

CS-671 Assignment-2

Group 18: Akhilesh Devrari, Chirag Vashist, Prabhakar Prasad

April 8, 2019

Contents

1	Aim	4
2	Datasets	4
3	Task-1 : 10-Class MNIST	4
3.1	Dataset Preparation	4
3.2	Classifier-1	4
3.2.1	Network Architecture	4
3.2.2	Results	5
3.2.3	Inferences	5
4	Task-1 : 96-Class Line-Dataset	7
4.1	Dataset Preparation	7
4.2	Classifier-1	7
4.2.1	Network Architecture	7
4.2.2	Results	7
4.2.3	Inferences	8
5	Task-2 : Multi-headed Classification for Line Data-set	9
5.1	Dataset Preparation	9
5.2	Classifier-1	9
5.2.1	Network Architecture	9
5.2.2	Results	10
5.2.3	Inferences	13
6	Task-3 : 10-Class MNIST Dataset	13
6.1	Architecture	13
6.2	Test Image 1	14
6.2.1	Visualizing Intermediate Layer Activations	14
6.2.2	Visualizing Convnet Filters	15
6.2.3	Visualizing Heatmaps of class activations	15
6.3	Test Image 2	16
6.3.1	Visualizing Intermediate Layer Activations	16
6.3.2	Visualizing Convnet Filters	17
6.3.3	Visualizing Heatmaps of class activations	17
6.4	Test Image 3	17
6.4.1	Visualizing Intermediate Layer Activations	18
6.4.2	Visualizing Convnet Filters	18
6.4.3	Visualizing Heatmaps of class activations	18
7	Task 3 : 96-Class Line Dataset	19
7.1	Architecture	19
7.2	Test Image 1	21
7.2.1	Visualizing Intermediate Layer Activations	21
7.2.2	Visualizing Convnet Filters	22
7.3	Test Image 2	22
7.3.1	Visualizing Intermediate Layer Activations	22
7.3.2	Visualizing Convnet Filters	23
7.4	Test Image 3	23

7.4.1	Visualizing Intermediate Layer Activations	23
7.4.2	Visualizing Convnet Filters	24

1 Aim

To make a CNN model from scratch to classify:

1. MNIST dataset into 10 classes.
2. Images of the line dataset consisting of 96 classes based on 4 kinds of variations(length, width, color, angle).

2 Datasets

Following datasets are provided:

1. 28*28 MNIST Dataset with 60000 images for training 10000 images for testing
2. 28*28*3 Coloured Lines on Black Background (CLBB) Dataset with 750 images of each class for training and 250 images of each class for testing.

3 Task-1 : 10-Class MNIST

3.1 Dataset Preparation

The standard MNIST dataset can be directly loaded by using `tf.keras.datasets.mnist.load_data()`. We reshaped training and testing data to 4-dimensions(60000, 28, 28, 1) to be able to work with Keras API.

1. Total Data-Samples : 70000
2. Training Samples : 60000
3. Testing Samples : 10000

3.2 Classifier-1

3.2.1 Network Architecture

1. Layer-1 : 7x7 Convolutional Layer with 32 filters and stride of 1
2. Layer-2 : ReLU Activation Layer
3. Layer-3 : Batch Normalization Layer
4. Layer-3 : 2x2 Max Pooling layer with a stride of 2
5. Layer-4 : fully connected layer with 1024 output units
6. Layer-5 : ReLU Activation Layer
7. Layer-5 : Flatten
8. Output-Layer : fully connected layer with 10 output units using softmax activation

Adam optimizer and `sparse_categorical_crossentropy` was used to compile the model.

3.2.2 Results

$$\text{Confusion Matrix} = \begin{bmatrix} 978 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1130 & 0 & 0 & 1 & 1 & 2 & 1 & 0 & 0 \\ 0 & 2 & 1019 & 0 & 2 & 0 & 1 & 7 & 1 & 0 \\ 2 & 0 & 1 & 988 & 0 & 16 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 974 & 0 & 2 & 0 & 2 & 4 \\ 1 & 0 & 0 & 1 & 0 & 888 & 1 & 0 & 1 & 0 \\ 3 & 3 & 0 & 0 & 3 & 6 & 941 & 0 & 2 & 0 \\ 0 & 3 & 5 & 1 & 1 & 0 & 0 & 1010 & 2 & 6 \\ 5 & 1 & 5 & 1 & 1 & 2 & 3 & 0 & 953 & 3 \\ 2 & 2 & 3 & 0 & 10 & 5 & 0 & 4 & 0 & 983 \end{bmatrix}$$

1. Training Accuracy : 99.88%
2. Testing Accuracy : 98.64%
3. Average F1-score : 98.63%
4. Average Precision : 98.62%
5. Average Recall : 98.64%

3.2.3 Inferences

1. From the accuracy-epochs graph, we can see the train accuracy of model increases, and train loss decreases. While test accuracy also increases, but not the same amount. This shows that model is getting over-fitted and probably we don't even need this more number of epochs.
2. Accuracy obtained is quite high i.e. 99.88% on training data and bit less i.e. 98.64% on test data, which also indicates over-fitting. To decrease over-fitting we added a 0.2 dropout layer before flattening, but results were similar. Nonetheless, accuracy of model is very good.

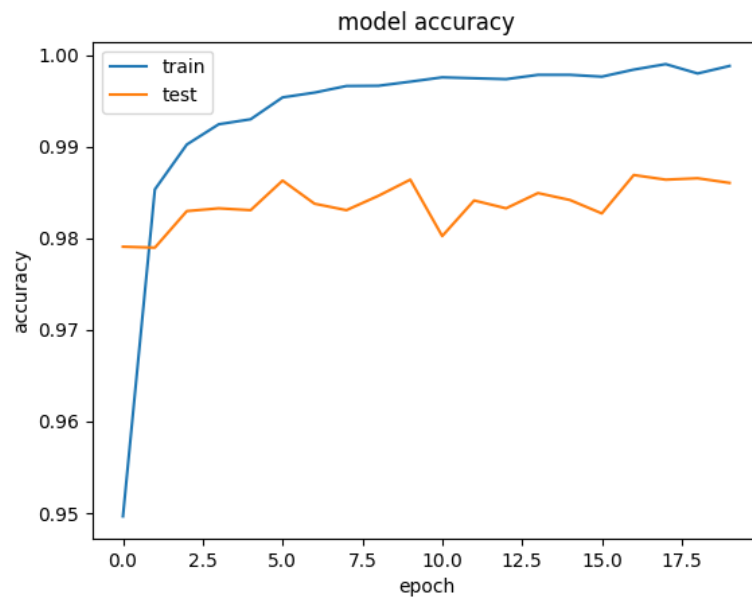


Figure 1: MNIST Data-Set : Accuracy vs Epochs

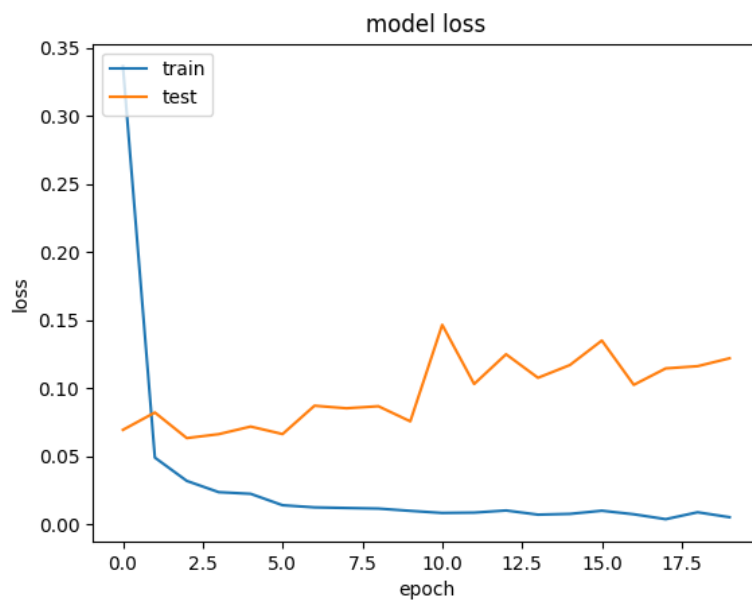


Figure 2: MNIST Data-Set : Loss vs Epochs

4 Task-1 : 96-Class Line-Dataset

4.1 Dataset Preparation

We created data for this classification in classifier's code itself. Creating the images is computationally less expansive than loading them from hard-disk and this does not affect the performance of classifier in any other way. We reshaped training and testing data to 4-dimensions(60000, 28, 28, 3) to be able to work with Keras API.

1. Total Data-Samples : 96000
2. Training Samples : 72000
3. Testing Samples : 24000

4.2 Classifier-1

4.2.1 Network Architecture

1. Layer-1 : 7x7 Convolutional Layer with 32 filters and stride of 1
2. Layer-2 : ReLU Activation Layer
3. Layer-3 : Batch Normalization Layer
4. Layer-3 : 2x2 Max Pooling layer with a stride of 2
5. Layer-4 : fully connected layer with 1024 output units
6. Layer-5 : ReLU Activation Layer
7. Layer-5 : Flatten
8. Output-Layer : fully connected layer with 96 output units using softmax activation

Adam optimizer and sparse_categorical_crossentropy was used to compile the model.

4.2.2 Results

1. Training Accuracy : 98.97%
2. Testing Accuracy : 98.96%
3. Average F1-score : 98.61%
4. Average Precision : 98.44%
5. Average Recall : 98.96%

4.2.3 Inferences

1. Using given architecture only, we good get quite good results. As train accuracy and test accuracy are quite similar, model is not over-fitted.

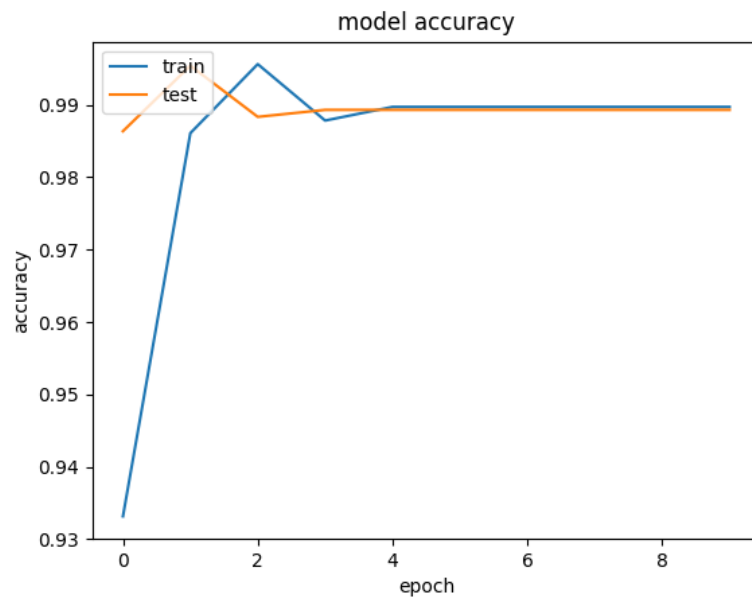


Figure 3: Line Data-Set : Accuracy vs Epochs

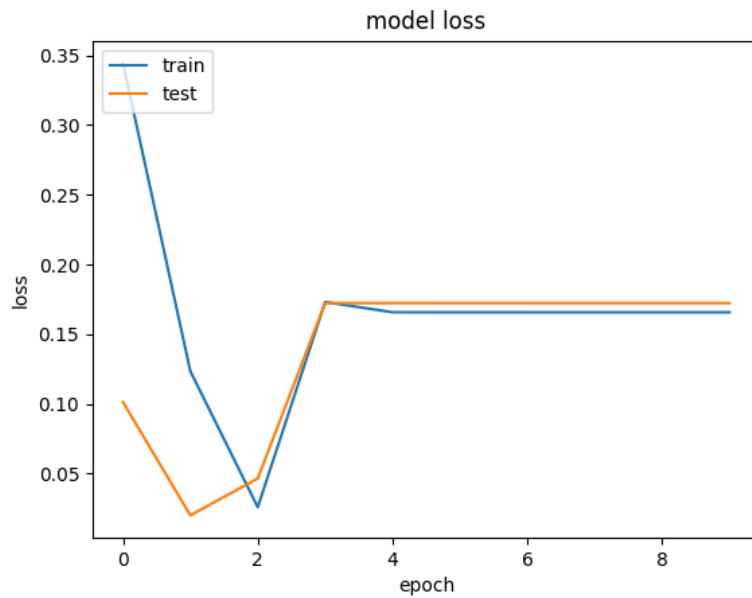


Figure 4: Line Data-Set : Loss vs Epochs

5 Task-2 : Multi-headed Classification for Line Data-set

5.1 Dataset Preparation

The data-set contained 96000 images of shape 28*28*3. In order to prepare the data, we first flattened each data image to a single vector with pixel values between 0 and 1. This resulted in a 2352-dimensional feature representing a single image. Next we partitioned data into training and testing set.

1. Total Data-Samples : 96000
2. Training Samples : 72000
3. Testing Samples : 24000

5.2 Classifier-1

5.2.1 Network Architecture

1. Colour-Head : Binary Cross-entropy loss with sigmoid activation.
2. Length-Head : Binary Cross-entropy loss with sigmoid activation.
3. Width-Head : Binary Cross-entropy loss with sigmoid activation.
4. Angle-Head : Categorical Cross-entropy loss with softmax activation.

Adam Optimizer was used. The learning rate was set to 0.0001 and decay to 0.00001 The model was run for 15 epochs.

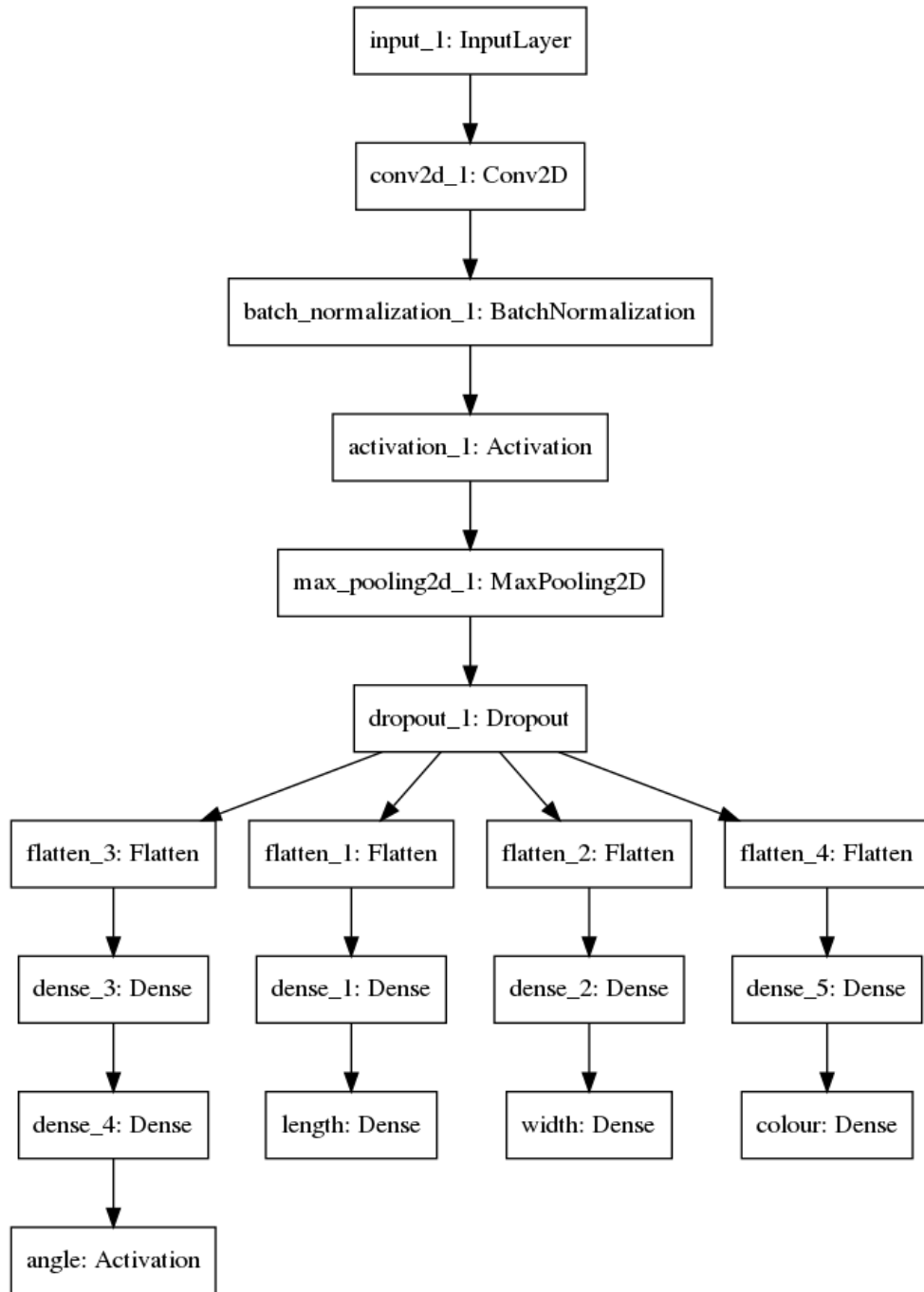


Figure 5: Multi-Head CNN Architecture

5.2.2 Results

1. Training Accuracy : 100.0%

2. Testing Accuracy : 100.00%

Since the all points are correctly classified, the recall, precision and f-score for all classes are 1.

Thus rather than making a table, I have put all the f-score in a plot.



Figure 6: Multi-head Classification : F-Score for all classes

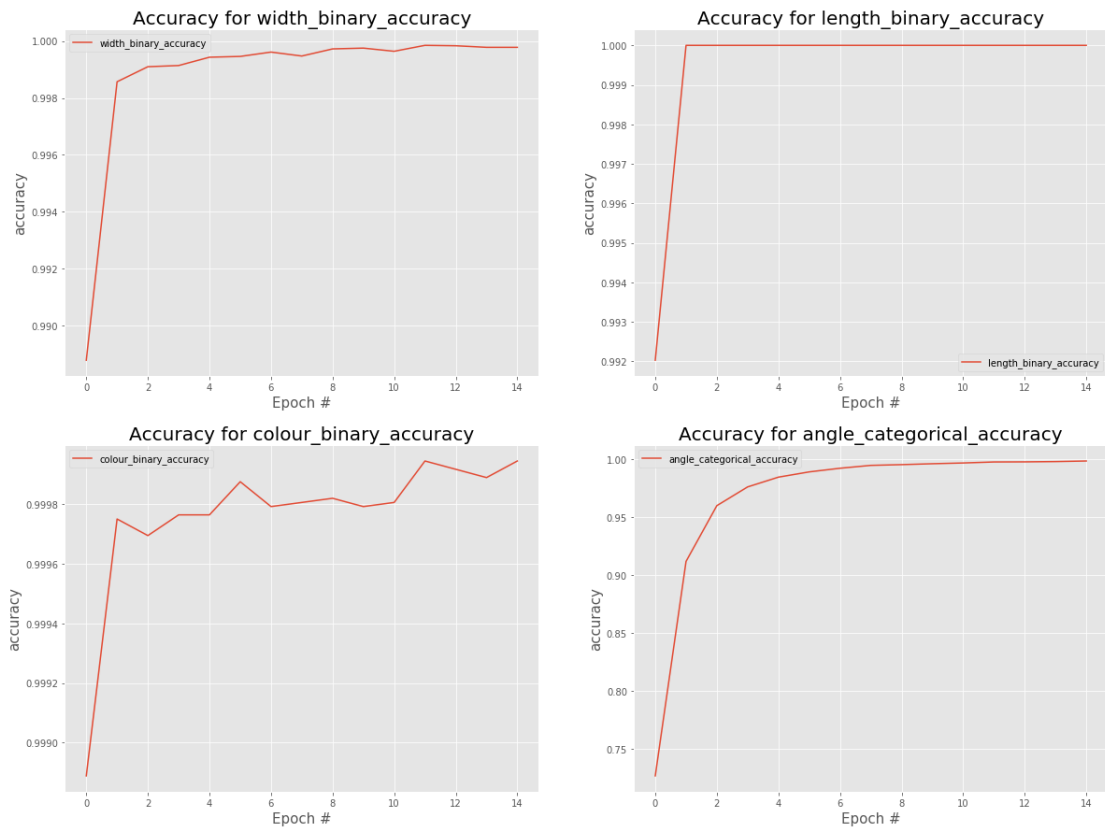


Figure 7: Multi-head Classification : Accuracy for various heads vs Epochs

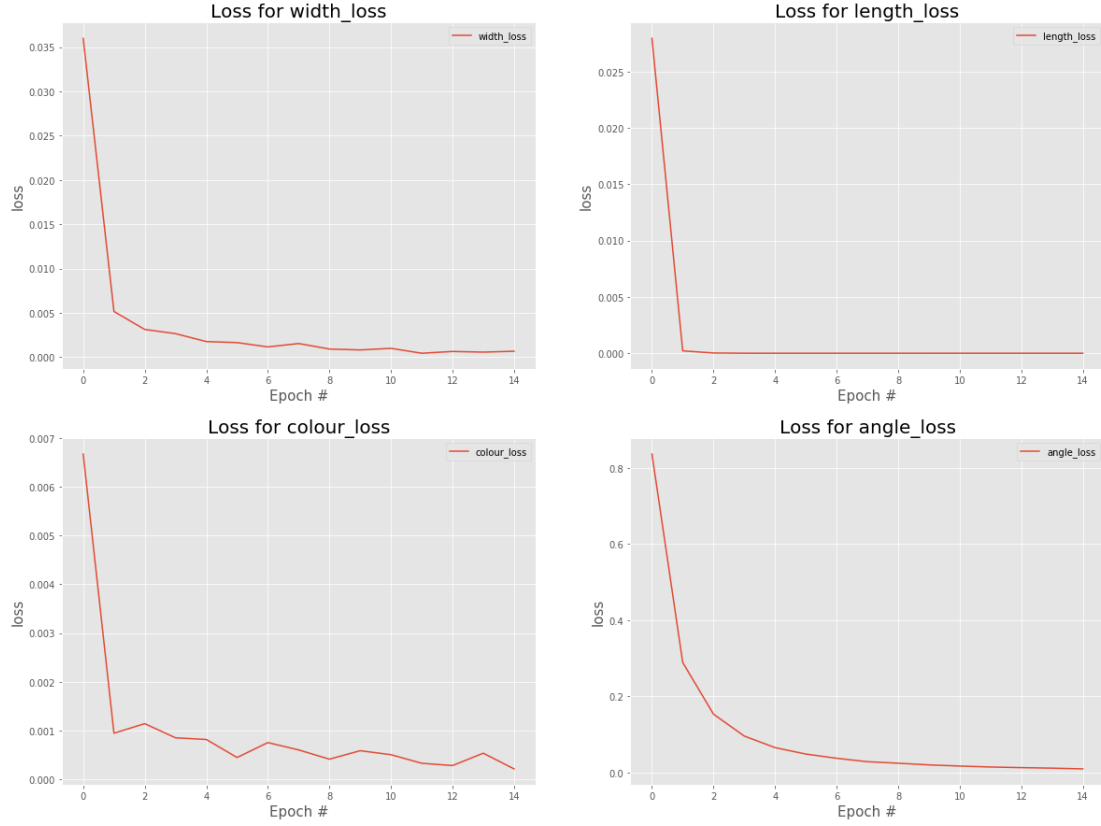


Figure 8: Multi-head Classification : Loss for various heads vs Epochs

5.2.3 Inferences

It is evident from the graphs that the bottle-neck for the accuracy are the angle classification head and the colour classification head. Having said that, the model performed very well, with no point being mis-classified.

One advantage of using Multi-head classification is follows:

Suppose we have trained we now want our model to detect Green Lines as well (they have been added to the data-set). Now, we only have to train the colour head of the model separately and the model would start recognizing the Green Lines. This is despite the fact that the other classification head have never actually seen a Green Line.

6 Task-3 : 10-Class MNIST Dataset

6.1 Architecture

Layer (type)	Output Shape	Param \#
conv2d_1 (Conv2D)	(None, 24, 24, 32)	832

activation_1 (Activation)	(None, 24, 24, 32)	0
batch_normalization_1 (Batch Normalization)	(None, 24, 24, 32)	128
max_pooling2d_1 (MaxPooling2D)	(None, 12, 12, 32)	0
dense_1 (Dense)	(None, 12, 12, 1024)	33792
activation_2 (Activation)	(None, 12, 12, 1024)	0
flatten_1 (Flatten)	(None, 147456)	0
dense_2 (Dense)	(None, 10)	1474570
Total params: 1,509,322		
Trainable params: 1,509,258		
Non-trainable params: 64		

6.2 Test Image 1

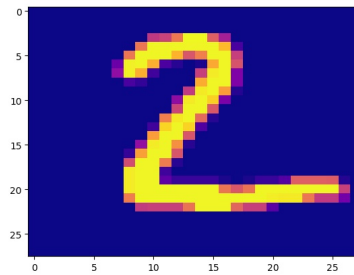


Figure 9: Test Image 1

6.2.1 Visualizing Intermediate Layer Activations

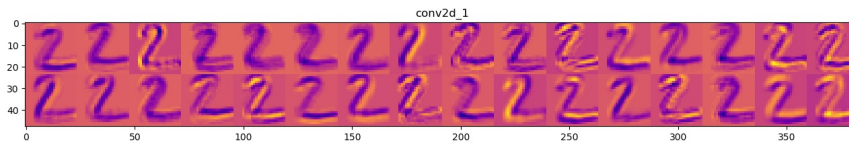


Figure 10: Conv2D Layer

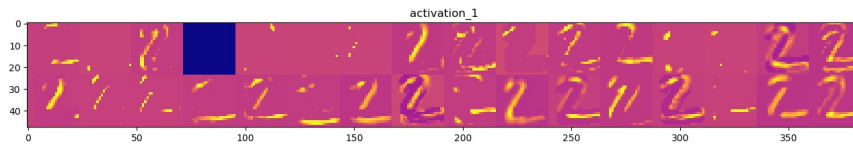


Figure 11: Activation Layer

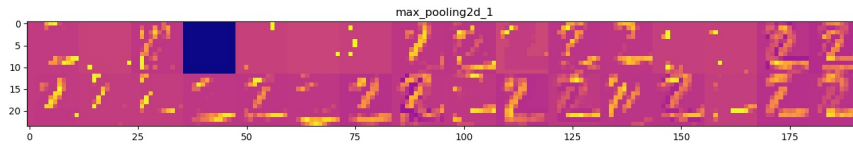


Figure 12: Max Pooling Layer

6.2.2 Visualizing Convnet Filters

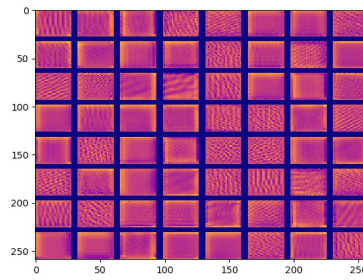
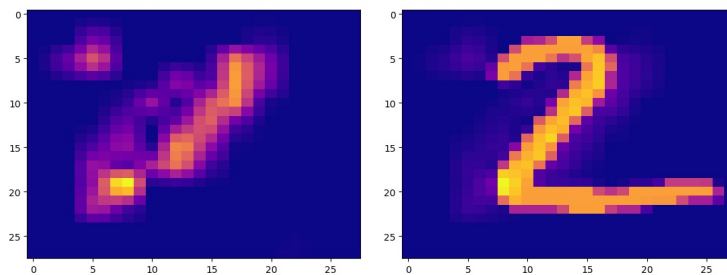


Figure 13: Conv net Filters of dense₁ Layer

6.2.3 Visualizing Heatmaps of class activations



(a) Heatmap of activation₂ Layer

(b) Superimposed Image

6.3 Test Image 2

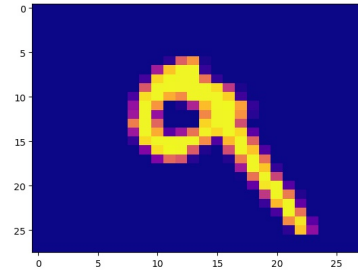


Figure 15: Test Image 2

6.3.1 Visualizing Intermediate Layer Activations

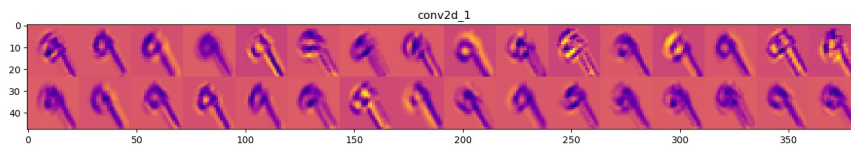


Figure 16: Conv2D Layer

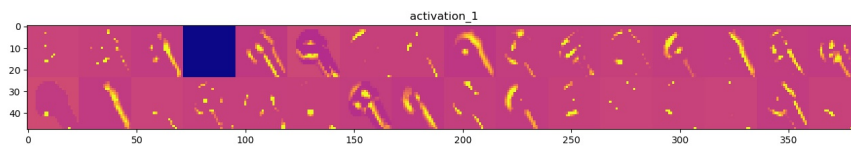


Figure 17: Activation Layer

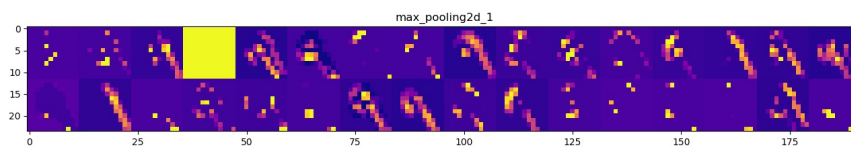


Figure 18: Max Pooling Layer

6.3.2 Visualizing Convnet Filters

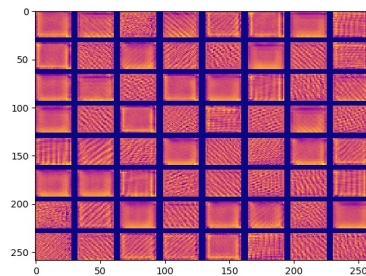
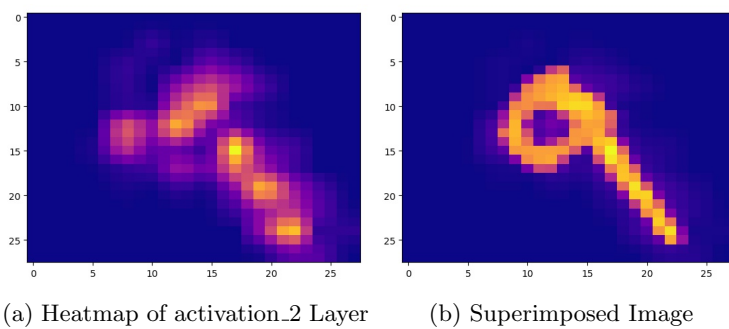


Figure 19: Convnet Filters of dense_1 Layer

6.3.3 Visualizing Heatmaps of class activations



(a) Heatmap of activation_2 Layer

(b) Superimposed Image

6.4 Test Image 3

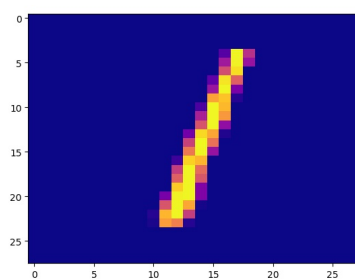


Figure 21: Test Image 3

6.4.1 Visualizing Intermediate Layer Activations

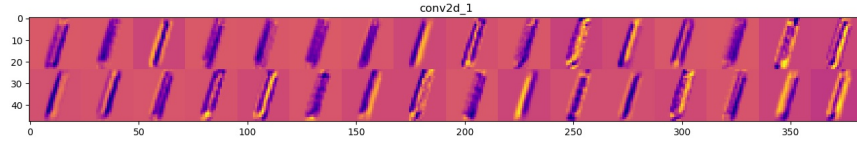


Figure 22: Conv2D Layer

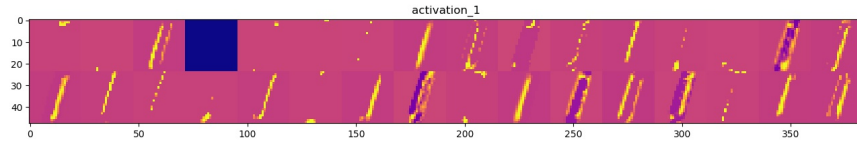


Figure 23: Activation Layer

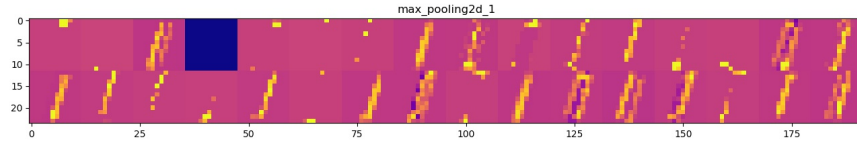


Figure 24: Max Pooling Layer

6.4.2 Visualizing Convnet Filters

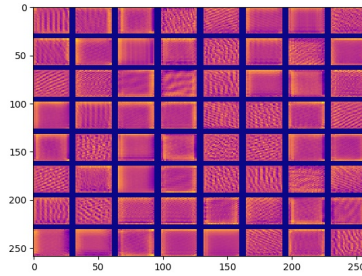
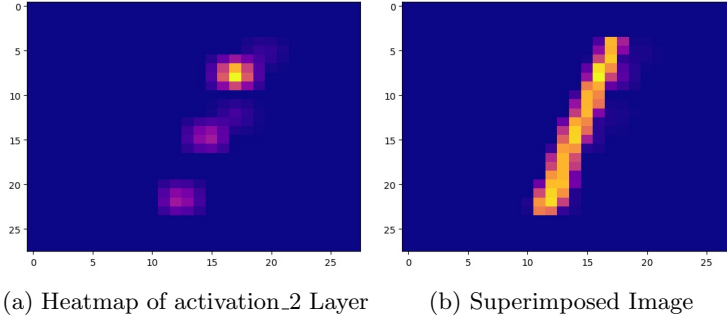


Figure 25: Convnet Filters of dense₁ Layer

6.4.3 Visualizing Heatmaps of class activations



7 Task 3 : 96-Class Line Dataset

7.1 Architecture

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 28, 28, 3)	0	
conv2d_1 (Conv2D)	(None, 28, 28, 32)	896	input_1[0][0]
activation_1 (Activation)	(None, 28, 28, 32)	0	conv2d_1[0][0]
batch_normalization_1 (BatchNor	(None, 28, 28, 32)	128	activation_1[0][0]
max_pooling2d_1 (MaxPooling2D)	(None, 9, 9, 32)	0	batch_normalization_1[0][0]
dropout_1 (Dropout)	(None, 9, 9, 32)	0	max_pooling2d_1[0][0]
conv2d_2 (Conv2D)	(None, 9, 9, 64)	18496	dropout_1[0][0]
activation_2 (Activation)	(None, 9, 9, 64)	0	conv2d_2[0][0]
batch_normalization_2 (BatchNor	(None, 9, 9, 64)	256	activation_2[0][0]
conv2d_3 (Conv2D)	(None, 9, 9, 64)	36928	batch_normalization_2[0][0]
conv2d_4 (Conv2D)	(None, 9, 9, 32)	9248	dropout_1[0][0]
conv2d_5 (Conv2D)	(None, 9, 9, 32)	9248	dropout_1[0][0]
conv2d_6 (Conv2D)	(None, 9, 9, 32)	9248	dropout_1[0][0]
activation_3 (Activation)	(None, 9, 9, 64)	0	conv2d_3[0][0]
activation_4 (Activation)	(None, 9, 9, 32)	0	conv2d_4[0][0]
activation_5 (Activation)	(None, 9, 9, 32)	0	conv2d_5[0][0]

activation_6 (Activation)	(None, 9, 9, 32)	0	conv2d_6[0][0]
batch_normalization_3 (BatchNor	(None, 9, 9, 64)	256	activation_3[0][0]
batch_normalization_4 (BatchNor	(None, 9, 9, 32)	128	activation_4[0][0]
batch_normalization_5 (BatchNor	(None, 9, 9, 32)	128	activation_5[0][0]
batch_normalization_6 (BatchNor	(None, 9, 9, 32)	128	activation_6[0][0]
max_pooling2d_2 (MaxPooling2D)	(None, 4, 4, 64)	0	batch_normalization_3[0][0]
max_pooling2d_3 (MaxPooling2D)	(None, 4, 4, 32)	0	batch_normalization_4[0][0]
max_pooling2d_4 (MaxPooling2D)	(None, 4, 4, 32)	0	batch_normalization_5[0][0]
max_pooling2d_5 (MaxPooling2D)	(None, 4, 4, 32)	0	batch_normalization_6[0][0]
dropout_2 (Dropout)	(None, 4, 4, 64)	0	max_pooling2d_2[0][0]
dropout_3 (Dropout)	(None, 4, 4, 32)	0	max_pooling2d_3[0][0]
dropout_4 (Dropout)	(None, 4, 4, 32)	0	max_pooling2d_4[0][0]
dropout_5 (Dropout)	(None, 4, 4, 32)	0	max_pooling2d_5[0][0]
flatten_1 (Flatten)	(None, 1024)	0	dropout_2[0][0]
flatten_2 (Flatten)	(None, 512)	0	dropout_3[0][0]
flatten_3 (Flatten)	(None, 512)	0	dropout_4[0][0]
flatten_4 (Flatten)	(None, 512)	0	dropout_5[0][0]
dense_1 (Dense)	(None, 12)	12300	flatten_1[0][0]
dense_2 (Dense)	(None, 1)	513	flatten_2[0][0]
dense_3 (Dense)	(None, 1)	513	flatten_3[0][0]
dense_4 (Dense)	(None, 1)	513	flatten_4[0][0]
angle_output (Activation)	(None, 12)	0	dense_1[0][0]
color_output (Activation)	(None, 1)	0	dense_2[0][0]
length_output (Activation)	(None, 1)	0	dense_3[0][0]
width_output (Activation)	(None, 1)	0	dense_4[0][0]

```
=====
Total params: 98,927
Trainable params: 98,415
Non-trainable params: 512
=====
```

7.2 Test Image 1

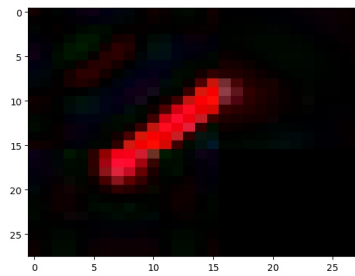


Figure 27: Test Image 1

7.2.1 Visualizing Intermediate Layer Activations

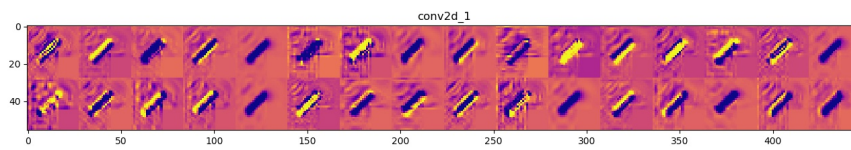


Figure 28: Conv2D.1 Layer

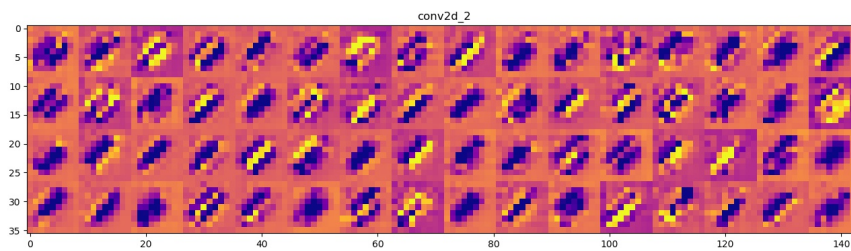


Figure 29: Conv2D.2 Layer

7.2.2 Visualizing Convnet Filters

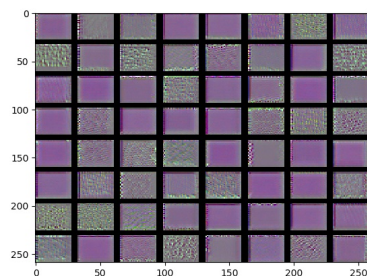


Figure 30: Conv net Filters of conv2d_2 Layer

7.3 Test Image 2

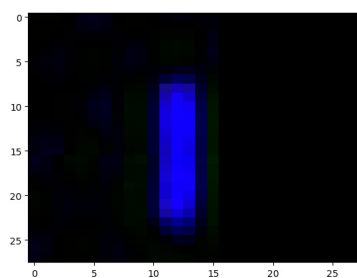


Figure 31: Test Image 2

7.3.1 Visualizing Intermediate Layer Activations

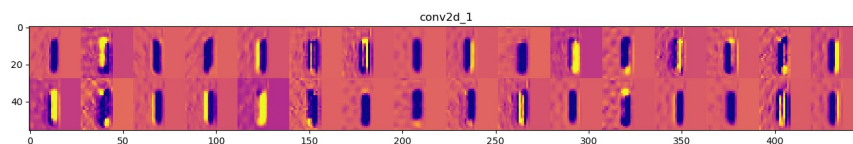


Figure 32: Conv2D.1 Layer

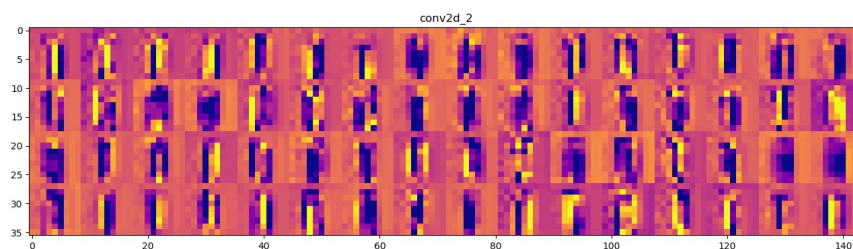


Figure 33: Conv2D.2 Layer

7.3.2 Visualizing Convnet Filters

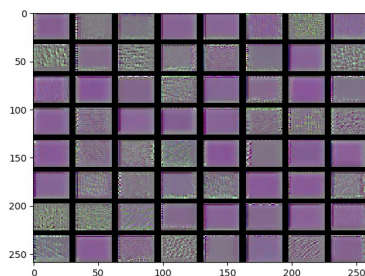


Figure 34: Conv net Filters of conv2d_2 Layer

7.4 Test Image 3

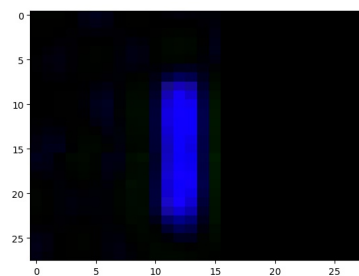


Figure 35: Test Image 3

7.4.1 Visualizing Intermediate Layer Activations

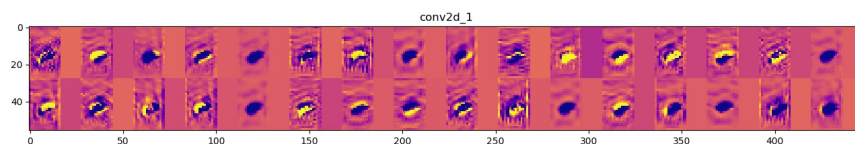


Figure 36: Conv2D.1 Layer

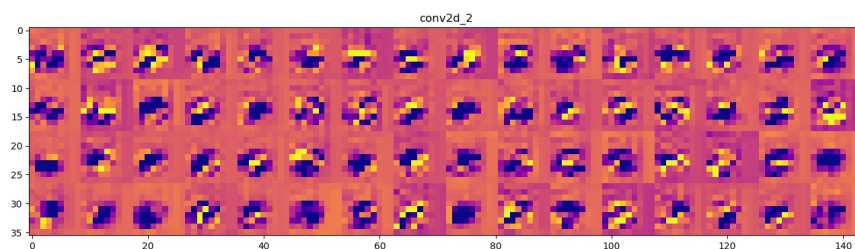


Figure 37: Conv2D.2 Layer

7.4.2 Visualizing Convnet Filters

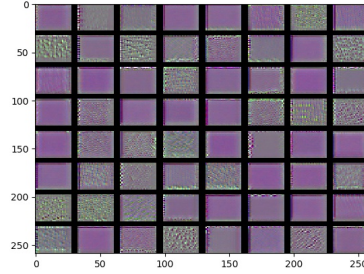


Figure 38: Conv net Filters of conv2d.2 Layer

References

- [1] www.pyimagesearch.com/2018/06/04/keras-multiple-outputs-and-multiple-losses/