al., 2018)) can potentially benefit many tasks in astronomy and astrophysics that use CNNs.

When applied to a particular task, CNNs (and neural networks in general) can be trained from scratch for the given task, or instead we can use a pretrained network. A pretrained CNN is one which has already been trained on a separate data set prior to its application to the given task (Aggarwal, 2018). The training of neural networks is, in general, an energy-intensive activity (Strubell, Ganesh, & McCallum, 2020; Borowiec, Harper, & Garraghan, 2022), and research is ongoing to quantify and reduce energy use (Yang, Chen, Emer, & Sze, 2017; García-Martín, Rodrigues, Riley, & Grahn, 2019). Training neural networks from scratch can require more training and therefore greater expense and resource usage

compared to using a pretrained network. Furthermore, this requirement for a large amount of training presupposes the availability of a sufficient amount of data within the intended domain for the desired amount of training. Thus, data availability itself can be a limitation which may preclude the possibility of a large amount of training being performed from scratch.

In astronomy, obtaining large, labelled training data sets is expensive, impractical, or both. As a result of the demands required to train deep learning models from scratch, pretraining may be an attractive alternative for astronomy. In recent years, transfer learning has been adopted for classification tasks in astronomy and astrophysics involving galaxy morphologies (Domínguez Sánchez et al., 2018), variable stars (Kim, Dae-Won, Yeo, Doyeob, Bailer-Jones, Coryn A. L., & Lee, Giyoung, 2021), and star clusters (Wei et al., 2020). However, the learning that is "transferred" in these cases is between different astronomical surveys. That is, a classifier is trained using data from one survey, and deployed, perhaps with modification, on test data arising from a different survey. For example, a model may be trained on Sloan Digital Sky Survey images and deployed on Dark Energy Survey images (Domínguez Sánchez et al., 2018).

An alternative type of transfer learning, which we refer to specifically as pretraining hereafter, involves the practice of training a neural network on another *unrelated* data set before applying the neural network to the particular data set of interest. This means that pretraining, using our definition, can exploit the vast effort undertaken to design CNNs for classifying large volumes of natural (everyday) images. Specifically, we will use a CNN trained on millions of non-astronomical images that comprise the ImageNet database (Deng et al., 2009); this CNN is known as AlexNet (Krizhevsky et al., 2012). We will demonstrate, through a series of numerical experiments, that for the task of galaxy morphology classification, a pretrained AlexNet outperforms an architecturally identical CNN that is trained from scratch using only galaxy morphology images. Transfer learning using training on a large data set of natural images has been explored in the context of analysis of data from the Laser Interferometer Gravitational-Wave Observatory (George, Shen, & Huerta, 2018) and in galaxy merger detection (Ackermann, Schawinski, Zhang, Weigel, & Turp, 2018). However, as far as we are aware, such transfer learning has not been explored in the context of galaxy morphology classification, which is the focus of this project.

The rest of this manuscript explores the utility of pretraining in the application of a CNN to galaxy morphology image classification. This domain represents a potential use case for a pretrained network due to the expense and difficulty of gathering and labeling images of galaxies (Cavanagh et al., 2021). We begin in Chapter 2 with an overview of galaxy morphology classification and the data we will use for our experiments. In Chapter 3 we describe CNNs, including the particular CNN used in this project, as well as data preparation procedures and tooling. Our numerical experiments and results are detailed

in Chapter 4. We discuss and summarize our contributions in Chapter 5, and also discuss directions for future research.

All code and materials necessary to reproduce our work can be found at: https://github.com/jsa378/01_masters.

Chapter 2

Galaxy Morphology Classification and Data

Images of galaxies are captured using either earthbound equipment or spacecraft. Traditionally, the images are classified by groups of experts who examine each image and come to a consensus regarding its classification (Cavanagh et al., 2021). In order to speed up the classification of images, other strategies have been used, such as the recruitment of enthusiastic amateurs (Lintott et al., 2010), and various automated classification techniques (Cheng et al., 2020).

In the near future, an expected deluge of data obtained by new spacecraft such as the European Space Agency's *Euclid* will overwhelm available resources for classification by humans (Silva, Cao, & Hayes, 2018). This adds urgency to the search for accurate, rapid, and automated classification techniques. However, the galaxy image data currently available for training deep learning models remains relatively small. For this work, we used 14,034 (labelled) galaxy images from the Sloan Digital Sky Survey Data Release 4 (Stoughton et al., 2002; York et al., 2000; Adelman-McCarthy et al., 2006); the data are described in detail in (Nair & Abraham, 2010). Each galaxy is labelled according to its morphology, or shape, as belonging to the class of:

1) *elliptical galaxies*, having a smooth, diffuse, and elliptical shape;

2) *spiral galaxies*, disk-like in appearance and with spiral arms;

3) *lenticular galaxies*, an intermediate class of galaxies between the elliptical and spiral categories; or

4) *irregular+miscellaneous (Irr+Misc) galaxies*, not meeting the membership criteria for any of the above three categories.

An example of each type of galaxy, taken from the set of 14,034, is presented in Figure 2.1. The 14,034 galaxy morphology images varied in size, but were often quite small—around 100×100 pixels, or 0.01 megapixel. The class breakdown of the images is as follows:

1) 2,738 images of elliptical galaxies (19.4% of the total),

2) 7,708 images of spiral galaxies (54.9% of the total),

3) 3,215 images of lenticular galaxies (22.9% of the total), and

4) 373 images of Irr+Misc galaxies (2.7% of the total).

While this is clearly a highly imbalanced data set as governed by the distribution of galaxies in the regions imaged, we did not attempt to correct for these imbalances because for this work we are only concerned with demonstrating the benefit of pretraining. Further discussion of class differences is in Subsection 4.1.[1]

[1]We note that (Cavanagh et al., 2021) applied CNNs to the same data set used in the present project. Methods and tools used in the present work are similar although not identical to those used in (Cavanagh et al., 2021), so while the peak accuracy results obtained in the present project are similar to those obtained in (Cavanagh et al., 2021), the results are not directly comparable and we therefore do not make such a comparison.

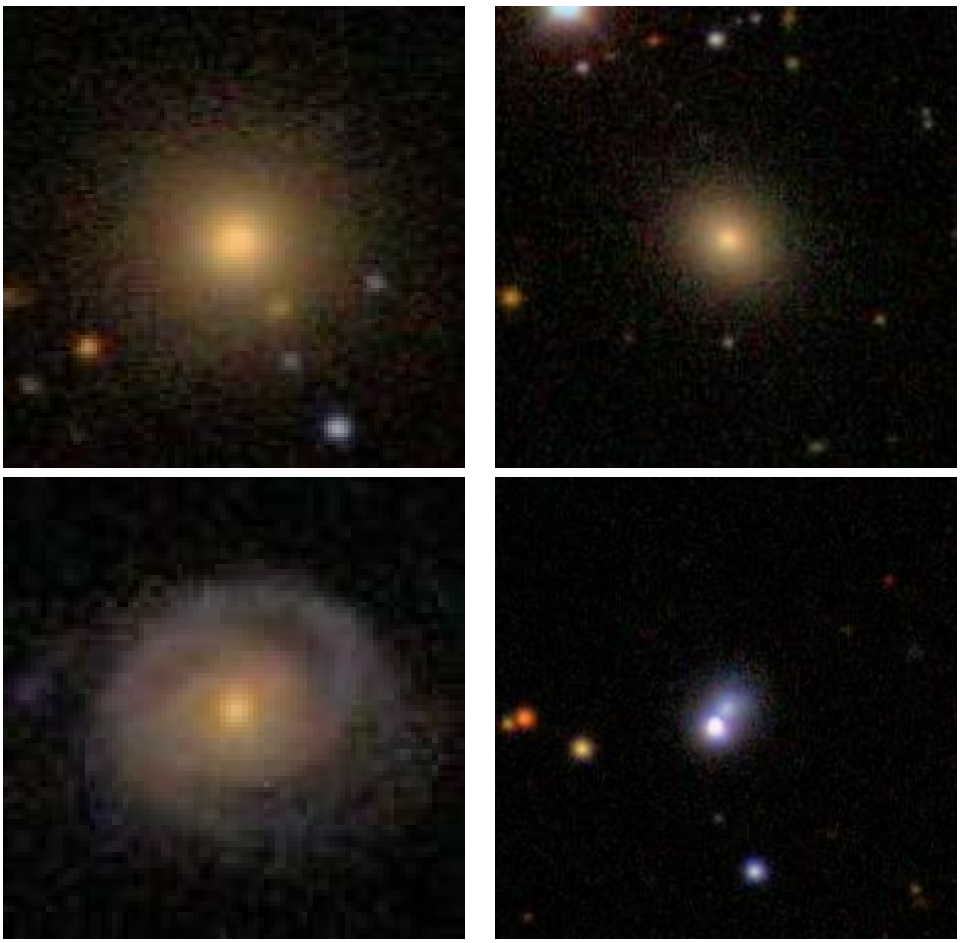

Figure 2.1: Examples of the four categories of galaxy from the Sloan Digital Sky Survey Data Release 4 (Nair & Abraham, 2010). Clockwise from top left: 1) Elliptical galaxy. 2) Lenticular galaxy. 3) Irregular+Miscellaneous galaxy. 4) Spiral galaxy.

Chapter 3

Methods and Data Preparation

The chapter provides a concise introduction to neural networks. We introduce the feedforward neural network in Section 3.1.1, followed by an overview of CNNs in Section 3.1.2. In Section 3.1.3 we introduce AlexNet (Krizhevsky et al., 2012), a particular CNN that we exploit for the current work. We conclude this chapter by detailing data preparation and computational tooling in Section 3.2.

Figures 3.1–3.4 were created in LaTeX.

3.1 Neural Networks: Exposition[1]

In this section (excluding the section title), "neural network" refers to a feedforward neural network, while CNN refers to a convolutional neural network.

3.1.1 Feedforward Neural Networks

Neural networks share fundamental similarities with CNNs. Indeed, the former are often a component of the latter. This section will discuss the basic components and operation of a neural network, many parts of which are directly applicable to CNNs.

Architecture

Example 1: Single-Layer Neural Network Figure 3.1 presents a simple, single-layer neural network for pedagogical purposes. The circles are referred to as *nodes*; the symbols x_1, x_2, x_3, x_4 are real valued and represent the components of an input vector $X = (x_1, x_2, x_3, x_4)$. The symbols w_1, w_2, w_3, w_4 are referred to as *weights* and are real valued. The symbol y_1 is real valued and represents the output of the network.[2] Thus, the neural

[1]The presentation in this section is a synthesis of material read in (Aggarwal, 2018) and the present author's thoughts on the material.

[2]In practice the number of input and output nodes can be tailored to the problem at hand.

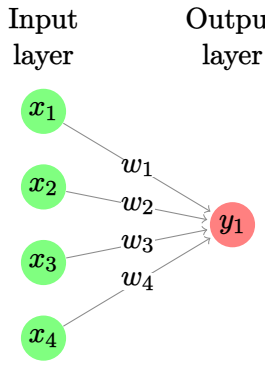

Figure 3.1: A single-layer neural network. In practice neural networks are rarely if ever this simple, but this architecture facilitates understanding of the basic operation of a neural network.

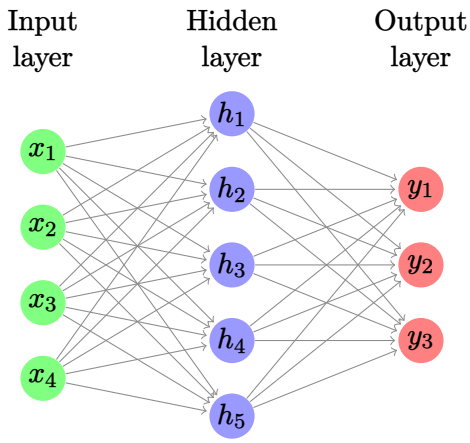

Figure 3.2: A fully connected feedforward neural network with one hidden layer.

network shown in Figure 3.1 is a function $g\colon \mathbb{R}^4 \times \mathbb{R}^4 \to \mathbb{R}$. Its definition is

$$(x_1, x_2, x_3, x_4), (w_1, w_2, w_3, w_4) \mapsto \Phi\left(\sum_{i=1}^{4} w_i x_i\right),$$

where $\Phi\colon \mathbb{R} \to \mathbb{R}$ is referred to as an *activation function*. Various choices for Φ exist, but a common one is the *Rectified Linear Unit* or ReLU:

$$z \mapsto \begin{cases} 0 \text{ if } z \leq 0 \\ z \text{ if } z > 0. \end{cases}$$

Example 2: Fully connected neural network with one hidden layer Figure 3.2 shows a neural network whose main complication compared to that shown in Figure 3.1 is

the presence of a *hidden layer*. Hidden layers add extra function compositions to the neural network and increase the network's flexibility. The weights are not shown in the figure to avoid clutter.

In order to describe the operation of the neural network shown in Figure 3.2 in a clean manner, matrix notation will be employed. Let X be a four-dimensional input vector as before. Let W be a 5×4 matrix whose i, j entry is the weight $w_{i,j}$ from hidden node i to input node j, and let W' be a 3×5 matrix whose k, l entry is the weight $w'_{k,l}$ from output node k to hidden node l.[3] Assume that the hidden nodes share the common activation function Φ_h and that the output nodes share the common activation function Φ_y.[4] Then, we can consider the neural network shown in Figure 3.2 to be a function $g \colon \mathbb{R}^4 \times \mathbb{R}^{5\times 4} \times \mathbb{R}^{3\times 5} \to \mathbb{R}^3$ with definition

$$(X, W, W') \mapsto \underbrace{\Phi_y\big(W' \underbrace{\Phi_h(WX)}_{5\times 1 \text{ vector}}\big)}_{3\times 1 \text{ vector}}.$$

Neural networks can be of essentially arbitrary size. It is not uncommon to encounter neural networks with thousands of nodes in a given layer, and dozens or possibly hundreds of hidden layers. The main practical constraints on a neural network's size are the problem at hand, the amount of data available, and certain computational and numerical considerations that are beyond the scope of this project.

Main Components of Neural Networks

Thus far we have described on a somewhat superficial level how a neural networks works; in practice more pieces are needed. Briefly, assuming a chosen network architecture, the pieces needed are as follows:

1. Input data, along with the associated output values.
2. An activation function. Typically the same activation function is used on all hidden nodes.[5]
3. A choice of loss function. The loss function is necessary in order to evaluate the network's performance. Common choices include squared loss and cross-entropy loss.

[3] This perhaps seemingly backwards matrix layout is used to facilitate matrix multiplication. Of course, the matrices could have been defined in a more normal way, but this latter choice would have necessitated the use of matrix transposes. The difference is merely one of notation.

[4] In matrix notation, the activation functions Φ operate component-wise on their arguments.

[5] Output nodes may require separate functions if their output values are to be interpreted as probabilities.

4. A learning mechanism. This is needed to adjust the network's weights in response to the information provided by the loss.

In order to gain more understanding of these pieces, we discuss how they are used in order to train a neural network.

Neural Network Training

Neural network training is essentially the process of adjusting the network weights in order to improve the network's performance. Training consists of two phases—a forward phase and a backward phase—which are repeated until some stopping criterion is met.

Forward Phase Mathematically, the forward phase occurs as described before, modulo computational details of implementation and optimization that are beyond the scope of this project. In this phase, data are fed to the network, which uses these data to predict the associated output values. The predicted output values and actual output values are then given to the loss function as arguments to measure the network's performance over the data it was fed.

A decision needs to be made regarding how much data to feed to the network in the forward phase. This is a hyperparameter referred to as the *batch size*. On the one extreme, the entire data set may be fed to the network. This has the advantage of giving the most complete picture possible of the network's performance, and it generally leads to faster training. The biggest disadvantage of this "full batch" approach is an increased propensity for overfitting, and hence diminished generalization ability.

Conversely, on the other extreme a single data observation may be fed to the network, so that the loss is computed only over that one observation. This generally requires less memory than the full batch approach, but its biggest benefit is an injection of randomness into the training process, which helps reduce overfitting.

Typically, because of the tradeoffs between these two approaches, intermediate batch sizes in the range of 2^5 to 2^8 observations are used (Aggarwal, 2018), assuming that a sufficient number of observations are available. (The powers of two are used for architectural reasons.)

Backward Phase The goal of the backward phase is to use the information gained in the forward phase to adjust the network's weights so that the network can "learn" in order to improve its performance. To this end, we will view the loss function L as a function of the weight values, even though more superficially it is a function of the predicted and actual output values.

Consider the neural network shown in Figure 3.3. We will use the node labels to represent the output of a given node, so that for example $y_1 = \Phi_y\left(\sum_{j=1}^{3} h_j w_{2,j,1}\right)$ and $h_1 = \Phi_h(g_1 w_{1,1,1})$.

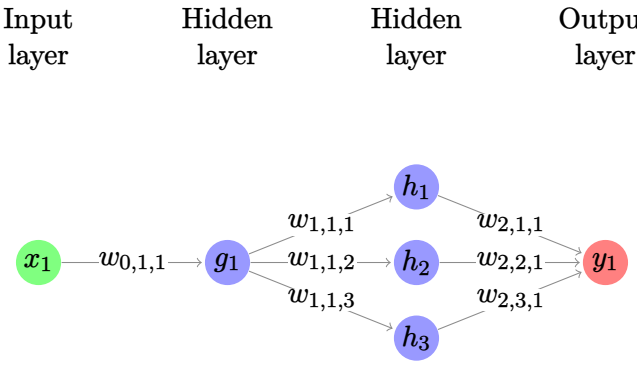

Figure 3.3: Examining this neural network illustrates how the derivative with respect to a given weight in the network is the sum of derivatives over all paths to that weight. Note that weight $w_{i,j,k}$ is in layer i and connects node j on the left with node k on the right.

Assume that the neural network has been fed some amount of data, that it has made prediction(s) $\hat{y}$, and let y be the corresponding observed value(s). The loss value (squared loss, for example), is computed as a function of the predicted and observed values, so we can write the loss value as $L(\hat{y}, y)$. However, the predictions $\hat{y}$ are a function of the weights $(w_{0,1,1}, \ldots, w_{2,3,1}) \in \mathbb{R}^7$, so we can rewrite the loss as $L(\hat{y}(w_{0,1,1}, \ldots, w_{2,3,1}), y)$.

Holding the data constant we can think of the loss as a function $L\colon \mathbb{R}^7 \to \mathbb{R}$, which enables us to envision a loss surface in seven-dimensional space, and attempt to find the minimum of this surface. Since this surface is in general not analytically tractable, an iterative approximation called *stochastic gradient descent* is used to try to minimize the loss (Aggarwal, 2018).[6] To this end, a method of calculating the partial derivatives of L with respect to all the weights is required.

Suppose that we would like to calculate $\frac{\partial L}{\partial w_{0,1,1}}$ in Figure 3.3. By the chain rule $\frac{\partial L}{\partial w_{0,1,1}} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_{0,1,1}}$, so it suffices to focus on differentiating the output of the network with respect to the weight. Skipping some mathematical details for brevity, one version of the chain rule states

$$\frac{df(C(t))}{dt} = \text{grad} f(C(t)) \cdot C'(t).$$

We are interested in $\frac{dy_1}{dw_{0,1,1}}$. Looking at the chain rule, we can let

$$C(w_{0,1,1}) = (h_1(w_{0,1,1}), h_2(w_{0,1,1}), h_3(w_{0,1,1})),$$

[6]This works because the gradient of L can be used to determine the direction of most rapid decrease at any point on the surface of L.

so

$$\frac{dy_1(C(w_{0,1,1}))}{dw_{0,1,1}} = \text{grad } y_1(C(w_{0,1,1})) \cdot C'(w_{0,1,1}) = \frac{dy_1}{dh_1}\frac{dh_1}{dw_{0,1,1}} + \frac{dy_1}{dh_2}\frac{dh_2}{dw_{0,1,1}} + \frac{dy_2}{dh_3}\frac{dh_3}{dw_{0,1,1}}.$$

Now, notice that, for example, $h_1(w_{0,1,1})$ is really $h_1(g_1(w_{0,1,1}))$. This means that we need to apply the single-variable chain rule. This gives us

$$\frac{dy_1(C(w_{0,1,1}))}{dw_{0,1,1}} = \underbrace{\frac{dy_1}{dh_1}\frac{dh_1}{dg_1}\frac{dg_1}{dw_{0,1,1}}}_{\text{first path}} + \underbrace{\frac{dy_1}{dh_2}\frac{dh_2}{dg_1}\frac{dg_1}{dw_{0,1,1}}}_{\text{second path}} + \underbrace{\frac{dy_2}{dh_3}\frac{dh_3}{dg_1}\frac{dg_1}{dw_{0,1,1}}}_{\text{third path}}.$$

Intuitively, to differentiate the output of a neural network with respect to a given weight, one finds all paths to a given weight, multiplies the partial derivatives backwards along a given path, and sums these products over all possible paths.

The *backpropagation algorithm* is used to compute the gradient in an efficient manner (Aggarwal, 2018). Essentially, one notices that the technique described above for computing derivatives farther back (i.e. close to the input layer) in a network involves much redundant differentiation, so by storing these intermediate derivatives, the gradient can be computed much more efficiently. Generically, if $\boldsymbol{w}_i$ refers to the vector of weights in a neural network at stage i of the learning process, and $\eta \in \mathbb{R}^+$, then the *backpropagation update* to the weight vector is as follows:

$$\boldsymbol{w}_{i+1} = \boldsymbol{w}_i - \eta \text{ grad} L(\boldsymbol{w}_i),$$

where the hyperparameter η is referred to as the *step size*.

In this way, one can imagine stepping down the loss surface with every backpropagation update. Unfortunately, this greedy algorithm is not guaranteed to find the global minimum of the loss surface.

Forward-Backward Iteration The entire training process consists of repeated alternation between the forward and backward phases described above, terminating when some convergence criterion is met (or perhaps when the operator's patience or resources have been exhausted). Each time the network has trained over the entire data set is referred to as an *epoch*. Depending on the size of the data set and the complexity of the network, training can proceed over tens or possibly hundreds of epochs.

Neural network training can be computationally demanding, since networks can have thousands, millions or even billions of parameters and observations. Training is typically done on graphics processing units (GPUs) due to their favorable architecture, which speeds up computations drastically compared to CPUs.

3.1.2 Convolutional Neural Networks

Introduction to CNNs

CNNs are a type of neural network that are well suited to image data (Goodfellow, Bengio, & Courville, 2016). They are so named because of the "convolution" operations applied within the network, although strictly speaking these operations are cross-correlations (Goodfellow et al., 2016). CNNs are perhaps the archetypal example of biologically-inspired artificial intelligence, because their conception was influenced by exploration of the visual cortex in the mid-20^{th} century (Aggarwal, 2018).

Although formal mathematical justification for CNNs is lacking, the common explanation is that CNNs function by detecting relatively crude features of an input image, such as lines, in the early layers of the network, and superimpose these features into progressively more complex features in later layers (Aggarwal, 2018).

CNN Operations

Like feedforward neural networks, CNNs consist of an input layer, one or more hidden layers and an output layer. The primary differences are the types of operations that the layers perform. The fundamental principles of neural networks—the forwards and backwards phases, and gradient-based optimization—also apply to CNNs. Furthermore, training methods consisting of feeding the entire training data set to the network multiple times, again referred to as an *epoch*, are similar for both types of networks. For brevity, therefore, we will focus on the unique mathematical operations employed in CNNs; these unique operations are convolution and pooling operations. We will also briefly discuss the regularization technique known as dropout, which is commonly used to avoid overfitting deep neural networks.

The Convolution (Cross-Correlation)[7] Consider a color image $\mathbf{I}$ of size $h \times w$ pixels, represented numerically as an array with dimensions $h \times w \times 3$. (The depth of 3 is for storage of the red, green and blue color values.) The convolution operation involves placing a smaller $h' \times w' \times 3$ ($h' \leq h$, $w' \leq w$) array $\mathbf{K}$, called the *kernel* or *filter*, at all possible positions overlaid on $\mathbf{I}$ and computing the component-wise dot product between $\mathbf{I}$ and $\mathbf{K}$.

More formally, the convolution of $h \times w \times 3$ image $\mathbf{I}$ (having i, j, l entry $I_{i,j,l}$) with $h' \times w' \times 3$ kernel $\mathbf{K}$ (having i, j, l entry $K_{i,j,l}$) is a function

$$*\colon \mathbb{R}^{h \times w \times 3} \times \mathbb{R}^{h' \times w' \times 3} \to \mathbb{R}^{(h-h'+1) \times (w-w'+1) \times 1}$$

[7]The convolution operation, for which CNNs are named, is actually a cross-correlation since neither of the functions in the operation's arguments are reflected. Despite this, we will adhere to the convention of referring to this operation as a convolution.

defined by

$$(\mathbf{I} * \mathbf{K})_{r,s} \coloneqq \sum_{i=1}^{h'} \sum_{j=1}^{w'} \sum_{k=1}^{3} I_{r+(i-1),s+(j-1),l} \cdot K_{i,j,l}.$$

To be clear, the symbol $\cdot$ above represents scalar multiplication.

Some notes regarding the convolution operation are as follows:

1. If the first two layers of the CNN are an input layer followed by a convolutional layer, then the output $\mathbf{I} * \mathbf{K}$ of the convolutional layer is fed into the third layer of the network, whatever that may be.[8]

2. The kernel $\mathbf{K}$ can only be placed in $(h - h' + 1) \times (w - w' + 1)$ positions over the image $\mathbf{I}$, so the convolution $\mathbf{I} * \mathbf{K}$ only has height $(h - h' + 1)$ and width $(w - w' + 1)$. Thus, repeated convolutions lead to significant "compression" of an array as it is fed forward through the network.

3. The application of a single kernel results in an output array having a depth of 1. In practice, p kernels $\mathbf{K}_1, \mathbf{K}_2, \ldots, \mathbf{K}_p$ may be applied within a given layer, meaning that the output array of all p convolutions has dimensions $(h - h' + 1) \times (w - w' + 1) \times p$. In practice a CNN may use tens or hundreds of kernels within a given convolutional layer.

4. Some hyperparameters associated with the convolution operation are as follows:

   (a) The dimensions of kernels used.[9]

   (b) The number of kernels to use.

   (c) The *stride*, which skips certain placements of the kernel over the image.

   (d) The *padding*, which is an augmentation of the image in response to the fact that information along the edges of the image is involved in fewer dot products with a kernel.[10]

The Max Pool The max pool operation involves a smaller array $\mathbf{P}$ similar to the kernel $\mathbf{K}$, except that $\mathbf{P}$ has a depth of 1. If $\mathbf{P}$ has dimensions $p \times q \times 1$ and acts on a layer $\mathbf{L}$ having

[8] An activation function, such as the ReLU, is often applied component-wise to the output of a convolution operation, but as in a feedforward network, the ReLU is rarely presented separately, as its own layer.

[9] Within a given convolutional layer, it is simpler to use only kernels with equal height and width dimensions, and furthermore to require all kernels to have the same dimensions, although more complicated arrangements are possible.

[10] For example, in Figure 3.4, a corner pixel in $\mathbf{I}$ will be involved in only one dot product with $\mathbf{K}$, while the center pixel will be involved in nine dot products with $\mathbf{K}$.

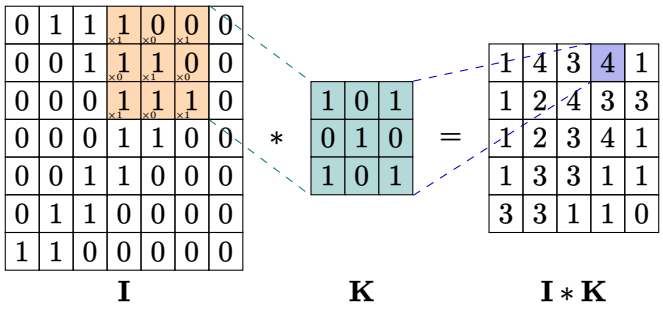

Figure 3.4: An example of part of a two-dimensional discrete convolution (or more properly, cross-correlation). The convolution $\mathbf{I} * \mathbf{K}$ is computed via the component-wise dot product of $\mathbf{K}$ with the portion of $\mathbf{I}$ that it overlaps. We can think of $\mathbf{I}$ as the image and $\mathbf{K}$ as the "filter" or "kernel", so that $\mathbf{I} * \mathbf{K}$ is the output of the first layer of the network, and therefore the input into the second layer. Notice that the output $\mathbf{I} * \mathbf{K}$ is smaller than the input $\mathbf{I}$, and a subsequent convolution $(\mathbf{I} * \mathbf{K}) * \mathbf{K}'$ would be smaller still. This is a consistent feature of convolutional neural networks. Note also that color images may be represented as 3-dimensional arrays as opposed to matrices (2-dimensional arrays) if the RGB color model is used, for example. In such a case $\mathbf{K}$ will also be a 3-dimensional array. Furthermore, in CNNs, multiple filters $\mathbf{K}$ are often used within a given layer.

dimensions $h \times w \times d$, then the pooling operation produces a layer $\mathbf{P}(\mathbf{L})$ having dimensions $(h - p + 1) \times (w - q + 1) \times d$. In particular,

$$\mathbf{P}(\mathbf{L})_{r,s,t} \coloneqq \max\{l_{i,j,t} \in \mathbf{L} \colon r \leq i \leq (r + p - 1), s \leq j \leq (s + q - 1)\}.$$

Some notes regarding the max pool operation are as follows:

1. Because it operates at all depths $1 \leq t \leq d$, the max pool is a depth-preserving operation. In other words, the output array $\mathbf{P}(\mathbf{L})$ has the same depth as the input array.

2. However, the max-pool is not a height- or width-preserving operation, for much the same reasons as the convolution.

3. The max pool has its own stride hyperparameter, much like the convolution.

Dropout Dropout is a regularization technique applied layer-wise during CNN training, which involves training random subsets of the overall network (Hinton, Srivastava, Krizhevsky, Sutskever, & Salakhutdinov, 2012; Srivastava, Hinton, Krizhevsky, Sutskever, & Salakhutdinov, 2014). If dropout is applied to layer l in the network, then each time the network is fed a training image, independent draws from a Bernoulli(p) distribution are made to determine which nodes in layer l are kept or discarded. The training prediction and backpropagation are then carried out only over the sub-network containing the remaining nodes. During training, repeated samples from the Bernoulli(p) distribution are drawn,

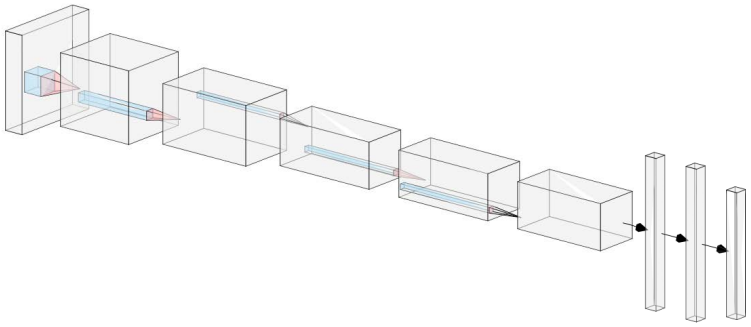

Figure 3.5: The AlexNet architecture. After receiving a $224 \times 224 \times 3$ input image, AlexNet applies a sequence of convolutions, ReLU operations, and pooling operations (in the portions represented by the deep rectangular prisms) before applying a set of linear layers with dropout (in the portions represented by the tall rectangular prisms). Figure created using NN-SVG.

which means that different subsets of the original network are trained. During testing, dropout is not applied, so the entire network is used. Using dropout significantly reduces network overfitting, and thus improves the network's ability to generalize to the test data (Srivastava et al., 2014).

3.1.3 AlexNet

The CNN that underpins our work is called *AlexNet* (Krizhevsky et al., 2012). Although CNNs date back to the 1980s, in 2012 the AlexNet CNN ushered in what is arguably the modern era in computer vision by winning the ImageNet Large Scale Visual Recognition Challenge, thoroughly surpassing past performers and challengers (Aggarwal, 2018). Since then, CNNs have served as the standard for image classification, as can be seen by the fact that subsequent winners of the ImageNet competition have also been CNNs (Aggarwal, 2018). Below, a brief summary of the AlexNet architecture is provided. Further details can be found in (Krizhevsky et al., 2012).

The AlexNet network can be broken into two broad parts: an early convolutional part and a later, more conventional feed-forward part. In the convolutional part, AlexNet takes as input a $224 \times 224 \times 3$ image and, over a number of layers, applies an increasing number of convolution filters which decrease in height and width. The first convolutional layer applies 64 filters of size $11 \times 11 \times 3$; a later convolutional layer applies 384 filters of size $3 \times 3 \times 256$. (As previously mentioned, formal mathematical justification for CNN operation is lacking, but the intuition is that larger numbers of smaller filters in later layers capture

more complex features of the input image.) AlexNet uses 5 convolutional layers in total. After every convolutional layer the ReLU operation is used, and a handful of max-pooling operations are layered in as well. In the feed-forward part, AlexNet contains 3 linear layers with ReLU activation functions before delivering its class probability calculations.[11] To reduce overfitting, dropout is used.

Because the ImageNet Challenge requires classifying images belonging to one of 1000 categories, AlexNet by default has 1000 nodes in its final layer. However, this can be modified when applying AlexNet for other purposes, such as classifying galaxy morphology into four categories.

The architecture of AlexNet is shown in a standard visual representation in Figure 3.5. Note that the original AlexNet architecture (shown in Figure 3.5) has 1,000 nodes in its output layer, corresponding to the 1,000 object categories in the ImageNet dataset. For this work, the number of output nodes was reduced to 4 in accordance with the number of categories for the problem at hand.

3.2 Data Preparation, Procedures and Tooling

For our work, all galaxy images were scaled to 256×256 in height and width to allow room for cropping to 224×224, which is the input size required by AlexNet. Regarding data augmentation, PyTorch is naturally set up to use random data augmentation, which involves the probabilistic application of standard data augmentation techniques. For example, the PyTorch function `transforms.RandomHorizontalFlip()` will, each epoch, horizontally flip a given image with probability 0.5.

The data set was split into 12,000 training images and 2,034 testing images uniformly at random. We constructed 100 such test/train splits. No hyperparameter tuning was done. (The decision to forgo hyperparameter tuning was made in order to make an even comparison between the pretrained and non-pretrained networks.) For each random split of the data set, which we refer to as a *run*, the pretrained AlexNet was trained for 200 epochs, while the non-pretrained AlexNet was trained for 400 epochs. This additional training time was given to the non-pretrained AlexNet so that it had a better opportunity to achieve optimal performance. The training used the standard cross-entropy loss via the PyTorch function `torch.nn.CrossEntropyLoss()`, which is described in the PyTorch documentation.

Standard Python-language tools including Matplotlib (Hunter, 2007), NumPy (Harris et al., 2020), pandas (pandas development team, 2020; Wes McKinney, 2010), and seaborn (Waskom, 2021) were used for performing our numerical experiments and compiling the results to be presented. In particular, PyTorch (Paszke et al., 2017) was used to obtain the

[11] The paper (Krizhevsky et al., 2012), which introduced the AlexNet architecture, makes clear that the size of AlexNet was limited by computational power and memory, and patience to endure long training times.

pretrained and non-pretrained AlexNet networks, and to train and test them. The networks used are available off the shelf in PyTorch, via the `torchvision.models` subpackage. Further, the Cedar computing system at Simon Fraser University, provisioned by the Digital Research Alliance of Canada and the BC DRI Group, was used to carry out the neural network training and testing.

Chapter 4

Results

In this chapter, we present and discuss the results obtained over the 100 runs, and compare the pretrained and non-pretrained networks on a variety of metrics. We begin with Figure 4.1, in which we present histograms of the peak test accuracy for all runs for both the pretrained and non-pretrained networks; this figure also provides the epoch in which peak test accuracy is achieved. We also compute the difference in peak test accuracy between the pretrained and non-pretrained networks, as well as the difference in the epoch number for which peak test accuracy was achieved for the two networks, and display the results in Figure 4.2. (For the latter calculation, only the first 200 epochs of training were considered for the non-pretrained network.)

To compare the pretrained and non-pretrained networks given a fixed training budget of 200 epochs, for each run we compute the difference between the peak accuracy of the pretrained network and the highest accuracy achieved by the non-pretrained network within its first 200 epochs of training. These results are presented in Figure 4.3. Examination of the histograms displayed in Figures 4.1–4.3 indicates that the pretrained AlexNet is preferred over the non-pretrained version. Pretraining leads to a higher overall accurracy, with an average peak accuracy (over the 100 runs) of 84.2% versus 82.4% for pretrained and non-pretrained, respectively. Furthermore, the pretrained network required significantly fewer epochs to reach its peak accuracy.

To further explore the efficiency gain of pretraining, we evaluate the performance of the pretrained network over a restricted portion of training, such as the first 20 or 50 epochs. Selected results are presented in Figure 4.4. The top left panel of Figure 4.4 is a histogram of the difference in peak test accuracy for the 200-epoch pretrained network and the 50-epoch pretrained network, showing that the average gain in accuracy from the additional 150 epochs of training is only about 1%, and the maximum gain over all 100 runs is less than 3%. If the pretrained network is limited to only 20 epochs, then the top right panel of Figure 4.4 shows that the average gain in accuracy from the additional 180 epochs of training increases to approximately 2% to 2.5%, with a maximum gain over the 100 runs of approximately 4%. This suggests that good performance can be achieved with relatively

few epochs, such as 20. Depending on the specific application and resource constraints, it may therefore be sufficient to train a pretrained network for a relatively small number of epochs.

The bottom row of Figure 4.4 displays the test classification accuracy curves for pretrained (left panel) and non-pretrained (right panel) models over all 100 runs. The classification accuracy curves for the pretrained network reinforce the finding that the vast majority of improvement is acquired within the first 10–20 epochs of training. The widths of the bands of lines (an informal measure of variability), indicate that there is much less variability when using a pretrained model than when using a non-pretrained model. The bottom right panel of the figure shows that the non-pretrained models require roughly 50 epochs of training in order to make any improvement at all; presumably this is the typical amount of training necessary to adjust a model's parameters sufficiently in order to begin changing its classification behavior.

Table 4.1 provides a summary of the results presented in Figures 4.1–4.4. The information presented in the Table is consistent with that conveyed in the Figures, namely that the pretrained network is more accurate and less variable in its performance than the non-pretrained network. Table 4.1 also provides more precise insight into the diminishing returns from training the pretrained network. Within the first 10% of training (20 epochs as opposed to 200), the pretrained network achieves an average peak test accuracy of 82.0%; within the first 25% of training (50 epochs), the pretrained network achieves a peak test accuracy of 83.1%. These means are within approximately 2% and 1%, respectively, of the peak test accuracy of 84.2%, and it is worth noting that despite the reduced training time, the standard deviations are essentially indistinguishable from those for the full 200 epochs' worth of training. This implies that there is no (or very little) variance penalty when doing a comparatively small amount of training of the pretrained network.

Figure 4.5 is presented as typical output for one of the 100 runs for both networks. (Run 49 was selected arbitrarily.) The upper panels present the train and test accuracy progression over all epochs of training, while the lower panels present train and test performance for both networks over all epochs from the perspective of the loss function. In general, the values of the loss function did not appear to provide any information not already apparent from the accuracy information, but the loss information is nevertheless presented in Figure 4.5 for additional illustration.

4.1 Classification Accuracy by Class and Analysis of Models

In order to develop a deeper understanding of model performance, the top-performing pretrained and non-pretrained models (by test accuracy) from each run were fed the test data set once again, and per-class accuracy figures were recorded. Figures 4.6 and 4.7 display histograms of this result.

	Number of Training Epochs					
Network	400	200	50	20	10	5
Pretr.	—	84.2%, 0.7%	83.1%, 0.7%	82.0%, 0.8%	80.8%, 1.1%	79.3%, 1.2%
Non-Pr.	82.4%, 0.9%	79.7%, 1.5%	—	—	—	—

Table 4.1: Selected figures summarizing the results from the present project. The percentages are the average peak accuracy (over 100 runs) and associated standard deviations. Certain figures are excluded from the table because they are not meaningful. Percentages are rounded to the nearest tenth of a percent; epoch numbers are rounded to the nearest epoch. The pretrained AlexNet clearly outperforms the non-pretrained AlexNet, but analyzed in terms of efficiency its advantage is even more striking: with just 20 epochs of training it is clearly superior to the non-pretrained AlexNet with 10 times as much training, and almost tied with the non-pretrained AlexNet with 20 times as much training.

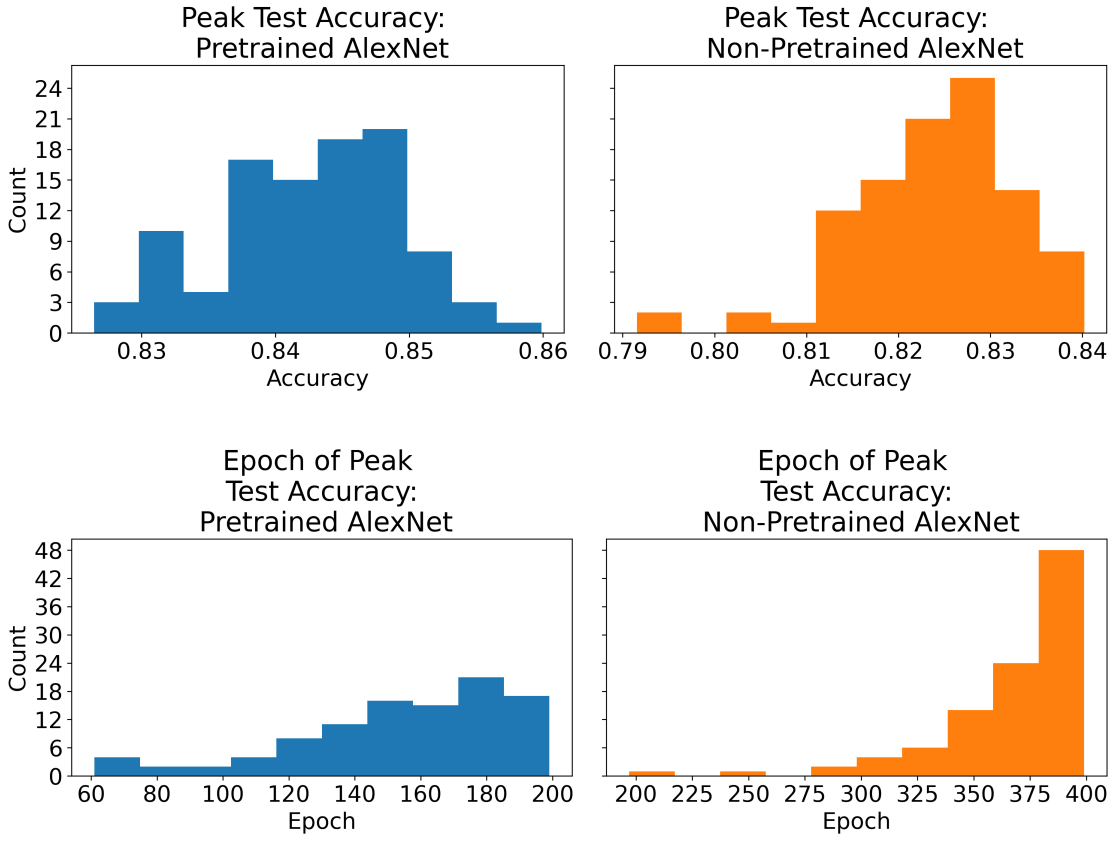

Figure 4.1: *Top:* Histograms of peak test accuracies for all runs for both the pretrained and non-pretrained networks. The pretrained networks are both more accurate on average and less varied in the peak accuracy that they achieve. *Bottom:* Histograms of the epoch in which peak test accuracy is achieved for each run, for both pretrained and non-pretrained networks. These histograms appear to suggest that 200 epochs of training are probably sufficient for the pretrained networks, while the non-pretrained networks might have benefited from even more than 400 epochs of training.

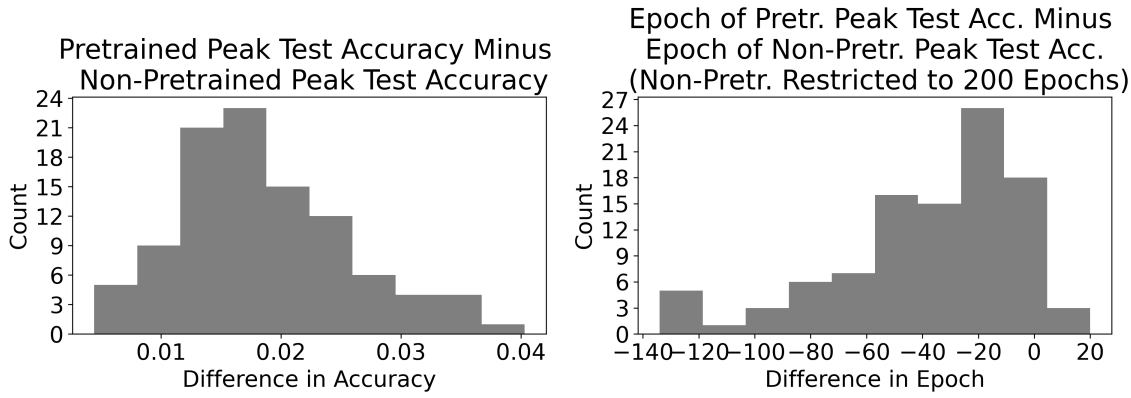

Figure 4.2: Histograms of the gains that result from using the pretrained network as opposed to the non-pretrained network. On the left is the peak test accuracy for the pretrained network minus the respective figure for the non-pretrained network. On the right is the epoch in which peak test accuracy was achieved for the pretrained network, minus the respective figure for the non-pretrained network, *if the non-pretrained network had been restricted to train for only 200 epochs*. It is clear that in terms of accuracy, the non-pretrained network never outperforms the pretrained network. In terms of speed in reaching peak accuracy, the pretrained network is almost always faster than the non-pretrained network, even when restricting the latter to 200 epochs of training.

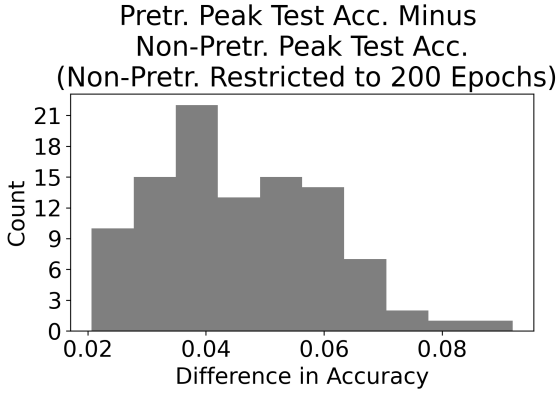

Figure 4.3: This figure is similar to the left histogram in Figure 4.2, except that we only consider the first 200 epochs for the non-pretrained networks. The advantage for the pretrained networks roughly doubles in this case.

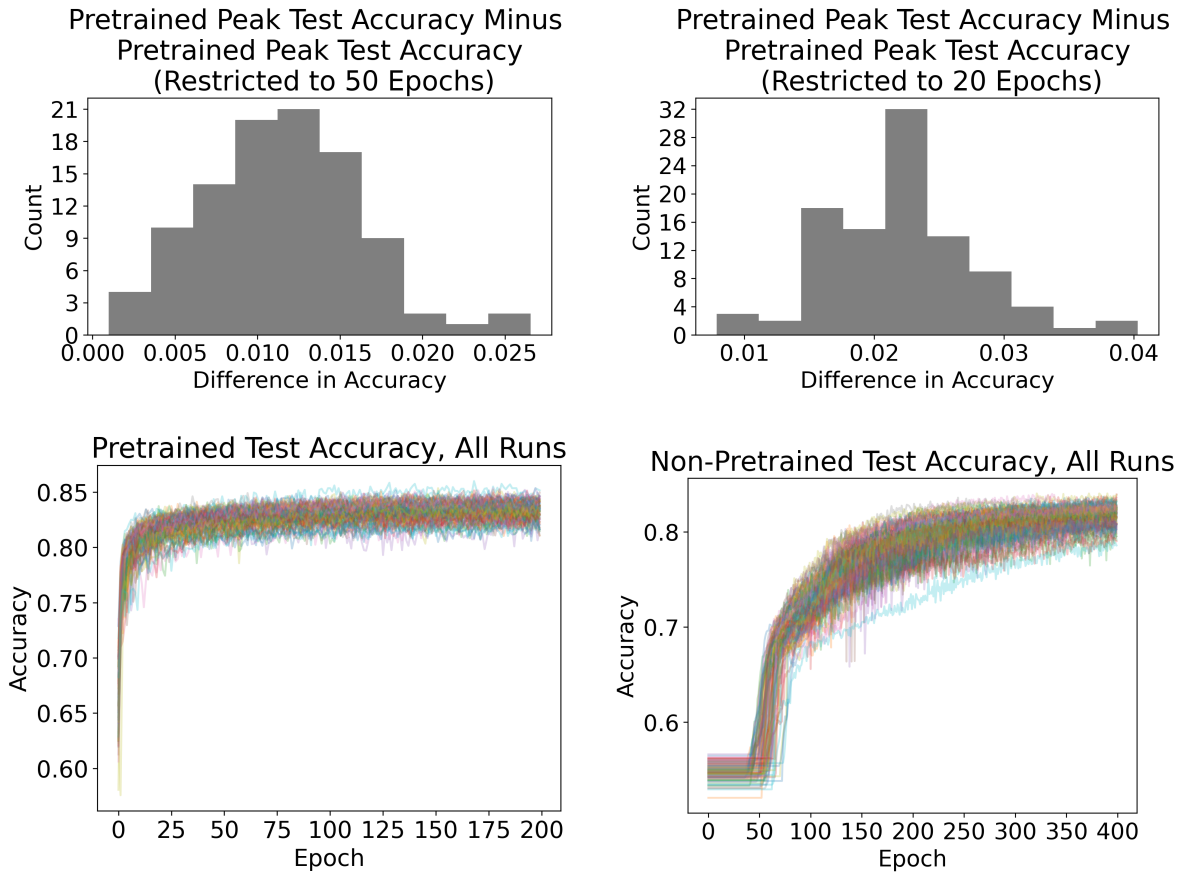

Figure 4.4: *Top Left:* Gain from letting the pretrained networks train for 200 epochs, as opposed to only 50 epochs. The gain is roughly 1%. *Top Right:* Gain from letting the pretrained networks train for 200 epochs, as opposed to only 20 epochs. The gain is roughly 2–2.5%. *Bottom Left:* Each line in this panel contains the progression in test accuracy for the pretrained AlexNet for 1 of the 100 runs performed. It is clear that most of the improvemnt occurs within the first 20–30 epochs of training. *Bottom Right:* Each line in this panel contains the progression in test accuracy for the non-pretrained AlexNet for 1 of the 100 runs performed. The width of the band of lines suggests that performance of the non-pretrained AlexNet is more variable than the pretrained AlexNet. Furthermore, it usually takes around 50 epochs of training for the non-pretrained AlexNet's parameters to adjust enough in order for the network's predictions to begin shifting.

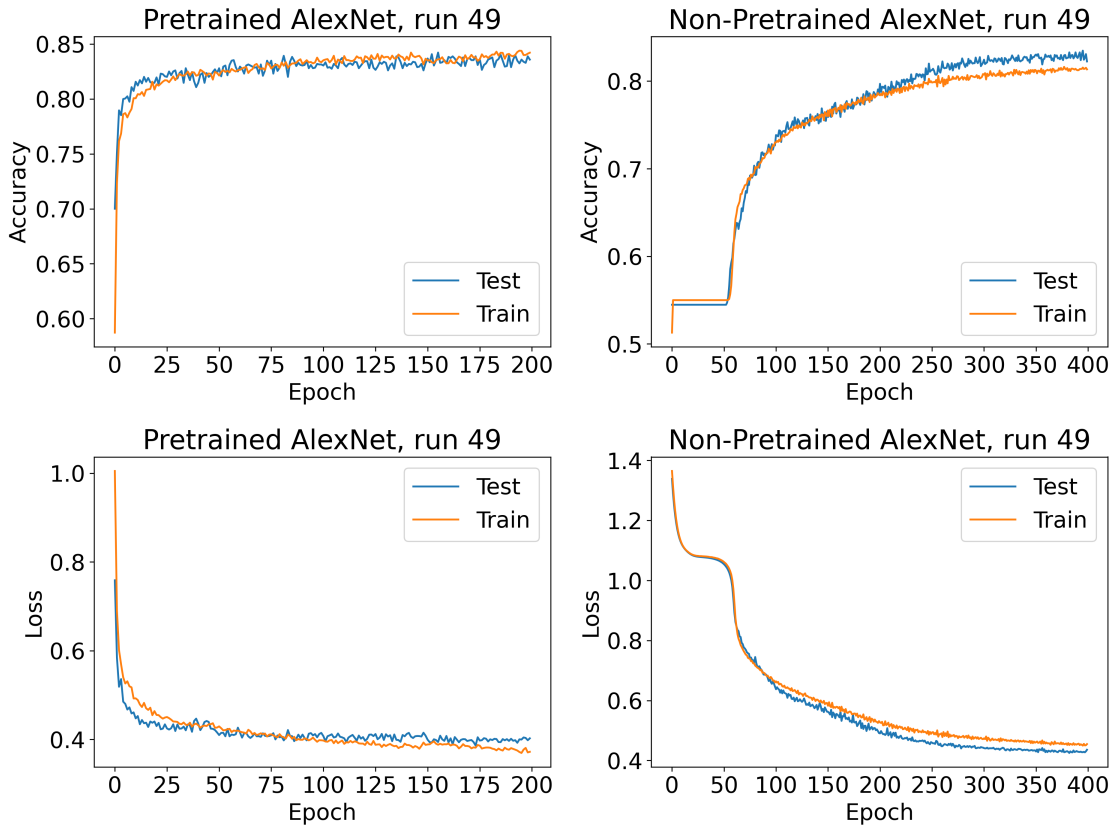

Figure 4.5: *Top left:* Train and test accuracies of run 49 over all epochs for the pretrained AlexNet. The blue line in this panel is one of the lines in the bottom-left panel in Figure 4.4. *Top right:* Equivalent information is presented for the non-pretrained AlexNet. *Bottom left:* Train and test loss values of run 49 over all epochs for the pretrained AlexNet. The blue line in this panel is one of the lines in the bottom-right panel in Figure 4.4. *Bottom right:* Equivalent information is presented for the non-pretrained AlexNet. Run 49 was chosen as an arbitrary representative of the 100 runs performed, although naturally there is some variation. Across most runs, performance of the pretrained network in training eventually surpasses performance in testing, although the point at which this occurs, and the eventual gap between the two, vary from run to run. This phenomenon is much weaker, perhaps nonexistent, for the non-pretrained networks, which may suggest that they would have benefited from more than 400 epochs of training. The non-pretrained networks also often take at least 50 epochs before the weights have adjusted enough for predictions to begin improving. (Initially the non-pretrained networks appear to classify all test images as being of spiral galaxies.)

The models were all roughly the same in that they were quite accurate when presented with images of spiral galaxies, less accurate when presented with images of elliptical galaxies, and less accurate still when presented with images of lenticular galaxies. Furthermore, they perform quite poorly when presented with images of irregular or miscellaneous galaxies, presumably due to the fact that there are few such images in the data set, and moreover the category itself is ill defined in the sense that it mostly serves as a grab-bag of images that fit in none of the preceding three categories.

However, besides the fact that pretrained networks classify images of elliptical, lenticular and spiral galaxies slightly more accurately than do the non-pretrained networks, there is one striking difference, namely that the pretrained networks classify images of irregular and miscellaneous galaxies almost twice as accurately as the non-pretrained networks. Even though the accuracy is still below 50%, this result might suggest that pretraining is potentially valuable for acquiring knowledge of uncommon or irregular examples in the application at hand. The fact that the pretrained networks offer a negligible improvement over the non-pretrained on spiral galaxies, by far the most common in the data set, might bolster this hypothesis.

Table 4.2 summarizes the results presented in Figures 4.6 and 4.7. This Table makes clear that not only is the pretrained network superior across all classes to the non-pretrained network in average class accuracy, but the pretrained network is also less variable in its classification performance. The only exception to this is the Irr+Misc class, for which the pretrained network is more variable than the non-pretrained network, but the pretrained network's almost twofold superiority in average accuracy for this class offsets its slightly higher variability.

Figure 4.8 presents confusion matrices for the top-performing pretrained and non-pretrained models from run 49. (This is the same run as that presented in Figure 4.5.) The information in this figure is consistent with that presented in Figures 4.6 and 4.7, namely the hierarchy in classification performance across the four categories of galaxy morphology, and the general superiority of the pretrained network. However, the confusion matrices also provide some insight into the nature of the *mis*classifications made by the networks. In particular, both pretrained and non-pretrained models tend to misclassify pictures of galaxies into adjacent morphological categories. For example, the majority of the misclassified spiral galaxies are classified as lenticular, as opposed to being classified as elliptical galaxies.

Figure 4.9 presents confusion matrix data over all 100 runs for the pretrained and non-pretrained AlexNets. (Note that as opposed to Figure 4.8, the totals have been converted to proportions.) Similar to Figure 4.8, Figure 4.9 provides information not only concerning how the models classify images of galaxies, but also how they *mis*classify images of galaxies. In the top-left panel of Figure 4.9, the value in each cell of the matrix is the average value for that cell from the 100 individual confusion matrices for the pretrained network. The bottom-left panel of Figure 4.9 contains an equivalent matrix for the non-pretrained network. The

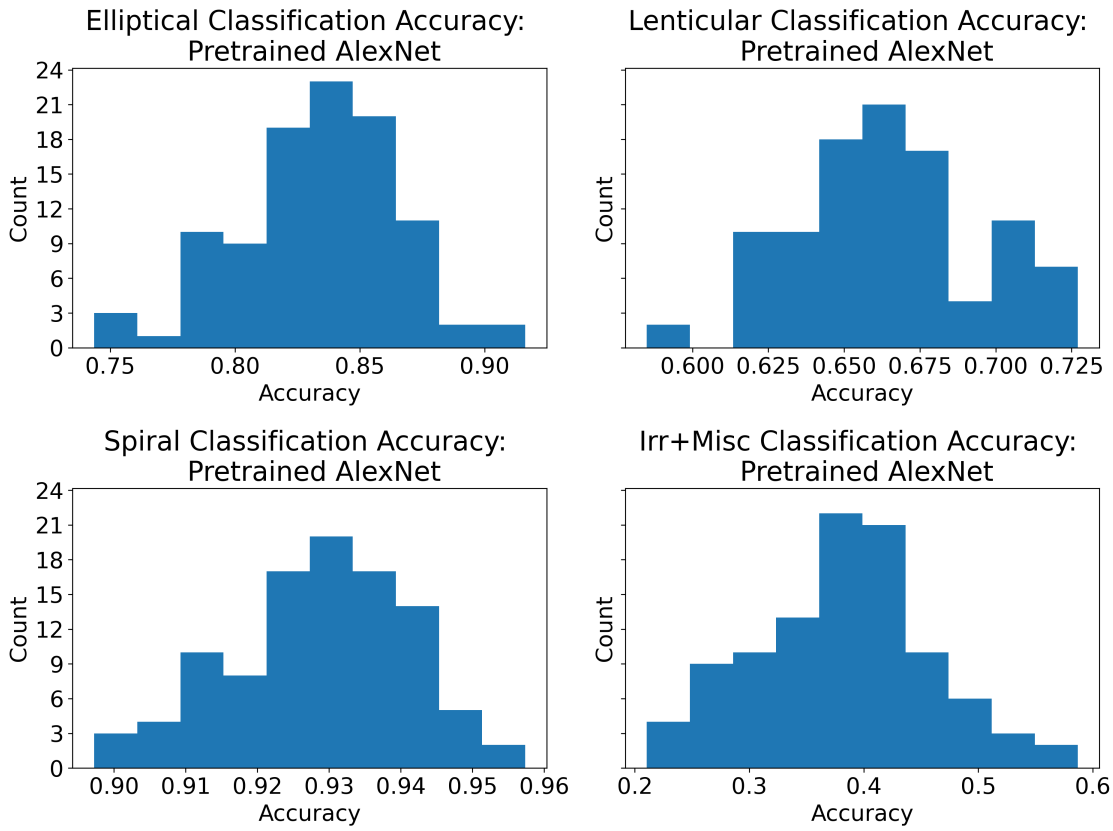

Figure 4.6: Histograms of class accuracies for the most accurate pretrained model of each run. There is a clear hierarchy in performance that is somewhat consistent with the distribution of the test data set (i.e. highest performance in the most frequenly occurring images), although the fact that the lenticular galaxies are in some sense "between" the elliptical and spiral galaxies seems to reduce lenticular accuracy.

top-right and bottom-right panels of Figure 4.9 contain the associated cell-wise standard deviations for the pretrained and non-pretrained networks, respectively. As with Figure 4.8, the most salient feature of Figure 4.9 is the demonstration that both pretrained and non-pretrained networks tend to misclassify images of galaxies into adjacent categories. This is sensible given that the morphological characteristics of these galaxies are thought to occur on a continuum, at least to some extent.

4.2 Statistical Significance Tests

In order to provide a quantitative measure of the differences in performance between the pretrained and non-pretrained AlexNets, *sign tests* are conducted below. (The sign test makes no assumptions about the distribution of the quantities being compared (Roussas, 1997)).

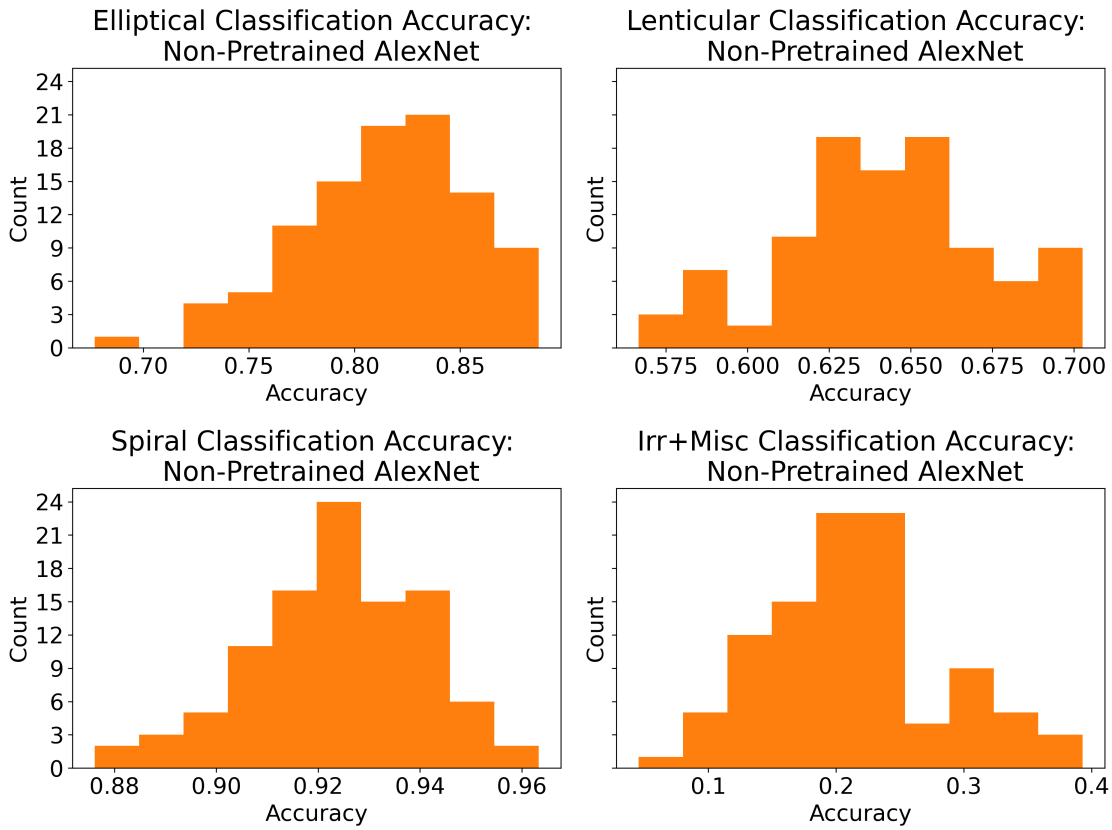

Figure 4.7: Histograms of class accuracies for the most accurate non-pretrained model of each run. These histograms are broadly similar to those in Figure 4.6, with the main exception being much poorer Irr+Misc performance.

Class	**Pretrained**	**Non-Pretrained**
Elliptical	83.3%, 3.2%	81.5%, 4.0%
Lenticular	66.4%, 3.0%	64.2%, 3.0%
Spiral	92.9%, 1.2%	92.4%, 1.7%
Irr+Misc	38.1%, 7.9%	21.4%, 6.7%

Table 4.2: Class accuracy averages and standard deviations across all 100 runs for both pretrained and non-pretrained networks. Percentages are rounded to the nearest tenth of a percent. The pretrained networks are more accurate on average across all categories, and also have smaller or equal standard deviations with the exception of the Irr+Misc category. Despite the larger standard deviation in that case, it seems clear that the pretrained networks are far superior for the Irr+Misc category.

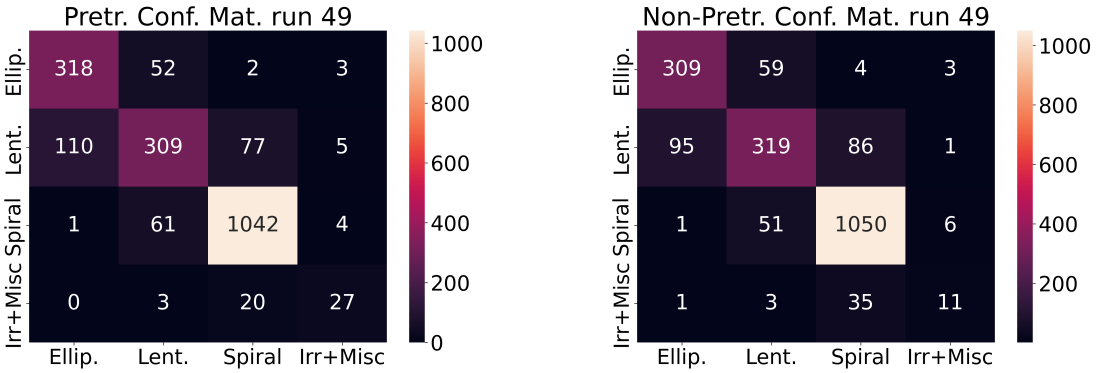

Figure 4.8: *Left:* Confusion matrix for the top-performing model from run 49 of the pretrained AlexNet. The sum of the numbers within a row is the number of images of that type within the test set for that run. For example, looking at the second row, in run 49 there were $110 + 309 + 77 + 5 = 501$ images of lenticular galaxies in the test set, and $309/501 \approx 61.7\%$ of those images were classified correctly. Furthermore, $110/501 \approx 22.0\%$ of images of lenticular galaxies were mis-classified as elliptical galaxies. The confusion matrices for all runs are roughly similar in that the models tend to mis-classify images of galaxies into adjacent categories, which is relatively sensible. *Right:* Confusion matrix for the top-performing model from run 49 of the non-pretrained AlexNet. Again, the sum of the numbers within a row is the number of images of that type within the test set for that run. For example, looking at the second row, in run 49 there were $95 + 319 + 86 + 1 = 501$ images of lenticular galaxies in the test set, and $319/501 \approx 63.7\%$ of those images were classified correctly. Furthermore, $95/501 \approx 19.0\%$ of images of lenticular galaxies were mis-classified as elliptical galaxies. As with the pretrained network, the confusion matrices for all runs are roughly similar in that the models tend to mis-classify images of galaxies into adjacent categories.

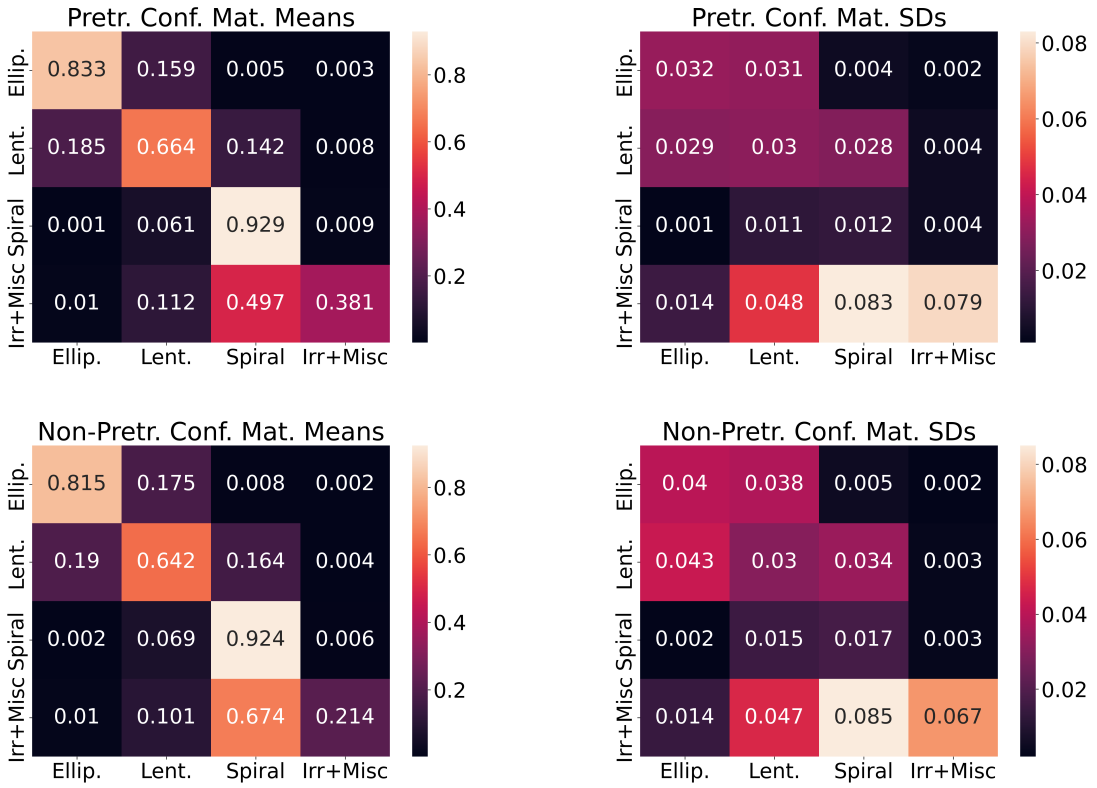

Figure 4.9: *Top Left:* The average confusion matrix for the pretrained AlexNet is shown. The entries in this confusion matrix are the means across all 100 confusion matrices for the pretrained network, one pertaining to each run. This figure shows that the network tends to misclassify images of galaxies into adjacent categories. *Top Right:* A matrix showing the standard deviations of the individual confusion matrices for the pretrained network over all 100 runs. *Bottom Left:* The average confusion matrix for the non-pretrained AlexNet. The entries in this confusion matrix are the means across all 100 confusion matrices for the non-pretrained network, one pertaining to each run. Like the top-left figure, the bottom-left figure shows that the non-pretrained network tends to misclassify images of galaxies into adjacent categories. *Bottom Right:* A matrix showing the standard deviations of the individual confusion matrices for the non-pretrained network over all 100 runs.

Let $X_0, X_1, \ldots, X_{99}$ be independent and identically distributed random variables with distribution function F, representing the distribution of peak test accuracy for the 100 runs of the pretrained AlexNet (over 200 epochs of training). Similarly, let $Y_0, Y_1, \ldots, Y_{99}$ be independent and identically distributed random variables with distribution function G, representing the distribution of peak test accuracy for the 100 runs of the *non*-pretrained AlexNet (over 400 epochs of training). We wish to test the hypothesis

$$H\colon F = G.$$

The result of the two-sided sign test is a p-value of approximately 1.58×10^{-30}, which provides strong evidence against the null hypothesis. We therefore have statistically significant evidence that the pretrained model is more accurate, even though the differences in accuracy may appear slight.

We can perform a similar test on differences in training time and energy used, using the number of epochs of training required to reach peak test accuracy as a proxy. Using similar definitions as above, the result of a two-sided sign test is the same p-value of approximately 1.58×10^{-30}, which again provides strong evidence against the null hypothesis. We therefore have statistically significant evidence that the pretrained model is not only more accurate but also more efficient.

The details of both sign tests are as follows. To test for a difference in accuracy, define the random variables

$$Z_j \coloneqq \begin{cases} 1, & \text{if } X_j > Y_j \\ 0, & \text{if } X_j < Y_j \end{cases}$$
$$Z \coloneqq \sum_{j=0}^{99} Z_j$$

for $j = 0, 1, \ldots, 99$, and the probability

$$p \coloneqq \mathbf{P}(X_j > Y_j).$$

Then we have $Z \sim \mathrm{Bin}(100, p)$ and the hypothesis H above is equivalent to testing $p = \frac{1}{2}$.

In the present case, we have $Z = 100$, so the p-value can be computed as follows:

$$\begin{aligned}
\mathbf{P}(Z \leq 0 \cup Z \geq 100) &= \mathbf{P}(Z \leq 0) + \mathbf{P}(Z \geq 100) \\
&= \binom{0}{100} \left(\frac{1}{2}\right)^{0} \left(1 - \frac{1}{2}\right)^{100-0} + \binom{100}{100} \left(\frac{1}{2}\right)^{100} \left(1 - \frac{1}{2}\right)^{100-100} \\
&= \left(\frac{1}{2}\right)^{100} + \left(\frac{1}{2}\right)^{100} \\
&= \frac{1}{2^{99}} \\
&\approx 1.58 \times 10^{-30}.
\end{aligned}$$

To test for a difference in efficiency, we make similar definitions to those made above, but using a subscript e for "epoch". We find that $Z_e = 0$ because in every case the non-pretrained AlexNet required more epochs to reach its peak test accuracy than did the pretrained AlexNet. The computations and resulting p-value are identical to those shown above.

Chapter 5

Summary and Outlook

The main objective of this work is to compare pretrained (on ImageNet) and non-pretrained versions of AlexNet by training them on galaxy images from the Sloan Digital Sky Survey Data Release 4 (as described in (Nair & Abraham, 2010)) and comparing their performance and efficiency. We note that while the overall classification accuracies achieved are comparable to or slightly surpass similar attempts (e.g., those described in (Cavanagh et al., 2021)), chasing the highest possible classification accuracy would lead us to consider other network architectures, hyperparameter tuning, etc. Rather, we have demonstrated the benefit to considering pretrained deep learning models for certain tasks. Our results are as follows:

1. The pretrained AlexNet had a consistent edge (compared to the non-pretrained AlexNet) in peak classification accuracy. It had an 84.2% average peak test accuracy, compared to an average peak test accuracy of 82.4% for the non-pretrained AlexNet.

2. The pretrained AlexNet was much more efficient (compared to the non-pretrained AlexNet) in that it attained peak test accuracy much more quickly. On average, the pretrained AlexNet achieved peak test accuracy in epoch 155 (standard deviation of 34 epochs), compared to epoch 367 (standard deviation of 33 epochs) for the non-pretrained AlexNet.

3. When considering only the first 200 epochs of training for the non-pretrained AlexNet, in order to provide a comparison with the pretrained AlexNet given an equal amount of training, the peak classification accuracy advantage for the pretrained AlexNet more than doubles, to about 4.6%.

4. The pretrained AlexNet achieves comparable performance to state-of-the-art methods, such as (Cavanagh et al., 2021), rather quickly. The pretrained AlexNet's average peak test accuracy after just 20 epochs of training is 82.0%, comparable with the headline 81–83% figures from (Cavanagh et al., 2021). After 50 epochs of training, the pretrained AlexNet's figure is 83.1%. This suggests that, taking advantage of pretraining, peak performance comparable to that from (Cavanagh et al., 2021) can

be achieved in as little as 10–60 minutes, depending on the computational resources at hand.

5. Regarding per-class accuracies, the most striking advantage for the pretrained AlexNet is that it often classifies the Irr+Misc images more than twice as accurately as the non-pretrained AlexNet. (Gains in classification accuracy for the other three categories are much smaller.)

Regarding the last point, as a neural network is somewhat of a black box, it is hard to know precisely how the pretrained AlexNet becomes so much more adept at identifying images of Irr+Misc galaxies. It seems reasonable to speculate that there is some sort of generalizable information within the unrelated ImageNet (pre)training set that is nevertheless applicable to classifying images of galaxies. Further speculating, it may be that pretraining on large, general, but unrelated data sets is of particular value in maximizing the ability to identify or classify rare cases in the particular application of interest, particularly when those rare cases are considered significant.

A challenge for galaxy morphology classification and many areas of astrophysical image classification more generally is the relative lack of training data. The number of galaxy images available for the present work, 14,034, is much smaller than the amount of data typically available for training deep learning models; ImageNet alone contains more than 14,000,000 images, the labeling of which is trivial compared to galaxy morphology classification by hand. While upcoming surveys such as those by *Euclid* will generate more data, proper labelling remains a challenge. However, the use of pretrained models, as we described in this project, offers the community a way of leveraging the significant effort already spent on developing and training deep learning models without sacrificing accuracy. Indeed, we suggest that the accuracy of a pretrained model may be slightly superior on common examples and vastly superior on rare examples, with much greater efficiency to boot.

Looking ahead we note that, in the deep learning community, AlexNet in particular and perhaps CNNs in general are no longer considered state of the art. This can be seen, for instance, in the progression of performance on the ImageNet data set over time. AlexNet is no longer close to the top-performing models on ImageNet, most of which are no longer CNNs. Transformer models (Vaswani et al., 2017) are often the highest-performing architectures currently and are much closer to the current state of the art. Exploration of their properties and performance is the subject of future work, especially the potential benefit of pretraining with them.

References

Ackermann, S., Schawinski, K., Zhang, C., Weigel, A. K., & Turp, M. D. (2018, 05). Using transfer learning to detect galaxy mergers. *Monthly Notices of the Royal Astronomical Society*, *479*(1), 415-425. Retrieved from `https://doi.org/10.1093/mnras/sty1398` doi: 10.1093/mnras/sty1398

Adelman-McCarthy, J. K., Agüeros, M. A., Allam, S. S., Anderson, K. S. J., Anderson, S. F., Annis, J., ... Zucker, D. B. (2006, jan). The fourth data release of the sloan digital sky survey. *The Astrophysical Journal Supplement Series*, *162*(1), 38. Retrieved from `https://dx.doi.org/10.1086/497917` doi: 10.1086/497917

Aggarwal, C. C. (2018). *Neural networks and deep learning: A textbook.* Gewerbestrasse 11, 6330 Cham, Switzerland: Springer.

Borowiec, D., Harper, R. R., & Garraghan, P. (2022, 01). The environmental consequence of deep learning. *ITNOW*, *63*(4), 10-11. Retrieved from `https://doi.org/10.1093/itnow/bwab099` doi: 10.1093/itnow/bwab099

Cabrera-Vives, G., Reyes, I., Förster, F., Estévez, P. A., & Maureira, J.-C. (2016). Supernovae detection by using convolutional neural networks. In *2016 international joint conference on neural networks (ijcnn)* (p. 251-258). doi: 10.1109/IJCNN.2016.7727206

Cavanagh, M. K., Bekki, K., & Groves, B. A. (2021, 06). Morphological classification of galaxies with deep learning: comparing 3-way and 4-way CNNs. *Monthly Notices of the Royal Astronomical Society*, *506*(1), 659-676. Retrieved from `https://doi.org/10.1093/mnras/stab1552` doi: 10.1093/mnras/stab1552

Cheng, T.-Y., Conselice, C. J., Aragón-Salamanca, A., Li, N., Bluck, A. F. L., Hartley, W. G., ... Tarle, G. (2020, 02). Optimizing automatic morphological classification of galaxies with machine learning and deep learning using Dark Energy Survey imaging. *Monthly Notices of the Royal Astronomical Society*, *493*(3), 4209-4228. Retrieved from `https://doi.org/10.1093/mnras/staa501` doi: 10.1093/mnras/staa501

Davies, A., Serjeant, S., & Bromley, J. M. (2019, 05). Using convolutional neural networks to identify gravitational lenses in astronomical images. *Monthly Notices of the Royal Astronomical Society*, *487*(4), 5263-5271. Retrieved from `https://doi.org/10.1093/mnras/stz1288` doi: 10.1093/mnras/stz1288

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 ieee conference on computer vision and pattern recognition* (pp. 248–255).

Domínguez Sánchez, H., Huertas-Company, M., Bernardi, M., Kaviraj, S., Fischer, J. L., Abbott, T. M. C., ... Zuntz, J. (2018, 12). Transfer learning for galaxy morphology from one survey to another. *Monthly Notices of the Royal Astronomical Society*,

484(1), 93-100. Retrieved from https://doi.org/10.1093/mnras/sty3497 doi: 10.1093/mnras/sty3497

García-Martín, E., Rodrigues, C. F., Riley, G., & Grahn, H. (2019). Estimation of energy consumption in machine learning. *Journal of Parallel and Distributed Computing*, *134*, 75-88. Retrieved from https://www.sciencedirect.com/science/article/pii/S0743731518308773 doi: https://doi.org/10.1016/j.jpdc.2019.07.007

George, D., Shen, H., & Huerta, E. A. (2018, May). Classification and unsupervised clustering of ligo data with deep transfer learning. *Phys. Rev. D*, *97*, 101501. Retrieved from https://link.aps.org/doi/10.1103/PhysRevD.97.101501 doi: 10.1103/PhysRevD.97.101501

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning.* One Broadway 12th Floor Cambridge, MA 02142, United States: The MIT Press.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., ... Oliphant, T. E. (2020, September). Array programming with NumPy. *Nature*, *585*(7825), 357–362. Retrieved from https://doi.org/10.1038/s41586-020-2649-2 doi: 10.1038/s41586-020-2649-2

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. R. (2012). *Improving neural networks by preventing co-adaptation of feature detectors.* arXiv. Retrieved from https://arxiv.org/abs/1207.0580 doi: 10.48550/ARXIV.1207.0580

Hubel, D. H., & Wiesel, T. N. (1959). Receptive fields of single neurones in the cat's striate cortex. *The Journal of Physiology*, *148*(3), 574-591. Retrieved from https://physoc.onlinelibrary.wiley.com/doi/abs/10.1113/jphysiol.1959.sp006308 doi: https://doi.org/10.1113/jphysiol.1959.sp006308

Hunter, J. D. (2007). Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, *9*(3), 90–95. doi: 10.1109/MCSE.2007.55

Kim, Dae-Won, Yeo, Doyeob, Bailer-Jones, Coryn A. L., & Lee, Giyoung. (2021). Deep transfer learning for the classification of variable sources. *A&A*, *653*, A22. Retrieved from https://doi.org/10.1051/0004-6361/202140369 doi: 10.1051/0004-6361/202140369

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In F. Pereira, C. Burges, L. Bottou, & K. Weinberger (Eds.), *Advances in neural information processing systems* (Vol. 25). Curran Associates, Inc. Retrieved from https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

Li, Q., Cai, W., Wang, X., Zhou, Y., Feng, D. D., & Chen, M. (2014). Medical image classification with convolutional neural network. In *2014 13th international conference on control automation robotics vision (icarcv)* (p. 844-848). doi: 10.1109/ICARCV.2014.7064414

Lintott, C., Schawinski, K., Bamford, S., Slosar, A., Land, K., Thomas, D., ... Vandenberg, J. (2010, 12). Galaxy Zoo 1: data release of morphological classifications for nearly 900 000 galaxies*. *Monthly Notices of the Royal Astronomical Society*, *410*(1), 166-178. Retrieved from https://doi.org/10.1111/j.1365-2966.2010.17432.x doi: 10.1111/j.1365-2966.2010.17432.x

Nair, P. B., & Abraham, R. G. (2010, February). A Catalog of Detailed Visual Morphological Classifications for 14,034 Galaxies in the Sloan Digital Sky Survey. , *186*(2), 427-456. doi: 10.1088/0067-0049/186/2/427

Paillassa, M., Bertin, E., & Bouy, H. (2020). Maximask and maxitrack: Two new tools for identifying contaminants in astronomical images using convolutional neural networks. *A&A*, *634*, A48. Retrieved from https://doi.org/10.1051/0004-6361/201936345 doi: 10.1051/0004-6361/201936345

pandas development team, T. (2020, February). *pandas-dev/pandas: Pandas.* Zenodo. Retrieved from https://doi.org/10.5281/zenodo.3509134 doi: 10.5281/zenodo.3509134

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., ... Lerer, A. (2017). Automatic differentiation in pytorch. In *Nips-w.*

Ribani, R., & Marengoni, M. (2019). A survey of transfer learning for convolutional neural networks. In *2019 32nd sibgrapi conference on graphics, patterns and images tutorials (sibgrapi-t)* (p. 47-57). doi: 10.1109/SIBGRAPI-T.2019.00010

Roussas, G. G. (1997). *A course in mathematical statistics, second edition.* 525 B Street, Suite 1900, San Diego, CA 92101-4495, United States: Academic Press.

Silva, P., Cao, L. T., & Hayes, W. B. (2018). Sparcfire: Enhancing spiral galaxy recognition using arm analysis and random forests. *Galaxies*, *6*(3). Retrieved from https://www.mdpi.com/2075-4434/6/3/95 doi: 10.3390/galaxies6030095

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, *15*(56), 1929–1958. Retrieved from http://jmlr.org/papers/v15/srivastava14a.html

Stoughton, C., Lupton, R. H., Bernardi, M., Blanton, M. R., Burles, S., Castander, F. J., ... Zheng, W. (2002, January). Sloan Digital Sky Survey: Early Data Release. , *123*(1), 485-548. doi: 10.1086/324741

Strubell, E., Ganesh, A., & McCallum, A. (2020, Apr.). Energy and policy considerations for modern deep learning research. *Proceedings of the AAAI Conference on Artificial Intelligence*, *34*(09), 13693-13696. Retrieved from https://ojs.aaai.org/index.php/AAAI/article/view/7123 doi: 10.1609/aaai.v34i09.7123

Taigman, Y., Yang, M., Ranzato, M., & Wolf, L. (2014). Deepface: Closing the gap to human-level performance in face verification. In *2014 ieee conference on computer vision and pattern recognition* (p. 1701-1708). doi: 10.1109/CVPR.2014.220

Tan, C., Sun, F., Kong, T., Zhang, W., Yang, C., & Liu, C. (2018). A survey on deep transfer learning. In V. Kůrková, Y. Manolopoulos, B. Hammer, L. Iliadis, & I. Maglogiannis (Eds.), *Artificial neural networks and machine learning – icann 2018* (pp. 270–279). Cham: Springer International Publishing.

Tsantekidis, A., Passalis, N., Tefas, A., Kanniainen, J., Gabbouj, M., & Iosifidis, A. (2017). Forecasting stock prices from the limit order book using convolutional neural networks. In *2017 ieee 19th conference on business informatics (cbi)* (Vol. 01, p. 7-12). doi: 10.1109/CBI.2017.23

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... Polosukhin, I. (2017). Attention is all you need. In I. Guyon et al. (Eds.), *Advances in neural information processing systems* (Vol. 30). Curran Associates, Inc. Retrieved from `https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf`

Waskom, M. L. (2021). seaborn: statistical data visualization. *Journal of Open Source Software*, *6*(60), 3021. Retrieved from `https://doi.org/10.21105/joss.03021` doi: 10.21105/joss.03021

Wei, W., Huerta, E. A., Whitmore, B. C., Lee, J. C., Hannon, S., Chandar, R., ... Congiu, E. (2020, 02). Deep transfer learning for star cluster classification: I. application to the PHANGS–HST survey. *Monthly Notices of the Royal Astronomical Society*, *493*(3), 3178-3193. Retrieved from `https://doi.org/10.1093/mnras/staa325` doi: 10.1093/mnras/staa325

Wes McKinney. (2010). Data Structures for Statistical Computing in Python. In Stéfan van der Walt & Jarrod Millman (Eds.), *Proceedings of the 9th Python in Science Conference* (p. 56 - 61). doi: 10.25080/Majora-92bf1922-00a

Yang, T.-J., Chen, Y.-H., Emer, J., & Sze, V. (2017). A method to estimate the energy consumption of deep neural networks. In *2017 51st asilomar conference on signals, systems, and computers* (p. 1916-1920). doi: 10.1109/ACSSC.2017.8335698

York, D. G., Adelman, J., John E. Anderson, J., Anderson, S. F., Annis, J., Bahcall, N. A., ... Yasuda, N. (2000, sep). The sloan digital sky survey: Technical summary. *The Astronomical Journal*, *120*(3), 1579. Retrieved from `https://dx.doi.org/10.1086/301513` doi: 10.1086/301513

Appendix A

Code and Data

The full extent of data produced during the computations for this project is about 27 TB, therefore sharing all of it is unfeasible.[1] Nevertheless, all figures and code have been made available, along with a master record containing sample information as well as accuracy and loss information. This information is available at GitHub:
https://github.com/jsa378/01_masters.

The list below describes the data available at the GitHub link above.

Note that below, `model_1` refers to the pretrained AlexNet and `model_8` refers to the non-pretrained AlexNet.

1. The files `comparison_run_0.png`, `comparison_run_1.png`, ..., `comparison_run_99.png` contain figures like Figure 4.5 for all 100 runs.
2. The files `confusion_matrix_model_1_run_0.png`, `confusion_matrix_model_1_run_1.png`, ..., `confusion_matrix_model_1_run_99.png` contain figures like Figure **??** (i.e. for the pretrained models) for all 100 runs.
3. The files `confusion_matrix_model_8_run_0.png`, `confusion_matrix_model_8_run_1.png`, ..., `confusion_matrix_model_8_run_99.png` contain figures like Figure **??** (i.e. for the non-pretrained models) for all 100 runs.
4. There are various `.png` files with the word `loss` in their name that show information similar to that shown in this project with regard to accuracy. For example, `loss_epoch_histogram_local.png` shows information like that shown in Figure **??**, but for the loss instead of the accuracy.
5. All the constituent images that comprise Figure 4.5 are available individually for all runs. For example, `loss_history_model_8_run_49.png` contains the plot from the lower-right of Figure 4.5.

[1]The large amount of data produced is due to saving every model after every epoch of training. The main information not being shared is the models themselves, since those files are hundreds of megabytes each and there are roughly 60,000 of them.

6. The files `model_1_8_ddp_0.py`, `model_1_8_ddp_1.py`, ..., `model_1_8_ddp_9.py` contain the Python code used for data preparation and model training.[2]

7. The file `collation.py` was used to create most of the plots shown in this project. (Some plots were created in the Python code mentioned above.)

8. The file `sign_test.py` was used to process the data to perform the sign tests.

9. The file `sample_perf_history_master.tar` can be loaded using PyTorch. In the following I will assume that it has been loaded as the variable `sample_perf_history`. Once loaded, it is a Python dictionary containing the following keys and information:[3]

   i `samples_list`: Indices for the training and test data for each run. For example, `sample_perf_history['samples_list'][49]['training_set_indices']` and `sample_perf_history['samples_list'][49]['test_set_indices']` respectively contain the indices of the training and test samples drawn when randomly splitting the data for run 49.

   ii `samples_counts`: The number of Elliptical, Lenticular, Spiral and Irr+Misc galaxies (in that order) in training and test sets for each run. Examples of use: `sample_perf_history['samples_counts'][49]['training_set']` and `sample_perf_history['samples_counts'][49]['test_set']`.

   iii `samples_percentages`: Percentage breakdowns of Elliptical, Lenticular, Spiral and Irr+Misc galaxies for training and test sets for each run. Examples of use: `sample_perf_history['samples_percentages'][49]['training_set']` and `sample_perf_history['samples_percentages'][49]['test_set']`.

   iv `pretrained_best_train_acc`: Peak training accuracy of pretrained AlexNet for all runs. Example of use: `sample_perf_history['pretrained_best_train_acc'][49]`.

   v `pretrained_best_test_acc`: Peak testing accuracy of pretrained AlexNet for all runs. Example of use: `sample_perf_history['pretrained_best_test_acc'][49]`.

   vi `pretrained_best_train_acc_epoch`: Epoch when peak training accuracy occurred for all runs. Example of use: `sample_perf_history['pretrained_best_train_acc_epoch'][49]`.

   vii `pretrained_best_test_acc_epoch`: Epoch when peak testing accuracy occurred for all runs. Example of use: `sample_perf_history['pretrained_best_test_acc_epoch'][49]`.

   viii `pretrained_best_train_loss`: Minimum training loss of pretrained AlexNet for all runs. Example of use: `sample_perf_history['pretrained_best_train_loss'][49]`.

   ix `pretrained_best_test_loss`: Minimum testing loss of pretrained AlexNet for all runs. Example of use: `sample_perf_history['pretrained_best_test_loss'][49]`.

[2]The work was split into 10 pieces in order to take advantage of available resources.

[3]Because of the often self-explanatory labeling system used, I will list all keys contained in the dictionary, but not describe all of them.

x `pretrained_best_train_loss_epoch`: Epoch when minimum training loss occurred for all runs. Example of use:
`sample_perf_history['pretrained_best_train_loss_epoch'][49]`.

xi `pretrained_best_test_loss_epoch`: Epoch when minimum testing loss occurred for all runs. Example of use:
`sample_perf_history['pretrained_best_test_loss_epoch'][49]`.

xii `non_pretrained_best_train_acc`

xiii `non_pretrained_best_test_acc`

xiv `non_pretrained_best_train_acc_epoch`

xv `non_pretrained_best_test_acc_epoch`

xvi `non_pretrained_best_train_loss`

xvii `non_pretrained_best_test_loss`

xviii `non_pretrained_best_train_loss_epoch`

xix `non_pretrained_best_test_loss_epoch`

xx `pretrained_cumulative_train_acc`: Lists of training accuracies for every epoch of every run. Example of use:
`sample_perf_history['pretrained_cumulative_train_acc'][49]` provides values for all epochs for run 49.

xxi `pretrained_cumulative_test_acc`: Lists of testing accuracies for every epoch of every run. Example of use:
`sample_perf_history['pretrained_cumulative_test_acc'][49]` provides values for all epochs for run 49.

xxii `pretrained_cumulative_train_loss`: Lists of training losses for every epoch of every run. Example of use:
`sample_perf_history['pretrained_cumulative_train_loss'][49]` provides values for all epochs for run 49.

xxiii `pretrained_cumulative_test_loss`: Lists of testing losses for every epoch of every run. Example of use:
`sample_perf_history['pretrained_cumulative_test_loss'][49]` provides values for all epochs for run 49.

xxiv `non_pretrained_cumulative_train_acc`

xxv `non_pretrained_cumulative_test_acc`

xxvi `non_pretrained_cumulative_train_loss`

xxvii `non_pretrained_cumulative_test_loss`

xxviii `pretrained_elliptical_acc`: Elliptical class accuracy for the best model from every run. Example of use:
`sample_perf_history['pretrained_elliptical_acc'][49]`.

xxix `pretrained_lenticular_acc`: Lenticular class accuracy for the best model from every run. Example of use:
`sample_perf_history['pretrained_lenticular_acc'][49]`.

xxx `pretrained_spiral_acc`: Spiral class accuracy for the best model from every run. Example of use:
`sample_perf_history['pretrained_spiral_acc'][49]`.

xxxi `pretrained_irrmisc_acc`: Irr+Misc class accuracy for the best model from every run. Example of use: `sample_perf_history['pretrained_irrmisc_acc'][49]`.

xxxii `non_pretrained_elliptical_acc`

xxxiii `non_pretrained_lenticular_acc`

xxxiv `non_pretrained_spiral_acc`

xxxv `non_pretrained_irrmisc_acc`