Deliverable #2

Karl Satchi E. Navida
Deep Learning - MSCS23A
April 10, 2025

I. Dataset

CIFAR-10 is a dataset composed of 60k images divided into five (5) batches and a test batch. It came from **Papers with Code** website and was downloaded directly from the CS Toronto Edu – CIFAR-10 website.

As previously stated, the CIFAR-10 dataset consists of 60k, 32x32 color images in 10 classes, with each class containing 6k images. The five batches contain a combined 50k images, with the 10k images under the test dataset. And under the test dataset, the 10k images is equally divided into 1k images per class.

II. Classification Results

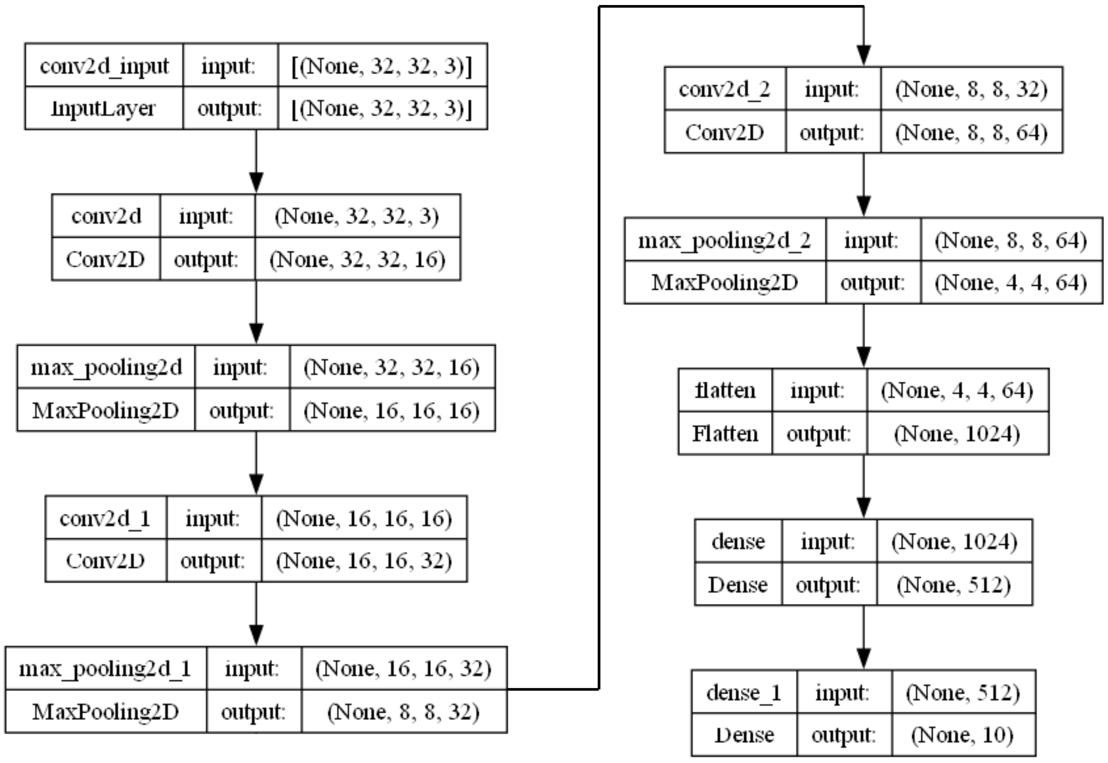
The results of this deliverable will be separated into five (5) sections, documenting each configuration used until the highest accuracy was attained. These configurations are only ran once and thus, may as well just be an approximation of accuracy around the provided one in this results.

1. **Initial**

   The **initial** configuration is the very first configuration used on its initial run and test. This configuration only has/uses three (3) `Conv2D` layers, with each `Conv2D` layer followed by a `MaxPooling2D` layer. In the hidden layer, `Dropout` layer was still not use in this stage.

   The `steps_per_epoch` (SPE) was also set to 100 in this run while the `validation_steps` (VS) was set to 50, leaving a lot of data unused during the fitting of the model.

   Figure 1 shows a simple model diagram of the created neural network for this deliverable.



conv2d_input | input: | [(None, 32, 32, 3)]
InputLayer | output: | [(None, 32, 32, 3)]
conv2d | input: | (None, 32, 32, 3)
Conv2D | output: | (None, 32, 32, 16)
max_pooling2d | input: | (None, 32, 32, 16)
MaxPooling2D | output: | (None, 16, 16, 16)
conv2d_1 | input: | (None, 16, 16, 16)
Conv2D | output: | (None, 16, 16, 32)
max_pooling2d_1 | input: | (None, 16, 16, 32)
MaxPooling2D | output: | (None, 8, 8, 32)
conv2d_2 | input: | (None, 8, 8, 32)
Conv2D | output: | (None, 8, 8, 64)
max_pooling2d_2 | input: | (None, 8, 8, 64)
MaxPooling2D | output: | (None, 4, 4, 64)
flatten | input: | (None, 4, 4, 64)
Flatten | output: | (None, 1024)
dense | input: | (None, 1024)
Dense | output: | (None, 512)
dense_1 | input: | (None, 512)
Dense | output: | (None, 10)

Figure 1: Model for "initial" configuration.

This configuration managed to reach only 59.09% accuracy as seen in Figures 2 and 3. This shows the without any regularization such as `Dropout` or `BatchNormalization`, the training will be less accurate due to some of the neurons having to learn broad features or the values not being normalized.

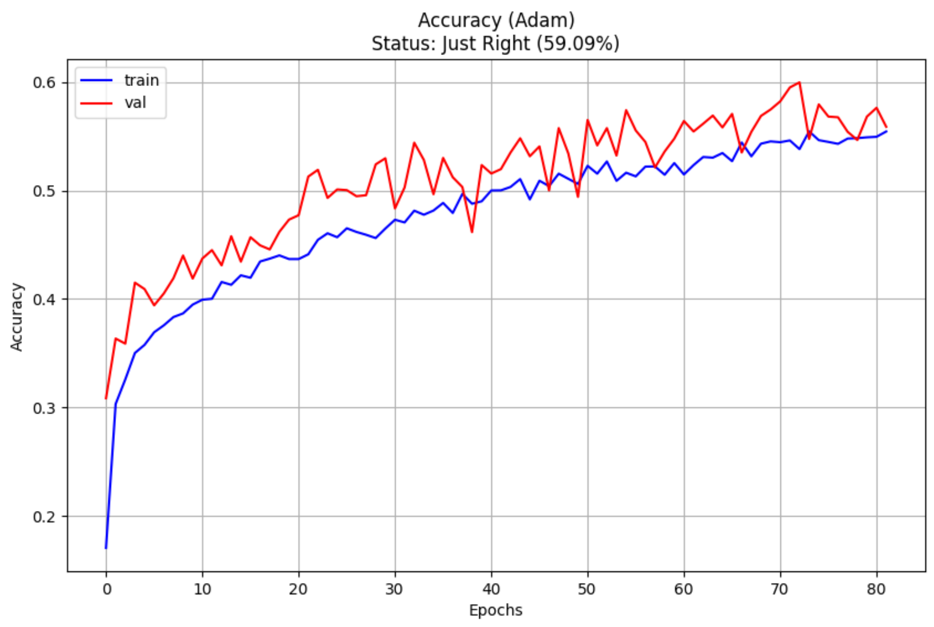
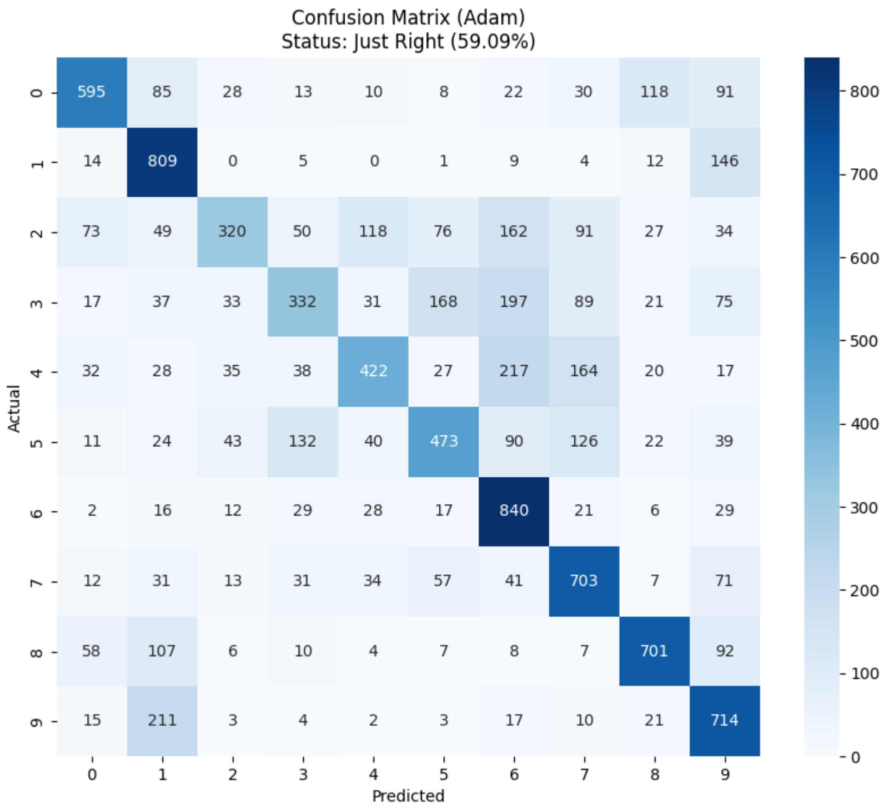


Figure 2: History Plot based on its `fit()` method.



	0	1	2	3	4	5	6	7	8	9
0	595	85	28	13	10	8	22	30	118	91
1	14	809	0	5	0	1	9	4	12	146
2	73	49	320	50	118	76	162	91	27	34
3	17	37	33	332	31	168	197	89	21	75
4	32	28	35	38	422	27	217	164	20	17
5	11	24	43	132	40	473	90	126	22	39
6	2	16	12	29	28	17	840	21	6	29
7	12	31	13	31	34	57	41	703	7	71
8	58	107	6	10	4	7	8	7	701	92
9	15	211	3	4	2	3	17	10	21	714

Figure 3: The confusion matrix of the "initial" model confriguration, showing the amount of correct prediction along the diagonal line starting from the top left.

2. **Training Steps Modification (No Dropout & Batch Normalization)**

The second configuration tested; it was after the initial configuration was modified lightly so that the SPE and VS utilizes the training data to its maximum. By doing this, the following formula was used:

$$floor(\frac{x}{y})$$

Whereas:

- $\boldsymbol{x}$ is the amount of train data (in this case, 40k for the training data and 10k for the validation data)
- $\boldsymbol{y}$ is the batch size (in this case, 32 for both the training and validation data)

Following the same model configuration, this configuration resulted to a much higher accuracy and output as seen in Figures 4 and 5.

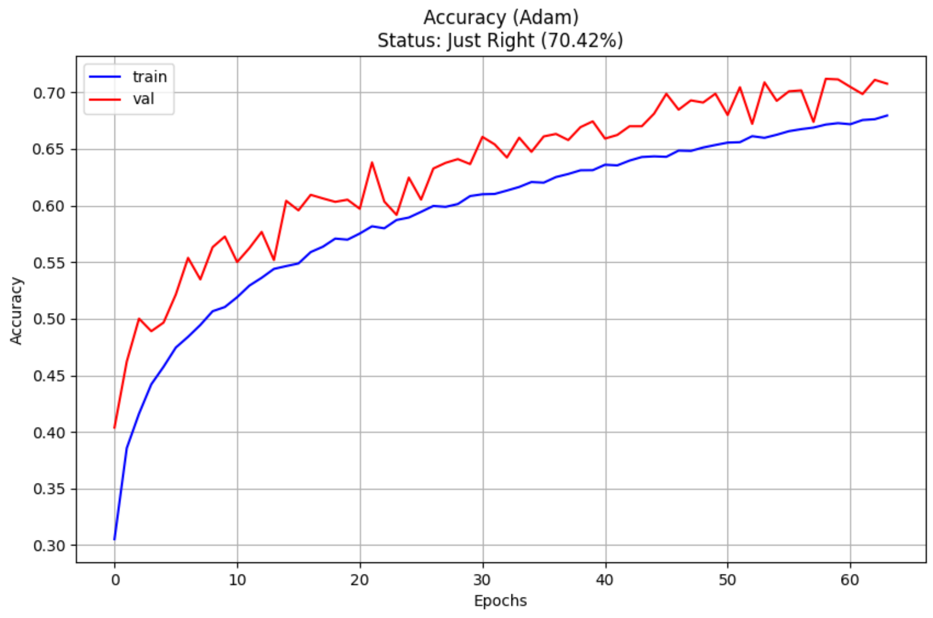


Figure 4: History Plot based on its `fit()` method.

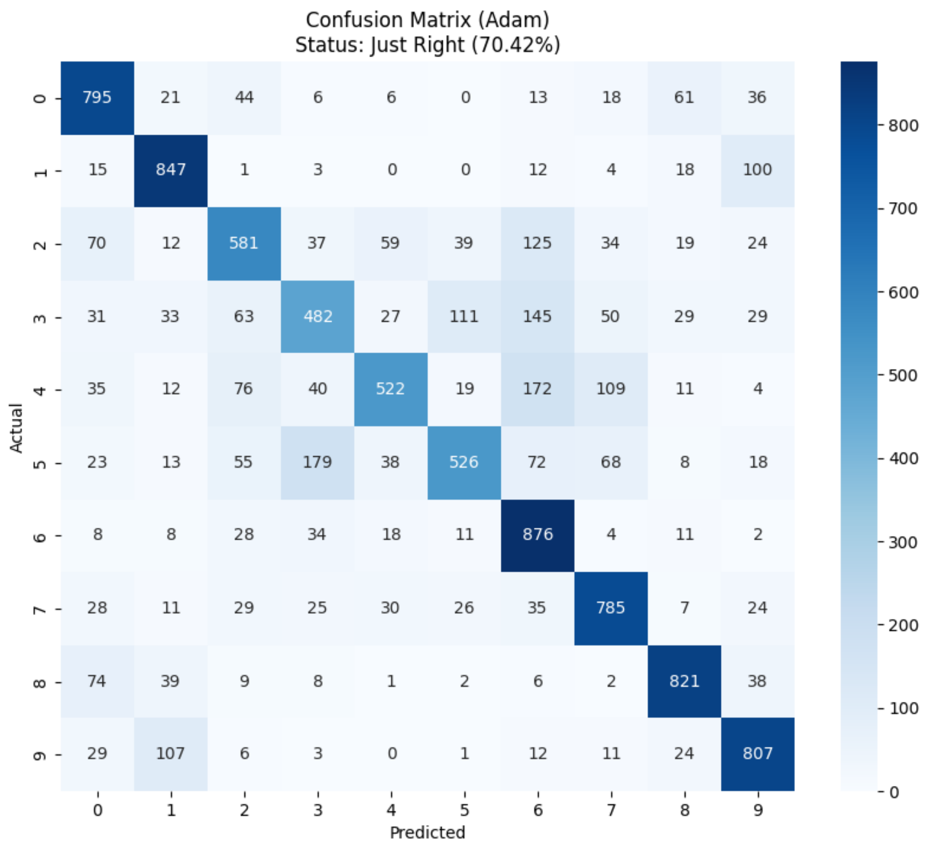


Figure 5: The confusion matrix of the second model configuration, showing the amount of correct prediction along the diagonal line starting from the top left.

This alone made a significance leap in improvement, showing a 28.87% increase in accuracy, making it jump up to 70.42% from the previous 50.09% accuracy.

3. **Introduction of Dropout and Batch Normalization (No Reduced Learning Rate)**
   This third configuration introduces the `Dropout` and `BatchNormalization` layers, resulting to a further small increase in accuracy. This new architecture introduces a `BatchNormalization` layer after every `Conv2D` layer, normalizing its values before feeding it to the `MaxPooling2D` layer. Finally, the `Dropout` layer is placed just before the output layer, which has a 40% rate. Figure 6 shows the updated model diagram while Figures 7 and 8 shows the training history and accuracy.

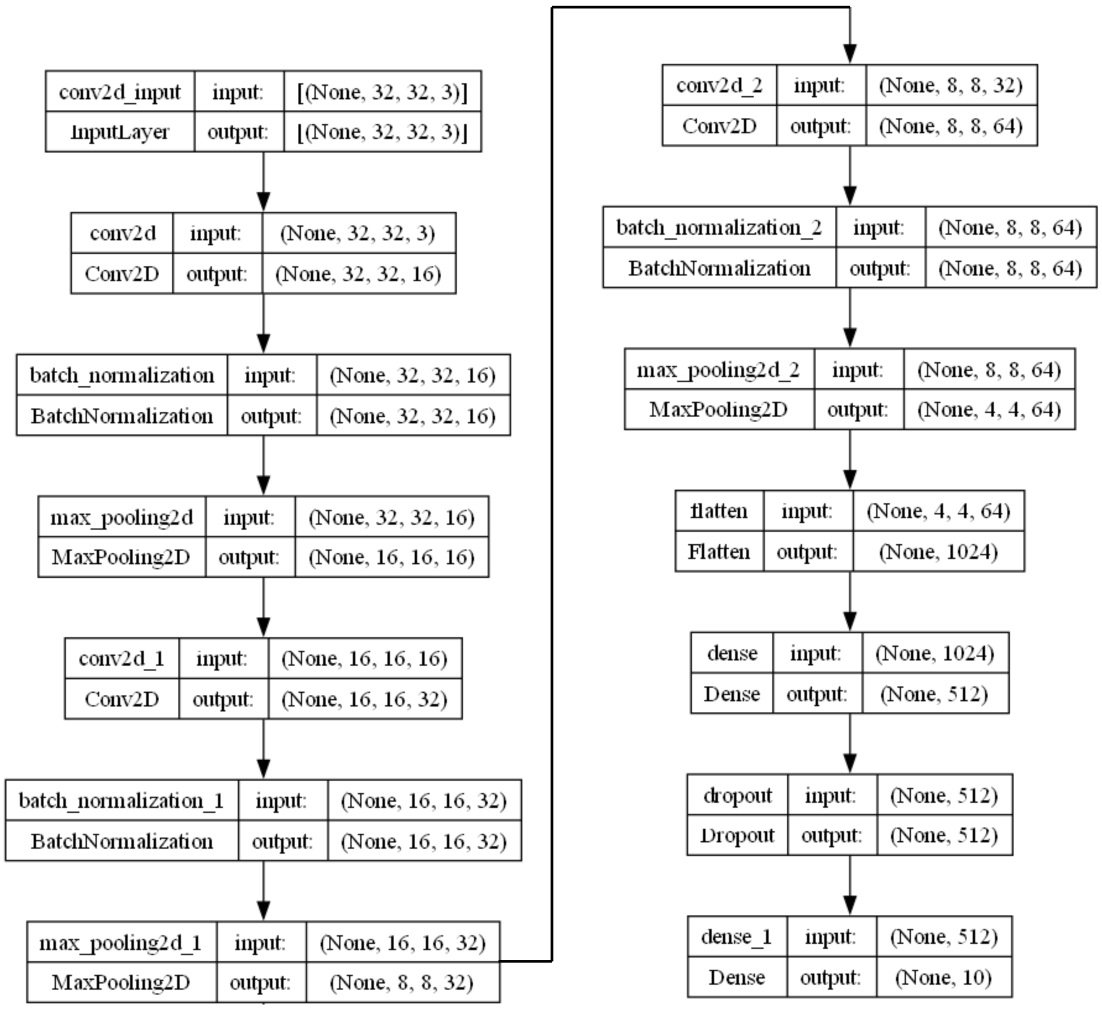

conv2d_input | InputLayer | input: [(None, 32, 32, 3)] | output: [(None, 32, 32, 3)]
conv2d | Conv2D | input: (None, 32, 32, 3) | output: (None, 32, 32, 16)
batch_normalization | BatchNormalization | input: (None, 32, 32, 16) | output: (None, 32, 32, 16)
max_pooling2d | MaxPooling2D | input: (None, 32, 32, 16) | output: (None, 16, 16, 16)
conv2d_1 | Conv2D | input: (None, 16, 16, 16) | output: (None, 16, 16, 32)
batch_normalization_1 | BatchNormalization | input: (None, 16, 16, 32) | output: (None, 16, 16, 32)
max_pooling2d_1 | MaxPooling2D | input: (None, 16, 16, 32) | output: (None, 8, 8, 32)
conv2d_2 | Conv2D | input: (None, 8, 8, 32) | output: (None, 8, 8, 64)
batch_normalization_2 | BatchNormalization | input: (None, 8, 8, 64) | output: (None, 8, 8, 64)
max_pooling2d_2 | MaxPooling2D | input: (None, 8, 8, 64) | output: (None, 4, 4, 64)
flatten | Flatten | input: (None, 4, 4, 64) | output: (None, 1024)
dense | Dense | input: (None, 1024) | output: (None, 512)
dropout | Dropout | input: (None, 512) | output: (None, 512)
dense_1 | Dense | input: (None, 512) | output: (None, 10)

Figure 6: Model diagram after adding the two new regularization layers.

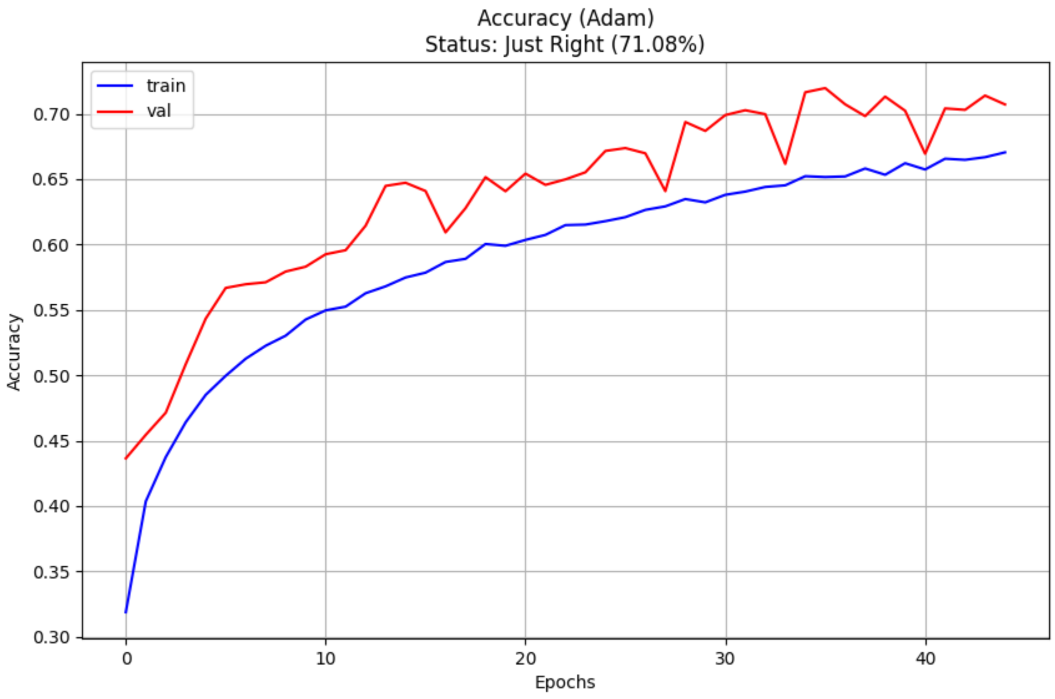


Figure 7: History Plot based on its `fit()` method.

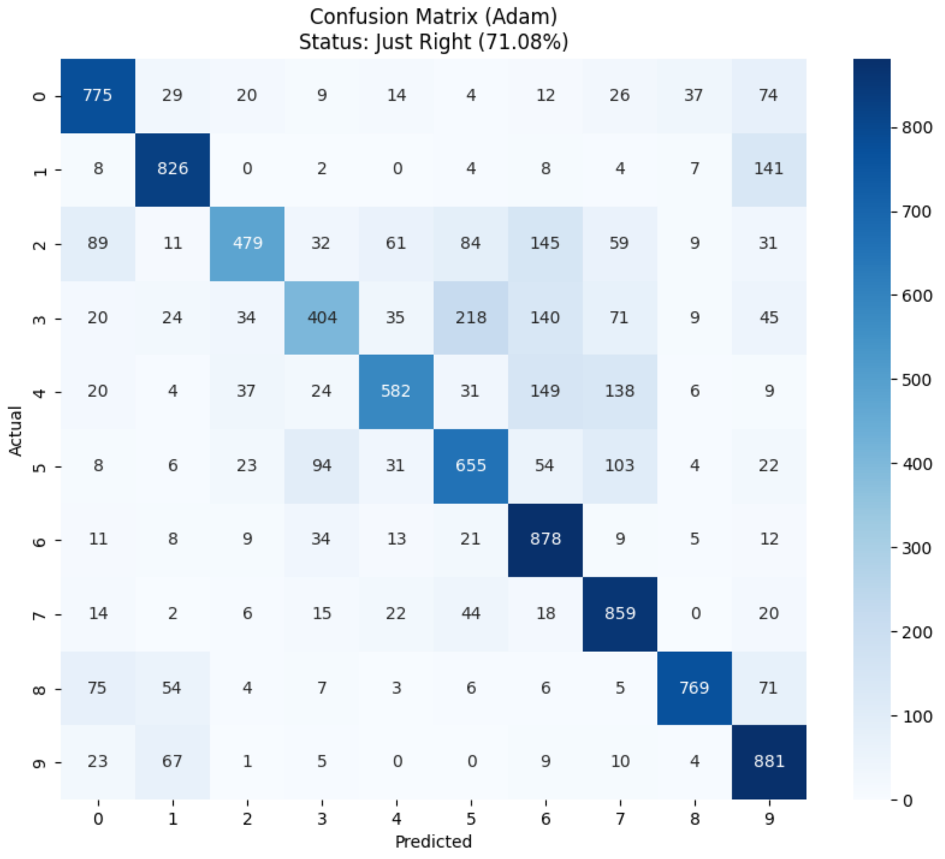


Figure 8: The confusion matrix of the third model configuration, showing the amount of correct prediction along the diagonal line starting from the top left.

4. **Addition of Callbacks (Three Convolutional 2D Layer)**

   This configuration introduces callbacks to the `fit()` method; adding both the `EarlyStopping` and `ReduceLROnPlateau` callbacks. Both callbacks monitor the `val_loss` metrics with a 10 and 5 `patience`, respectively.

   The `EarlyStopping` callback was set higher than the `ReduceLROnPlateau` to allow the model to have a wiggle room before the former callback kicks in. Furthermore, the `ReduceLROnPlateau` has its `factor` set to 50% (0.5) and its minimum learning rate to 1e-6 (0.0000001).

   This application provided minimal increase in accuracy, but the result may just be its minimum in its range.

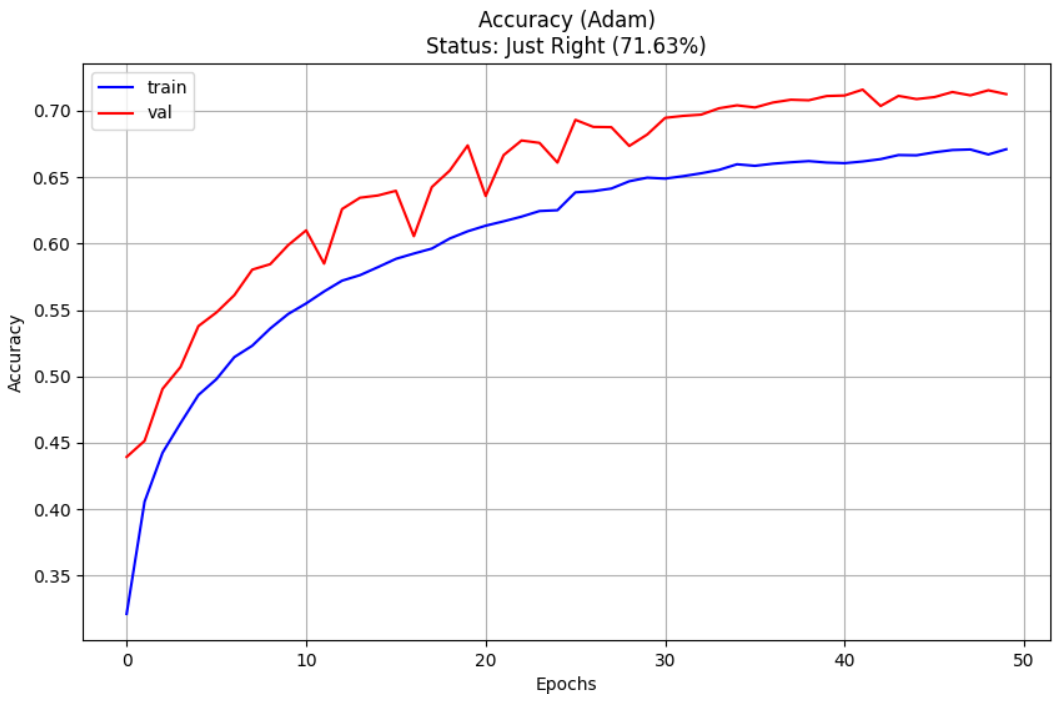


Figure 9: History Plot based on its `fit()` method.

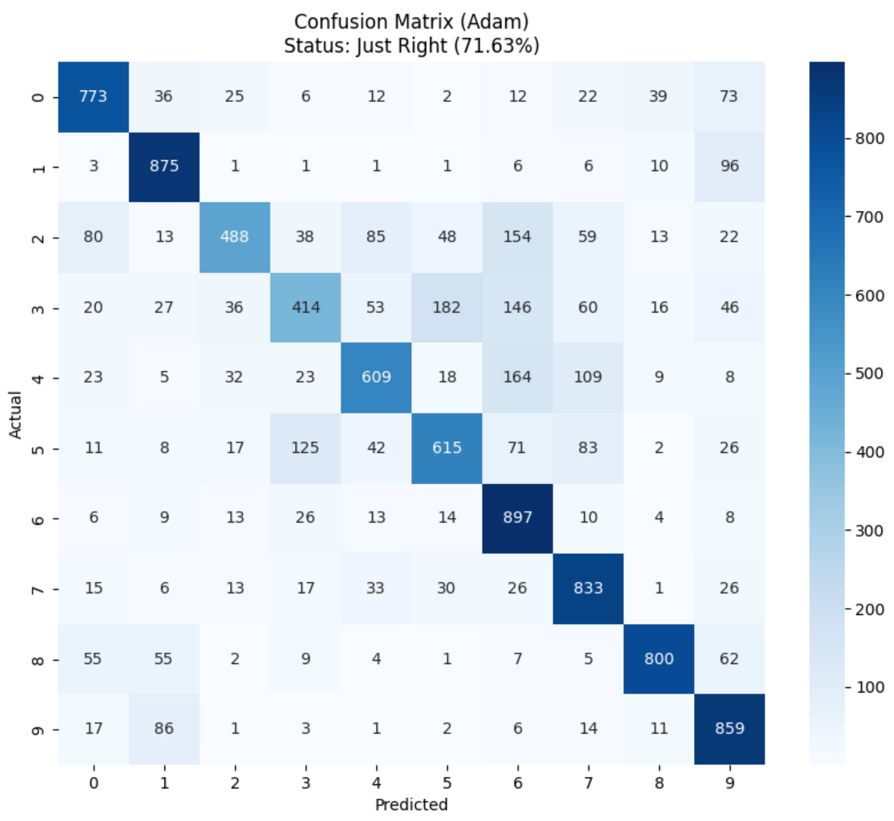
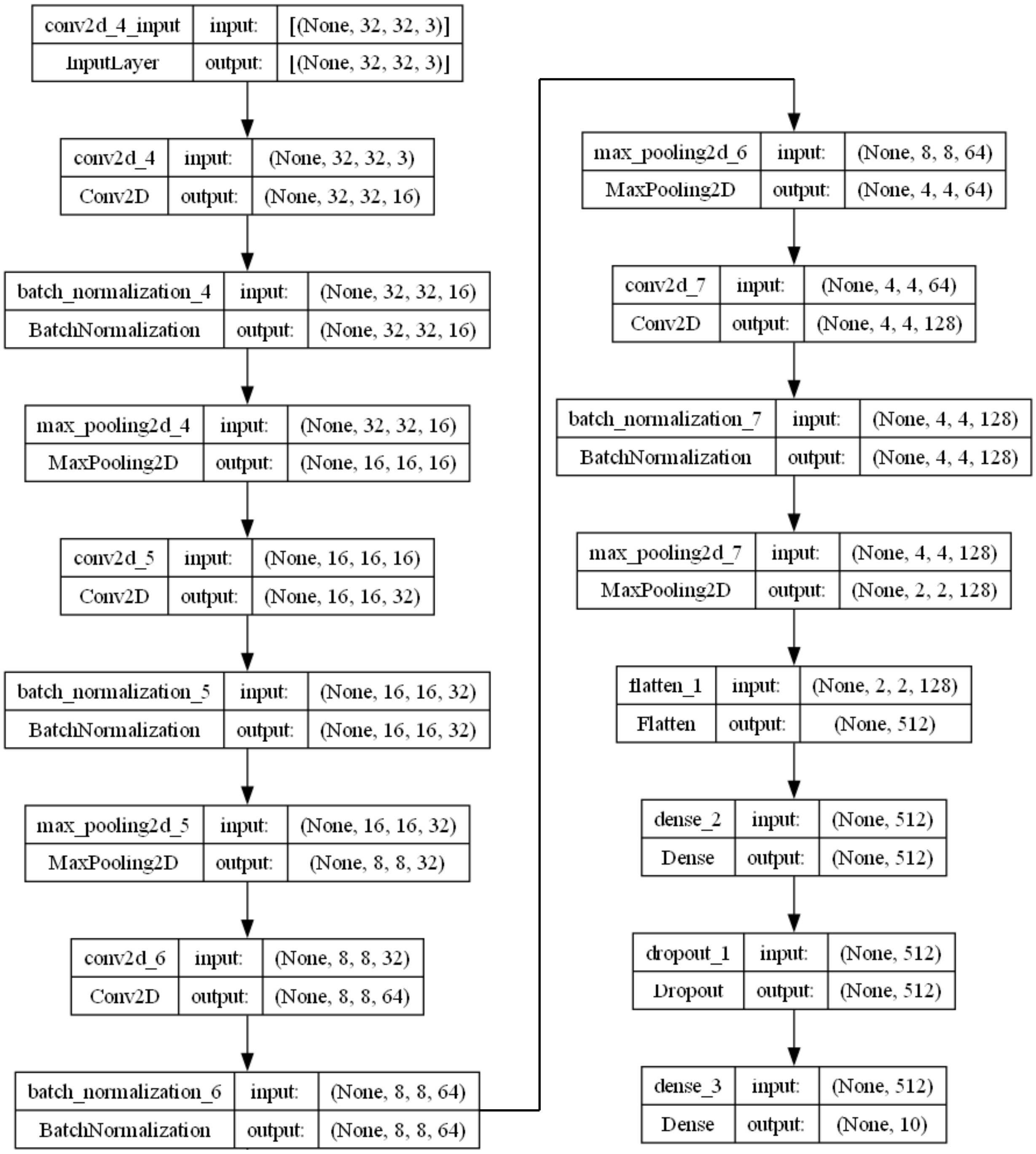


Figure 10: The confusion matrix of the fourth model configuration, showing the amount of correct prediction along the diagonal line starting from the top left.

5. **Final (Original)**

The final configuration has the highest accuracy of them all. Simply due to the addition of a fourth `Conv2D` layer with its `BatchNormalization` and `MaxPooling2D` layers.

The updated model diagram in Figure 11 shows the changes made to the model's architecture.



conv2d_4_input | input: | [(None, 32, 32, 3)]
InputLayer | output: | [(None, 32, 32, 3)]
conv2d_4 | input: | (None, 32, 32, 3)
Conv2D | output: | (None, 32, 32, 16)
batch_normalization_4 | input: | (None, 32, 32, 16)
BatchNormalization | output: | (None, 32, 32, 16)
max_pooling2d_4 | input: | (None, 32, 32, 16)
MaxPooling2D | output: | (None, 16, 16, 16)
conv2d_5 | input: | (None, 16, 16, 16)
Conv2D | output: | (None, 16, 16, 32)
batch_normalization_5 | input: | (None, 16, 16, 32)
BatchNormalization | output: | (None, 16, 16, 32)
max_pooling2d_5 | input: | (None, 16, 16, 32)
MaxPooling2D | output: | (None, 8, 8, 32)
conv2d_6 | input: | (None, 8, 8, 32)
Conv2D | output: | (None, 8, 8, 64)
batch_normalization_6 | input: | (None, 8, 8, 64)
BatchNormalization | output: | (None, 8, 8, 64)
max_pooling2d_6 | input: | (None, 8, 8, 64)
MaxPooling2D | output: | (None, 4, 4, 64)
conv2d_7 | input: | (None, 4, 4, 64)
Conv2D | output: | (None, 4, 4, 128)
batch_normalization_7 | input: | (None, 4, 4, 128)
BatchNormalization | output: | (None, 4, 4, 128)
max_pooling2d_7 | input: | (None, 4, 4, 128)
MaxPooling2D | output: | (None, 2, 2, 128)
flatten_1 | input: | (None, 2, 2, 128)
Flatten | output: | (None, 512)
dense_2 | input: | (None, 512)
Dense | output: | (None, 512)
dropout_1 | input: | (None, 512)
Dropout | output: | (None, 512)
dense_3 | input: | (None, 512)
Dense | output: | (None, 10)

Figure 11: Model diagram after adding the new layers.

With this new configuration, the model's accuracy jumped a measly 6.62% from its previous 71.63%, returning a 76.71% accuracy with this modification.

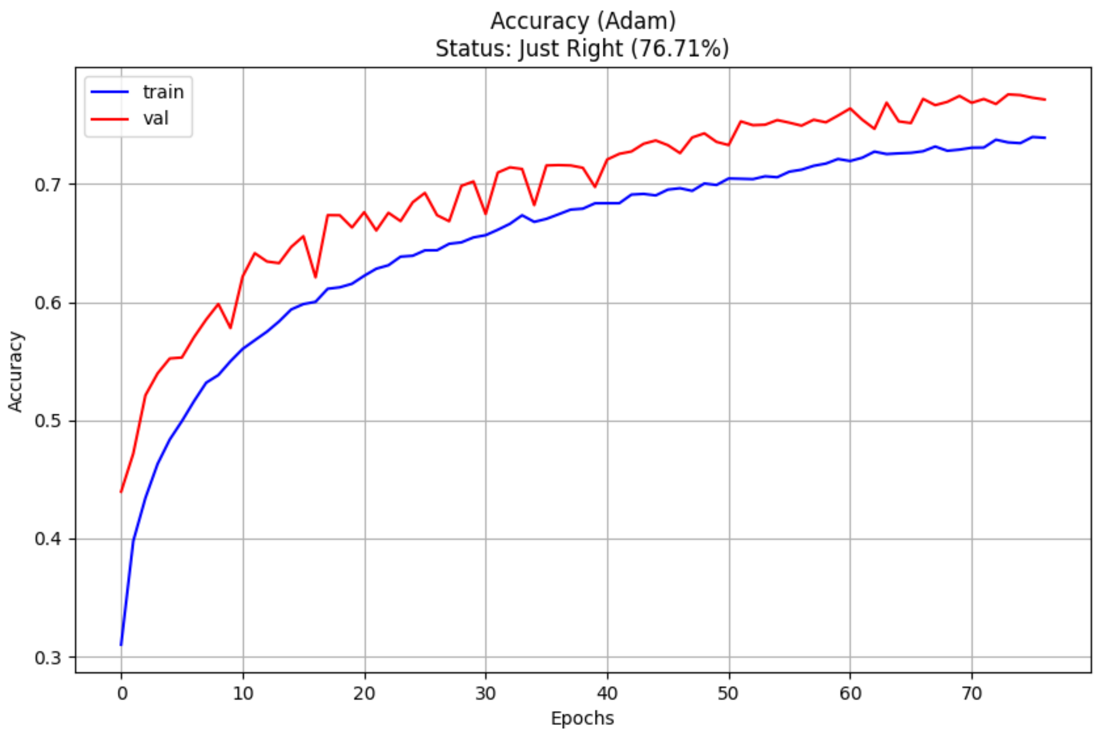
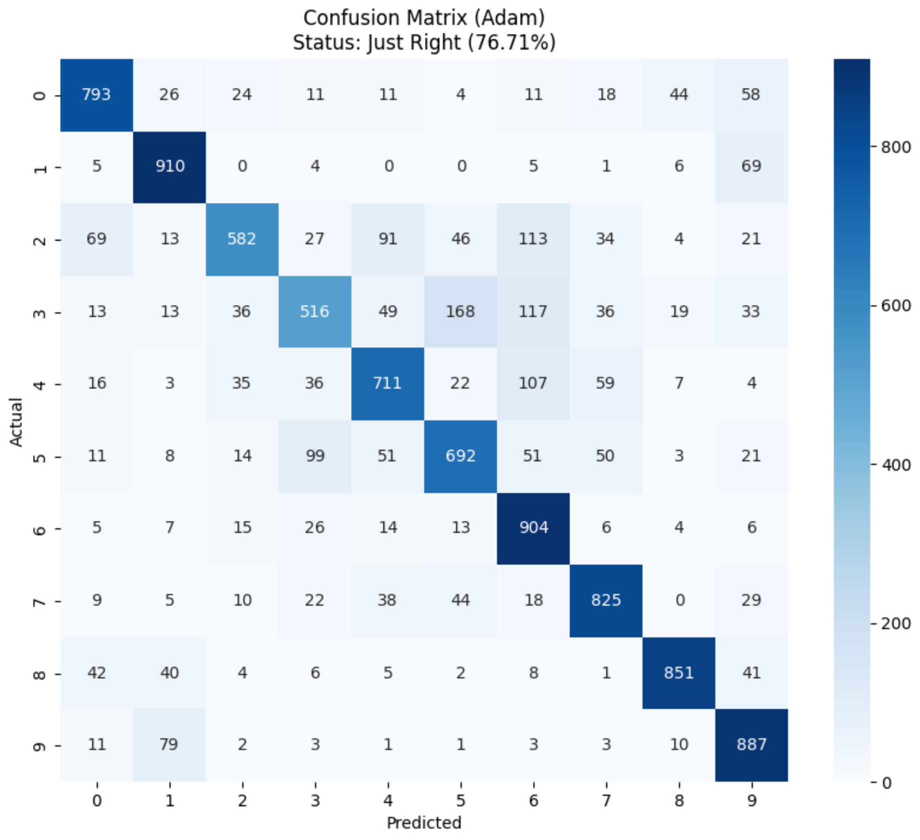


Figure 12: History Plot based on its `fit()` method.



	0	1	2	3	4	5	6	7	8	9
0	793	26	24	11	11	4	11	18	44	58
1	5	910	0	4	0	0	5	1	6	69
2	69	13	582	27	91	46	113	34	4	21
3	13	13	36	516	49	168	117	36	19	33
4	16	3	35	36	711	22	107	59	7	4
5	11	8	14	99	51	692	51	50	3	21
6	5	7	15	26	14	13	904	6	4	6
7	9	5	10	22	38	44	18	825	0	29
8	42	40	4	6	5	2	8	1	851	41
9	11	79	2	3	1	1	3	3	10	887

Figure 13: The confusion matrix of the fifth and last model configuration, showing the amount of correct prediction along the diagonal line starting from the top left.