CS671 - Deep Learning and its Applications

Programming Assignment 2 : CNNs

Sabin Kafley (D18032)
Dhanunjaya Varma (S18023)
Hritik Gupta (B16097)

April 7, 2019

Contents

1. Task - 1

1.1 Problem Statement

To make a CNN model from scratch to classify images of the line dataset into the respective 96 classes and MNIST dataset into 10 classes

1.1.1 Part - 1

In this part, the network architecture of the model is specified apriori. Adam optimiser has to be used with categorical cross-entropy loss.

1.1.2 Part - 2

In this part, the motive is to improve the model accuracy via our own network architecture. Experimentation with hyperparameters, architecture, loss functions, and optimisers to train the model needs to be performed

1.2 Dataset

Each dataset (MNIST and Lines) have been split to 60% test and 40% train.

1. MNIST

   No. of Classes : 10

   Train Images : 42000

   Test Images : 28000

2. Lines

   No. of Classes : 96

   No. of items in each Class : 1000

   Train : 600 for each Class

   Test : 400 for each Class

1.3 Learning Curves

1.3.1 Part-1

Both the models were trained for 10 epochs. The results obtained are as below :

Dataset	Training Accuracy	Training Loss	Testing Accuracy	Testing Loss
MNIST	0.9952	0.0211	0.9848	0.0957
Lines	0.9762	0.2165	0.9657	0.2558

The training and testing curves are as reported below :

FOR MNIST DATASET

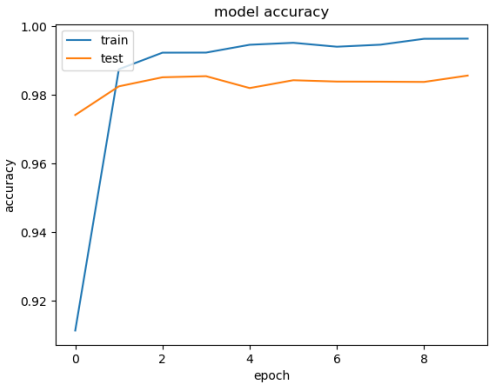


Figure 1: Accuracy Curve

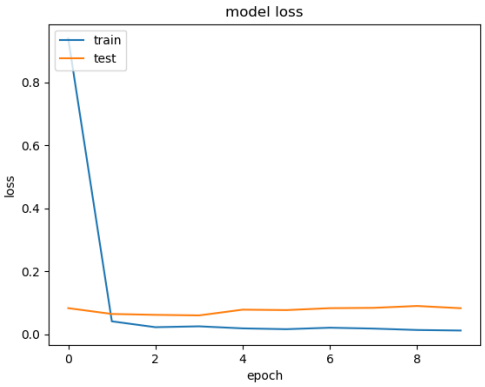


Figure 2: Loss Curve

FOR LINES DATASET

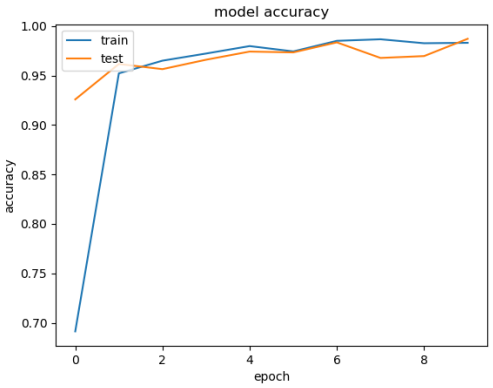


Figure 3: Accuracy Curve

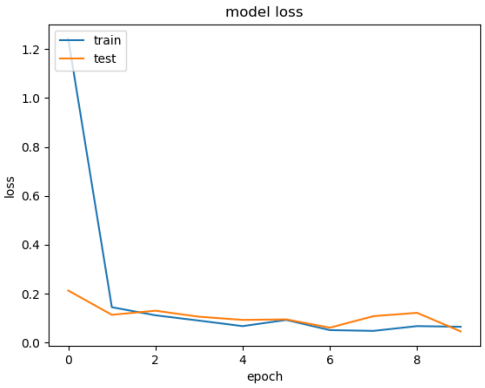


Figure 4: Loss Curve

1.3.2 Part-2

Both the models were trained for 20 epochs. The results obtained are as below :

Dataset	Training Accuracy	Training Loss	Testing Accuracy	Testing Loss
MNIST	0.9894	0.0352	0.991429	0.0329
Lines	0.9613	0.1394	0.9939	0.0211

The training and testing curves are as reported below :

FOR MNIST DATASET

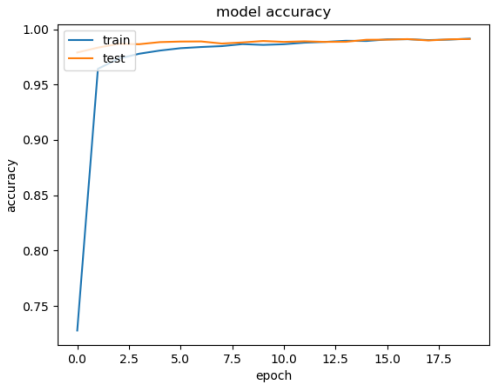


Figure 5: Accuracy Curve

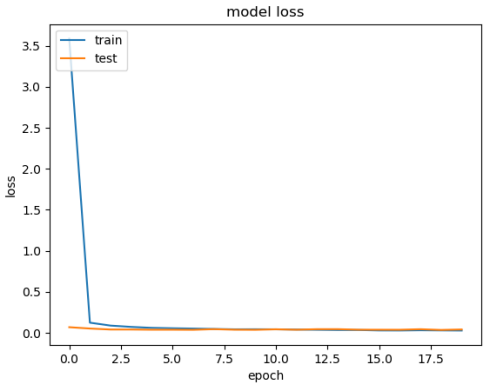


Figure 6: Loss Curve

FOR LINES DATASET

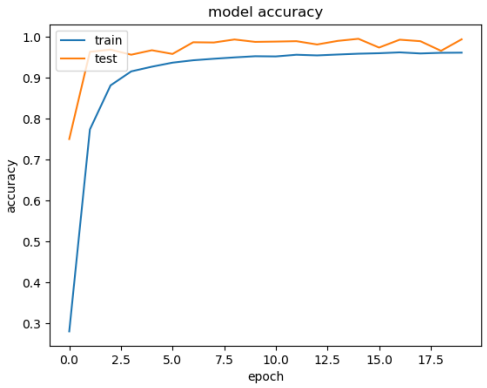


Figure 7: Accuracy Curve

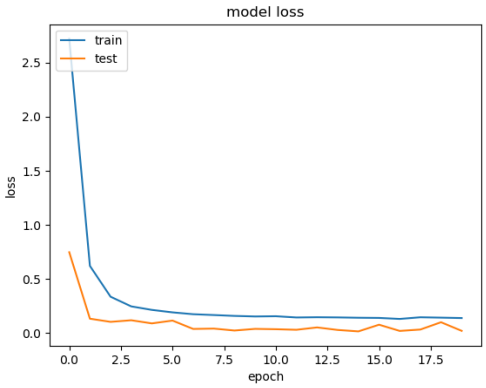


Figure 8: Loss Curve

1.4 F-Scores

1.4.1 Part-1

MNIST

Parameter	Class1	Class2	Class3	Class4	Class5	Class6	Class7	Class8	Class9	Class10
Precision	0.99378655	0.99190939	0.98491379	0.98878374	0.99252895	0.98387731	0.98496779	0.97506739	0.98802843	0.96495489
Recall	0.98657475	0.98679974	0.97963558	0.98121739	0.97077092	0.98893281	0.99350181	0.99041752	0.98251488	0.98722045
F Score	0.99016752	0.98934797	0.9822676	0.98498603	0.98152937	0.98639858	0.98921639	0.98268251	0.98526394	0.97596069

LINES

Since, there are 96 classes, it is not possible to show the F-Score.

1.4.2 Part-2

MNIST

Parameter	Class1	Class2	Class3	Class4	Class5	Class6	Class7	Class8	Class9	Class10
Precision	0.99496946	0.99214331	0.99176218	0.99791014	0.99519941	0.99021526	0.98925528	0.98619738	0.98883929	0.98760623
Recall	0.99568501	0.99684244	0.99282897	0.9852132	0.98790323	0.99060298	0.9970127	0.9927058	0.99105145	0.98412138
F Score	0.9953271	0.99448732	0.99229529	0.99152102	0.9915379	0.99040908	0.99311884	0.98944089	0.98994413	0.98586073

LINES

Since, there are 96 classes, it is not possible to show the F-Score.

1.5 Confusion Matrix

1.5.1 Part-1

MNIST

$$\begin{bmatrix} 2719 & 1 & 8 & 4 & 0 & 0 & 15 & 1 & 8 & 0 \\ 0 & 3065 & 11 & 2 & 2 & 1 & 1 & 21 & 1 & 2 \\ 2 & 7 & 2742 & 10 & 1 & 1 & 0 & 28 & 6 & 2 \\ 0 & 0 & 3 & 2821 & 0 & 24 & 1 & 6 & 8 & 12 \\ 0 & 4 & 2 & 0 & 2657 & 0 & 5 & 8 & 1 & 60 \\ 3 & 0 & 0 & 6 & 2 & 2502 & 9 & 0 & 0 & 8 \\ 6 & 1 & 2 & 2 & 2 & 3 & 2752 & 0 & 2 & 0 \\ 1 & 2 & 9 & 3 & 3 & 0 & 0 & 2894 & 1 & 9 \\ 2 & 5 & 7 & 4 & 5 & 7 & 8 & 1 & 2641 & 8 \\ 3 & 5 & 0 & 1 & 5 & 5 & 3 & 9 & 5 & 2781 \end{bmatrix}$$

LINES

Since, there are 96 classes, it is not possible to show the Confusion matrix (96x96).

1.5.2 Part-2

MNIST

$$\begin{bmatrix} 2690 & 1 & 3 & 0 & 0 & 3 & 21 & 0 & 0 & 2 \\ 0 & 1129 & 1 & 2 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1028 & 0 & 0 & 0 & 1 & 3 & 0 & 0 \\ 0 & 0 & 2 & 1004 & 0 & 2 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 977 & 0 & 1 & 0 & 1 & 2 \\ 0 & 0 & 0 & 4 & 0 & 885 & 1 & 0 & 2 & 0 \\ 2 & 4 & 0 & 0 & 2 & 1 & 946 & 0 & 3 & 0 \\ 0 & 2 & 9 & 0 & 0 & 0 & 0 & 1017 & 0 & 0 \\ 1 & 0 & 2 & 2 & 1 & 0 & 0 & 1 & 965 & 2 \\ 0 & 3 & 0 & 1 & 8 & 3 & 0 & 5 & 3 & 986 \end{bmatrix}$$

LINES

Since, there are 96 classes, it is not possible to show the Confusion matrix (96x96).

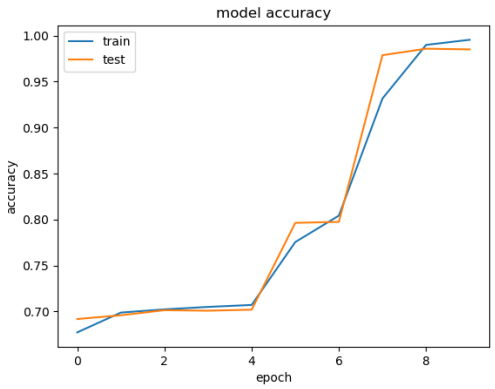
1.6 Variations Tried

1.6.1 Part-1

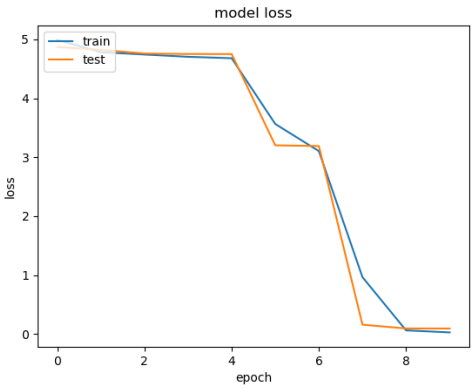
None, for this part.

1.6.2 Part-2

- **Filter Size :** Tried varying the filter size to smaller value. On experimenting with 2x2, the training loss reduced significantly (0.0124), but the validation loss was much greater (0.1184). The network probably learns too many intricate features, and thus overfits.
- **Number of Filters :** Filters are like feature detectors in the model. Hence a greater number of feature detectors tries to learn more number of features. As a result, more feature maps are generated. We doubled the number of filters from 32 to 64. The model converges slowly with higher number of filters, as shown clearly in the below graph :



(a) Accuracy Curve



(b) Loss Curve

- **Pooling vs Strided Convolution :** Both are down-sampling techniques. Pooling reduces the spatial size of the representation, significantly reducing the number of parameters and the computation overhead. We tried replacing Max Pooling with strided Convolution, which was reportedly slow (with a lower stride). We found that Max Pooling can simply be replaced by a convolutional layer with increased stride without loss in accuracy.

- **Network architecture :** Increased the number of layers in the network : We tried replicating the architecture of VGG19, by introducing a 5 layers of trainable parameters, the architecture as shown :

```
model = keras.Sequential()

model.add(Conv2D(64, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2), strides=2))

model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2), strides=2))

model.add(Conv2D(256, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2), strides=2))

model.add(Flatten())
model.add(Dense(1024, activation='relu'))
model.add(Dense(1024, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))
```

  This model performed with a very high validation losses (12 - 14 percent). We reduced the number of neurons to 128, and the loss reduced significantly, giving a training accuracy >98% within 7 epochs.

- **Use Learning Rate Decay :** Learning Rate is a hyperparameter that controls how much we are adjusting the weights with respect to the loss gradient. A smaller value makes us travel slower downward the slope. This may lead to slow convergence or no convergence at all. Hence a decay parameter was tweaked in the 'keras.optimiser.Adam()' (increased to 0.5), which lead to faster convergence per epoch.

- **Regularization :** Tried having Dropouts in the architecture. Dropouts drops output of random neurons during the training phase. Hence it was observed that the training loss and accuracies were substantially lower than the validation loss and accuracies.

1.7 Inference

1.7.1 Part-1

We learned to use higher level APIs to create model architecture, train and test it.

1.7.2 Part-2

The variations on the hyperparameters to enhance the accuracy of the model enabled us to gain an understanding of the role of each parameter.

2. Task - 2

2.1 Problem Statement

To design a non-sequential convolutional neural network for classifying the line dataset. The network will have 4 outputs based on the 4 kind of variations(length, width, color, angle).
The network architecture needs to be divided to:

1. Feature Network : responsible for extracting the required features from the input and attached to it would be the four classification heads one for each variation.

2. Classification Heads : The first 3 classification heads are for 2 class problems namely length, width and color classification. In all these the final layer contains a single neuron with a sigmoid activation followed by binary crossentropy loss. The last classification head is a 12 class problem for each 12 angles of variation. In this the final layer contains 12 neurons with softmax activation and Categorical Cross entropy loss.

2.2 Dataset

The lines dataset has been split to 60% test and 40% train.

1. Lines

   No. of Classes : 96

   No. of items in each Class : 1000

   Train : 600 for each Class

   Test : 400 for each Class

2.3 Learning Curves

Feature	Testing Accuracy	Testing Loss
Length	0.9982	0.00753
Width	0.9987	0.00562
Color	0.9999	0.000375
Angle	0.9588	0.2262

Total Loss: 0.2398

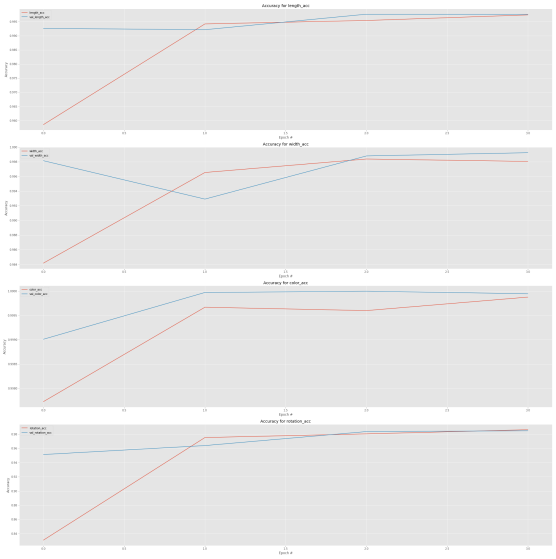


Figure 10: Accuracy Curve

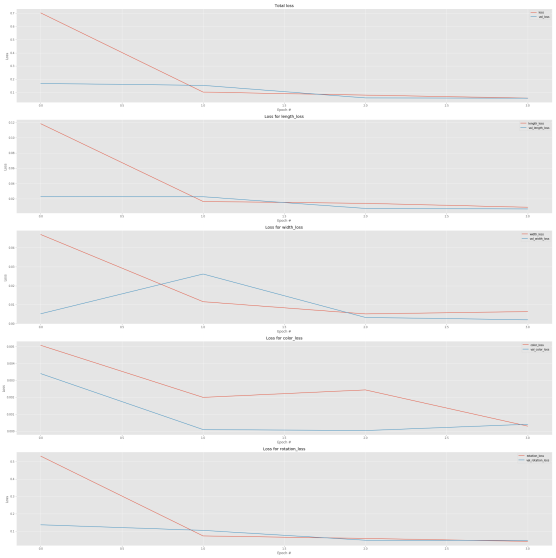


Figure 11: Loss Curve

2.4 F-Scores

2.4.1 Length Head Parameters

	Class 1	Class2
Precision	1.	0.74821714
Recall	0.66348958	1.
F-Measure	0.79770813	0.85597735

2.4.2 Width Head Parameters

	Class 1	Class2
Precision	1.	0.75249853
Recall	0.67109375	1.
F-Measure	0.80317906	0.85877222

2.4.3 Color Head Parameters

	Class 1	Class2
Precision	1.	0.99683298
Recall	0.99682292	1.
F-Measure	0.99840893	0.99841398

2.4.4 Rotation Head Parameters

	0°	15°	30°	45°	60°	75°	90°	105°	120°	135°	150°	165°
Precision	0.9860	0.9849	0.9831	0.9545	0.9713	0.9875	0.9259	0.9643	0.9837	0.8447	0.9941	0.9540
Recall	0.9921	0.9821	0.9837	0.9834	0.9743	0.8903	0.9725	0.9793	0.8125	0.9846	0.9578	0.9921
F-Measure	0.9890	0.9835	0.9834	0.9687	0.9728	0.9364	0.9486	0.9717	0.8899	0.9093	0.9756	0.9727

2.5 Confusion Matrix

2.5.1 Length Head

$$\begin{bmatrix} 12739 & 6461 \\ 0 & 19200 \end{bmatrix}$$

2.5.2 Width Head

$$\begin{bmatrix} 12885 & 6315 \\ 0 & 19200 \end{bmatrix}$$

2.5.3 Color Head

$$\begin{bmatrix} 19139 & 61 \\ 0 & 19200 \end{bmatrix}$$

2.5.4 Rotation Head

$$\begin{bmatrix} 3175 & 5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 20 \\ 27 & 3143 & 30 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 31 & 3148 & 20 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 5 & 23 & 3147 & 12 & 6 & 0 & 0 & 0 & 7 & 0 & 0 \\ 0 & 0 & 0 & 78 & 3118 & 4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 45 & 80 & 2849 & 220 & 6 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 26 & 3112 & 62 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 29 & 3134 & 30 & 6 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 47 & 2600 & 553 & 0 & 0 \\ 2 & 0 & 0 & 6 & 0 & 0 & 0 & 1 & 13 & 3151 & 16 & 11 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 13 & 3065 & 122 \\ 15 & 7 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 3175 \end{bmatrix}$$

2.6 Variations Tried

Hyperparameters like the ones listed in the Q-1(b) were tweaked to improve the accuracy of the model. Loss functions and final layer activation function has been kept the same as asked in the question.

2.7 Inferences

Performance improved when we reduced the number of classes from 96 (in the 1st part) to merely 4, concentrating majorly on the 4 features : line, width, color and rotation. We have used a single layer of trainable parameter, followed by another layer for each branch.

3. Task - 3

3.1 Problem Statement

Visualization of the networks:

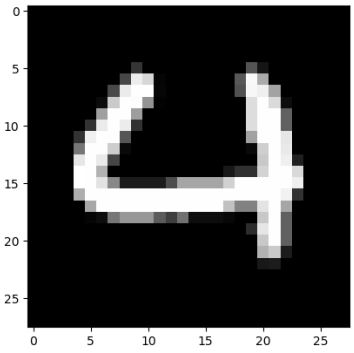
1. **Visualizing Intermediate Layer Activations :** Intermediate layer activations are the outputs of intermediate layers of the neural network. Intermediate activations of the layers of the neural networks need to be plotted.

2. **Visualizing Convnet Filters :** To visualize the filters in the convolutional neural network. This is done by running Gradient Descent on the value of a convnet so as to maximize the response of a specific filter, starting from a blank input image.

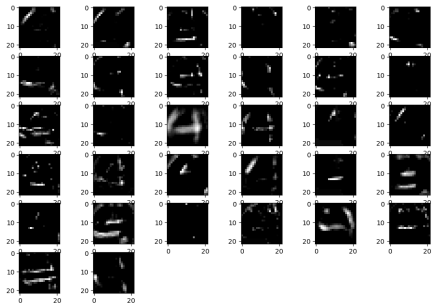
3. **Visualizing Heatmaps of class activations :** To plot heatmaps of class activations over input images. This helps to visualize the regions of the input image the convnet is looking at. A class activation heatmap is a 2D grid of scores associated with a particular output class, computed for every location for an input image, indicating how important is each location is with respect to that output class

3.2 Intermediate Layers

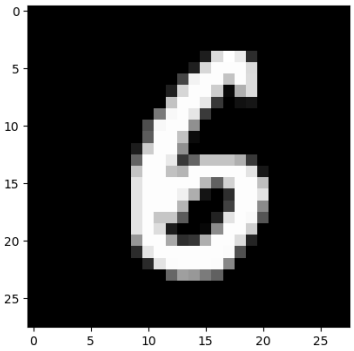
3.2.1 MNIST



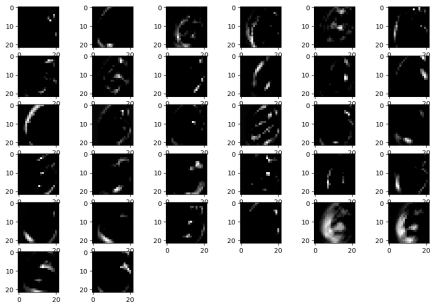
(a) Input Image



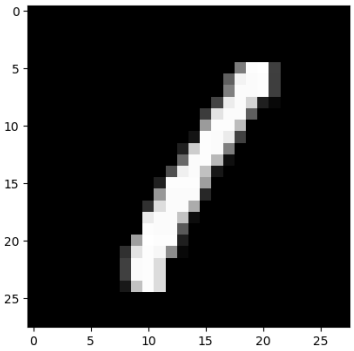
(b) Intermediate Activation Layer Output



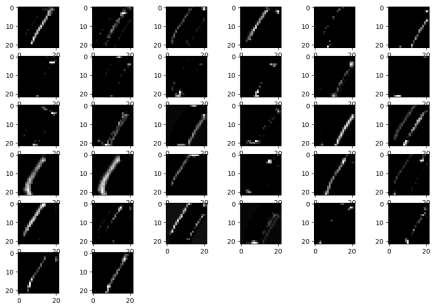
(a) Input Image



(b) Intermediate Activation Layer Output



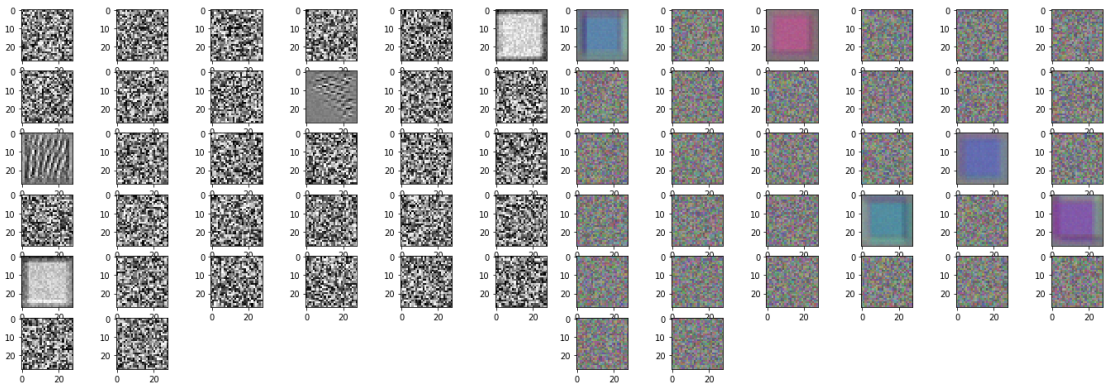
(c) Input Image



(d) Intermediate Activation Layer Output

3.3 Convnet Filters

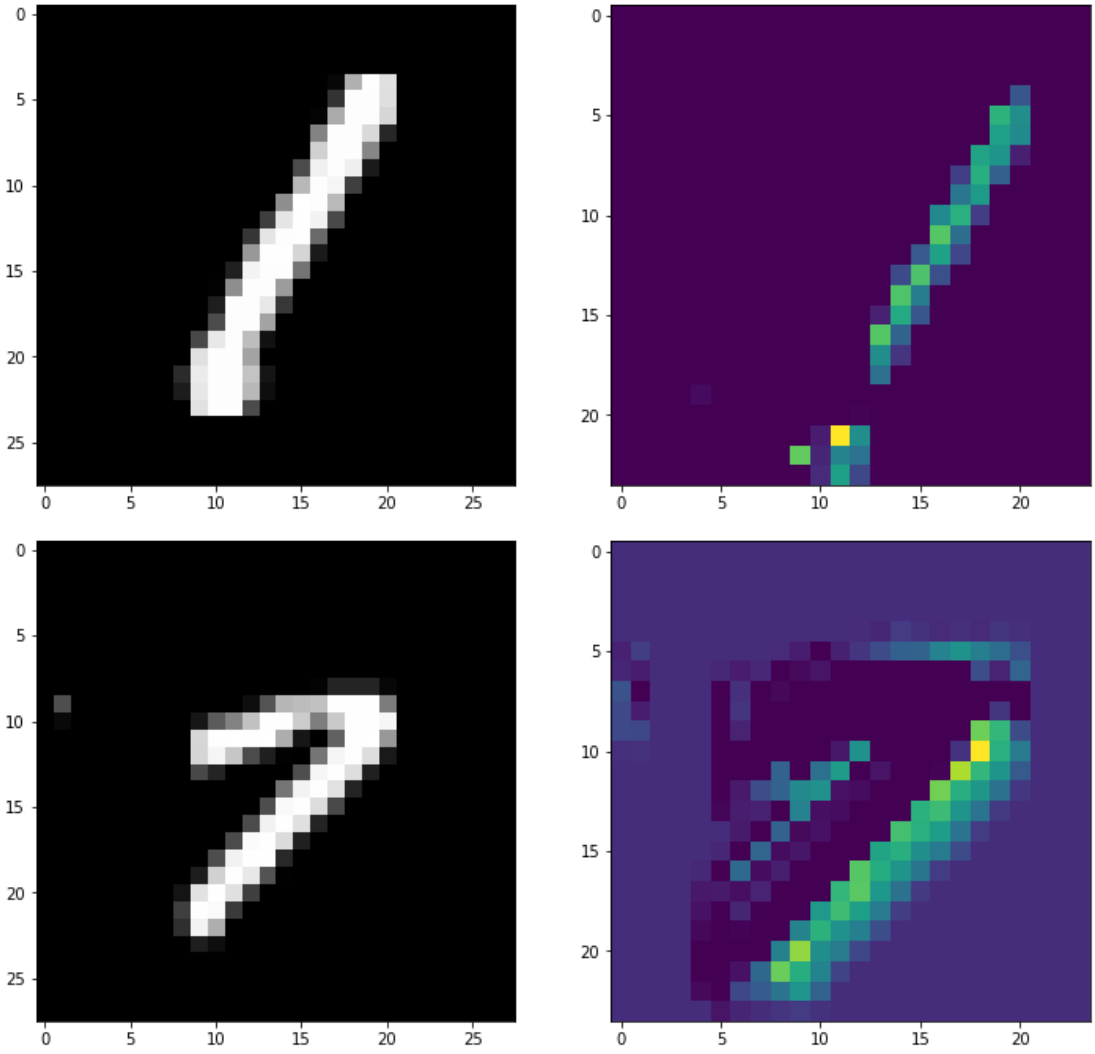
3.3.1 MNIST and Lines

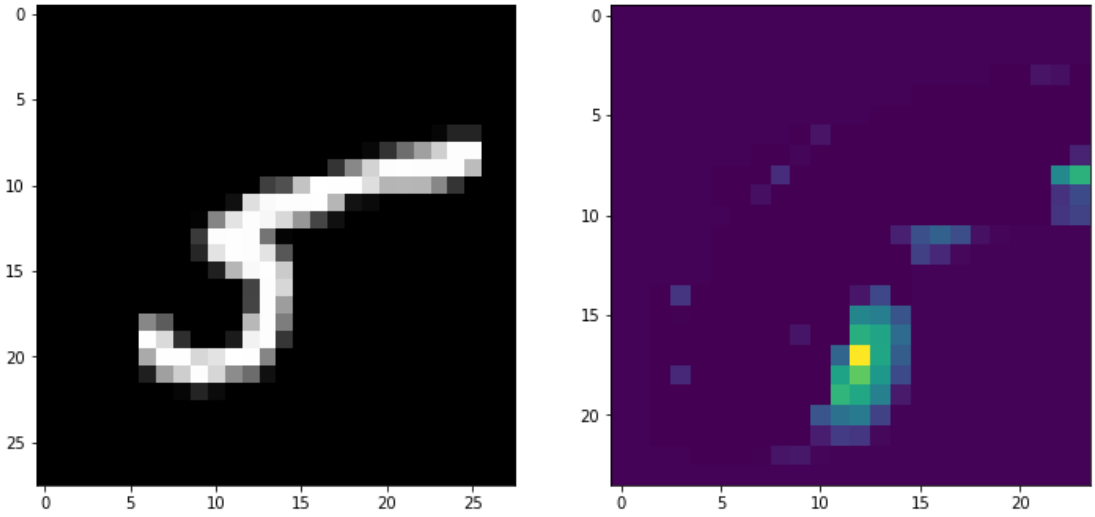


(a) Filter Output for MNIST (b) Filter Output for Line

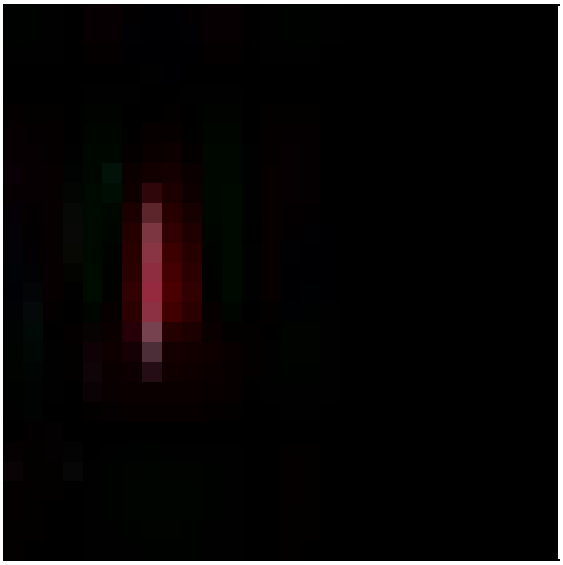
3.4 Heatmaps

3.4.1 MNIST

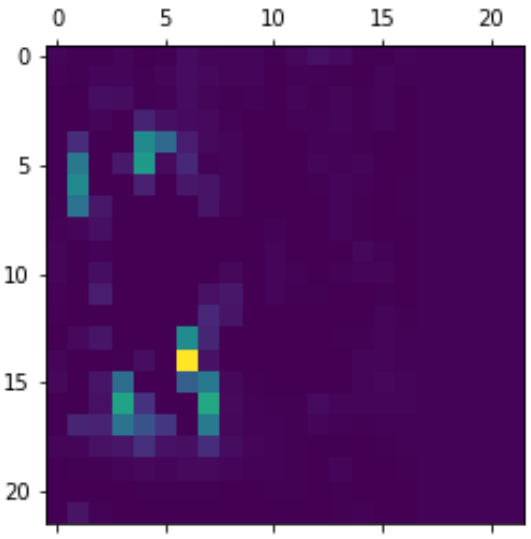




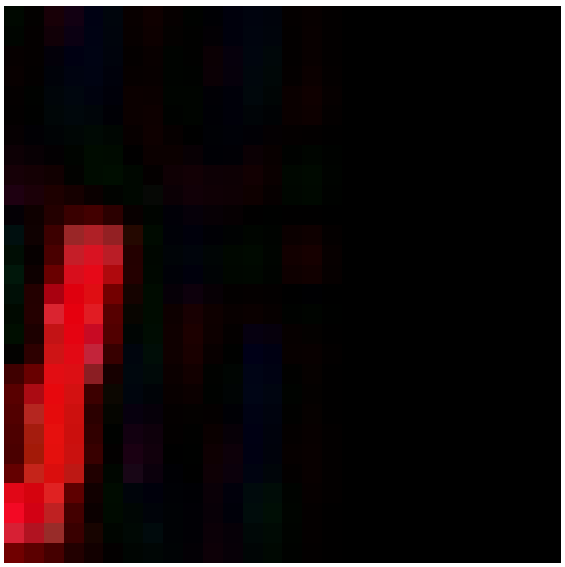
3.4.2 Lines



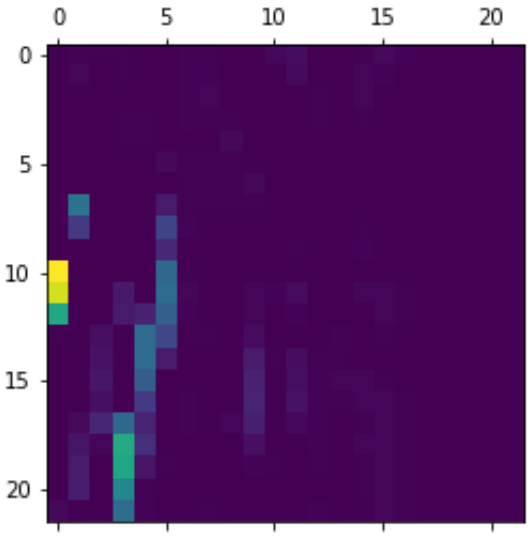
(a) Filter Output for MNIST



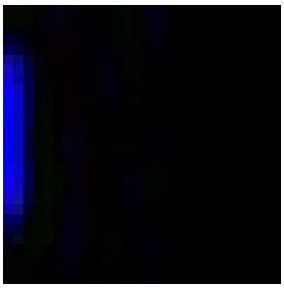
(b) Filter Output for Line



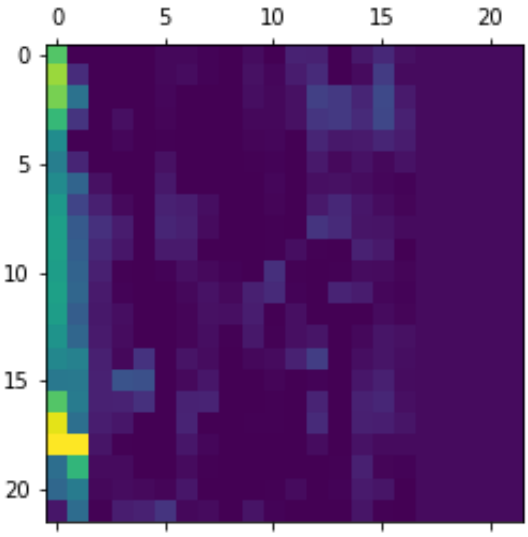
(a) Filter Output for Line



(b) Filter Output for Line



(a) Filter Output for Line



(b) Filter Output for Line