Classifying CIFAR-100 Images

Priyanka Chakraborti *University of Nebraska-Lincoln*
Bryan Moore *University of Nebraska-Lincoln*

I. Introduction

The main goal of this study is to find a suitable architecture of convolutional and fully connected layers which can classify images from the CIFAR-100 dataset with high accuracy. To this end we explore 2 convolutional architectures built and trained from scratch. This includes pre-training an autoencoder architecture ecture, on the imagenet dataset, before replacing the decoder layers with our fully-connected dense layers and retraining on the CIFAR-100 dataset. The CIFAR-100 dataset is described in detail in Section II. For each convolutional architecture we perform a grid-search of 3 different sets of hyperparameters using statistical benchmarking. For the transfer model we explore two different approaches for retraining on the CIFAR-100 dataset.

We utilize 95% confidence intervals to decide whether the differences in accuracy between models are statistically significant. To do this we implement 5-Fold Cross-Validation, and assume a student-t distribution, to calculate the 95% confidence limit for the estimated difference in accuracy between all considered models. This process is described in detail in Section IV. The findings are discussed in Section V and the conclusions in Section VI.

II. Data Summary

The main dataset used in this study is the CIFAR-100 dataset [1]. Every image in this set is a 32x32 color image with 3 channels. The value for each pixel in these images is an integer between 0 and 255. For use in our model we normalize each pixel value to the range 0 to 1. The dataset contains 100 unique classes, of which there are 20 superclasses. These superclasses are aquatic mammals, fish, flowers, food containers, fruit and vegetables, household electrical devices, household furniture, insects, large carnivores, large man-made outdoor things, large natural outdoor scenes, large omnivores and herbivores, medium-sized mammals, non-insect invertebrates, people, reptiles, small mammals, trees, vehicles 1, and vehicles 2. Each class has an equal number of images such that there is no class imbalance.

We read in 50,000 images from this set and randomly shuffle the order of the images. As we are utilizing 5-fold cross-validation, this means that for each fold 20% of the data is aside to serve as an evaluation set. It is worth noting that the number of samples in each set puts us far above the threshold where the central limit theorem begins to apply. We can thus safely approximate that the underlying distribution is Gaussian.

For pre-training the autoencoder we use the imagenet dataset [2]. This contains 1.33 million unlabeled color images. The details of this architecture are given in Section III.

III. Architectures Employed

For this study we have chosen to create four unique architectures. We will consider each architecture one at a time. Each expects an input tensor of dimension (None,32,32,3). Note that we have intentionally omitted specifying the number of filters in many of the convolutional layers, and the number of neurons in the dense layers. This information is not mentioned in this section as we will explore different variations of these as our hyperparameters. These hyperparameters, along with the investigative process we follow, are described in Section IV.

All architectures employed, with the exception of the autoencoder, are trained by minimizing the cross-entropy loss function. The loss function for the autoencoder is specified in Subsection C of this section. For the other architectures we use cross-entropy to update the weights in our graph, and compare the effects of hyper-parameters and differences in architecture. Note that we use cross-entropy, even though our main target is actually the classification accuracy, due to the discontinuous derivatives of the raw classification accuracy. These discontinuities make updating the weights impossible, thus leading us to rely on cross-entropy.

There are many hyperparameters which are held the same for all architectures discussed below. For ease of reading we will list them here. We used the Adam optimizer, and left the momentum parameters at default. We explored manipulating these, but did not see any significant benefit, so left them at default. We did however adjust the learning-rate to 0.0001. All convolutional layers and dense layers implemented in these architectures use L2 kernel regularization with a regularization scale of 0.001. This was decided by manually adjusting the scaling and benchmarking the prediction accuracy. We also explored using Early-Stopping, but found that although it led to lower cross-entropy, that we were able to achieve higher accuracy on the test-set with this turned off. Hence, we did not implement Early-Stopping.

A. Convolutional Architecture 1

The first architecture is referred to as Arch1 in the tables below. The following defines our first convolutional block.

(a) The first layer is a convolutional layer with a 5x5 kernel. This is followed by passing the output through a leaky-RELU activation function. Then we reduce dimensionality using a maxpooling layer with pool size of 2, followed by batch-normalization. Finally we pass this through a dropout layer with 80% dropout rate to reduce overfitting.
(b) The second convolutional block is identical to the first except that we use a different number of filters in the convolutional layer.
(c) The third convolutional block is a convolutional layer with a 3x3 kernel. The output is passed through a leaky-RELU activation function and then passed through a dropout layer with 80% dropout rate.
(d) The fourth convolutional block is identical to the third.

The output of this layer is then flattened and sent into three densely connected layers. Each of these dense layers has a leaky-RELU activation function and is followed by a dropout layer with 80% dropout rate. As the number of neurons in each layer is adjusted during the investigation we will discuss this in Section IV. After these dense layers the output is sent into the final dense layer, which has 100 neurons, one for each class. This output is followed by a softmax layer to determine the predicted class.

During our investigation we will alter the number of filters, and neurons, in each layer to determine the effect the width of a convolutional architecture has on the prediction accuracy. The main idea of this model is to extract features from the images using the convolutional layers first, increasing the number of filters as we specialize, before utilizing the dense layers to essentially compute how to weight all combinations of these features.

B. Convolutional Architecture 2

We will refer to this architecture as Arch2 in the tables below. This architecture is intentionally chosen to be very similar to Architecture 1. We leave the width and general setup of the architecture the same as for Convolutional Architecture 1, and make it deeper by adding additional convolutional layers. The layers are set up as described below.

(a) For our first convolutional block the first layer is a convolutional layer with a 5x5 kernel. This is followed by passing the output through an exactly identical convolutional layer. The idea is to simulate a dense net with stacked convolutions. The output of this is then passed through a leaky-RELU activation function, followed by dimensionality reduction using a maxpooling layer of pool size 2. This is followed by batch-normalization. Finally, we pass this through a dropout layer with 80% dropout rate.
(b) The second convolutional block is identical to the first except that we use a different number of filters in the convolutional layers.

(c) Likewise, the third convolutional block begins with a convolutional layer with a 3x3 kernel, and the output is passed through an exactly identical convolutional layer. This is followed by a leaky-RELU activation function and a dropout layer with 80% dropout rate.
(d) The fourth convolutional block is identical to the third.
(e) The output layer follows the same schema as architecture 1.

As before we will treat the number of filters, and neurons, as our hyperparameters. We believe that deepening the layers, as described above, will allow us to pull out more features from the images, and provide an improvement in accuracy over convolutional architecture 1.

C. Autoencoder Architecture

This will be used for pre-training on the images from the imagenet dataset. Every convolutional layer utilizes L2-regularization. The output of each convolutional layer passes through a relu activation function, followed by batch normalization.

(a) The first layer is a convolutional layer with 64 filters, a 3x3 kernel, and a stride of 1.
(b) From here we send the output into two parallel convolutional layers, creating a tower-like architecture. The first branch is a convolutional layer with 128 filters, a 3x3 kernel, and a stride of 2. The second branch has 128 filters, a 5x5 kernel, and a stride of 1. The output of these branches passes through maxpooling layers, with a pool size of 2, before sent to the relu activation function. The individual branches are then concatenated along the 3rd dimension.
(c) The tower is followed by a convolutional layer with 256 filters, a 3x3 kernel, and a stride of 2. The output of this is then flattened and sent into the code layer, which has 512 neurons.
(d) For decoding this code layer we first send the output into a transpose convolutional layer with 256 filters, a 3x3 kernel, and a stride of 1. We then send the output into two parallel transverse convolutional layers. Each is identical with 128 neurons, a 3x3 kernel, and a stride of 2.
(e) The output of these is sent into the final transverse convolutional layer. This has 64 filters, a 5x5 kernel, and a stride of 2. The output is then sent into the final transverse convolutional layer.
(f) To match the input dimensions (32x32x3), the output layer of the decoder has 3 filters, a 3x3 kernel, and a stride of 1.

For training the autoencoder our loss function is defined as

$$Loss_{total} = Loss_{reconstruction} + Loss_{sparsity} \tag{1}$$

in which $Loss_{reconstruction}$ is defined as

$$Loss_{reconstruction} = mean[(pred_{image} - actual_{image})^2] \tag{2}$$

where we are taking the average over all pixels in the image. $Loss_{sparsity}$ is defined as the L1-norm of the code layer, multiplied by a constant. In our approach we multiplied the L1-norm by $5 \cdot 10^{-3}$ when calculating $Loss_{sparsity}$.

The main motivation for constructing an autoencoder of this type is that we believe that using this parallel architecture will allow the information to be processed in different ways, hopefully allowing us to identify features that otherwise may have been lost or averaged out.

D. Transfer Architecture

For our final architecture we start by taking the trained autoencoder architecture. We then extract the tensor just after it passes through the leaky-RELU activation function after the code layer. As the question in the assignment specified that the layers after the code can only be connected layers, we have excluded the use of convolutional layers. We have added the following dense layers on top of the code layer from the autoencoder architecture.

Each of these dense layers has a leaky-RELU activation function and L2-regularization. A dropout layer with a dropout rate of 80% has been inserted between each of the dense layers. There are five dense layers in total before the final layer. For this architecture we rely on the encoder layers of the autoencoder to extract the features from the images, and we use these added dense layers to determine the importance which should be given for combinations of these features.

Noting that the code layer has 512 neurons, we chose to follow this by setting the number of neurons in our first dense layer to 1024 neurons. The second dense layer then has 512 neurons, and the third has 512 neurons. The belief is that the large number of neurons in each layer will ensure we do not lose the ability to identify complex features. We now reduce the width of the layers, hopefully forcing the network to specialize.

For the fourth dense layer we use 256 neurons, and for the fifth we again use 256 neurons. The final output layer has 100 neurons and passes through a leaky-RELU activation layer. This is followed by a softmax layer to determine the predicted class. We believe that retraining these layers should provide high enough complexity to transfer the weights we learned from the imagenet dataset, while still being able to specialize on those in the CIFAR-100 dataset.

IV. METHODS FOR COMPARING MODELS

When read in the class labels are denoted as integers between 0 and 99. We one-hot-encode the labels so that we can use cross-entropy when evaluating our predictions on the CIFAR-100 dataset. Of course, for training the autoencoder on the imagenet data there are no labels. For both training sets we have chosen to use a batch size of 50 for weight updates. This was chosen to balance lowering the run-time and ensuring that we have enough variance in weight updates to avoid being stuck in a local minima. We used 50 epochs as our maximum when training on the CIFAR-100 data. This was chosen because learning curves, such as those shown in Fig. 2, clearly show that past about 30 epochs there is only small improvement on the test set accuracy. We thus chose 50 epochs to ensure that we would not cut our learning short when comparing different architectures and sets of hyperparameters.

For the CIFAR-100 dataset all images are randomly shuffled, with the seed set to 42, such that although the shuffling is effectively random, it is also repeatable. For comparing the considered models on the CIFAR-100 dataset we use 5-fold cross validation for each. This means that for each fold we have a training set of 40,000 images, and an evaluation set of 10,000. Both are solidly within the regime where the central limit theorem is valid. We do not shuffle any further. This ensures that we can directly compare the same folds across all models we are testing.

For Convolutional Architecture 1 and Convolutional Architecture 2 we explored adjusting the width of the convolutional layers as our 3 sets of hyperparameters. We explored the same sets of hyperparameters for each architecture. As we have four convolutional blocks in each architecture, we denote the number of filters in each convolutional layer within that block as an entry in a list. Each architecture also has the same number of dense layers, and we adjusted the number of neurons in each layer using the same scale as for the convolutional filters.

Thus, we also denote the number of dense layers as entries in a list. To make the notation easy to understand we will include each set of hyperparameters in the form:
[number of convolutional filters],[number of neurons]

We thus explored the following sets of hyperparameters; $\{[32, 64, 128, 128], [1024, 512, 256]\}$, $\{[64, 128, 256, 256], [2048, 1024, 512]\}$, and $\{[128, 256, 512, 512], [4096, 2048, 512]\}$. We refer to these as hyperparameter sets 1, 2, and 3. These combinations are chosen as we are interested to see if we increasing the width of our model, thus drastically increasing its complexity without adding additional layers, has a significant impact on the prediction accuracy.

For the transfer model, due to time-limitation we were not able to explore too many hyperparameters. However, we did investigate the differing effects of retraining only those dense layers we added on top of the code layer, and retraining the entire model. Those results are detailed below. We denote the model where we retrained all layers as T_Arch1, and the model where we retrained only the dense layers we added as T_Arch2.

When performing k-fold cross-validation, each fold can be considered an independent investigation. This is valid as we reset the TensorFlow graph before each fold begins. At the end of training a particular fold the accuracy and cross entropy for that fold is computed and saved for statistical evaluation.

We compare all models by calculating the estimate of the accuracy between each model with every other model. We would normally compare the error for each model, but as we intentionally allowed over-fitting to raise the accuracy, comparing based on accuracy makes more sense. Note that the k-fold splits will create the same groups every time, as we seeded our random permutation before the shuffle. We are therefore comparing the same subset of images every time for each model.

We denote each of these estimated accuracy differences as p_i, where i denotes the fold number. We then average these into a single variable denoted p, which represents the expected difference in accuracy between the two models. It is very important to note that we can only do pair-wise comparisons. The below formula is used to estimate the minimum value the accuracy difference at the 95% confidence interval, assuming a Student-t distribution of the differences in accuracy.

$$p + t_{c,K-1} s_p \tag{3}$$

where K is the number of folds (5) and s_p is

$$s_p = \sqrt{\frac{1}{K(K-1)} \sum_{i=1}^{K} (p_i - p)^2} \tag{4}$$

For our desired 95% confidence $t_{c,K-1}$ is 2.132. We use this one-way t-test by locating the smallest value for $p + t_{c,K-1} s_p$ from each row. This means the model with the smallest value outperformed the model specific by that row. Also, if the optimal model for that row is found to be the same as the row, it means that it outperformed all models it competed against. For an example of how this is carried out see Table II.

Again, note that technically this can only tell us anything within a single row. However, to generalize our comparisons we implement logical comparisons to trace back which combination is actually best overall. To do this we first find which model is optimal for each row. We then look at the results for each row and see if there is a particular model which consistently outperforms all others. For example, if the same model is found to be optimal for all rows, then that model is clearly better overall. This is the sort of trend we are looking for when implementing this comparison approach.

After finding the overall optimal model we train it on our training set, and evaluate it on the test set. The performance on our set-aside test set will provide a reasonable estimate of what the model performance will be on the test-set held by the instructors.

V. Results

A. *Choose Best Model*

The average results across all 5-folds are shown in Table I for each model. From this we observe that for improving accuracy Convolutional Architecture 2, with hyperparameter set 3, is the best choice of all considered models. It is interesting to note that this model does not correspond to that which minimizes the cross-entropy, but as we have intentionally decided to increase overfitting to improve the accuracy this is not necessarily too surprising. For a more in-depth investigation of the difference in accuracy between models we implement the methodology described in Section IV. These results are displayed in Table II.

TABLE I: Average Accuracy and Cross-Entropy for Models

	Average Accuracy (%)	Average Cross-Entropy
Arch1-Hyp1	35.07	6.58
Arch1-Hyp2	39.29	5.49
Arch1-Hyp3	41.28	5.09
Arch2-Hyp1	36.33	6.42
Arch2-Hyp2	40.81	5.59
Arch2-Hyp3	41.94	5.52
T_Arch1	29.21	7.92
T_Arch2	20.38	5.74

Each row represents a different model. Arch_1 and Arch_2 are referring to Convolutional Architecture 1 and Convolutional Architecture 2, as described in Section III. Hyp1, Hyp2, and Hyp3 are referring to the 3 sets of hyperparameters described in Section IV. T_Arch1 refers to the transfer model described in Section III, where we have allowed all layers to be retrained, while T_Arch2 is the same architecture and only retrains the added dense layers.

TABLE II: Confidence Intervals for Models

	Arch1-Hyp1	Arch1-Hyp2	Arch1-Hyp3	Arch2-Hyp1	Arch2-Hyp2	Arch2-Hyp3	T_Arch1	T_Arch2
Arch1-Hyp1	0.0000	-0.0422	-0.0621	-0.0126	-0.0574	-0.0687	0.0586	0.1470
Arch1-Hyp2	0.0422	0.0000	-0.0199	0.0296	-0.0152	-0.0265	0.1008	0.1892
Arch1-Hyp3	0.0621	0.0199	0.0000	0.0495	0.0047	-0.0066	0.1207	0.2091
Arch2-Hyp1	0.0126	-0.0296	-0.0495	0.0000	-0.0448	-0.0561	0.0712	0.1596
Arch2-Hyp2	0.0574	0.0152	-0.0047	0.0448	0.0000	-0.0113	0.1160	0.2043
Arch2-Hyp3	0.0687	0.0265	0.0066	0.0561	0.0113	0.0000	0.1273	0.2156
T_Arch1	-0.0586	-0.1008	-0.1207	-0.0712	-0.1160	-0.1273	0.0000	0.0883
T_Arch2	-0.1470	-0.1892	-0.2091	-0.1596	-0.2043	-0.2156	-0.0883	0.0000

Each row represents a different model. Arch_1 and Arch_2 are referring to Convolutional Architecture 1 and Convolutional Architecture 2, as described in Section III. Hyp1, Hyp2, and Hyp3 are referring to the 3 sets of hyperparameters described in Section IV. T_Arch1 refers to the transfer model described in Section III, where we have allowed all layers to be retrained, while T_Arch2 is the same architecture and only retrains the added dense layers. For each row, the column which corresponds to the smallest value in that row represents the model which outperformed all models for that row, as judged by the condition imposed by equation (3).

As we can see from these results, Convolutional Architecture 2, with hyperparameter set 3, achieves the highest average accuracy, and is statistically found to be optimal when compared to every row-model within Table II. This model is described in Section III, and the optimal hyperparameters have 128 filters in convolutional block 1, 256 filters in convolutional block 2, 512 filters in convolutional block 3, and 512 filters in convolutional block 4. The following dense layers have the following numbers of neurons. Dense layer 1 has 4096 neurons, dense layer 2 has 2048 neurons, and dense layer 3 has 512 neurons.

We noted that regardless of whether we retrained the entire model, or only the dense layers we added, the Transfer Model was not able to outperform the convolutional architectures. As it was pre-trained on 1.33 million images from the imagenet dataset we had expected that it would have learned a very large number of very relevant features, which would directly contribute to an increase in accuracy when used again on the CIFAR-100 dataset.

Retraining the entire encoder stack performed better than training solely the added dense layers, which makes sense. However, our convolutional architectures vastly outperform even the fully retrained transfer model. It is hard to gauge the exact cause for this as the CIFAR 100 dataset is not vastly different from Imagenet. In such a case, we would have expected possible overfitting if we retrain all the layers, but superior performance when compared to a convolutional model trained only on the single dataset. We surmise that perhaps we did not use a complex enough architecture for the dense layers that were stacked on top of the code layer, or that we should have utilized a VAE. An in-depth grid-search of autoencoder architectures, and hyperparameters, would be the focus of continued work in this area.

B. Characterize Chosen Model

We now characterize this chosen model. The confusion matrix for our class-predictions is shown in Fig 1. From this confusion matrix we see that overall our predictions are not too bad, but that there are many classes for which we are missing almost all. We conjecture that these may correspond to classes belonging in the same superclass.

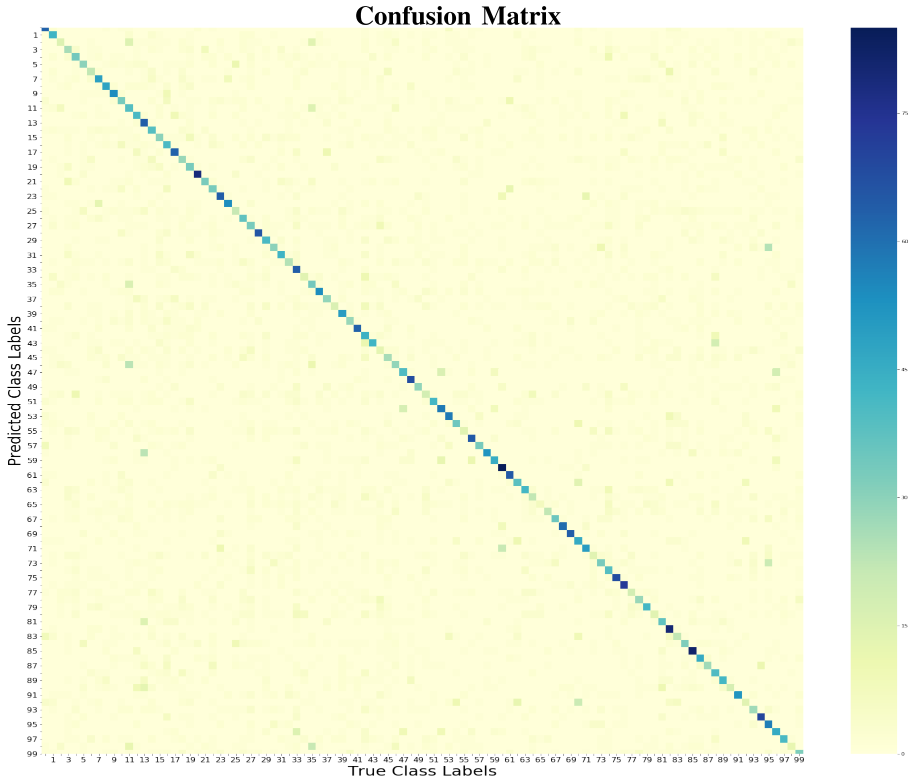

Fig. 1: This shows the percentage of each class correctly predicted by our chosen model. True positive rates for each class lie on the diagonal.

In Fig. 2 we show the learning curve for our chosen model. For this we use 80% of the data for training, and 20% for our test set. From these we see that the train set continues to learn more and more from the samples after approximately epoch 5, improving the train accuracy significantly. The cross-entropy plot makes this even more clear, as after epoch 5 the error on the validation set stops decreasing, and actually begins increasing rapidly for a time before leveling out. However, the train cross-entropy continues to drop well past that point.

However, note that although the cross-entropy on the test-set increases dramatically, the accuracy on the test-set actually still improves slightly. This is why we chose to allow our model to overfit to the train data. By allowing the overfitting we are still able to gain some improvement in terms of accuracy on the test set. This approach also led to increased performance on the instructor's holdout test-set. Overall, we achieve an accuracy of 44.06% on the instructor's holdout test-set, which we believe that for a 100 class prediction problem such as this is quite good.

Learning Curves - Chosen Model

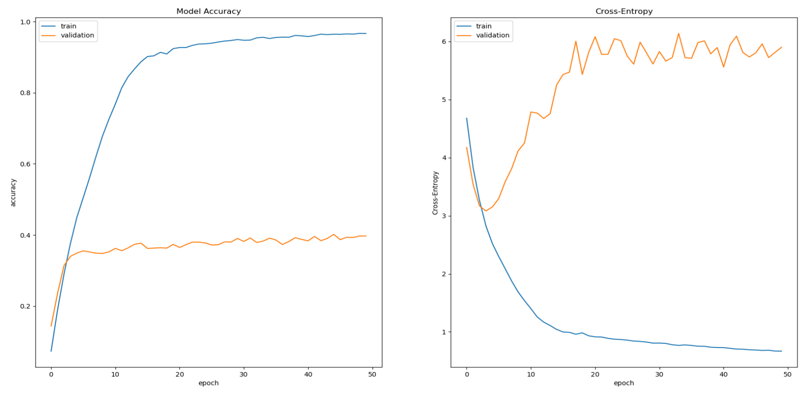

Fig. 2: This shows the learning curves for Convolutional Architecture 2 with hyperparameter set 3. Note the drastic overfitting, which we intentionally chose to allow.

VI. Conclusion

In conclusion, we have conducted a statistical investigation of three architectures, and a total of seven models, for classifying the CIFAR-100 dataset. All have been shown to achieve sufficiently good accuracy. The transfer architecture was built off of an autoencoder trained on 1.33 million images from the imagenet dataset. We implemented transfer learning by adding our dense layers on top of the code-layer from the pre-trained autoencoder architecture. Interestingly, the results showed that this model was vastly outperformed by the convolutional architectures trained only on the CIFAR-100 dataset. Statistical comparison shows that Convolutional Architecture 2, with hyperparameter set 3, is our optimal model.

This optimal model is described in detail Section III. As a brief overview, this has 128 filters in convolutional block 1, 256 filters in convolutional block 2, 512 filters in convolutional block 3, and 512 filters in convolutional block 4. The following dense layers have the following numbers of neurons dense layer 1 has 4096 neurons, dense layer 2 has 2048 neurons, and dense layer 3 has 512 neurons. This model achieves 44.06% on the instructor's holdout test-set.

We deduce that any further improvements, both in terms of accuracy and overfitting, likely require increased complexity of the chosen model. However, due to increasing length of training time we were unable to explore more complex architectures within the time-constraints imposed on this assignment. We thus leave this as possible improvements for the future, and hope this investigation has helped to shed light on how a relatively simple combination of convolutional and dense layers can already provide reasonable accuracy on even a 100 class problem.

References

[1] A. Krizhevsky, "Learning multiple layers of features from tiny images," tech. rep., 2009.
[2] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR09*, 2009.