AN OBJECTIVE COMPARISON OF FEATURE REDUCTION METHODS FOR NEURAL NETWORK IMAGE CLASSIFICATION

AN OBJECTIVE COMPARISON OF FEATURE REDUCTION METHODS FOR NEURAL NETWORK IMAGE CLASSIFICATION

A project report submitted in partial
fulfillment of the requirements for the degree of
Master of Science

By

Drake Young
University of Colorado Denver, 2021
Bachelor of Science. in Computer Science

April 2021

University of Colorado Denver

ABSTRACT

When using a neural network for image classification, the number of features input into the classifier can greatly impact the performance of the model. Reducing the number of features can yield significant performance gains on inference time, model size, and training overhead. These gains come at an additional cost in the preprocessing step, however.

In this project, the exact performance metrics, in terms of training and validation accuracy and loss as well as the overhead time cost at each step of the program's use, will be measured and reported in order to construct an objective comparison of various preprocessing approaches for the reduction of the feature space. The experiments will be performed identically using two different convolutional neural network (CNN) architectures as well as across two different datasets in order to evaluate how obvserved behaviors translate across these changes.

This Project Report is approved for recommendation to the Graduate Committee.

Project Advisor:

_____

Name1, e.g., Jane Doe

MS Project Committee:

_____

Name2

_____

Name3

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION

1.1 Problem

From an objective standpoint, image data tends to be large and noisy. Much of the pixel data within an image is unnecessary, and results in orders of magnitude more parameters needing to be trained in a neural network. The current iPhone 11 camera takes RGB photos which are 3024x4032 pixels in dimension, which results in over 36 million features present in each image that a model would need to train on if unprocessed. Training on so many parameters affects the training time of a model, and the amount of noisy features in large images may impact the accuracy of the model. There already exist several techniques for preprocessing the image data in order to reduce the feature space and improve training-time-performance of models trained on those features. The intent of this project is to directly compare the performance and the overhead of these methods.

Though there exist three many methods for reducing the feature space of images, three specific techniques are evaluated in this project to act as representatives for their given scopes. One such technique is the reconstruction of the images as grayscale (reduction from three color channels to one color channel that indicates brightness/darkenss in an image) paired with resizing the image (reduce the number of pixels for the height and width of the image) and regularization (perform min-max scaling to assure pixel values in the input are between 0 and 1); this technique is used as the standard implementation when preprocessing is necessary, as demonstrated by Dey et al [1], Omidiora et al [2], and Chrabaszcz et al [3]. Another such technique is Principal Component Analysis (PCA) combined with variance scaling, which involves extracting the dominant patterns in the matrix (image) in terms of a complementary set of score and loading plots, as described by

Wold et al [4], and such components are filtered down to a smaller number that still captures a sufficient degree of the dataset variance. A final methodology is considered from the field of data science, referred to as Locality Sensitive Hashing (LSH), which utilizes probabilistic methods on Gaussian random projections to perform fast, scalable indexing by reducing the dimensionality of high-dimensional data for fast comparison while preserving locality sensitivity/similarities among data units, as explained by Jiang et al [5]; the theory for using LSH as a preprocessing methodology is to evaluate whether a neural network is capable of learning on these similarity-preserving "hashes" or "Gaussian random projections," which could provide high volumes of image data information in a small number of input features. In all three of these methods, the overall goal is to reduce the number of features present in an a high-dimensional set of data in order to obtain a reduced feature space which is able to approximate most of the relevant information presented within the original data while being able to improve both the computation cost and the data footprint by the use of the reduced feature set.

Though all these techniques are designed in order to achieve the same goals, their relative performance must be compared in order to objectively determine the methodology which should be preferred in most, if not all, cases. Or alternately, to demonstrate the performance-dependence of these methods relying strictly on the problem scope, demonstrating the need to experiment with some or all approaches in the problem-specific scope before use in a full implementation.

1.2 Problem Statement

Within the context of a neural network image classification projeblem, the intent of this project is to evaluate the overhead time and memory costs of various feature

reduction/extraction methods when used as dataset preprocessing before training the network, as well as evaluating the training time performance, reduction in memory cost, the overhead cost when using a pre-trained model for inference, and the overall change in the training performance of the model both in terms of accuracy and loss. The goal is to comparatively and quantitatively evaluate the cost and performance of these preprocessing techniques to objectively determine the "best" preprocessing methodologies for image classification applications when used with neural networks.

1.3 Approach

In order to objectively compare the use of these three methodologies, a series of experiments will be conducted in order to provide the necessary data for an objective, quantitative, comparison between the methods. Each experiment performs image classification with the use of a CNN classifier which is trained on images preprocessed with the various methodologies explored for a total of 25 training epochs in order to achieve baseline benchmark performances of the experiments. Each experiment is conducted in Jupyter Notebook running through Anaconda Shell on Python version 3.7.0. The neural networks are constructed ant trained on the Keras Python package with the use of a Tensorflow-GPU backend, and is trained on a system with an Intel i7-8700k CPU and accelerated with an NVIDIA RTX 2070 GPU for hardware acceleration.

Four methodologies are experimented within this project. As a baseline, the first methodology used is a "No Preprocessing" approach, where the only transformation made to the dataset before use is to ensure that all images are the same dimensions, reshaping them all to the average dimensions of the unaltered training dataset. This methodology is used to act as a benchmarking baseline for all of the measured costs/gains extracted

throughout the experiments. In the No Preprocessing method, no alterations beyond the initial reshape-on-read are conducted. The second method evaluated demonstrates the current standard implementation – conversion to grayscale, resizing to smaller dimensions, and regularizing/scaling the pixel values. Before experimentation, it is hypothesized that this metric will be the highest achieving in terms of cost and overhead performance, only being a slight overhead cost compared to No Preprocessing, yet achieving scores relatively-close to the No Preprocessing baseline. The third method explored is the use of PCA against scaled features to extract 1,024 of the most impactful features for use as the input. Finally, an LSH experiment is performed, which generates 1,024 hashes to represent the input images for use as the input.

The four experiments listed above are conducted on each of two datasets. The first dataset is intended a binary image classification between cats and dogs. The specific dataset used is called "Dogs & Cats Images," an open-source dataset hosted on Kaggle for the use of image classification by user chetanimravan [6]. The set consists of 8,000 training images and 2,000 validation images, each with proper labeling of either the "cat" or the "dog" class. In this dataset, there is exactly 50% of both the training and validation set as cats and 50% as dogs (4,000 of each class in the training set and 1,000 of each in the validation set). Each image in the dataset has varying pixel dimensions. The second dataset used is another open-source dataset provided on Kaggle. This dataset is called "Intel Image Classification," which consists of 14,034 training images and 3,000 validation images, for use in multi-class image classification; the dataset is posted by Kaggle user Puneet Bansal [7]. Each image is 150x150 pixels in dimension. The images belong to one of six classes: buildings,

forest, glacier, mountain, sea, and street. For each of the six classes, the training set consists of at least 2,000 samples, and the validation dataset consists of at least 400 samples.

Finally, each of the four experiments, on both of the datasets are computed on each of two CNN architectures. The first being a shallow, custom, model; the intent of this small model is to provide a simle demonstration of the effectiveness of the various methods when the network size is relatively small with fewer parameters than would be expected in other architectures. The second being an implementation of the VGG19 model architecture; this architecture is chosen because of the name-recognition of VGG19, and the likelihood of use in fully implemented applications. The VGG19 model is significantly deeper than the custom model used, allowing an example of a model with many trainable parameters to potentially demonstrate the impact on models which would otherwise be significantly large and slow in some use cases. These two model architectures are explained in greater detail in future sections.

Finally, with the use of the Kerastuner Python package, light hyperparameter tuning is performed on each of the 16 experiments in order to determine the optimal learning rate of 10 tested per experiment in order to simulate the results of parameter tuning and its effects on the overhead cost as well as performance. These are completed in separate experiments from their non-tuning variants in order to extract and compare both the non-tuned and tuned variants for use in the evaluation step.

In total, 32 experiments are conducted in order to test the four preprocessing techniques across two distinct datasets, each using both of two model architectures, and each trained once without hyperparameter tuning, and once with hyperparameter tuning. The results of these experiments are consolidated and analyzed in future sections.

1.4 Organization of this Project Report

Section 2 of this project will expand the background insights on the key concepts of CNNs, PCA, and LSH, in order to provide the baseline understanding necessary for the remainder of this report. This section is required in order for readers to become familiar with the key concepts addressed within the experimentations and already briefly discussed within the paper. Section 2 also discusses the known works related to the subject at hand. This includes the relevant information on image preprocessing techniques, image classification, PCA implementation, and LSH implementations. This section is necessary for expanding the greater context of the problem and the other relevant fields of research revolving around the problem scope of this work.

Section 3 will provide a more thorough description of the methodologies used, as well as the model architectures implemented for experimentation. It will provide greater detail on the custom and VGG19 model architectures implemented, including the use of specific figures for representation of the architectures. This section will explain in-depth the reasoning for these choices, and the specific implementations around these architecutres within the context of the experiments performed for this project. This section is necessary for providing the architectural understanding of the CNN models used, enabling the reader to recreate the experiments.

Section 4 will further elaborate the methodology of the experiments performed. This section will explain in detail each of the 32 experiments performed, how they vary, and the reasoning for any experiment-specific design decisions made for the experiment implementations. This section is necessary for providing the reader with the technical understanding and the design decisions necessary for recreation of the experiments

performed within the scope of this project. After introducing the methodologies used within these experiments, the results and all relevant figures will both be provided and analyzed for the reader in order to gain an interpretation of the effectiveness of the experiments performed as well as the allowing the reader to critically analyze the interpretations made for the experiment results in order to affirm the conclusions drawn.

Finally, section 5 of this report will contain the concluding statements. These statements are necessary for the aggregation of all experiment results into a brief overview, which enables the reader to gain a collective understanding of all experiments and all metrics reported in section 4, as well as the expansion of potential future works which did not see an opportunity for implementation within the current scope of the project at the time of submission. This section is necessary for the reader to understand the overall contributions provided by this experimentation, as well as the potential room for other works to further expand upon this implementation.

2. BACKGROUND

2.1 Key Concepts

The reader must understand the the following areas of research in order to fully understand the contents of this project report: Convolutional Neural Networks, Principal Component Analysis, and Locality Sensitive Hashing – three concepts which were briefly introduced in earlier sections. It is necessary to understand each of these three concepts in order to fully understand their use within the scope of the experiment set performed, as described in greater detail in later sections. Section 2.1.1 will provide an in-depth background for the use of CNNs. Section 2.1.2 will provide an in-depth background for the use of principal component analysis as a form of feature space reduction, as will be used for the relevant experiments. Finally, Section 2.1.3 will provide an in-depth background for the use of Locality Sensitive Hashing with respect to the approaches used for the relevant experiments performed in this project.

2.1.1 Convolutional Neural Networks

For the sake of brevity, it is assumed that readers have a familizrity with the use of Artificial Neural Networks (ANNs). For this section, the knowledge of ANNs will be expanded upon to create a baseline understanding of the implementation of a CNN. CNNs are a relatively novel form of implementation for neural networks which have reached high levels of popularity in modern computing due to the increases in performance gains and accessibility in the past ten years. The primary benefit to the use of CNNs over other neural network architectures is the use of convolution layers. Convolution layers allow CNNs to consider the relative neighborhood around pixels, by scanning a small matrix filter over

the original image, and determining the resulting computation of the filter's neighborhood at each stride. These convolutions can allow the network to use relative positioning to learn broader contexts within the network, such as grouping individual pixels into lines, lines into angles, angles into shapes, and so on. As explained by Albawi et al [8], a CNN is able to obtain abstract features as the input of the network is propagated into deeper layers of the network. The use of CNNs has gained prominence thanks to their groundbreaking performance in image classification applications.

When given an input image, at the convolution layers of a CNN, the given image is scanned using a fixed-shape kernel filter. Standard filter dimensions tend to be as small as 3x3 filter windows. Various strides are used in order to define how far the window moves at each step of the convolution. It can be safely assumed that whenever the stride is not specified, it has a value of 1; a stride value of 1 indicates that the filter will shift its field of view 1 pixel to the right of the previous filter or realign itself to the left a single row downward if the end of a row is achieved. Because the convolution step groups a kernel filter of image pixels together into a single representative value, the output of a convolution layer will be of lower dimensionality than the input, but with a depth equal to the number of filters to be learned. The weights of a filter are akin to a node within a hidden layer in an ANN. When the window is passed over a filter, matrix multiplication is performed between the filter and the window at each filter, and the results are passed through an activation function before being passed to the next layer of the network. The activation functions available for convolution layers are identical to those used with ANN architecutres. The Rectivied Linear Unit (ReLU) activation function is typically preferred due to its demonstrated performance over the past decade.

Because of the high volume of filters, which add significant depth to a CNN, increasing the number of parameters to train, the use of pooling is encouraged when designing CNN architectures. In a pooling layer, the filter weights are condensed into a smaller representation. For example, if a 2x2 pool is used, the input to the layer is partitioned into non-overlapping 2x2 sub-matrices. The local maxima of each sub-matrix is then selected to represent the entire sub-matrix and is positioned into an output matrix based on the relative positioning of the sub-matrix it represents. In other terms, the pooling layer will pool-together weights into a single representative for each pool. This allows the number of features to be cut in half for subsequent layers, allowing for the depth of CNNs to expand far beyond what memory constraints would allow without the use of pooling. Pooling layers are an aggregation step which contain no weights for the model to learn.

Finally, once the convolution and pooling layers have reached a desirable depth for the CNN architecture planned, the output is flattened into a single vector and passed through one or more fully-connected layers. These fully connected layers operate identically to a standard ANN in terms of both forward and back propagation.

Training a CNN uses the same backpropagation approach that is used for training ANNs. In the convolution layers, the gradient of the loss is backpropagated to each kernel filter's weights in the same way that the gradient would be back propagated to nodes of a hidden layer in an ANN. For a further understanding of the technical details and the formulae relevant to the use of CNNs, please reference the work by Albawi et al [8].

2.1.2 Principal Component Analysis

For the purposes of brevity, it is assumed that the reader has a background understanding of statistical terminology such as variance. It is also assumed that the reader

is familiar with the use of vector and matrix operations, like the transpose operations and eigenvalues. PCA is a statistical approach which performs a multivariate analysis on the contents of high-dimensional data in order to extract the most important feature information from datasets. As described by Abdi and Williams [9], the principal components (the features extracted by PCA) are a set of orthogonal variables extracted from the original data which are extracted from the eigen-decomposition of the data. That is, determining the immutable features which represent the underlying patterns of the data, and can be combined back with the principal components extracted from specific data points in order to reconstruct the original value, or a close approximation thereof.

For the computation of the principal components, a Singular Value Decomposition (SVD) calculation is performed on the dataset given. SVD is the process of factoring large matrices into the product of two smaller matrices, called the Factors Score Matrix and the Loading Matrix. The decomposition into the product of two matrices is also called a Bilinear Decomposition. Within the context of the applications, we will be using the Factors Scores Matrix for training, and using libraries to perform the analysis. An interesting behavior that occurs when performing PCA on datasets is that multiplying the Loading Matrix by the original data produces that data point's factor scores. This behavior is due to inherent properties of the matrices used in the SVD operation. This means, obtaining the Loading Matrix though analysis will allow us to determine the principal components of both observed/known data as well as unknown/new data.

The value of the Loading Matrix must be learned through analysis of the dataset in order to find the orthogonal weights from within the original dataset. Since PCA and SVD are dimensionality reduction methods, it can be expected that these values are merely close

approximations to the true orthogonal eigenvalues of the dataset, as information is guaranteed to be lost as the dimensionality is reduced. However, with a sufficiently learned Loading Matrix, this impact will be mitigated to minimal values. For the purposes of the experiments performed in this project, the learning of the Loading Matrix weights is not necessary, as that step is controlled-for with the use of the same python package implementation for each PCA experiment. The specific methodology for learning the weights is not as necessary, since the experiment simply uses the abstracted library as a representative of the implementations and used in a consistent manner across all relevant experiments. Further explanation of the usage of this library is provided in future sections.

2.1.3 Locality Sensitive Hashing

The use of Locality Sensitive Hashing is an approach used within the field of data science in order to compare the similarity of data entries in high dimensional spaces. That is, LSH is the use of carefully selected hashing functions in order to achieve sublinear time-performance on nearest neighbor searching. LSH is one of the most widely used algorithms for data points which are highly dimensional in nature, in which even a linear time cost can be prohibitively expensive. The theory applied in this paper is that a CNN may be able to train on these hashes instead of the full piece of data, significantly improving the cost of training and using the CNN with data entries that preserve the locality information on which a CNN's convolution layers benefit from.

The LSH process behaves similalrly to indexing schemes on a dataset. The process has two distinct phases, the hash generation phace and the query phase. During the hash generation phase, many various algorithms are proposed in order to optimize the the hash for a fair balance between the number of remaining features and the volume of information

conveyed by the hashes generated for each point of data. The query phase behaves similarly to any other nearest neighbor search algorithm, comparing the hashes generated by the query against the precomputed hashes generated for the known dataset. Though LSH is typically used for document querying, similar principles can be applied to the highly dimensional pixel values of images, which consist of equivalent highly dimensional data volumes.

For the purposes of this experiment, we only are concerned with the hash generation process, as that step is used to reduce the dimensionality of the images. We will then use the generated hashes for the original images as input for training a CNN for the relevant experiments. Though multiple algorithms are possible under the use of LSH [5, 10, 11], the specific approach used more closely resembles that described by Har-Peled et al [12]. Relying on the idea that the probability of random collisions of similar objects is high – even in high dimensional space. That is, if selecting a random subset of feature indices is probabilistically likely to produce similar or identical subsets of features when the original data points are high similarity. This process is translated into use for images by treating each pixel (or each pixel in each color channel for colored images) as a potential feature to extract from the subset. Finally, in order to represent similarity, the values of the subsets, each selected feature value is mapped to either a 0 or a 1 based on a predefined threshold. By doing this, the subset of features becomes a vector of bits, which can be concatenated into a single base-two number representing a hash of the original datapoint which preserves its similarity to the other data points in the set. This is the hash generation scheme which will be used for all LSH related experiments within this project. When selecting hashes, the probability of collision increases when multiple different subsets of the feature space

are selected and converted into hash strings by the aforementioned process. In order to tune the performance of LSH for achieving an optimal performance in terms of both time and effectiveness, the algorithm may be modified to use a more optimal number of subsets selected (hashes generated), which improves the accuracy of the search, but increases the time-cost of querying. Additional tuning may be performed against the size of the hash strings generated (number of features selected for the random subsets), which improves the query time-cost but increases the amount of false-negatives in nearest neighbor searches. For the sake of simplicity, whenever LSH is referred to in the remainder of this paper, it is specifically talking about this hash generation step performed within the context of the experiment code, performed on the dataset-specific image pixels, and the resulting hash strings are used as the feature-reduced representations of the dataset to be used as input for the CNN.

2.2 Related Work

The use of grayscale image preprocessing is a commonly used methodology for training neural networks for image-processing tasks, and its use is commonly combined with resizing in order to further improve the overhead costs of training models. The work of Chrabaszcz et al [3] utilizes this preprocessing method is used for training a deep reinforcement learning model to play Atari video games, demonstrating significant gains in game scores against their baseline learning methods, showing the effectiveness of this preprocessing scheme. The work of Tuncer et al [13] utilizes similar preprocessing techniques for use in a image classification problem, preprocessing images of lung X-ray scans using the VGG-19 CNN architecture among others in order to classifiy the presence of COVID-19 within the X-rays. Their implementation was capable of achieving an

accuracy of 92.8% to 99.6% across their experiments performed. Their success in this methodology is another point of evidence that the grayscale and resizing preprocessing technique is effective in real-use case scenarios. The work of Sajjad [14] utilizes similar preprocessing techniques for use in the recognition and localization of vehicle license plates within images. The process used performs image segmentation through the Optical Character Recognition Engine in order to interpret the exact value of detected plates. In the experiments performed, a success ratio of at least 92% across all experiments was achieved, further demonstrating the effectiveness of this preprocessing technique. Across the applications examined, we have demonstrated success in reinforcement learning [3], image classification [13], and in image segmentation [14], showing a breadth of use-cases, in which this method is utilized for high performance. For the sake of our own experiments, we will utilize a similar preprocessing scheme on a controlled system against other preprocessing methods to determine the objective relative performance of the grayscale and resize approach within the broader scope of image preprocessing methods.

The use of PCA is demonstrated by the works of Omidiora et al [2] and Oladele et al [15]. Both works address a regression problem on which the program must estimate the ages of individuals based on images of their faces. For both works, the use of PCA is implemented as a stage in the preprocessing step before performing the feature analysis/training. The extracted features resulting from the PCA extraction are used in both works as input to various neural networks similarly to the methodology used within our own respective PCA experiments. Though the experiment results vary between the two works, both are capable of demonstrating the relative effectiveness of PCA for use as training input, allowing the respective methods to learn to discriminate the age of the

person in an image significantly better than random chance, which proves the viability of the PCA approach, and thus, it is included in the experimentations performed by this project. The work of Long et al [16] uses a hybrid approach, combining the use of PCA with the use of an existing preprocessing technique known as Fisher's Linear Discriminant to perform feature extraction on microscope images be used as input for a neural network with the intent to identify the presence of cells and cell clusters within the image. The results of this work demonstrate a convergence of the mean-squared error on the test set across the training. This convergence further signifies the ability for a neural network to learn based on the principal components extracted from images rather than the original image itself, while providing a significan reduction in the size of the feature space. Due to the demonstrated performance of PCA across these various applications, it has been selected as another experimentation performed here. Similar to the works of Omidiora et al [2] and Oladele et al [15], in the experimentations performed for this project, the images will be converted to grayscale before the PCA is performed in order to improve the component analysis overhead time during the use of the applications.

The work of Chen et al [17] demonstrate the use of an LSH preprocessing workflow to be used with neural networks. Unlike the previous approaches considered in the other LSH works listed for various hash generation algorithms not considered in this project [5, 10, 11], the approach used here involves dynamically changing hashes that must be recomputed as the weights of the neural network update. Unlike this work, we will be considering the use of a pre-hashed dataset with static hashes for performing the learning on. LSH is again used as a preprocessing technique for input in CNNs within the context of vehicular vision applications within the work of Qiao et al [18]. Unlike the claims made

by Cheng et al [17], this work makes the claim that the use of LSH preprocessing on the images within the dataset allow the network to achieve real-time performance. The results found by Qiao et al [18] determined that their experimetns were able to yield notable gains in performance for querying sequence matching through the use of their model architecture. Though the time overhead was not measured, the promise of real-time performance and the potential gains in model performance are an indication that the LSH technique is worth conducting experiments on in order to determine the relative effectiveness of LSH compared to other preprocessing methodologies with controlled experiments.

The work of Rozenstein et al [19] examines multiple preprocessing techniques and their use in classification problems. The experiments performed compare multiple preprocessing techniques with the use of multiple different classification methods in order to attempt to demonstrate model independence. Though we take a similar approach to Rozenstein et al within their experiments, there are many areas in which this project deviates. The experiments performed in this project compare two different convolutional neural network approaches, as opposed to comparing a statistical and a random forest approach. Additionally, this project attempts to demonstrate whether these techniques are dataset dependent or dataset independent by repeating each experiment each on once of two different image datasets. Finally, the experiments of Rozenstein et al focus on the use of various scaling and normalization techniques as preprocessing, whereas this project focuses more on feature extraction methods of preprocessing.

The work of Medjahed [20] performs comparison to various feature extraction methods within the context of image classification. There are points of divergence between the work of this project and the work of Medjahed as well. Primarily, the different methods

compared between the two projects: the comparison between no preprocessing, grayscale with resizing, PCA, and LSH is performed within this project, whereas Medjahed explores less general approaches for feature extraction without exploring the overhead costs of these approaches (such as shape feature and texture feature extraction methods). Additionally, Medjahed explores image classification with the use of various Support Vector Machine models for binary classification problems, whereas this project compares CNN models for both binary and multi-class classification problems. These experiments paired together can provide a broad perspective into both the generalized and problem-specific feature exptraction methods for both machine learning and deep learning image classification problems.

3. MODEL ARCHITECTURE AND EXPERIMENT APPROACH

3.1 High Level Experiment Design

This project focuses on the immediate effects of modifying the preprocessing techniques in order to determine whether the use of any specific methodology results in an objectively better performance in terms of training time, inference time, memory footprint, classification crossentropy loss, and classification accuracy. Because of this, we are not as concerned with the direct values of any experiments; it is nearly guaranteed that with proper tuning, model curation, and sufficient training duration, the accuracy and loss scores could achieve greater values than those performances reported in the paper. However, as previously mentioned here, our concern is not about the direct values, but the changes to the values from the baseline as the experiments are performed.

For each of the experiments performed, the same process is followed. Using the experiment-specific dataset (Dogs & Cats Images [6] or Intel Image Classification [7]), the experiment code first generates a list of all training and validation images to be used for the experiment by listing the contents of the specific subdirectories used by the datasets. Next, a multi-dimensional array is populated by iterating over the list of files, using the Open Computer Vision (OpenCV) library [21], storing the pixel-data of each image into system memory. Any image transformations necessary for experiment-specific uses (conversion to grayscale and resizing the image when necessary) are performed during the image-read step. The overhead time cost in seconds is measured for the reading/processing step for the image. Once all the image pixel data is in system memory, experiment specific preprocessing is performed. This preprocessing step is only necessary for PCA and LSH. For the PCA experiments, this step is used for scaling the dataset in order to perform

extraction of the principal components. For the LSH experiments, this step is used for the generation of the hash functions, and extracting the hash representation of each image. For both the PCA and LSH implementations, the vector of components/hashes is then reshaped into 32x32 matrices. The selection of this size is because it is the smallest possible square matrix with power of two dimensions which can be processed by the model architectures used. For these preprocessing steps, the time-cost is again measured and logged for evaluation purposes. Now that the data has been fully preprocessed for each experiment, the experiment specific neural network architecture is constructed and trained for 25 epochs on the dataset using the dataset-defined training and validation partitions. The runtime for training the CNN model is recorded for future reporting of the training overhead. Additionally, the training/validation loss and accuracy are recorded in order to show a direct model performance comparison – a fast model is not useful if it performs significantly worse in terms of accuracy/loss. Finally, for each experiment, a random sample of images is selected, loaded, preprocessed, and the inference is performed. For these experiments, the end-to-end runtime for the use of a trained model is measured for a single inference, 10 inferences at once, 100 inferences at once, and 1000 inferences at once. This will measure the real, post-training performance of the models for each esperiments in terms of runtime overhead.

A total of 32 experiments are conducted within this project. There are two datasets previously discussed for use in the experiments – the Cats & Dogs Images [6] and the Intel Image Classification [7] datasets. For each of the two datasets, the experiments are performed once on each of two separate CNN model architectures – the customly-defined shallow model architecture and the VGG19 model architecture. For each of the four

dataset-architecture pairs, four experiments are conducted to evaluate the different preprocessing methodologies – no preprocessing, grayscale with resizing, PCA, and LSH. Finally each of the 16 dataset-architecture-preprocessing combinations is performed twice, once with no adjustments to the hyperparameters, in order to evaluate whether there is a direct change with all parameters controlled for, and once in which the learning rate is tuned to maximize validation accuracy (tuned to achieve the largest possible accuracy for the specific experiment at the 10^{th} epoch when examining 10 different learning rates). The tuning experiments are an attempt to show whether hyperparameter tuning affects the results of the experiments or alters the potential gains/losses from using the various preprocessing schemes. The loss function measured for each esperiment is specific to the dataset used in the experiments. Since the Dogs vs Cats dataset is intended for binary classification problems, the binary crossentropy loss function is utilized for the exepriments performed on the Cats vs Dogs dataset. Since the Intel Images dataset is intended for multi-class classification, the categorical crossentropy loss function is instead utilized for experiments using the Intel Images datasets. Each experiment is run using Python 7 within Jupyter Notebook through the Anaconda Shell. At the completion of each experiment, the logged results and measurements of each experiment are aggregated into the tables and figures used in the results section of this report.

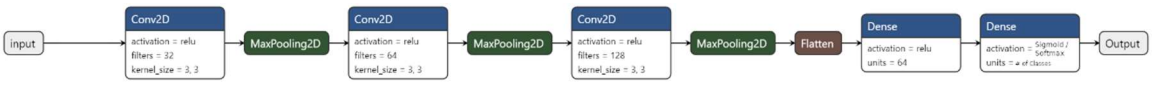

Figure 1: Custom CNN Architecture. Diagram Generated by Netron [22]

3.2 Custom Model Architecture

Figure 1 demonstrates the custom architecture used for the first CNN experiment implementations. Throughout the duration of the experiments, this architecture will be referred to as "custom model." This custom model is used to represent a shallow, generic implementation of a CNN architecture. The shape of the input is defined specifically for each specific preprocessing method and dataset used. For the Dogs & Cats dataset, the input shape is ?x400x400x3 for the no preprocessing experiment, ?x80x80x1 for the grayscale resizing experiment, ?x32x32x1 for the PCA experiments, and ?x32x32x1 for the LSH experiments. For the Intel Image Classification experiments, the input dimensions are 150x150x3 for the no preprocessing experiment, 75x75x1 for the grayscale resizing experiment, 32x32x1 for the PCA experiment, and 32x32x1 for the LSH experiment. The input shape is defined by the post-processing dimensions of the dataset, and does not affect the remaining architecture, it only affects the number of parameters that the model must learn.

After taking the input in, the model contains two convolution blocks. A convolution block is to be defined as any number of convolution (Conv2D) layers prior to a pooling (MaxPooling2D) layer. For the first convolution block, the model uses a single convolution layer. The convolution layer in the first convolution block contains 32 convolution filters with a 3x3 kernel, followed by a pooling layer with a 2x2 pooling size. The activation function used for the convolution layer in the first convolution block is ReLU. For the second convolution block, only a single convolution layer is present. For the convolution layer in the second convolution block, 128 filters are used with a 3x3 kernel, followed by

a pooling layer with a 2x2 pooling size. The activation of the convolution layer in the second convolution block is, again, the ReLU function.

After the two convolution blocks, the result is flattened from a matrix into a vector. This vector is passed through two fully-connected (Dense) layers. Which perform the classification. The first of the dense layers contains 64 neurons and performs the ReLU activation function. The final dense layer is used to generate the output. The shape of this layer and its activation is dependent on the dataset used. The Dogs vs Cats dataset is a binary classification problem, so for experiments which use the Dogs & Cats dataset, the layer contains a single neuron with the sigmoid activation function. Alternately, the Intel Images dataset is a multi-class classification problem with 6 classes, and therefore, the final layer for experiments performed on the Intel Images dataset contains 6 neurons, and utilizes the softmax activation function in order to complete the classification.

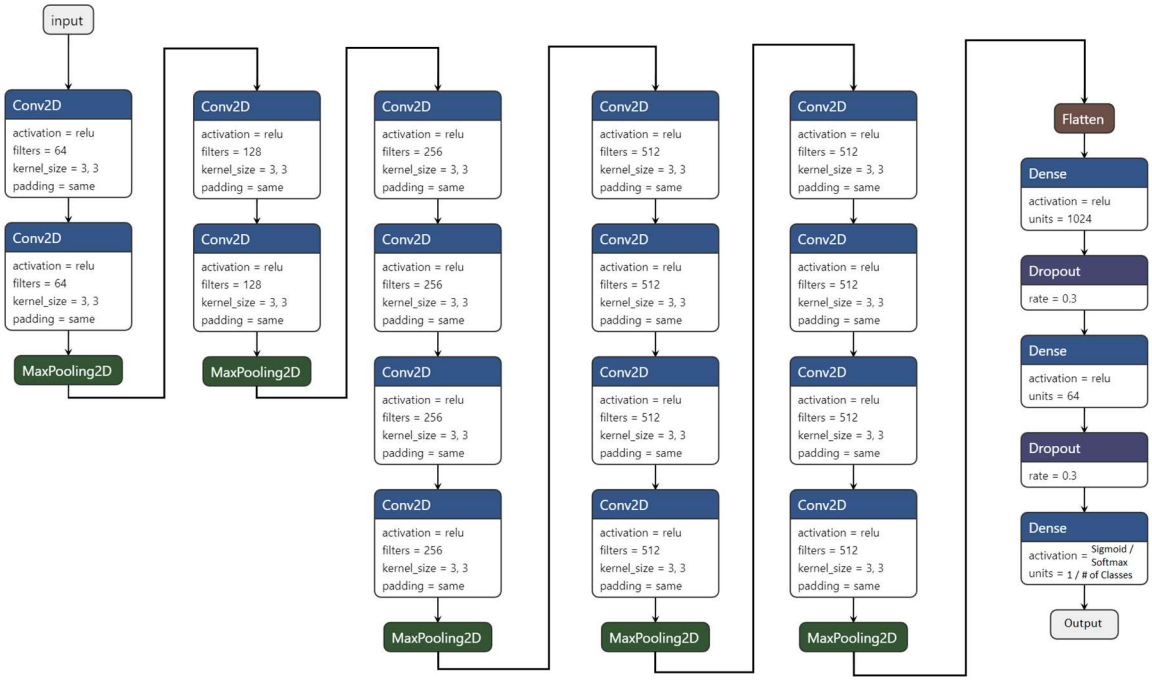

Figure 2: VGG19 Architecture. Diagram Generated by Netron [22]

3.3 VGG19 Model Architecture

Figure 2 demonstrates the CNN architecture used in the VGG19 model which was adapted for use in the relevant esperiments performed within this project. Throughout the experiments, this model architecture will be referred to as VGG19. It represents one of the more well-known, standard model architectures utilized for image classification. This model is a deep, complex network implementations which is expected to contain a high volume of parameters for which the network must learn the weights of. The shape of the input is defined specifically for each specific preprocessing method and dataset used. For the Dogs & Cats dataset, the input shape is ?x400x400x3 for the no preprocessing experiment, ?x80x80x1 for the grayscale resizing experiment, ?x32x32x1 for the PCA experiments, and ?x32x32x1 for the LSH experiments. For the Intel Image Classification experiments, the input dimensions are 150x150x3 for the no preprocessing experiment, 75x75x1 for the grayscale resizing experiment, 32x32x1 for the PCA experiment, and 32x32x1 for the LSH experiment. The input shape is defined by the post-processing dimensions of the dataset, and does not affect the remaining architecture, it only affects the number of parameters that the model must learn.

After the input layer, the neural network passes through five separate convolution blocks. A convolution block is defined the same for the VGG19 model as it was for the custom model. That is, a convolution block consists of any number of convolution layers groupded together, and the end of a convolution block is indicated by the presence of a pooling layer. The first convolution block consists of two identical convolution layers. Both convolution layers within the first block contain 64 filters with a kernel size of 3x3. For both of the convolution layers within the first block, the outputs are padded with zeros

in order to maintain the original shape of the input. Finally, both convolution layers within the first block of the network use the ReLU activation function. The end of the first convolution block is indicated by by a 2x2 pooling layer.

Like the first convolution block, the second block also contains two identical convolution layers. However, unlike the first block, both of the convolution layers within the second convolution block utilize 128 filters each, and both with a 3x3 kernel. Yet again, both of the convolution layers in the second convolution block for the VGG19 architecture perform zero-padding in order to maintain the original input shape, and perform the ReLU activation function. Like the first convolution block, the second block also finishes with the use of a 2x2 pooling layer.

Unlike the previous two convolution blocks, the third convolution block contains a total of four convolution layers. Each of the four convolution layers within this block are identical to one-another. The layers contain 256 filters with a 3x3 kernel. Each convolution within this layer performs zero-padding in order to maintain the original shape of the input. For each of the four convolution layers, the ReLU activation function is used. The third convolution block ends with a 2x2 pooling layer.

The fourth and fifth convolution blocks are identical. Like the third convolution block, both the fourth and fifth convolution blocks contain four convolution layers. Each of the four convolution layers in both the fourth and fifth convolution blocks contain 512 filters and a 3x3 kernel. All of the layers within these convolution blocks perform the ReLU activation function. Both the fourth and fifth convolution layers are terminated with the use of a 2x2 pooling layer.

After the five convolution blocks, the results are flattened from matrix representation into a single vector in order to be passed through a series of fully-connected, dense, layers. The first dense layer contains 1,024 neurons. The first dense layer performs the ReLU activation function on its results. After the first dense layer, a dropout is performed, which removes 30% of the output features from the previous layer in order to reduce the possibility of bias or overfitting to specific features since the number of features present in the network is so large at this point. After performing the dropout, the remaining features are passed into the second fully-connected layer, which contains a total of 64 neurons. Upon completing the processing of the second fully-connected layer, the ReLU activation function is applied. Yet again, after the second fully-connected layer, another 30% dropout is performed, further reducing the feature space.

After the second dropout, the results are finally fed into the last fully-connected layer to generate the output. The shape and activation of the final layer are dependent on the experiment-specific dataset being used. The Dogs & Cats dataset is intended for use in a binary classification context, so for the experiments which use the Dogs & Cats dataset, the final dense layer utilizes a single neuron with the sigmoid activation function. Alternately, the Intel Images dataset is intended for use in multi-class classification between 6 classes, so when the experiment uses the Intel Images dataset, the final dense layer utilizes 6 neurons with the softmax activation function.

4. METHODOLOGY, RESULTS AND ANALYSIS (OR SIMILAR TITLE)

Each of the experiments introduced in section 3.1 are conducted in a separate Jupyter Notebook. Each Jupyter Notebook is run through Anaconda and is used for interpreting Python 3.7. The training and use of the CNN models is enabled by the Keras library [23] with a Tensorflow backend that is accelerated by an NVIDIA RTX2070 graphics processing unit (GPU). During the runtime of each experiment, the relevant metrics for comparison are measured and stored in comma-separated value (CSV) files in order to be aggregates after all experinments reach completion. After completing each of the 32 experiments, the CSV log files are aggregated into graphs and tables which are more easily interpretable than the raw data. These generated visualizations of the data are shown in section 4.2 and analyzed in section 4.3.

4.1 Methodology

The first experiment is conducted on the Dogs vs Cats dataset using the custom CNN architecture described in section 3.2, and this experiment performs no preprocessing on the dataset. First, the list of image files for the training and test partitions of the dataset are extracted into system memory. Once in system memory, the image pixel data is read into memory as colored images (three-dimensional matrices) of shape 400x400x3. The 400x400 specifies the height and width of the images. Though the Dogs vs Cats dataset contains images of varying dimensions, analysis showed that the average dimensions of the entire dataset were 360x400. In order for the the images to be processed by the neural network, it was necessary for the images to be of the same dimensions, so for this experiment, all images are resized into squares of the larger average dimension. The x3

refers to the three color channels of the images, which are interpreted as the red-green-blue (RGB) color channels. Following the experiment design specified in section 3.1, the 400x400x3 images are used to train the neural network for 25 epochs with a learning rate calculated with proportion to the network depth and number of parameters. It was decided that his dynamically calculated learning rate would be fair for the purposes of the experiments, since each network has order of magnitude different numbers of parameters to train, it would be sensical to not use the same learning rate across all experiments. In order to calculate the learning rate, simply divide the number of layers in the networkby the number of parameters in the same network. After training, the overhead time required to inference 1, 10, 100, and 1000 images separately are measured, and the overhead time costs of each are logged for future aggregation.

The second experiment is conducted on the Dogs vs Cats dataset using the custom CNN architecture described in section 3.2, and this experiment performs the grayscale with resizing preprocessing approach. First, the list of image files for the training and test partitions of the dataset are extracted into system memory. Once in system memory, the image pixel data is read into memory as grayscale images of shape 80x80x1. The 80x80 specifies the height and width of the images. Understanding the use of 400x400 for the image height and width as justified for the first experiments, the image dimensions are resized by the largest factor of the dimensions that produces dimensions less than 100 pixels. That is, the height and width of the image are scaled to 1/5 the original dimensions to produce the 80x80 resized shape. The x1 refers to the single color channel of the grayscale images (a brightness/vibrance channel). Following the experiment design specified in section 3.1, the 80x80x1 images are used to train the neural network for 25

epochs with a learning rate calculated with proportion to the network depth and number of parameters. The justification for the selection for the learning rate value is described in the explanation of experiment one. The formula for learning rate calculation is consistent across the first 16 experiments, which perform no tuning. After training, the overhead time required to inference 1, 10, 100, and 1000 images separately are measured, and the overhead time costs of each are logged for future aggregation.

The third experiment is conducted on the Dogs vs Cats dataset using the custom CNN architecture described in section 3.2, and this experiment performs the PCA preprocessing approach on the dataset. First, the list of image files for the training and test partitions of the dataset are extracted into system memory. Once in system memory, the image pixel data is read into memory as grayscale images of shape 400x400x1. The 400x400 specifies the height and width of the images. The reasoning for these dimensions is justified in the description of experiment 1. The x1 refers to the single color channel of the grayscale images (a brightness/vibrance channel). Next, the Scikit-Learn Python library [25] is used to perform scaling on the dataset in order to ensure a standardized mean and deviation across the pixels in the image. PCA requires scaled dataset in order to effectively determine the principal components which have the greatest impact on the variance (closest to orthogonal). Finally, the Scikit-Learn Python library is again utilized to perform the PCA process on the dataset, learning the principal components, then transforming both the training and validation datasets into their reduced-dimensionality components. For the PCA experiments, 1,024 principal components are selected to be trained upon. The reasoning for using this number is because it can be reshaped into a 32x32 matrix of components, which is the smallest possible square matrix whose dimensions are a power

of two that all experimented network architectures are capable of processing. Once the components are transformed into the 32x32x1 matrices for each image, they are fed into the CNN for training. The network is trained for 25 epochs with a learning rate calculated with proportion to the network depth and number of parameters. The justification for the selection for the learning rate value is described in the explanation of experiment one. The formula for learning rate calculation is consistent across the first 16 experiments, which perform no tuning. After training, the overhead time required to inference 1, 10, 100, and 1000 images separately are measured, and the overhead time costs of each are logged for future aggregation.

The fourth experiment is conducted on the Dogs vs Cats dataset using the custom CNN architecture described in section 3.2, and this experiment performs the LSH preprocessing approach on the dataset. First, the list of image files for the training and test partitions of the dataset are extracted into system memory. Once in system memory, the image pixel data is read into memory as grayscale images of shape 400x400x1. The 400x400 specifies the height and width of the images. The reasoning for these dimensions is justified in the description of experiment 1. The x1 refers to the single color channel of the grayscale images (a brightness/vibrance channel). Next, 1024 hash functions are generated. Each hash function is specified to select 8 random pixels and report either 0 or 1 depending on whether the selected pixel has a value less than a specified threshold. Finally these eight 0s and 1s are aggregated into a single string of 8 bits, converted to a base 10 representation (0-255), and divided by 255 in order to perform a Min-Max scaling on the hashed values. The random pixels and thresholds stay constant after the function is initialized, meaning these values do not change for any future calls to the same hash

function. The reasoning for the selection of 1,024 hashing functions is in order to generate 1,024 hash strings to represent each image. This number was chosen for the same reasoning as the 1.024 principal components in the third experiment. Once the hashing function have been generated, both the training and validation partitions of the dataset are fed into each, extracting the 1,024 hash strings to represent each image. Finally, the list of hash strings for each image are reshaped into 32x32x1 matrices to be fed into the CNN for training. The network is trained for 25 epochs with a learning rate calculated with proportion to the network depth and number of parameters. The justification for the selection for the learning rate value is described in the explanation of experiment one. The formula for learning rate calculation is consistent across the first 16 experiments, which perform no tuning. After training, the overhead time required to inference 1, 10, 100, and 1000 images separately are measured, and the overhead time costs of each are logged for future aggregation.

For experiments 5, 6, 7, and 8, the same procedures as the first four experiments are utilized. The difference between these four experiments and the previous four is the model architecture being evaluated. For experiments 5-8, the VGG19 architecture, as described in section 3.3, is utilized instead of the custom model architecture. The same four preprocessing techniques (no preprocessing, grayscale with resizing, PCA, and LSH) are performed again using the same training/validation dataset (Dogs & Cats Images [6]). Similar to the previous four experiments, the same overhead time costs are measured and logged for future aggregation.

For experiments 9, 10, 11, and 12, the same four preprocessing methodologies are evaluated again. However, for these experiments, the custom CNN architecture described in section 3.2, as used in the first for experiments) is used again. The difference between

these four experiments and the first four experiments is the dataset used for training/validation in the experiment. For experiments 9 through 12, the Intel Image Classification [7] dataset is used. This alters the initial steps of the experiments, but the training and measuring processes remain unchanged. The Intel Images dataset utilizes images which are all 150x150 colored images. Because of this, the no preprocessing experiment (experiment 9) uses an input shape of 150x150x3 for its experiments instead of 400x400x3 and the grayscale with resizing experiment (experiment 10) uses an input shape of 75x75x1 instead of 80x80x1. The PCA and LSH experiments still use 32x32x1 components and hashes respectively, but the components/hashes are extracted from 150x150x1 grayscale images instead of from 400x400x1 grayscale images. As with all experiments, the measured and logged values are stored for future aggregation.

For experiments 13, 14, 15, and 16, the same four preprocessing methodologies are yet again measured. Similar to the previous four methods, the Intel Images dataset is processed for these experiments. The difference between these four experiments and the previous four experiments is the use of the VGG19 model architecture as described in section 3.3 instead of the custom model architecture. All other aspects of the experiments remain unchanged. As with the previous experiments, all overhead times are measured and logged in order to be used in aggregation in a future notebook. Now, each of the two datasets has an experiment with each of the two model architectures while perofming each of the four preprocessing methods.

Finally, experiments 17 through 32 are near-identical copies of the firs 16 experiments. The difference between the original experiments and these is the use of hyperparameter tuning of the models. In order to avoid modifying the model architectures,

only the learning rate is tuned. The Kerastuner python library [26] is used for tuning the learning rate. It is done using a hyperband approach to determine which of 10 possible learning rates results in the greatest validation accuracy after 10 epochs of training. Once the tuned learning rate is determined, the model is re-trained on the experiment-specific dataset/preprocessed input for a total of 25 epocs. These experiments are used to determine whether modifying the selection of learning rate has an impact on the performance gains/losses of using specific preprocessing methods. Finally, in addition measuring the same overhead and metrics values as the previous experiments, these experiments also measure the overhead cost of hyperparameter tuning the learning rate.

Once all 32 experiments have been conducted, a final notebook is used for aggregating the recorded data from each experiment into graphs and tables for easy data visualization and interpretation. The data visualization results are shown in the figures presented in the following section.

4.2 Results

Within this section, we will include the chats and tables of the metrics measured by the experiments conducted and their aggregated visualization as performed in the final notebook described in the previous section. Each figure presented will have a brief description of the contents of the charts/tables, but any analysis and observations beyond statements of fact will be reserved for section 4.3. Graphs were generated using the Matplotlib Python library [27], and tables are organized based on the raw results.

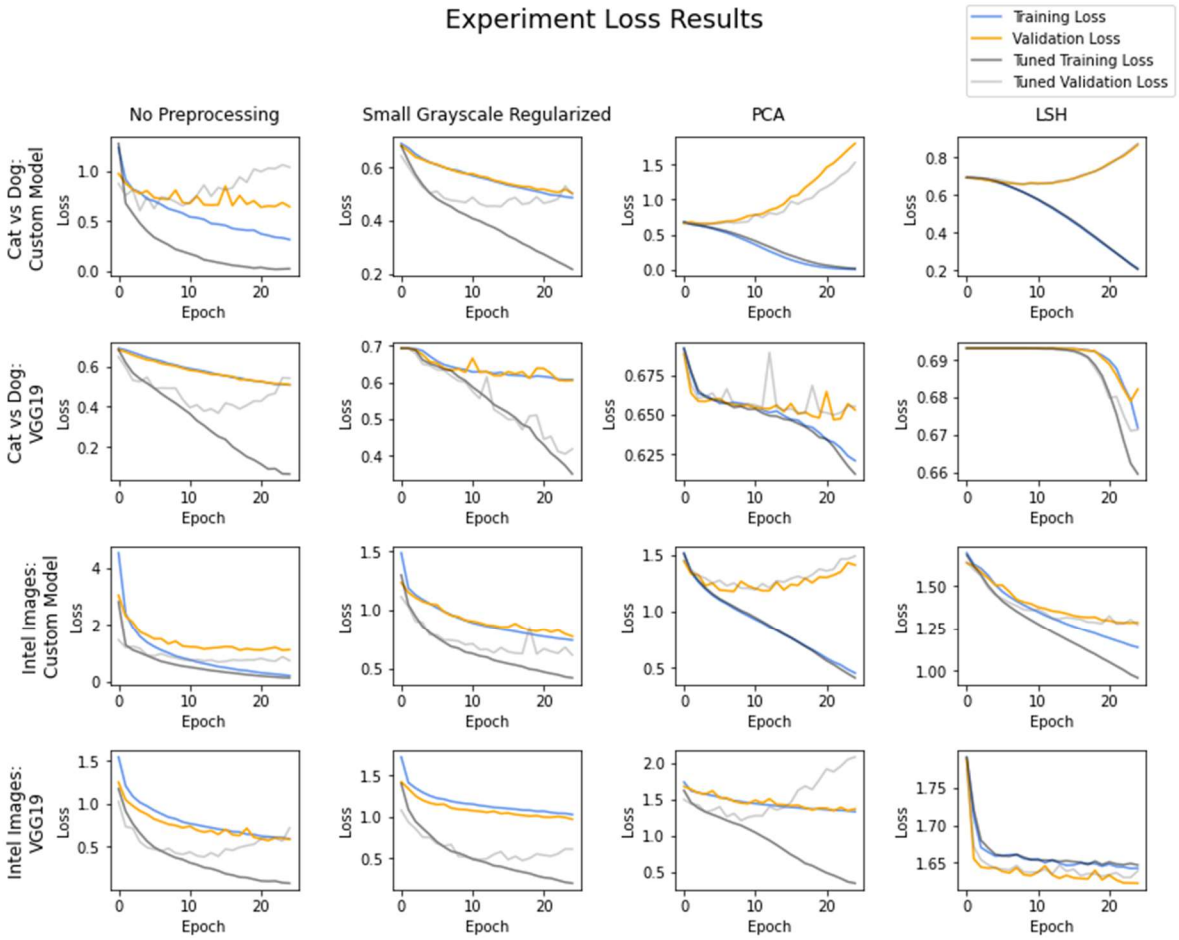

Figure 3: Visualizing Experimental Loss Behavior on Shared Axes

Figure 3 demonstrates the training loss and validation loss for each of the 32 experiments on shared axes. Each column denotes the preprocessing method used for the experiment, and each row denotes the dataset and model architecture used for the experiment. For example, the first column denotes the No Preprocessing approach, and the first row, denotes the Cats vs Dogs dataset being used to train the custom CNN architecture. Therefore, the upper-left graph in the grid contains plots of the loss functions related to the No Preprocessing experiment when performed on the Cat vs Dog dataset using the custom CNN model architecture. For each graph within the grid, the X axis denotes the epoch

number that the measurement was taken, and the Y axis denotes the loss value (binary crossentropy for the Cat vs Dog datasets or categorical crossentropy for the Intel Images datasets). Each graph on the grid contains four plots: the blue plot indicates the loss value of the training partition calculated at the end of each epoch, the orange plot indicates the loss value of the validation partition calculated at the end of each epoch, the dark gray plot indicates the loss value of the training partition for the experiment variant which performed hyperparameter tuning on the learning rate, and the light gray plot indicates the loss value of the validation partition for the experiment variant which performed hyperparameter tuning on the learning rate.

Examining the first row of graphs in the grid, the following observations can be made. In the No Preprocessing column, the training partition's loss converged for both the original and the tuned experiments, the validation partition's loss converged for the origina experiment, but the validation partition's loss on the tuned experiment begins to diverge after approximately 10 epochs. For the Small Grayscale Regularized column, identical convergence behavior to the previous column is observed, with different steepness of the slopes, the loss of the validation partition of the tuned experiment diverges beginning around approximately the 10^{th} epoch, and the loss of the validation partition for the original experiment more closely adheres to the values of the training partition for the same experiment. When observing the PCA column, both losses for the training partitions converge to near-zero, whereas the validation partitions' losses diverge for the duration of the experiment beginning from the initial epoch. When observing the LSH column, nearly identical behavior to the PCA column is observed in terms of plot behavior, but the final values of the plots vary from PCA in that the training partition losses did not converge fully

to zero, and the validation partition losses did not diverge to as extreme of values as the PCA experiments for this configuration.

Examining the second row of graphs in the grid, the following observations can be made. In the No Preprocessing and Small Grayscale Regularized columns, a nearly identical behavior compared to the first row's implementations of the same preprocessing methods can be observed: all of the loss functions converge with the exception of the validation partition loss of the tuned variant, which begins to diverge at approximately the tenth epoch. When observing the PCA column, both losses for the training partitions converge with the shape of a negative cubic function for the entirety of the 25 epochs of training, whereas the validation partitions' losses neighter converge nor diverge, remaining at approximately the same loss value (ignoring occasional spikes which return in the following epoch) for the entirety of the training duration. This behavior differs from the PCA experiments observed in the previous row. When observing the LSH column, all loss values remain near-constant for the first 15-20 epochs before all losses begin to converge. At the final epoch, the losses of both validation partitions begin an upward divergence from the losses of the training partitions. This behavior differs from that observed in the previous row's experiments.

Examining the third row of graphs in the grid, the following observations can be made. In the No Preprocessing and Small Grayscale Regularized columns, a nearly identical behavior compared to the first two rows' implementations of the same preprocessing methods can be observed: all of the loss functions converge with the exception of the validation partition loss of the tuned variant, which begins to diverge at approximately the tenth epoch. When observing the PCA column, the behavior of the losses

is nearly identical to that of the first row's PCA experiments. That is, the losses of both training partitions converge toward zero while validation partition losses diverge beginning at the first epoch. When observing the LSH column, a trend more similar to the No Preprocessing and Small Grayscale Regularized columns than to the previous LSH experiments is observed. That is, all four losses converge for the duration of the 25 epochs wit the exception of the validation partition's loss on the tuned variant of the experiment, which begins to diverge at approximately the tenth epoch.

Finally, examining the last row of graphs in the grid, the following observations can be made. In the No Preprocessing and Small Grayscale Regularized columns, a nearly identical behavior compared to all three prior rows' implementations of the same preprocessing methods can be observed: all of the loss functions converge with the exception of the validation partition loss of the tuned variant, which begins to diverge at approximately the tenth epoch. When observing the PCA column, the behavior of the losses more closely resembles the behavior of the No Preprocessing and Small Grayscale Regularized methods. That is, the losses of all four experiments converge, but starting at approximately the tenth epoch, the loss of the validation partition for the tuned experiment variant begins to diverge. When observing the LSH column, all four losses converge within the first epoch. The resulting converged loss remains steady for all four plots throughout the duration of the 25 epochs. This behavior differs from all other observations across all other experiments.

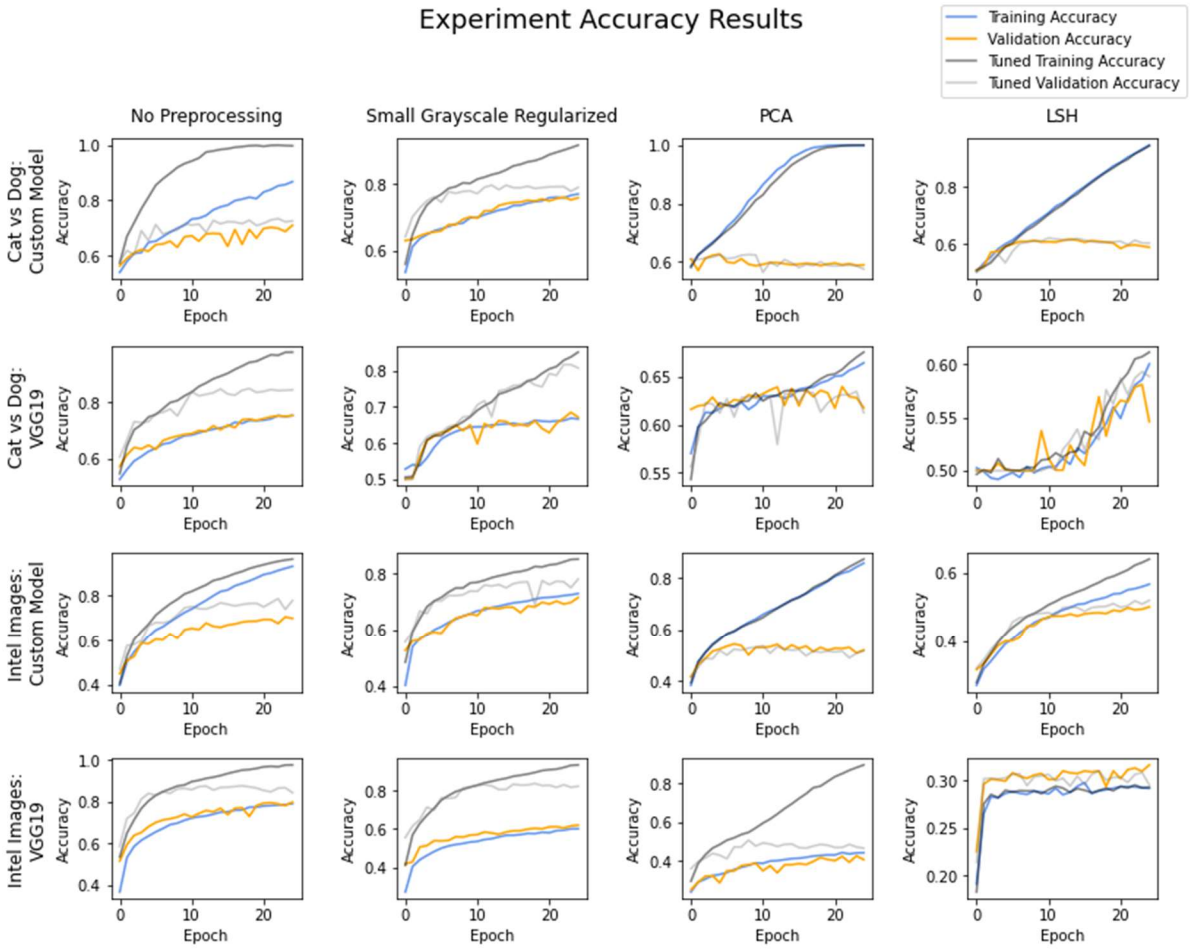

Figure 4: Visualizing Experimental Accuracy Behavior on Shared Axes

Figure 4 demonstrates the training accuracy and validation accuracy for each of the 32 experiments on shared axes. Each column denotes the preprocessing method used for the experiment, and each row denotes the dataset and model architecture used for the experiment. For example, the first column denotes the No Preprocessing approach, and the first row, denotes the Cats vs Dogs dataset being used to train the custom CNN architecture. Therefore, the upper-left graph in the grid contains plots of the accuracy functions related to the No Preprocessing experiment when performed on the Cat vs Dog dataset using the custom CNN model architecture. For each graph within the grid, the X axis denotes the

epoch number that the measurement was taken, and the Y axis denotes the accuracy value (0 being completely incorrect, and 1 being perfectly correct). Each graph on the grid contains four plots: the blue plot indicates the accuracy value of the training partition calculated at the end of each epoch, the orange plot indicates the accuracy value of the validation partition calculated at the end of each epoch, the dark gray plot indicates the accuracy value of the training partition for the experiment variant which performed hyperparameter tuning on the learning rate, and the light gray plot indicates the accuracy value of the validation partition for the experiment variant which performed hyperparameter tuning on the learning rate.

Examining the first row of graphs in the grid, the following observations can be made. In the No Preprocessing column, the training partition's accuracy converged to nearly 1 for the tuned variant of the experiment, the training partition's accuracy for the original experiment was in the process of a near-linear convergence of the duration of the 25 epochs, and accuracies of both validation partitions converged to approximately 0.70, which remained stable for the 25 epochs of training. For the Small Grayscale Regularized column, the accuracies for both the training and validation partitions of the original experiments converge near-linear in behavior for the duration of the experiments, whereas the tuned variants yield a rapid convergence of both accuracies for the first five epochs, followed by a linear convergence for the training partition's accuracy and a stable accuracy for the validation partition's accuracy. When observing the PCA column, both accuracies of the training partitions converge to nearly one by the 15^{th} epoch, whereas the accuracies of both validation partitions remain stable for all 25 epochs. When observing the LSH column, both accuracies of the training partitions converge toward 1 at a linear rate for the

duration of the 25 epochs, whereas the accuracies of the validation partitions converge for approximately the first five epochs before becoming stable.

Examining the second row of graphs in the grid, the following observations can be made. In the No Preprocessing column, the training partition's accuracy converged to nearly 1 for the tuned variant of the experiment, the training and validation partitions' accuracies for the original experiment converged with similar behavior, but toward a lower final value, and the validation partition's accuracy of the model converged to approximately 0.8 within the first 10 epochs before stabilizing. For the Small Grayscale Regularized column, the accuracies for both the training and validation partitions of the original experiments converge drastically for the first few epochs before stabilizing for the remainder of the training, whereas the accuracies of the tuned variants of the experiments converge dramatically for the first few epochs before a near-linear convergence for the remainder of the training. When observing the PCA column, both accuracies of the training partitions follow a cubic convergence for the duration of the training, whereas the accuracies of the validation partitions remain approximately stable (with the exception of jitters) for the entirety of training. This observed behavior differs from the behavior of the PCA experiments observed in the first row. When observing the LSH column, all four accuracies adhere to a quadratic or exponential training curve, which contains a slowly increasing slope for the duration of the training. This behavior differs from the LSH behavior observed in the first row.

Examining the third row of graphs in the grid, the following observations can be made. In the No Preprocessing columns, the observed behavior is nearly identical to the behaviro of the No Preprocessing experiments demonstrated in the first row, with the

exception of training partition's accuracy in the original experiment more closely adhering to the trend of the tuned variant's training partition's accuracy. For the Small Grayscale Regularized column, the observed behavior is nearly identical to that of the Small Grayscale Regularized results demonstrated in the first row. When observing the PCA column, a behavior similar to the first row's LSH accuracy behavior is observed. That is, both training parition's accuracies converge in a near-linear pattern, whereas both validation accuracies converge for the first five epochs before stabilizing. When observing the LSH column, a pattern more similar to the No Preprocessing column of this same row is observed. That is, each of the accuracies convergefor the duration of the training, but both validation partition accuracies begin to stabilize after approximately 10 epochs. For the LSH variant, the values which the accuracies converge to are much lower than that of the no preprocessing variants. This behavior for LSH accuracy is unique across all LSH experiments conducted.

When examining the last row of graphs in the grid, the following observations can be made. The No Preprocessing column shows near-identical behavior to the behavior of the second row in the same figure. That is, the original experiment accuracies converge much slower than the accuracies of the tuned variants, and to lower final values after the 25 epochs of training. For the Small Grayscale Regularized column of this row, the observed behavior is nearly identical to that of the first and third rows of the same column. That is, the tuned partitions converge much more rapidly and to much larger values than that of the original experiments, however, all accuracies converge for the duration of the experiment, with the exception of the tuned variant's validation partition accuracy, which stabilizes after approximately 10 epochs. When observing the PCA column of the final

row, both accuracies of the original experiments converge to approximately the same value, the validation parition's accuracy for the tuned variant converges to this value in the first few epochs before stabilizing, whereas the training parition's accuracy converges near-linearly to a much higher value than the other accuracies. When observing the LSH column for the same row, all four accuracies converge after the first epoch. At which point, the accuracies become stable for the remainder of the training.

Dataset	Model	Method	Max Training Accuracy	Max Validation Accuracy	Min Training Loss	Min Validation Loss	Max Tuned Training Accuracy	Max Tuned Validation Accuracy	Min Tuned Training Loss	Min Tuned Validation Loss
Cat vs Dog	Custom	No Preprocessing	0.867875	0.710500	0.313935	0.638147	0.999625	0.734500	0.015159	0.605599
Cat vs Dog	Custom	Small Gray Regularized	0.770125	0.761000	0.486085	0.502933	0.916500	0.796500	0.216351	0.452856
Cat vs Dog	Custom	PCA	1.000000	0.626000	0.006917	0.656489	1.000000	0.624500	0.018837	0.652192
Cat vs Dog	Custom	LSH	0.948125	0.616500	0.210245	0.657620	0.946000	0.621000	0.205796	0.658582
Cat vs Dog	VGG19	No Preprocessing	0.753125	0.753500	0.507873	0.509692	0.977125	0.848500	0.065436	0.368069
Cat vs Dog	VGG19	Small Gray Regularized	0.668375	0.684500	0.607305	0.603955	0.850500	0.816500	0.349252	0.404243
Cat vs Dog	VGG19	PCA	0.664125	0.639500	0.620835	0.646857	0.675250	0.634500	0.612338	0.649893
Cat vs Dog	VGG19	LSH	0.600250	0.581000	0.671911	0.678976	0.611375	0.593000	0.659489	0.671043
Intel Images	Custom	No Preprocessing	0.930953	0.705333	0.215904	1.102662	0.963161	0.787333	0.139609	0.687411
Intel Images	Custom	Small Gray Regularized	0.729372	0.715333	0.739184	0.771263	0.851076	0.780667	0.420022	0.613120
Intel Images	Custom	PCA	0.858415	0.542667	0.455484	1.178614	0.874163	0.536000	0.413564	1.192485
Intel Images	Custom	LSH	0.566196	0.500333	1.136454	1.282292	0.638948	0.519333	0.958207	1.272556
Intel Images	VGG19	No Preprocessing	0.790366	0.797333	0.590207	0.571217	0.975987	0.876000	0.074297	0.379473
Intel Images	VGG19	Small Gray Regularized	0.600470	0.619667	1.024908	0.968209	0.932735	0.836000	0.197984	0.472184
Intel Images	VGG19	PCA	0.441927	0.424667	1.328807	1.340595	0.894328	0.506333	0.345684	1.206905
Intel Images	VGG19	LSH	0.297349	0.316000	1.642660	1.622854	0.294927	0.309667	1.647169	1.630612

Figure 5: Table of Peak Experimental Accuracy/Loss Values

Figure 5 depicts a table containing the minim loss values for each experiment and the maximum accuracy values for each experiment – both in terms of the training partition and the validation partition of each experiment, as well as the variants of the experiments which perform hyperparameter tuning on the dataset. Each row of the table corresponds to a single experiment and its hyperparameter tuned variant.

The "Dataset" column of the dable indicates which of the two experiment datasets that the experiment corresponding to the table row utilizes. If this column contains a "Cat vs Dog," then the Dogs & Cats dataset is used, and if the column contains "Intel Images," then the Intel Image Classification dataset is used. The "Model" column indicates which of the two CNN model architectures is levereaged by the experiment in the corresponding row. If this column contains the value "Custom," then the custom CNN architecture described in section 3.2 is used, and if the column contains the value "VGG19," then the VGG19 CNN architecture described in section 3.3 is used instead. The "Method" column indicates which of the four preprocessing methods that the corresponding row's experiments utilize: No Preprocessing, Small Grayscale Regularization, PCA, or LSH. Finally all remaining columns in the table contain either the minimum ("Min") or maximum ("Max") loss/accuracy values for both the original experiment and for the variant which performed hyperparameter tuning using the same configurations. The values examined in order to generate this table are the same values used to generate the plots for Figures 3 and 4. This table is used to indicate the ideal values which the models could have converged to during the training process if intelligent stopping was utilized instead of a fixed 25 epoch training period. The rows of the table are first organized by the dataset used, and further organized within the dataset groups by the model architecture used.

Dataset	Model	Method	Load/Process Dataset (Sec)	Method Preprocessing Overhead (Sec)	Tuning Time (Sec)	Training (Sec)	Total Time (Sec)
Cat vs Dog	Custom	No Preprocessing	19.26	0.00	551.18	756.54	1326.97
Cat vs Dog	Custom	Small Gray Regularized	11.50	0.00	47.20	47.70	106.40
Cat vs Dog	Custom	PCA	13.34	98.55	40.99	41.85	194.73
Cat vs Dog	Custom	LSH	12.40	207.92	39.72	38.88	298.93
Cat vs Dog	VGG19	No Preprocessing	19.54	0.00	6361.85	7897.04	14278.43
Cat vs Dog	VGG19	Small Gray Regularized	11.20	0.00	485.06	585.24	1081.50
Cat vs Dog	VGG19	PCA	13.93	106.41	267.32	311.65	699.31
Cat vs Dog	VGG19	LSH	12.41	207.87	205.57	305.68	731.52
Intel Images	Custom	No Preprocessing	11.83	196.71	202.85	241.10	652.49
Intel Images	Custom	Small Gray Regularized	83.73	0.00	85.31	97.89	266.93
Intel Images	Custom	PCA	12.28	32.54	65.82	70.91	181.56
Intel Images	Custom	LSH	11.86	364.02	64.51	69.15	509.54
Intel Images	VGG19	No Preprocessing	11.57	0.00	1930.20	2389.45	4331.23
Intel Images	VGG19	Small Gray Regularized	11.47	0.00	805.66	988.38	1805.51
Intel Images	VGG19	PCA	11.32	31.75	443.11	533.08	1019.26
Intel Images	VGG19	LSH	11.32	350.47	438.06	526.40	1326.24

Figure 6: Table of Training Overhead Time Costs

Figure 6 depicts a table containing the overhead time cost in seconds for the various stages of the model processing and training times for each of the experiments run. Each row of the table corresponds to a single experiment. For the sake of brevity, only the running times for the experiment variants which perform hyperparameter tuning are reported. The reasoning for this decision is because the overall running time of the tuned variant should require the same overhead time costs as the non-tuned variant, and the tuning time is reported as a separate measurement. This decision allows the reporting of the

overhead times for each configuration without the need to include 16 redundant rows for the non-tuned variants of each experiment.

The “Dataset” column of the dable indicates which of the two experiment datasets that the experiment corresponding to the table row utilizes. If this column contains a “Cat vs Dog,” then the Dogs & Cats dataset is used, and if the column contains “Intel Images,” then the Intel Image Classification dataset is used. The “Model” column indicates which of the two CNN model architectures is levereaged by the experiment in the corresponding row. If this column contains the value “Custom,” then the custom CNN architecture described in section 3.2 is used, and if the column contains the value “VGG19,” then the VGG19 CNN architecture described in section 3.3 is used instead. The “Method” column indicates which of the four preprocessing methods that the corresponding row’s experiments utilize: No Preprocessing, Small Grayscale Regularization, PCA, or LSH. The “Load/Process Dataset” column contains the time cost in seconds for reading the image pixel data into memory with the proper color/dimensions (this time includes potential resizing and potential conversion to grayscale in the overhead, as these transformations are performed as the data is read into system memory). The “Method Preprocessing Overhead” column contains the time cost in seconds for preprocessing the dataset based on the methods specified. For the No Preprocessing and the Small Grayscale Regularized methods, this value is 0 because all processing was performed as the image was loaded into memory. For the PCA experiments, this time cost includes the overhead for scaling the dataset, extracting the principal components, and transforming the dataset into the 32x32 principal component matrices. For the LSH experiments, this time cost includes the overhead for generating the hash functions, hashing each image, and reshaping the dataset

into the 32x32 matrix of hashes. The “Tuning Time” column contains the overhead time cost for performing hyperparameter tuning on the dataset for optimizing learning rate, where a hyperband approach attempts to maximize validation accuracy after 10 epochs with early stopping mechanisms in place, this time is the entirety of the tuning method, recorded at the moment that the optimal learning rate is determined. The “Training” column contains the overhead time cost for training the model using the tuned learning rate for exactly 25 epochs. Finally The “Total Time” column contains the aggregate overhead for the entire process. That is, it contains the sum-cost of all other columns within the table for the corresponding row.

Dataset	Model	Method	1x Inference (Sec)	10x Inference (Sec)	100x Inference (Sec)	1,000x Inference (Sec)
Cat vs Dog	Custom	No Preprocessing	0.207219	0.284083	0.988960	3.135347
Cat vs Dog	Custom	Small Gray Regularized	0.124565	0.099055	0.286398	1.295611
Cat vs Dog	Custom	PCA	0.316769	0.293242	0.632698	3.067829
Cat vs Dogs	Custom	LSH	0.143059	0.307734	2.359649	22.186182
Cat vs Dog	VGG19	No Preprocessing	0.886392	1.917221	6.097037	14.410130
Cat vs Dog	VGG19	Small Gray Regularized	0.414690	0.390347	0.989822	2.052068
Cat vs Dog	VGG19	PCA	0.583645	0.479919	1.055694	3.614403
Cat vs Dog	VGG19	LSH	0.396659	0.505955	2.815671	23.080500
Intel Images	Custom	No Preprocessing	0.131967	0.119255	0.329906	0.818770
Intel Images	Custom	Small Gray Regularized	0.118294	0.092615	0.234790	0.760401
Intel Images	Custom	PCA	0.149803	0.122615	0.269565	0.972127
Intel Images	Custom	LSH	0.143431	0.309200	2.406225	21.392704
Intel Images	VGG19	No Preprocessing	0.584501	0.756817	1.984969	2.506454
Intel Images	VGG19	Small Gray Regularized	0.414227	0.370429	0.939264	1.280994
Intel Images	VGG19	PCA	0.399005	0.323580	0.703694	1.254572
Intel Images	VGG19	LSH	0.373771	0.511416	2.725614	21.721442

Figure 7: Table of Inference Time Costs

Figure 7 depicts a table containing the overhead time cost in seconds in order to perform inference using the trained model. In these trials, a random set of 1, 10, 100, or 1000 images are selected from the list of files for the validation partition, those images are read into memory, the necessary preprocessing is performed, and the model is used to perform inference on those images. The overall time for the entire process is measured and reported in this table. The intent for these measurements is to demonstrate the relative usability and scalability in real-use post training environments. The training time for a model is not necessarily as important as the time-cost of using the model after training, therefore, these results are all recorded presented for analysis in section 4.3.

The “Dataset” column of the dable indicates which of the two experiment datasets that the experiment corresponding to the table row utilizes. If this column contains a “Cat vs Dog,” then the Dogs & Cats dataset is used, and if the column contains “Intel Images,” then the Intel Image Classification dataset is used. The “Model” column indicates which of the two CNN model architectures is levereaged by the experiment in the corresponding row. If this column contains the value “Custom,” then the custom CNN architecture described in section 3.2 is used, and if the column contains the value “VGG19,” then the VGG19 CNN architecture described in section 3.3 is used instead. The “Method” column indicates which of the four preprocessing methods that the corresponding row’s experiments utilize: No Preprocessing, Small Grayscale Regularization, PCA, or LSH. The remaining columns each correspond to the inference time cost for an experiment which perorms a batch inference of the specified size (1x for a single inference, 10x for a batch of 10, 100x for a batch of 100, and 1000x for a batch of 1000 inferences performed at once.

4.3 Analysis

Evaluating the results of figures 3 and 4 show that for the No Preprocessing and the Small Grayscale Regularized methods, the shapes of the accuracy and validation cuves for both methods do not change across the various experiment configurations. In both of these processing methods, the behavior is constant across all four rows for these two columns, indicating that the use of a resized grayscale image as input to the model has no affect on the training behavior of the model, and these results are both dataset independent and model architecture independent. When examining the peak values for validation loss/accuracy as shown in Figure 5, varying behavior emerges. For only the Cat vs Dog dataset with the custom model, the Small Grayscale Reglularized approach achieved greater tuned validation accuracy than the No Preprocessing variant. For all other configurations, the accuracy of the Small Grayscale Regularized approach achieved between 0.007 and 0.04 less accuracy than the No Preprocessing variant. This behavior indicates that with more rhigorous tuning, it is possible for the models to perform equivalently regardless of whether the dataset is in its original form or whether the dataset is resized and converted to grayscale. This behavior is both dataset independent and model architecture independent. Additionally, the Small Grayscale Regularized approach took a fraction of the overhead time for training compared to the No Preprocessing approach (less than $1/10^{th}$ the total time for the Cat vs Dog dataset, and less than half the time for the Intel Images dataset), as well as performing individual and batch inference faster than all other approaches. This allows the conclusion to be drawn that every image classification project that utilizes CNNs will benefit from converting the dataset to grayscale and resizing the image to smaller

dimensions, regardless of the dataset used and regardless of the model architecture used (both for binary and multi-class classification problems).

Analyzing the PCA performance yields different conclusions. The loss behavior show in Figure 3 for PCA diverged in both of the custom model implementations, but the PCA approach had very slight convergence when used on the VGG19 implementations. Showing the behavior of the loss for the PCA implementations was dataset independent, but dependent on model architecture. However, the accuracy behavior shown in Figure 4 for PCA shows that almost no learning is performed beyond the initial epoch, and only overfitting occurs, which the model is not able to extrapolate beyond the converged validation values for each configuration. It is hypothesized that part of the model's difficulty in learning is due to the use of a convolutional neural network, which is attempting to find patterns in the arrangement of the principal components rather than the values themselves. Additionally, the use of more than 1,000 principal components is not standard practice for use of PCA in machine learning contexts; it is expected that the minimum number of components necessary to to encapsulate the target percentage of explained variance should be extracted (such as maintaining 95 percent of the information rather than a predetermined number of parameters. Though its use cannot be ruled out for other experiments, it can be concluded that the use of PCA, is not suited for use in CNN image classification tasks, as the accuracy does not converge to comparable values as the previously examined approaches. This result was disappointing when considering the training overhead time costs for PCA, which achieved the lowest total time for three of the four experiment configurations – only performing second fastest in the Cat vs Dog dataset with the custom CNN model. These training performance gains, however did not translate

into real-use gains, as the overhead for extracting the principal components outweight the gains in inference time, resulting in less performance than the small grayscale regularized approach. These time cost results and the performance results for the PCA implementation allow us to definitively conclude that PCA is not suitable for use in CNN image classification problems.

Finally, the results for the LSH implementations yielded interesting conclusions. In Figure 3, the loss functions for the various LSH experiments performed differently across all four configurations, showing both a dataset dependence and a model architecture dependence on the behavior of the training and validation losses, both with and without hyperparameter tuning. More interestingly, the LSH losses for the three implementations which did not diverge (all except Cat vs Dog dataset with custom CNN architecture) achieved validation loss values which were within 15% of the losses of either the No Preprocessing or the Small Grayscale Regularized implementation for three of the experiments conducted prior to hyperparameter tuning (the results were farther without tuning). These results indicate that more thorough tuning better optimized for LSH preprocessing may see performances equivalent to the results expected with No Preprocessing. When observing the reported accuracy functions of Figure 4, it is noticed that yet again, the behavior of the accuracy is both application dependent and model architecture dependent. However, in each of the observed behaviors, a convergence is demonstrated, showing that CNNs are capable of learning from LSH processed data. In th Cat vs Dogs Custom Model experiment, the accuracy was only able to converge to a peak of 0.62 after tuning, which is more than 0.10 less than the no preprocessing variant for the same configuration. Since the accuracy was stable throughout the training for this

configuration both with and without tuning, it can be concluded that for this configuration, though LSH hashes can be used to converge, it is not as effective for learning as an unprocessed input. For the Intel Images VGG19 experiments, similar behavior is shown, with the model converging to a stable validation accuracy of only 0.31 after tuning. These results were less than half of the accuracy achieved by the non-tuned Small Grayscale Regularized implementation of the same configuration. These results allow us to conclude that LSH is also not well-suited for learning in this configurations. However, in both the Cat vs Dog VGG19 experiment and the Intel Images Custom Model experiment, the LSH implementation was still in the process of converging by the end of the 25^{th} epoch, indicating the potential for even further gains beyond those reported here. Though the accuracies are lower than those achieved in the other implementations, since convergence was still underway, it is hypothesized that more refined tuning in these configurations could allow LSH to perform on-par with the No Preprocessing and Small Grayscale Regularized methods if allowed to tune more specifically for the use of application. Due to the dependence on both the dataset used and the model architecture, the relative performances of LSH are mostly inconclusive – performance is objectively worse than the other experiments performed, but the presence of convergence shows the potential for future experimentation to be made with more rigorous tuning. In order to draw a more direct conclusion. The overhead time cost and the inference time draw more definitive conclusions, however. When observing Figures 6 and 7, it is noticed that LSH required the largest preprocesing overhead of all experiments conducted, and a training time equivalent to that of PCA typically resulting in either the second or third fastest training time for each of the four configurations. However, since the preprocessing was the largest overhead cost,

the inference time for LSH showed interesting behavior – it was the fastest at performing a single inference (a result consistent across all configurations except the Cat vs Dog Custom model Small Grayscale Regularized implementation which could perform single inference 0.02 seconds faster for the same configurations). However, the inference performance of LSH does not scale, achieving consistently the worst when the batch size was 1000x. Because of this, even though the model accuracy and loss performances have the potential of being tuend to perform equivalently to the other methods, the overhead cost of LSH preprocessing makes the use of LSH not feasible for most CNN image classification problems unless single input inference time is more valuable than classification accuracy/loss, in which case, the use of LSH may be feasible to achieve mildly higher performances. Another note to make about this conclusion is the choice to use 1024 different 8-bit hashes for the hashing method. The standard use of the LSH algorithm used is intended for generating a small batch (10-20) of longer hash strings (16-24 bits). This leaves the potential for future experimentations with respect to LSH as described in greater detail in section 5.3.

The final conclusion that can be drawn from analyzing the results of the experimentations is that converting image data to grayscale, resizing it to smaller dimensions, and regularizing the data achieves loss and accuracy performance equivalent to that of performing no preprocessing at all, while also achieving significant reductions in both training and inference time, allowing the conclusion that transforming input image data into small, grayscale, regularized variants of the original data is not only preferred to no preprocessing at all, but over all preprocessing methods experimented. These concluded results remain true independent of the image dataset used, the model architecture being

used, and the type of classification being performed (binary or multi-class). It is concluded that these results allow the conclusion that this preprocessing method is the objectively best method of all methods examined in the conducted experiments.

5. CONCLUSIONS

5.1 Summary

Throughout the 32 experiments performed, definitive conclusions can be drawn about the objective, relative performance of three preprocessing techniques against the avoidance of any preprocessing at all. Experiments conducted were able to determine which cases held a dataset or model architecture dependence, and which preprocessing approaches had an independence in these regards. For each of these experiments, the overhead time costs for both training and for use of the trained model were also measured in order to determine which preprocessing method was capable of performing with objectively the lowest overhead in most scenarios.

The overall results described in the above analysis allowed the following conclusions to be drawn for the use of the grayscale with resizing preprocessing approach. Using this preprocessing approach did not affect the overall behavior of the model during training, both in terms of accuracy and loss function trends and values. The use of this preprocessing method was capable of achieving comparable performance to the use of no preprocessing independent of the dataset used and the model architecture used. The use of this preprocessing methods achieved significantly lower overhead time costs when training and when used for inference. The use of a grayscale resizing is objectively preferred to no preprocessing for use in classification when using CNNs to perform the classification.

The overall results described in the above analysis allowed the following conclusion to be drawn for the use of PCA preprocessing. The use of PCA achieved the same loss and accuracy behavior independent of both the dataset used and the model architecture used. In all cases, the PCA preprocessing method did not allow the model to

perform learning the principal components, with no validation convergence. The preprocessing overhead during the training step for PCA is a significantly large amount of time relative to the overall training time, but the inference use time of this preprocessing technique achieved the lowest overhead time in nearly all trials. The conclusion is that the use of PCA may be sufficient in other machine learning methods thanks to the acceleration of inference time, but does not provide meaningful learning when used in the context of CNN-enabled image classification.

The overall results described in the above analysis made drawing conclusions about the LSH preprocessing method to be difficult. The use of LSH preprocessing created results which were both dependent upon dataset and dependent upon the model architecture used. In nearly all experiments performed, the CNN demonstrated the capability of learning from the generated hashes, allowing the loss and accuracy metrics to be capable of converging. The use of LSH preprocessing gained the largest preprocessing overhead time for each experiment conducted, but training time was equivalent to that of PCA. The inference overhead time cost for LSH preprocessing was one of the lowest of all experiments perfomed when the number of inferences performed in a single batch was small. The conclusion is that an LSH approach may be capable of performing sufficiently well in these contexts, but was incapable of achieving comparable loss and accuracy compared to the non-tuned variants of the no preprocessing and the grayscale with resizing preprocessing approaches. This allows us to conclude that despite the potential ability to achieve better with more specialized tuning and training, it requires a more involved development (more care is required for the tuning process) than the other preprocessing approaches, allowing the conclusion that LSH is objectively worse that performing no preprocessing and

grayscale with resizing preprocessing for use in image classification via CNNs. The ability for the LSH approach to learn from the hashes allows us to also conclude that the use of LSH preprocessing is objectively better than the PCA preprocessing approach in the context of using CNNs for image classification.

To aggregate all of the conclusions made above, it is determined that using grayscale with resizing is objectively the best image preprocessing approach to be used for image classification, achieving comparable performance to not using any preprocessing, while requiring less training/inference overhead. The LSH approach was the objective second-best preprocessing approach of the three examined in this project, showing the capability of learning, but not achieving accuracy or loss levels comparable to no preprocessing or small grayscale regularized. Finally, the PCA preprocessing implementation resulted in the CNN models being unable to learn from the principal components, allowing the conclusion that not only was PCA the worst of the evaluated preprocessing approaches for use in CNN image classification, but that PCA is not suitable for the use of any CNN image classification problem.

5.2 Contributions

When training a neural network, one of the most important design decisions that the developer must make is the preprocessing methodology to use on the dataset prior to training and evaluations. When working with high-dimensional data, such as images, the need for preprocessing to reduce the dimensions of the input becomes significant. In an ideal scenario, a preprocessing methodology could significantly reduce the number of features which the model must learn and improve the overhead time of training and inferencing without impacting the loss or accuracy of the model. Within this project, it was

objectively demonstrated that the use of resizing a grayscale with resizing preprocessing approach is capable of achieving these idealized goals for preprocessing. This preprocessing method was the best of those evaluated in this project, and the use of it is strongly recommended over no preprocessing. The conclusions of this project make the decision of preprocessing methodologies to use an easy decision for developers who chose to follow the conclusions of this paper.

The source code of the project, the datasets used, all generated results, and all trained models are made publicly accessible from a public Google Drive [28], or with the Jupyter Notebooks viewable from GitHub [29]. This allows anyone willing to conduct the experiments themselves to either verify the results or expand upon them. Providing this available resource allows these conclusions to be verified by others without being taken at face-value. Allowing open access facilitates the opportunity for future works as described in more detail in section 5.3.

5.3 Future Work

While this project report provides a high volume of granular experimentations to determine the performance of various preprocessing techniques within the context of image classification, there exist many opportunities for future expansions and works which build upon the experiments performed in order to generate further refied conclusions for the problem being evaluated.

Only three separate preprocessing mechanisms were consizered as representatives of broader categories of feature pruning methodologies. For example, only a single LSH algorithm was selected of the many available. There exists the opportunity to expand upon the work in this project by conducting additional experiments on other preprocessing

algorithms outside those evaluated. Such examples include other LSH algorithms, dropping features which have little impact on classification results, and so on. The inclusion of more methods experimented upon, while verifying the dataset and model architecture independence would help provide a broader understanding of the overall behavior of preprocessing methodologies, and increase the confidence in the conclusions drawn.

For both LSH and PCA experiments, the number of features remaining after preprocessing were significantly higher thant he number of features which these preprocessing techniques are intended to produce. In both cases, beyond 1,000 features is considered irresponsibly large. Future experiments open the potential for considering smaller CNN models which are able to train better on feature sets that these methods excel at producing (for example, only 16-36 features with sufficiently small CNNs which can handle such small input shapes). Such experiments may yield sufficiently better or worse results than those produced within this experiment, and would be worth exploring in future expansions.

The hash strings produced in the LSH preprocessing approaches each considered only 8 features from the original image for the processing. Under normal circumstances, the hashes would not be so short for images of the size used in the experiments. Future works could explore the impact of increasing the length of the generated hashes by the LSH preprocessing approach to determine whether increasing the length of the hash strings produces better or worse performance in terms of learning.

All experimetns performed were conducted on convolutional neural networks for the use of classification. Such networks are common when performing image classification,

however, the use of other types of networks are also possible in the scope of image classification. Future works could introduce two separate model architectures of a different type of neural network (such as a fully connected artificial neural network) in order to determine whether these conclusions are independent or dependent of the type of network used for the classification and learning process.

The two datasets conducted involve two different types of classification problems. The Dogs & Cats dataset utilized a binary classification problem, whereas the Intel Image Classification dataset was a multi-class classification problem. In order to further bolster the conclusions drawn from the experiments performed, additional experimentation would be useful for these types of problems, adding a second binary classification dataset and a second multi-class classification dataset would further bolster the conclusion of this project by allowing direct comparison of sub-categories of classification instead of simply treating all classification problems as the same.

The experiments performed were more concerned with evaluating the changes in performance across the different metrics used. Because of this intention of the experiments conducted, none of the models or preprocessing experiments were directly designed to achieve optimal loss/accuracy in the datasets used, and measure whether the preprocessing results are consistent when performed on the optimal hyperparameters for the respective image classification problems, rather than simply in general use-cases. Future work can expand the results by conducting more refined hyperparameter tuning than the rudimentary tuning performed for the experimetns in this project in order to determine how consistend the results are with better-tuned variants of the implementations used.

Finally, only two datasets and only two model architectures were considered in the context of experimentation. Though these are sufficient for drawing high-level conclusions, the confidence of these conclusions could be further bolstered by the increased grandularity of experimentation in these aspects. Determining whether these results remain consistent across more than two model architectures and across more than two different image datasets in order to more confidently conclude the objserved behavior across a broader spectrum of granular experimentation.

REFERENCES

[1] A. Dey and S. K. Bandyopadhyay, "Automated Glaucoma Detection Using Support Vector Machine Classification Method", *JAMMR*, vol. 11, no. 12, pp. 1-12, Oct. 2015.

[2] E. O. Omidiora, M. O. Oladele, T. M. Adepoju, A. A. Sobowale, and O. A. Olatoke, "Comparative Analysis of Back Propagation Neural Networks and Self Organizing Feature Maps in Estimating Age Groups Using Facian Features", *CJAST*, vol. 15, no. 1, pp. 1-7, Mar. 2016.

[3] P. Chrabaszcz, I. Loshchilov, and F. Hutter, "Back to Basics: Benchmarking Canonical Evolution Strategies for Playing Atari", *arXiv preprint arXiv: 1802.08842*, pp. 1-8, Feb. 2018.

[4] S. Wold, K. Esbensen, and P. Geladi, "Principal Component Analysis", *Chemometrics and intelligent laboratory systems*, vol. 2, no. 1-3, pp. 37-52, Aug. 1987.

[5] K. Jiang, Q. Que, and B. Kulis, "Revisiting Kernalized Locality-Sensitive Hashing for Improved Large-Scale Image Retrieval", *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4933-4941, 2015.

[6] chetanimravan, *Dogs & Cats Images*, Version 1, Kaggle: chetanimravan, 2018. [Dataset]. Available: https://www.kaggle.com/chetankv/dogs-cats-images.

[7] P. Bansal, *Intel Image Classification*, Version 1, Kaggle: Puneet Bansal, 2019. [Dataset]. Available: https://www.kaggle.com/puneet6060/intel-image-classification.

[8] S. Albawi, T. A. Mohammed and S. Al-Sawi, "Understanding of a Convolutional Neural Network," *2017 International Conference on Engineering and Technology (ICET)*, Antalya, Turkey, 2017, pp. 1-6, doi: 10.1109/ICEngTechnol.2017.838186.

[9] H. Abdi and L. J. Williams, "Principal Component Analysis," *Wiley interdisciplinary reviews: computational statistics*, vol. 2, no. 4. pp. 433-459, 2010.

[10] M. Datar, N. Immorlica, P. Indyk, V. S. Mirrokni, "Locality-Sensitive Hashing Scheme Based on p-Stable Distributions," *Proceedings of the twentieth annual symposium on Computational geometry*. pp. 253-262, 2004.

[11] N. Sundaram, A. Turmukhametova, N. Satish, T. Mostak, P. Indyk, S. Madden, and P. Dubey, "Streaming Similarity Search over one Billion Tweets using Parallel Locality-Sensitive Hashing," *Proceedings of the VLDB Endowment*, vol. 6, no. 14. pp. 1930-1941, 2013.

[12] S. Har-Peled, P. Indyk and R. Motwani, “Approximate Nearest Neighbor: Towards Removing the Curse of Dimensionality,” *Theory of computing*, vol. 8, no. 1. pp. 321-350, 2012.

[13] T. Tuncer, S. Dogan and F. Ozyurt, “An Automated Residual Exemplar Local Binary Pattern and Iterative ReliefF Based COVID-19 Detection Method Using Chest X-Ray Image,” *Chemometrics and Intelligent Laboratory Systems*, vol. 203. pp. 104054, 2020.

[14] K. M. Sajjad, “Automatic License Plate Recognition Using Python and OpenCV,” *Department of Computer Science and Engineering MES College of Engineering*. 2010.

[15] M. O. Oladele, E. O. Omidiora and A. O. Afolabi “A Face-Based Age Estimation System Using Back Propagation Neural Network Technique,” *Journal of Advances in Mathematics and Computer Science*, pp. 1-9, 2016.

[16] X. Long, W. L. Cleveland and Y. L. Yao “A New Preprocessing Approach for Cell Recognition,” *IEEE Transactions on Information Technology in Biomedicine*, vol. 9, no. 3, pp. 407-412, 2005.

[17] B. Chen, Z. Liu, B. Peng, Z. Xu, J. L. Li, T. Dao, Z. Song, A. Shrivastava, and C. Ré, “MONGOOSE: A Learnable LSH Framework for Efficient Neural Network Training,” *ICLR (oral)*, 2020.

[18] Y. Qiao, C. Cappelle, Y. Ruichek, and T. Yang, “ConvNet and LSH-Based Value Localization Using Localized Sequence Matching,” *Sensors*, vol. 19, no. 11, pp. 2439, May 2019.

[19] O. Rozenstein, T. Paz-Kagan, C. Salbach, and A. Karnieli, “Comparing the Effect of Preprocessing Transformations on Methods of Land-Use Classification Derived From Spectral Soil Measurements,” *IEEE Journal of Selected Topics in Applied Earth Observation and Remote Sensing*, vol. 8, no. 6, pp. 2393-2404, Jun. 2015, doi: 10.1109/JSTARS.2014.2371920.

[20] S. A. Medjahed, “A Comparative Stude of Feature Extraction Methods in Image Classification,” *International journal of image, graphics and signal processing*, vol. 7, no. 3, pp. 16-23, 2015.

[21] G. Bradski, “The OpenCV Library,” *Dr. Dobb’s Journal of Software Tools*, 2000.

[22] L. Roeder (2021) Netron (Release 4.8.9) [Source code]. https://github.com/lutzroeder/netron.

[23] F. Chollet (2020) Keras (Release 2.4.0) [Source code]. https://github.com/keras-team/keras.

[24] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: A System for Large Scale Machine Learning," in *12th USENIX Symposium on Operating System Design and Implementation (OSDI 16)*, pp. 265-283, 2016.

[25] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825-2830, 2011.

[26] T. O'Malley, E. Bursztein, J. Long, F. Chollet, H. Jin, L. Invernizzi, and others (2019) Keras Tuner (Release 1.0.2) [Source code]. https://github.com/keras-team/keras-tuner.

[27] J. D. Hunter, "Matplotlib: A 2D Graphics Environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90-95, 2007, doi: 10.1109/MCSE.2007.55.

[28] D. Young (2021) An Objective Comparison of Feature Reduction Methods for Neural Network Image Classification [Source code]. https://drive.google.com/drive/folders/1KPBMunLt7tYlkAyQJSSTqPbe3Vu2zZ9W?usp=sharing.

[29] D. Young (2021) An Objective Comparison of Feature Reduction Methods for Neural Network Image Classification [Code repository]. https://github.com/drake-young/An_Objective_Comparison_of_Feature_Reduction_Methods_for_Neural_Network_Image_Classification.

This is the final page of a Project Report and should be a blank page