EXPLORING DEEPER NEURAL NETWORKS IN ANALYSIS OF CIFAR-10 DATASET

Michael Mistarz

Northwestern University, MSDS458: Artificial Intelligence & Deep Learning

GitHub: https://github.com/mistmr7/MSDS458_Assignment2_CIFAR10_CNN

February 9, 2025

1. Abstract

This study explored deep learning neural networks to build an efficient model for classifying images in the Keras CIFAR-10 dataset. Layers explored included dense, 2D-convolutional, 2D-max pooling, dropout, batch normalization, and L1 and L2 regularization. The goal of this experimentation was to build a model capable of color image recognition and classification of both vehicle types and animal types. The model will be built for a photography company looking to decrease overhead costs by employing an artificial intelligence model to quickly classify images, greatly decreasing the need for manual classification.

Through experimentation, dense neural layers were not found to be adequate at image classification, as has been discussed in literature (Pratama 2020). Max pooling layers were found to pair well with convolutional layers. Dropout layers were found to regularize the model better than batch normalization or L1 and L2 regularization, leading to a base layer structure of two-dimensional convolutional layer, two-dimensional max pooling layer, and dropout regularization. The final model had this structure repeated multiple times, leading to a final test accuracy of 0.7666 while limiting the size of the model to 4.36 MB.

2. Introduction

The architecture of neural networks can be altered to best fit the input data. For smaller datasets, especially labeled texts, a shallow neural network can be utilized to build the model. Shallow neural networks are composed of a single hidden layer, while deep neural networks are composed of multiple hidden layers. Deep neural networks are needed to analyze larger input datasets, especially images (geeksforgeeks 2024). Using deep neural networks leads to a larger parameter count, more computational resources required, a higher risk of overfitting, and greater

difficulty in determining the inner workings of the model (Geeksforgeeks 2024). This requires more time and effort in building and optimizing the model, with more parameters to alter and more configurations to try than a shallow learning network.

For image processing, convolutional neural networks outperform dense neural networks (Pratama 2020). Convolutional neural networks involve running a filter through the image and manipulating the input data using linear combination. This allows the model to pull out features systematically, and using a filter window to move across the image allows for easier pattern recognition and therefore better image classification (Watson 2017). Convolutional neural networks also allow for automatic feature extraction, requiring less manual manipulation of the input data and allowing for more robustness in the output model (Tamanna 2023).

3. Literature Review

Just as the MNIST dataset is a classic dataset for beginning to understand neural networks, the CIFAR-10 dataset is seen as a good next step down the neural network path (Ombaval 2024; Khan 2024; Gopalakrishnan 2022; Franky 2022). Books, videos, and step-by-step guides for creating and running convolutional neural networks can be found, even with step-by-step guides to training on the CIFAR-10 dataset (Shanmugamani 2018; Watson 2017; AI Guy 2021). CIFAR-10 provides an ideal dataset for entry into image processing, giving a large, labeled dataset of small image sizes (32 x 32 pixels) for efficient model training and preprocessing.

In a similar study to this one, TensorFlow was used to study whether a convolutional neural network could outperform K-nearest neighbor (KNN) clustering in a labeled dataset of 2000 images of ten image types (Zhao and Li 2021). In the study, the convolutional neural network was able to predict accuracy on the entire dataset of 89 percent while the KNN model

was only able to achieve 46 percent accuracy. Despite only having 2000 total images in the dataset, this study showed how convolutional neural networks provide a better path forward for image processing (Zhao and Li 2021).

4. Methods

The input data used in this study was loaded from the CIFAR-10 dataset provided in TensorFlow Keras built-in datasets, accessed using python (TensorFlow n.d.). This dataset provides 60000 32x32 color images from ten different categories: airplanes, automobiles, birds, cats, deer, dogs, frogs, horses, ships, and trucks. The dataset is split into 50000 training images and 10000 test images. An example of the input data can be seen in Figure 1.

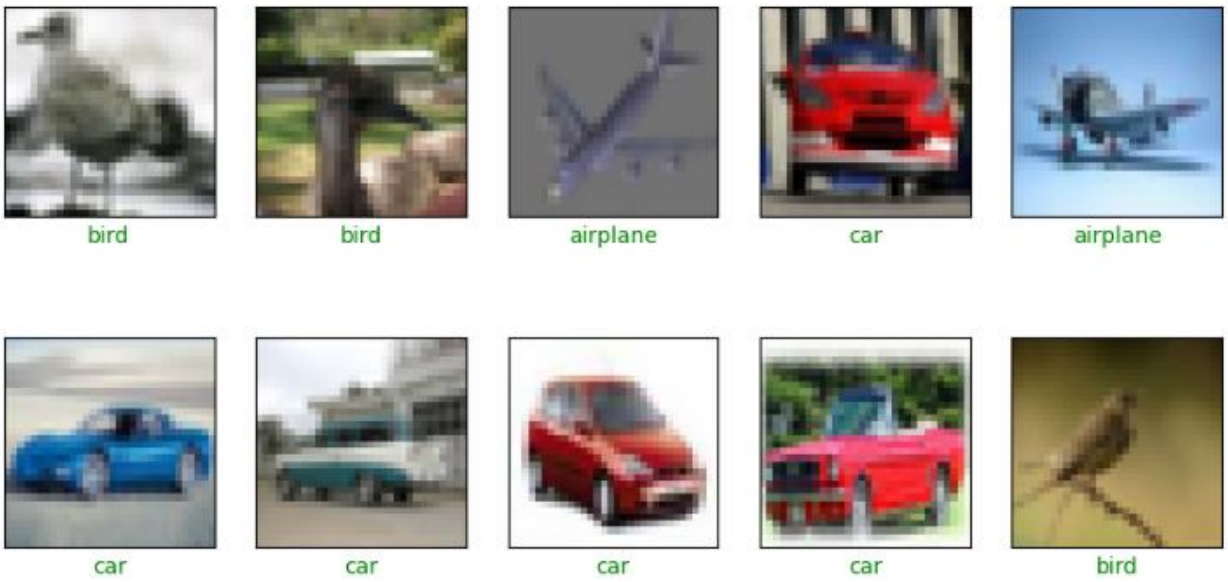

Figure 1: Example images and corresponding labels from the CIFAR-10 dataset

4.1 Data Preparation

The training images, training labels, test images, and test labels were all extracted directly from the TensorFlow Keras CIFAR-10 dataset. The training images and labels were then split using Scikit-learn using the train-test-split function, where 5000 of the images and corresponding labels were randomly selected as validation data. This created a training image set with shape

(45000, 32, 32, 3) corresponding to 45000 images of 32 x 32 pixels with 3 different colors. Each of the image sets were then normalized by dividing by 255 to get a value between 0 and 1, with a higher value representing more saturated pixels.

From earlier experimentation on the MNIST dataset (Mistarz 2025), several improvements were added to all models to decrease exploration time and minimize GPU processing requirements. These improvements included adding an early stopping monitor to the model, halting the model when the validation accuracy decreased for three straight epochs. This allowed the ability to stop the model when the validation accuracy started to differ from the training accuracy or when continuing to train the model did not lead to an improvement in overall accuracy.

4.2 Exploring a Single Hidden Layer

Now that the data has been reshaped and normalized, the focus began with comparing a single hidden layer. A single dense layer with node sizes of 4, 8, 16, 32, 64, 128, 256, 512, and 1024 was tested on the dataset. This was repeated with a two-dimensional convolutional layer with filter sizes identical to the node sizes above. A third experiment was set up with a two-dimensional max pooling layer after the convolutional layer. Each of these experiments were repeated using a regularization technique, testing out dropout layers, L1 and L2 normalization, and batch normalization. An example of one of the more complex single-layer models can be seen in Figure 2.

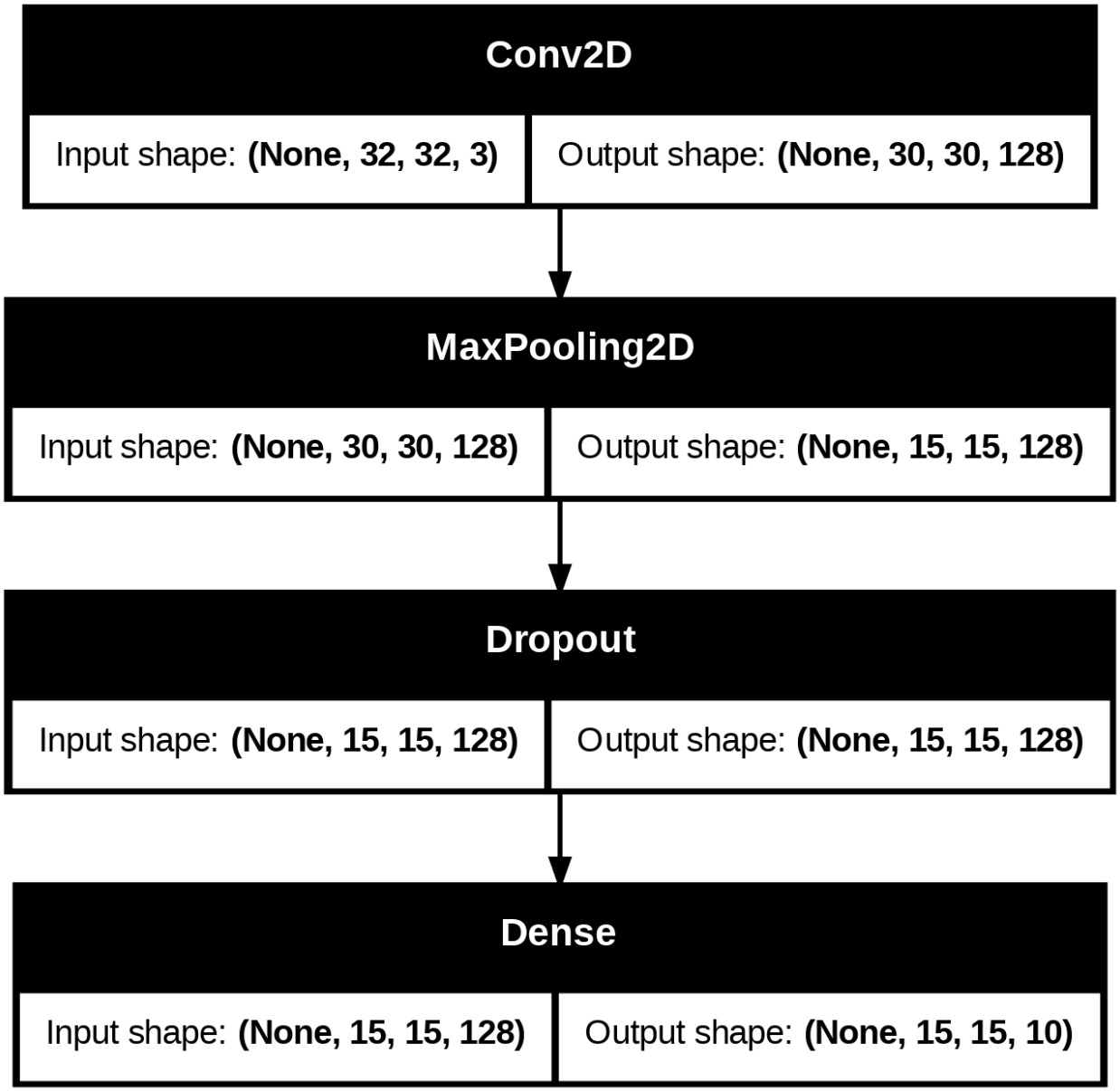

Figure 2: A 2D-Convolutional Layer with a 2D-Max Pooling Layer and Dropout regularization

4.3 Exploring Multiple Layers

The first experiments were repeated by adding one or more hidden layers to the initial hidden layer. Dense layers were tried with convolutional layers, multiple dense layers were tried, multiple convolutional layers were tried, and convolutional layers coupled with max pooling layers were tried as well. Each of these setups was run with and without regularization

techniques, and the best results were explored further in later examples as well. The best overall structure was taken for further exploration in creating the final model.

4.3 Testing Other Parameters

Multiple experimental models were run with small alterations, including altering the kernel size of the convolutional layer, altering the activity regularizer on both the dense and convolutional layers, changing the size of the dropout layers, and many extra experiments checking different filter size setups. Each of these setups was tried with different parameters in determining how to test for the setup of the final best model.

4.4 Determining the Best Model

In creating the final models, the optimal model from testing was chosen along with several other models with similar structures while differing in complexity of parameters. These models were all run ten times each to determine the best model to choose going forward. Processing time, training accuracy, training loss, validation accuracy, validation loss, and testing accuracy of each of the sets were analyzed and a best model was selected.

4.5 Analysis of the Model

After the best model was chosen, a confusion matrix was built showing the true label of the images in the set and the predicted label of the model, highlighting where the model struggled most in its predictions. Original images that were predicted incorrectly were plotted to show where errors were occurring and where the model could be improved.

5. Results

5.1 Shallow Learning Models

The initial experiments looked at whether a shallow learning model would be adequate at predicting the CIFAR-10 images. A single layer dense neural network with 4 nodes was initially

tried, followed by altering the layer to have 8, 16, 32, 64, 128, 256, 512, and 1024 nodes. This was repeated for a two-dimensional convolutional hidden layer, altering the number of filters instead of the number of hidden nodes. The tables for the results of shallow learning for the dense and convolutional neural networks can be seen in Figure 3 and Figure 4 respectively. As can be seen in the tables, adding nodes to the dense neural network beyond 128 nodes did not significantly alter the validation accuracy or results though did increase the processing time. Similar results were seen with the convolutional neural network, with 128 filters or less providing similar processing time while slightly increasing accuracy as filters were increased.

model	accuracy	val_accuracy	test_accuracy	loss	val_loss	test_loss	time
DNN	0.453	0.415	0.417	1.597	1.687	1.670	30.760
DNN_8_nodes	0.484	0.448	0.455	1.491	1.582	1.554	28.051
DNN_16_nodes	0.499	0.458	0.473	1.440	1.543	1.502	38.472
DNN_32_nodes	0.496	0.467	0.476	1.457	1.527	1.489	19.331
DNN_64_nodes	0.543	0.485	0.481	1.329	1.500	1.480	32.487
DNN_128_nodes	0.547	0.475	0.475	1.319	1.512	1.507	36.486
DNN_256_nodes	0.579	0.489	0.484	1.228	1.509	1.514	65.084
DNN_512_nodes	0.577	0.488	0.482	1.236	1.532	1.537	90.682
DNN_1024_nodes	0.580	0.488	0.478	1.223	1.537	1.536	187.932

Figure 3: Single dense hidden layer output table for CIFAR-10 dataset with altering number of hidden nodes

model	accuracy	val_accuracy	test_accuracy	loss	val_loss	test_loss	time
initial	0.487	0.492	0.494	1.470	1.435	1.445	54.484
CNN_8	0.562	0.556	0.569	1.256	1.256	1.244	35.631
CNN_16	0.650	0.631	0.637	1.008	1.054	1.061	61.715
CNN_32	0.617	0.587	0.587	1.115	1.185	1.186	22.460
CNN_64	0.659	0.617	0.614	0.988	1.119	1.127	33.272
CNN_128	0.667	0.613	0.599	0.970	1.108	1.132	31.851
CNN_256	0.739	0.647	0.643	0.754	1.044	1.080	69.515
CNN_512	0.735	0.637	0.624	0.764	1.084	1.124	85.789
CNN_1024	0.793	0.637	0.633	0.594	1.173	1.212	232.270

Figure 4: Single convolutional hidden layer output table for CIFAR-10 dataset with altering number of filters

5.2 Exploring Deep Learning Models

In exploring deeper models, the first experiments added a second layer to the initial dense neural network or convolutional neural network. Adding a second dense layer to the 64-node dense neural network did not appreciably increase the accuracy of the model. Results from these experiments can be seen in Figure 5. For the convolutional neural network, a similar experimentation setup was tested using the 32-filter convolutional layer as the first hidden layer and attempting different hidden layers afterwards. This was attempted with a max pooling layer with or without adding a second convolutional layer. Unlike the dense neural network, adding a second layer to the convolutional neural network greatly increased training accuracy while slightly raising validation and test accuracy, though overfitting became more problematic with adding a second hidden layer. These results can be seen in Figure 6.

model	accuracy	val_accuracy	test_accuracy	loss	val_loss	test_loss	time
DNN_64_4_nodes	0.464	0.424	0.444	1.566	1.651	1.605	27.527
DNN_64_8_nodes	0.497	0.459	0.471	1.462	1.554	1.523	43.371
DNN_64_16_nodes	0.518	0.473	0.489	1.410	1.511	1.479	37.811
DNN_64_32_nodes	0.530	0.480	0.492	1.383	1.517	1.486	29.874
DNN_64_64_nodes	0.580	0.484	0.488	1.256	1.533	1.514	54.106
DNN_64_128_nodes	0.557	0.486	0.489	1.311	1.513	1.503	44.237
DNN_64_256_nodes	0.563	0.492	0.500	1.294	1.498	1.477	61.046
DNN_64_512_nodes	0.571	0.488	0.490	1.274	1.532	1.511	95.996
DNN_64_1024_nodes	0.566	0.484	0.485	1.285	1.511	1.499	168.276

Figure 5: Multi-layer dense neural network for the CIFAR-10 dataset

model	accuracy	val_accuracy	test_accuracy	loss	val_loss	test_loss	time
CNN_32_4_nodes	0.595	0.540	0.548	1.159	1.289	1.295	50.951
CNN_32_8_nodes	0.674	0.589	0.574	0.930	1.198	1.243	78.339
CNN_32_16_nodes	0.734	0.613	0.614	0.764	1.129	1.145	44.272
CNN_32_32_nodes	0.795	0.613	0.615	0.596	1.210	1.209	37.210
CNN_32_64_nodes	0.843	0.610	0.605	0.465	1.291	1.327	44.449
CNN_32_128_nodes	0.889	0.647	0.645	0.331	1.197	1.246	55.631
CNN_32_256_nodes	0.840	0.658	0.642	0.477	1.102	1.148	55.118
CNN_32_512_nodes	0.944	0.661	0.652	0.177	1.381	1.421	113.277
CNN_32_1024_nodes	0.888	0.651	0.645	0.336	1.229	1.270	213.050
CNN_32_MP	0.721	0.645	0.645	0.814	1.041	1.047	43.365
CNN_32_MP_256	0.796	0.658	0.661	0.593	1.039	1.049	54.637

Figure 6: Multi-layer convolutional neural network for the CIFAR-10 dataset

5.3 Exploring Regularization Functions

After exploring the multi-layer networks, it became clear that overfitting was a consistent problem as the model's complexity grew. Comparing four runs on similar models, Figure 7 shows the results of adding a max pooling layer between convolutional layers, adding a dropout

layer between convolutional layers and adding a batch normalization layer between convolutional layers. Only the dropout and max pooling layers brought the training and validation curves closer, so those two functions were chosen for our final model to explore further.

model	accuracy	val_accuracy	test_accuracy	loss	val_loss	test_loss	time
CNN_32_256_nodes	0.840	0.658	0.642	0.477	1.102	1.148	55.118
CNN_32_MP_256	0.796	0.658	0.661	0.593	1.039	1.049	54.637
CNN_32_DO_256	0.783	0.636	0.632	0.631	1.115	1.149	76.352
CNN_32_BN_256	0.952	0.597	0.599	0.148	2.375	2.532	71.470

Figure 7: Exploration of how adding regularization between convolutional layers could improve overfitting issues

After exploring these regularization techniques, further exploration into L1 and L2 normalization was performed. Both L1 and L2 regularization techniques were explored, though almost every L1 regularization technique destroyed the performance of the model, producing model accuracy plots that struggled to reach 10 percent accuracy as can be seen in Figure 8. More than 30 different configurations of L2 normalization were attempted, and a small subset can be seen in Figure 9. None of the L1 or L2 regularization techniques reliably brought the training and accuracy closer together without significantly reducing both, so neither of these techniques was chosen for the final model.

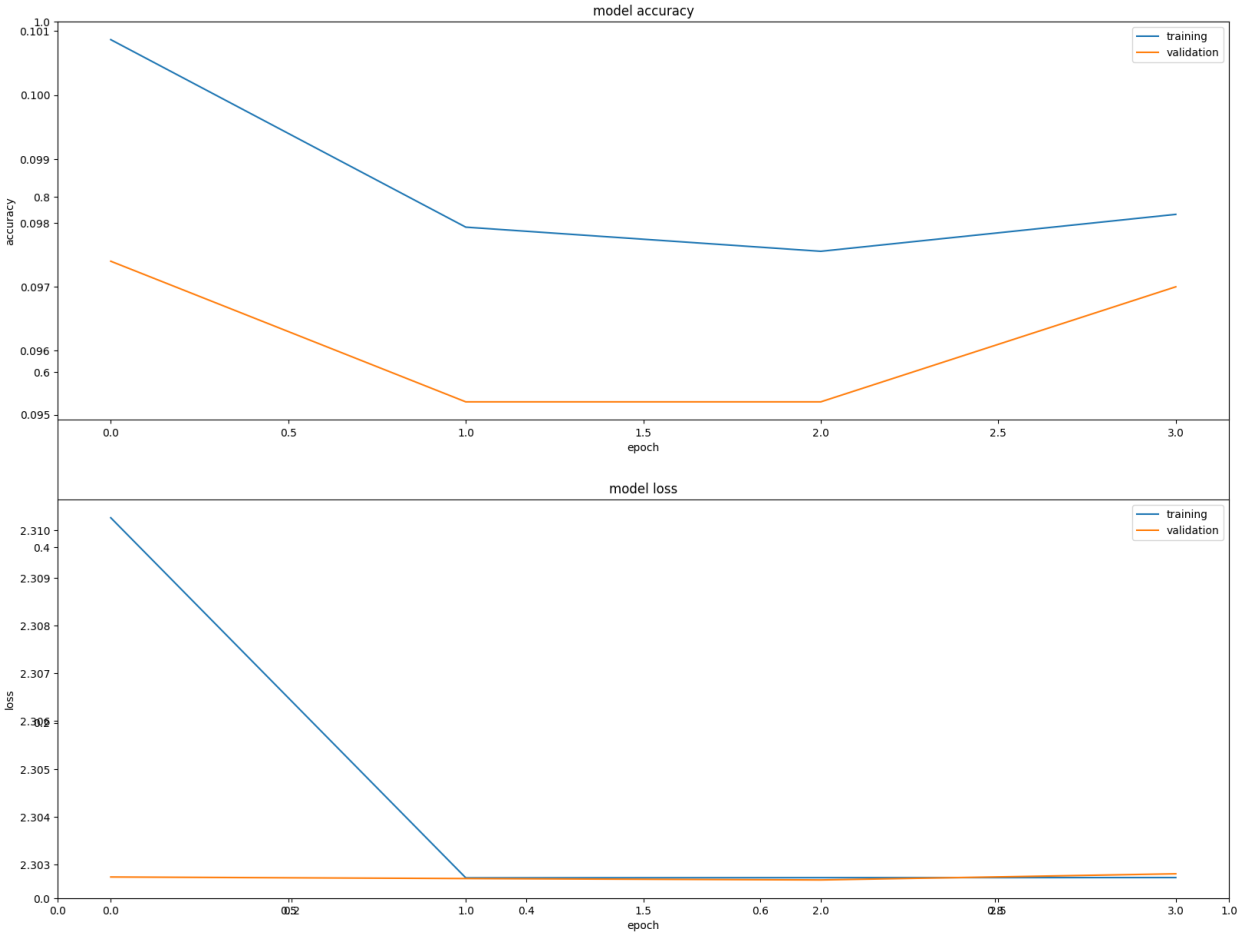

Figure 8: Accuracy and loss plots for multi-layer convolutional neural network employing L1 regularization.

model	accuracy	val_accuracy	test_accuracy	loss	val_loss	test_loss	time
CNN_32_256_256_DNN_reg_0	0.930	0.716	0.716	0.193	1.180	1.213	97.608
CNN_32_256_256_DNN_reg_0.1	0.098	0.097	0.100	2.303	2.303	2.303	28.501
CNN_32_256_256_DNN_reg_0.01	0.884	0.690	0.689	0.470	1.076	1.037	82.355
CNN_32_256_256_DNN_reg_0.001	0.947	0.715	0.708	0.252	1.059	1.064	87.927
CNN_32_256_256_DNN_reg_0.0001	0.933	0.747	0.733	0.255	0.905	0.955	52.052
CNN_32_256_256_DNN_reg_1e-05	0.952	0.737	0.729	0.172	1.084	1.067	72.002
CNN_32_256_256_DNN_reg_1e-06	0.896	0.732	0.736	0.303	0.942	0.968	63.639
CNN_32_256_256_DNN_reg_0_slot1	0.898	0.749	0.738	0.288	0.891	0.934	57.416
CNN_32_256_256_DNN_reg_0.1_slot1	0.707	0.568	0.563	0.972	1.437	1.394	148.867
CNN_32_256_256_DNN_reg_0.01_slot1	0.805	0.650	0.642	0.684	1.231	1.188	87.052
CNN_32_256_256_DNN_reg_0.001_slot1	0.909	0.711	0.709	0.356	1.174	1.136	72.745
CNN_32_256_256_DNN_reg_0.0001_slot1	0.947	0.743	0.743	0.207	1.122	1.102	89.572
CNN_32_256_256_DNN_reg_1e-05_slot1	0.910	0.738	0.730	0.282	0.969	1.021	59.468
CNN_32_256_256_DNN_reg_1e-06_slot1	0.936	0.742	0.737	0.188	1.077	1.130	70.776
CNN_32_256_256_DNN_reg_0_slot2	0.900	0.727	0.736	0.279	0.932	0.958	59.800

Figure 9: Exploring L1 and L2 regularization techniques for the multi-layered convolutional neural network.

5.4 Exploring Final Models

At this point in the experimentation, a pattern of convolutional layer, max pooling layer, then a dropout layer was chosen as the best framework for sharpening the model's accuracy. The model that performed best in the initial setup was the model shown in Figure 10. As such, similar models to this one were tested to try to pick the best model for correctly classifying the images. All models tested follow the pattern described above and shown in Figure 10 alongside a summary result table as shown in Figure 11. The initial model chosen had 32 filters, 256 filters, and 256 filters for the three convolutional layers. The other tested structures were 32-64-128 filters, 64-128-256 filters, 128-256-512 filters, 256-512-1024 filters. Each model was run 10 times, allowing for an average of accuracy, test accuracy, validation accuracy, and processing

time. Each of these four model setups can be seen in Appendix A, along with the ten-run summary tables showing the variance of the models.

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 30, 30, 32)	896
max_pooling2d (MaxPooling2D)	(None, 15, 15, 32)	0
dropout (Dropout)	(None, 15, 15, 32)	0
conv2d_1 (Conv2D)	(None, 13, 13, 256)	73,984
max_pooling2d_1 (MaxPooling2D)	(None, 6, 6, 256)	0
dropout_1 (Dropout)	(None, 6, 6, 256)	0
conv2d_2 (Conv2D)	(None, 4, 4, 256)	590,080
max_pooling2d_2 (MaxPooling2D)	(None, 2, 2, 256)	0
dropout_2 (Dropout)	(None, 2, 2, 256)	0
flatten (Flatten)	(None, 1024)	0
dense (Dense)	(None, 10)	10,250

Total params: 2,025,632 (7.73 MB)
Trainable params: 675,210 (2.58 MB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 1,350,422 (5.15 MB)

Figure 10: Best model structure from initial testing

	accuracy	val_accuracy	test_accuracy	loss	val_loss	test_loss	time
count	10.000000	10.00000	10.000000	10.000000	10.000000	10.000000	10.000000
mean	0.815200	0.76910	0.764300	0.522800	0.681900	0.708000	129.697300
std	0.029705	0.01159	0.011567	0.086974	0.033141	0.037956	32.049542
min	0.753000	0.74700	0.743000	0.426000	0.648000	0.672000	95.556000
25%	0.809500	0.76125	0.755750	0.448750	0.660250	0.682250	103.967000
50%	0.815000	0.77200	0.769000	0.524000	0.668500	0.690000	119.060500
75%	0.839500	0.77850	0.772750	0.539000	0.699250	0.728500	159.085750
max	0.849000	0.78300	0.779000	0.704000	0.749000	0.789000	176.365000

Figure 11: Results from 32-256-256 convolutional model over ten runs

After aggregating the results of each of the final models, the runs were compared to see which model to choose. Model size, processing time, model accuracy and testing accuracy were all considered in the selection. This was tabulated in Figure 12 and shown graphically in Figure 13, where all four models were comparable in test accuracy. However, due to model size and processing time, the final model chosen was a 64-128-256 filter convolutional neural network.

model	accuracy	test_accuracy	time (s)	size (MB)
32-256-256	0.8152	0.7643	129.6973	7.73
32-64-128	0.7376	0.7404	112.7025	1.13
64-128-256	0.8065	0.7666	111.9240	4.36
128-256-512	0.8387	0.7663	186.1798	17.16
256-512-1024	0.8372	0.7558	558.6481	68.07

Figure 12: Comparison table of averages of ten runs for each of the final models

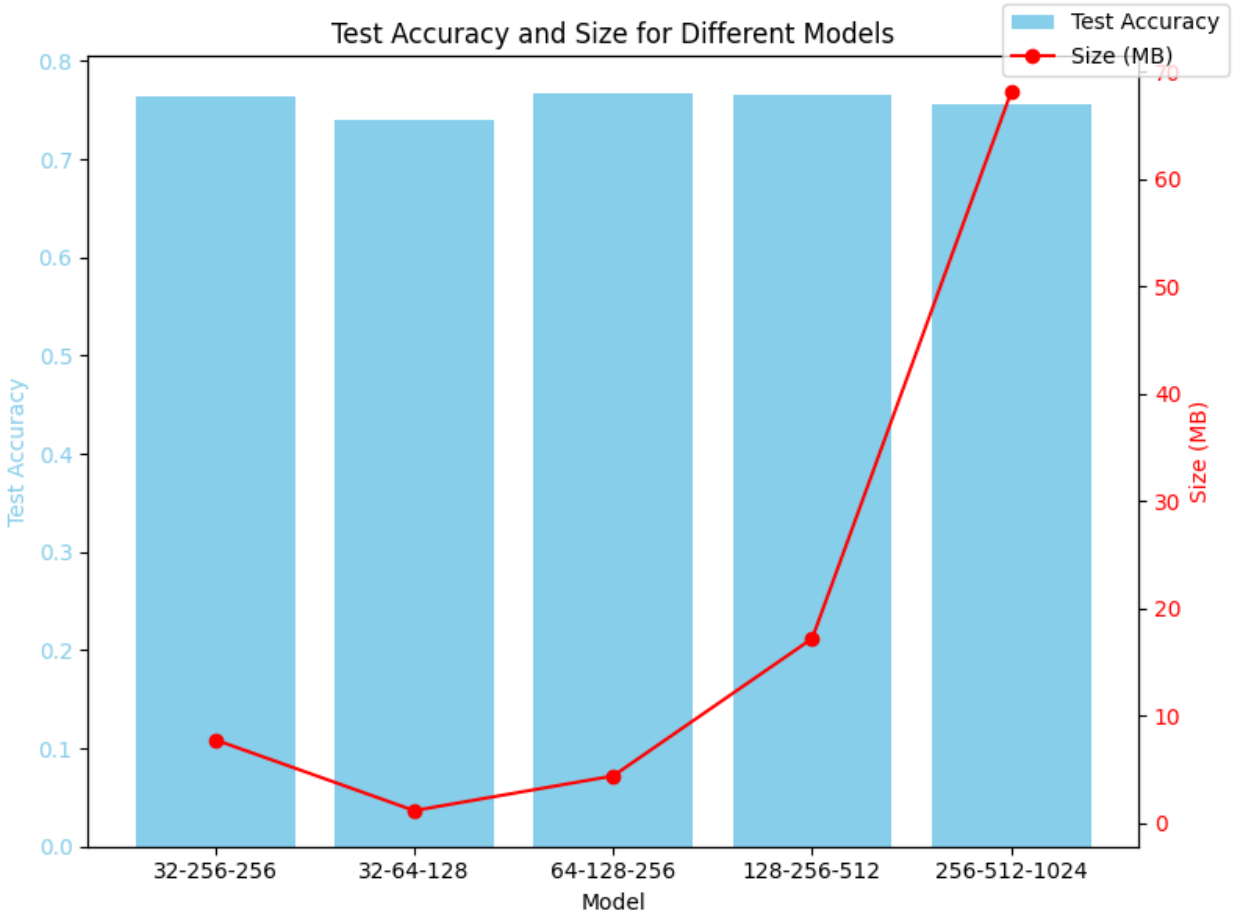

Figure 13: Comparison of test accuracy and model size for each of the final models

5.5 Analysis of the Final Model

The final model chosen was analyzed by creating a confusion matrix of the predicted values of the images versus the true value of the images, which can be seen in Figure 14. This confusion matrix shows that image 3, corresponding to a cat, and image 5, corresponding to a dog, were often confused by the model. Images where the model correctly classified cats and incorrectly classified dogs as cats can be seen in Figure 15. More figures from analysis of the final model can be found in Appendix B.

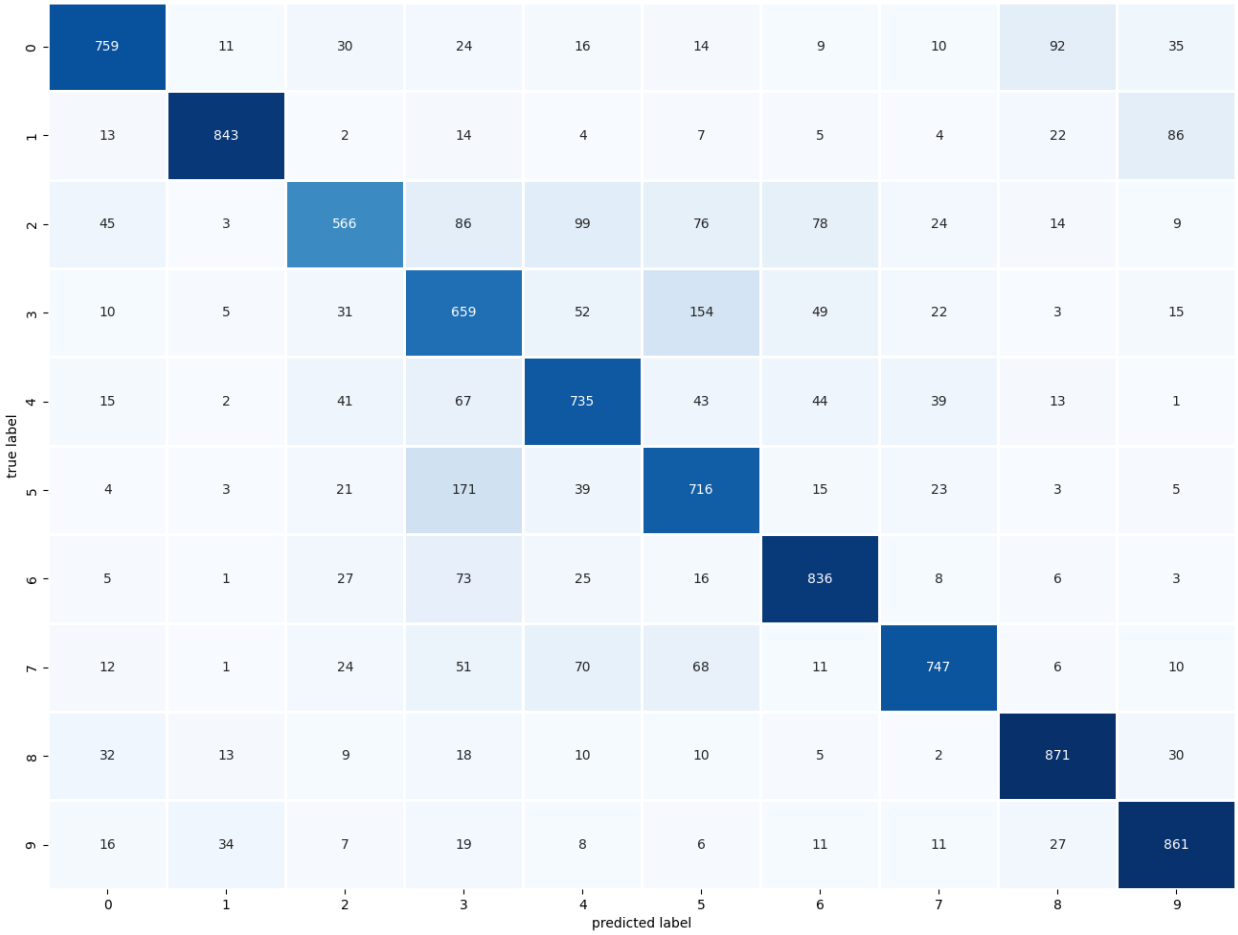

Figure 15: Confusion matrix of correct and incorrect predictions for the model by image type

Figure 15: Cats classified correctly (L) versus dogs classified as cats (R) by the final model

6. Conclusions

Shallow learning neural networks provide ample computing power and complexity to analyze smaller datasets, especially those that have fewer dimensions. These neural networks are not adequate in modeling more complex data, with more layers needed to sort through the data piece by piece to build a successful model. Building deeper neural networks by adding extra layers allows for introducing more parameters to the model, increasing performance in pattern recognition and relationships between data points. This leads to needing more processing power and introducing new error sources, so optimization of the model is key.

Dense layers do not work as well for image processing as convolutional and max pooling layers, and use of patterns of convolutional and max pooling layers allows for better model performance. However, by increasing the complexity of the neural network, overfitting becomes a larger issue as well. Regularization techniques can be used to help prevent overfitting, and dropout regularization, where certain neurons are randomly disabled to try to prevent the model from lining up too closely to the training data at the expense of model robustness, was found to

be the best method for the CIFAR-10 dataset. In conclusion, using a pattern of convolutional layers to classify images can significantly reduce the need for manual classification, though model optimization is necessary to reduce overfitting and lower processing requirements.

Appendix A

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 30, 30, 32)	896
max_pooling2d (MaxPooling2D)	(None, 15, 15, 32)	0
dropout (Dropout)	(None, 15, 15, 32)	0
conv2d_1 (Conv2D)	(None, 13, 13, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 6, 6, 64)	0
dropout_1 (Dropout)	(None, 6, 6, 64)	0
conv2d_2 (Conv2D)	(None, 4, 4, 128)	73,856
max_pooling2d_2 (MaxPooling2D)	(None, 2, 2, 128)	0
dropout_2 (Dropout)	(None, 2, 2, 128)	0
flatten (Flatten)	(None, 512)	0
dense (Dense)	(None, 10)	5,130

Total params: 295,136 (1.13 MB)
Trainable params: 98,378 (384.29 KB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 196,758 (768.59 KB)

Figure A.1: Final model testing for 32-64-128 filter Convolutional neural network

	accuracy	val_accuracy	test_accuracy	loss	val_loss	test_loss	time
count	10.000000	10.000000	10.000000	10.000000	10.000000	10.00000	10.00000
mean	0.737600	0.744100	0.740400	0.750700	0.736300	0.75430	112.70250
std	0.010844	0.015103	0.012817	0.031116	0.038555	0.03646	20.65412
min	0.722000	0.713000	0.711000	0.705000	0.676000	0.70600	79.18800
25%	0.731000	0.735250	0.734750	0.736250	0.713750	0.73125	98.02825
50%	0.738000	0.746500	0.743500	0.750000	0.734500	0.75150	114.92800
75%	0.741750	0.756750	0.747500	0.774000	0.748500	0.76425	121.32700
max	0.755000	0.760000	0.757000	0.796000	0.821000	0.83900	146.12300

Figure A.2: Summary table of ten runs from model in Figure A.1

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 30, 30, 64)	1,792
max_pooling2d (MaxPooling2D)	(None, 15, 15, 64)	0
dropout (Dropout)	(None, 15, 15, 64)	0
conv2d_1 (Conv2D)	(None, 13, 13, 128)	73,856
max_pooling2d_1 (MaxPooling2D)	(None, 6, 6, 128)	0
dropout_1 (Dropout)	(None, 6, 6, 128)	0
conv2d_2 (Conv2D)	(None, 4, 4, 256)	295,168
max_pooling2d_2 (MaxPooling2D)	(None, 2, 2, 256)	0
dropout_2 (Dropout)	(None, 2, 2, 256)	0
flatten (Flatten)	(None, 1024)	0
dense (Dense)	(None, 10)	10,250

Total params: 1,143,200 (4.36 MB)
Trainable params: 381,066 (1.45 MB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 762,134 (2.91 MB)

Figure A.3: Final model testing for 64-128-256 filter Convolutional neural network

	accuracy	val_accuracy	test_accuracy	loss	val_loss	test_loss	time
count	10.000000	10.00000	10.000000	10.000000	10.000000	10.000000	10.000000
mean	0.806500	0.77130	0.766600	0.549700	0.664800	0.691400	111.924000
std	0.022535	0.01212	0.011597	0.064462	0.029877	0.025149	21.393714
min	0.757000	0.74500	0.739000	0.497000	0.632000	0.670000	71.859000
25%	0.795500	0.76525	0.761750	0.500750	0.645500	0.674500	99.719750
50%	0.816500	0.77250	0.771000	0.519500	0.661000	0.684500	124.195500
75%	0.822750	0.78100	0.774500	0.581500	0.674750	0.698750	127.104000
max	0.826000	0.78500	0.778000	0.691000	0.733000	0.755000	131.120000

Figure A.4: Summary table of ten runs from model in Figure A.3

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 30, 30, 128)	3,584
max_pooling2d (MaxPooling2D)	(None, 15, 15, 128)	0
dropout (Dropout)	(None, 15, 15, 128)	0
conv2d_1 (Conv2D)	(None, 13, 13, 256)	295,168
max_pooling2d_1 (MaxPooling2D)	(None, 6, 6, 256)	0
dropout_1 (Dropout)	(None, 6, 6, 256)	0
conv2d_2 (Conv2D)	(None, 4, 4, 512)	1,180,160
max_pooling2d_2 (MaxPooling2D)	(None, 2, 2, 512)	0
dropout_2 (Dropout)	(None, 2, 2, 512)	0
flatten (Flatten)	(None, 2048)	0
dense (Dense)	(None, 10)	20,490

Total params: 4,498,208 (17.16 MB)
Trainable params: 1,499,402 (5.72 MB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 2,998,806 (11.44 MB)

Figure A.5: Final model testing for 128-256-512 filter Convolutional neural network

	accuracy	val_accuracy	test_accuracy	loss	val_loss	test_loss	time
count	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000
mean	0.838700	0.772500	0.766300	0.457800	0.678300	0.707500	186.179800
std	0.015812	0.006042	0.005034	0.046714	0.011643	0.013294	37.120014
min	0.814000	0.762000	0.760000	0.402000	0.662000	0.695000	122.774000
25%	0.828000	0.767250	0.762250	0.418750	0.669500	0.697250	169.999500
50%	0.840500	0.774000	0.765500	0.447500	0.679500	0.704500	183.836000
75%	0.852500	0.778000	0.769750	0.488750	0.684000	0.715250	214.137750
max	0.858000	0.779000	0.774000	0.531000	0.699000	0.736000	239.728000

Figure A.6: Summary table of ten runs from model in Figure A.5

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 30, 30, 256)	7,168
max_pooling2d (MaxPooling2D)	(None, 15, 15, 256)	0
dropout (Dropout)	(None, 15, 15, 256)	0
conv2d_1 (Conv2D)	(None, 13, 13, 512)	1,180,160
max_pooling2d_1 (MaxPooling2D)	(None, 6, 6, 512)	0
dropout_1 (Dropout)	(None, 6, 6, 512)	0
conv2d_2 (Conv2D)	(None, 4, 4, 1024)	4,719,616
max_pooling2d_2 (MaxPooling2D)	(None, 2, 2, 1024)	0
dropout_2 (Dropout)	(None, 2, 2, 1024)	0
flatten (Flatten)	(None, 4096)	0
dense (Dense)	(None, 10)	40,970

Total params: 17,843,744 (68.07 MB)
Trainable params: 5,947,914 (22.69 MB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 11,895,830 (45.38 MB)

Figure A.7: Final model testing for 256-512-1024 filter Convolutional neural network

	accuracy	val_accuracy	test_accuracy	loss	val_loss	test_loss	time
count	10.000000	10.00000	10.00000	10.000000	10.000000	10.00000	10.000000
mean	0.837200	0.76310	0.75580	0.462600	0.713800	0.74180	558.648100
std	0.021238	0.00674	0.00577	0.062352	0.017769	0.02034	122.372486
min	0.815000	0.75100	0.74600	0.345000	0.692000	0.71200	351.332000
25%	0.820250	0.75750	0.75300	0.421000	0.699000	0.72775	499.147500
50%	0.832000	0.76500	0.75500	0.473000	0.713000	0.74050	580.230000
75%	0.852000	0.76825	0.75775	0.517250	0.727000	0.75300	628.572750
max	0.878000	0.77100	0.76700	0.526000	0.740000	0.78100	762.143000

Figure A.7: Summary table of ten runs from model in Figure A.7

Appendix B

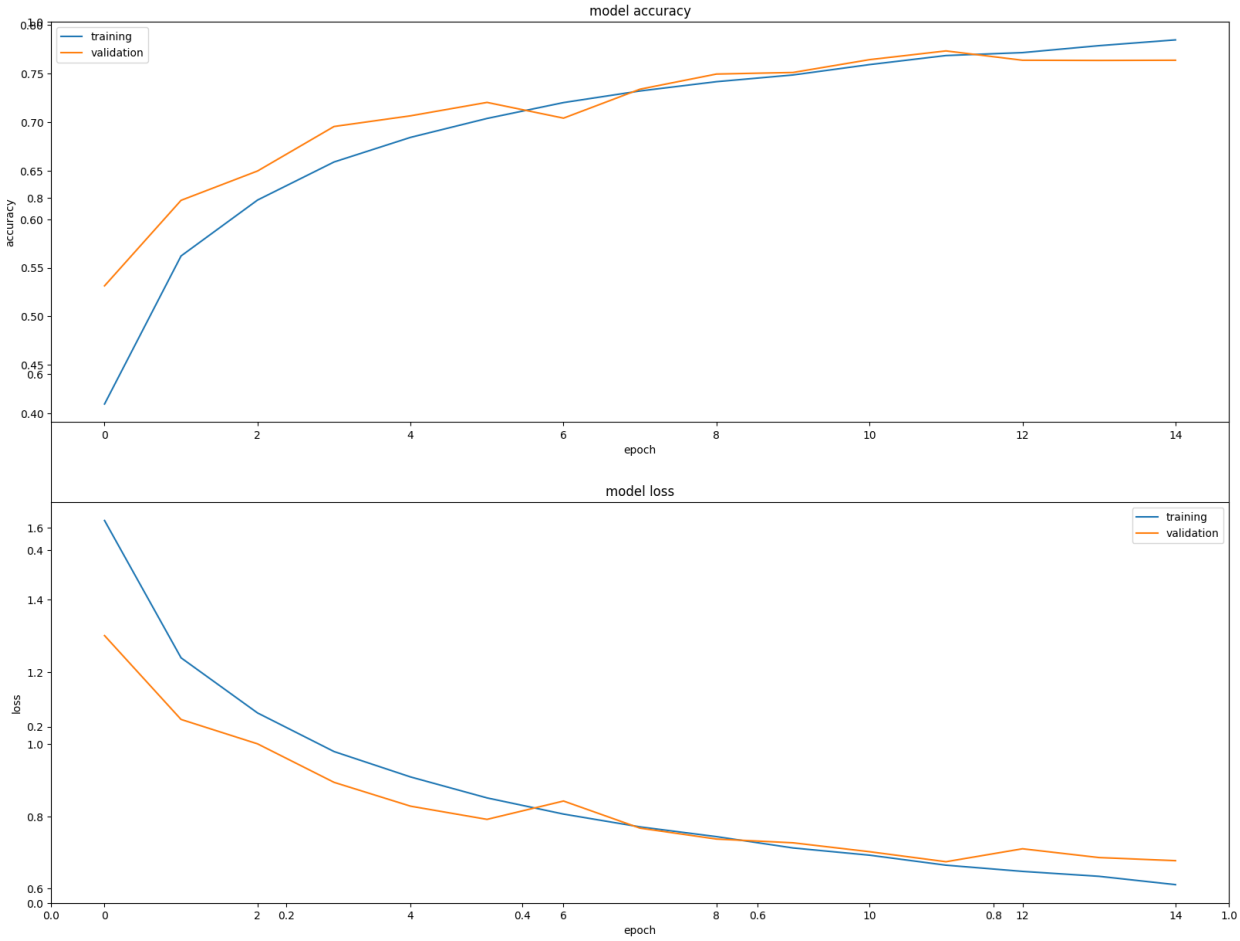

Figure B.1: Training and validation accuracy and loss plots for the best output model

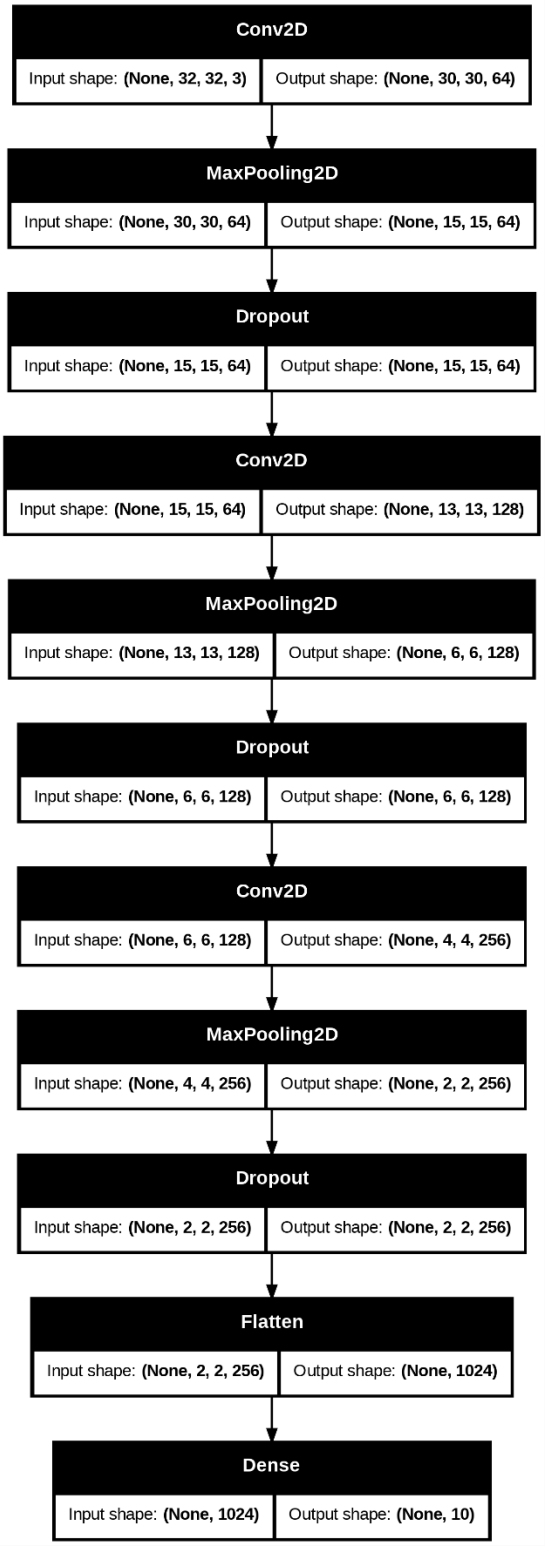

Figure B.2: Keras model plot of final model

References

AI Guy. 2021. “Classification on CIFAR-10.” *Medium.* Published June 5, 2021. https://medium.com/nerd-for-tech/classification-on-cifar-10-32abe456302.

Franky. 2022. “Once Upon a Time in CIFAR-10.” *Medium.* Published August 31, 2022. https://franky07724-57962.medium.com/once-upon-a-time-in-cifar-10-c26bb056b4ce.

Geeksforgeeks. 2024. “Difference between Shallow and Deep Neural Networks.” Updated July 19, 2024. https://www.geeksforgeeks.org/difference-between-shallow-and-deep-neural-networks/.

Gopalakrishnan, Ramamurthi. 2022. “A Simple Nueral Network on MNIST dataset using Pytorch.” *Medium.* July 3, 2022. https://medium.com/@ramamurthi96/a-simple-neural-network-model-for-mnist-using-pytorch-4b8b148ecbdc.

Khan, Azim. 2024. “A Beginner’s Guide to Deep Learning with MNIST Dataset.” *Medium.* April 16, 2024. https://medium.com/@azimkhan8018/a-beginners-guide-to-deep-learning-with-mnist-dataset-0894f7183344.

Mistarz, Michael. 2025. “Identifying Handwritten Digits Using a Single Layer Dense Neural Network.” Published January 26, 2025. https://github.com/mistmr7/MSDS458_Single-Layer-DNN-on-MNIST-Data.

Ombaval. 2024. “Building a Simple Neural Network from Scratch for MNIST Digit Recognition without using TensorFlow/PyTorch only using Numpy.” *Medium.* May 14, 2024. https://medium.com/@ombaval/building-a-simple-neural-network-from-scratch-for-mnist-digit-recognition-without-using-7005a7733418.

Pratama, Muhammad Adisatriyo. 2020. “Comparing Image Classification with Dense Neural Network and Convolutional Neural Network.” *Medium*. Published December 28, 2020. https://medium.com/analytics-vidhya/comparing-image-classification-with-dense-neural-network-and-convolutional-neural-network-5f376582a695.

Shanmugamani, Rajalingappaa. 2018. *Deep Learning for computer vision: expert techniquest to train advanced neural networks using TensorFlow and Keras.* Birmingham, England: Paths International Ltd. 2018 (1): Ch. 1. https://learning.oreilly.com/library/view/deep-learning-for/9781788295628/78521051-5153-4a75-b149-5219677cdd70.xhtml

Tamanna. 2023. “Exploring Convolutional Neural Networks: Architecture, Steps, Use Cases, and Pros and Cons.” *Medium*. April 24, 2023. https://medium.com/@tam.tamanna18/exploring-convolutional-neural-networks-architecture-steps-use-cases-and-pros-and-cons-b0d3b7d46c71.

TensorFlow. n.d. “Module: tf.keras.datasets.” Accessed February 4, 2025. TensorFlow documentation v2.16.1. https://www.tensorflow.org/api_docs/python/tf/keras/datasets.

TensorFlow. n.d. “tf.keras.callbacks.EarlyStopping.” Accessed January 24, 2025. TensorFlow documentation v2.16.1. https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/EarlyStopping.

Watson, Nells. 2017. “Understanding Convolutional Neural Networks (CNNs).” Infinite Skills. https://learning.oreilly.com/course/understanding-convolutional-neural/9781491978931/.

Zhao, Jiantao, and Lili Li. 2021. “Research on Image Classification Algorithm Based on Convolutional Neural Network-TensorFlow.” September 2021. *Journal of Physics: Conference Series* 2010 (1): 1-9. https://iopscience.iop.org/article/10.1088/1742-6596/2010/1/012096/pdf.