MATH 6373 Spring 2021 MSDS
Homework 3

Jacob Gogan
Sara Nafaryeh
Thomas Su

All authors played an equal part

Introduction

For our Homework 3 submission, we will select five fonts from the *fonts* data set and create a convolutional neural network (CNN) and train it for classification. As discussed in HW1, Multilayer Perceptron (MLP) is a type of feedforward Artificial Neural Network Model that introduces one or more hidden layers on the basis of a single-layer neural network. The hidden layer is located between the input layer and the output layer. Therefore, the neurons in the hidden layer are fully connected to each input in the input layer, and the neurons in the output layer and each neuron in the hidden layer are also fully connected. What our CNN will do is apply filters to our images (the digitized font) to transform the data before interacting with the aforementioned layers.These convolved and pooled features are then fed into the MLP where we can train the model for classification.

The files were downloaded from a zip file from the following link: *https://archive.ics.uci.edu/ml/machine-learning-databases/00417/*

The font types that we chose were: CAMBRIA, FRANKLIN, LUCIDA, SWIS721, and TAHOMA. Each has 412 column features along with the following total observed cases.

```
print(CAMBRIA.shape)
print(FRANKLIN.shape)
print(LUCIDA.shape)
print(SWIS721.shape)
print(TAHOMA.shape)
```

```
(9598, 412)
(15720, 412)
(15171, 412)
(13440, 412)
(13292, 412)
```

Each case describes numerically a digitized image of some specific character typed in each of the fonts. The images have 20x20 = 400 pixel sizes, each with its own “gray level” indicated by an integer value of 0 to 255. All the fonts have 412 feature columns, where 400 of them describe the 400 pixels named:

{ r0c0, r0c1,r0c2, … , r19,c17, r19c18, r19c19}

"rLcM" = gray level image intensity for pixel in position **{Row L, Column M}**.

Finally, the last important columns in the data set we will be looking at are the strength and italic columns. The strength column lists 0.4 = normal character and 0.7 = bold character. The italic column lists 1 = italic character and 0 = normal character.

STEP 1: Data Setup

We prep our data for analysis in Python. Other than the 400 columns that are associated with the pixels, the data set font files each have the following 12 names:

{ font, fontVariant, m_label, strength, italic, orientation, m_top, m_left, originalH, originalW, h, w }

Of these 12 we need to discard the following 9:

{fontVariant, m_label, orientation, m_top, m_left, originalH, originalW, h, w}

And keep the following 3: ***{font, strength, italic}*** as well as the 400 pixel columns named: ***{ r0c0, r0c1,r0c2, … , r19,c17, r19c18, r19c19}*** therefore we are left with 403 columns. After these steps are completed, we define three CLASSES on images of the "normal" character where we extract all the rows in which our three fonts have both strength of 0.4 and italic of 0:

CL1 = all rows of **CAMBRIA.csv** file for which {strength = 0.4 and italic=0}

CL2 = all rows of **FRANKLIN.csv** file for which {strength = 0.4 and italic=0}

CL3 = all rows of **LUCIDA.csv** file for which {strength = 0.4 and italic=0}

CL4 = all rows of **SWIS721.csv** file for which {strength = 0.4 and italic=0}

CL5 = all rows of **TAHOMA.csv** file for which {strength = 0.4 and italic=0}

And left with the following case outputs:

```
n1 = len(CL1)
n2 = len(CL2)
n3 = len(CL3)
n4 = len(CL4)
n5 = len(CL5)
N = (n1+ n2+ n3+ n4+ n5)
print(n1,n2,n3,n4,n5)
print(N)
```

```
3208 3931 3794 3360 3323
17616
```

The respected case sizes for the 5 classes are named

n1, n2, n3, n4, n5 = 3208, 3931, 3794, 3360, and 3323 and their sum → N = 17616

We combine them all together into a data set named DATA using the function **rbind()** and see it has dimensions of 1930 rows and 403 columns: the 403 being font, strength, italic, plus 400 pixels we will call features X1, X2, … X400. Each such feature Xj is observed N times, and its N observed values are listed in column "j" of DATA.

```
DATA = pd.concat([CL1, CL2,CL3, CL4, CL5])
print(DATA)
```

	font	strength	italic	r0c0	...	r19c16	r19c17	r19c18	r19c19
0	CAMBRIA	0.4	0	255	...	84	115	255	255
1	CAMBRIA	0.4	0	1	...	255	255	255	227
2	CAMBRIA	0.4	0	1	...	255	255	255	227
3	CAMBRIA	0.4	0	1	...	255	255	247	218
4	CAMBRIA	0.4	0	1	...	255	255	249	218
...	...	...	...	...	...	...	...	...	...
3318	TAHOMA	0.4	0	1	...	255	184	66	1
3319	TAHOMA	0.4	0	1	...	1	1	1	1
3320	TAHOMA	0.4	0	1	...	1	1	1	1
3321	TAHOMA	0.4	0	255	...	255	255	242	1
3322	TAHOMA	0.4	0	255	...	255	255	255	255

[17616 rows x 403 columns]

STEP 2: Input Reshaping

Recall from HW2, each input vector X had a flattened dimension of 400 pixels. We can now reshape each input vector X as an image of 20x20 matrix pixel intensities. As seen below, we have a total cases of 17616 reshaped into 20x20 matrices, and

Y = The true output of 3 font classes as the following

[1 0 0 0], [0 1 0 0 0], [0 0 1 0 0], [0 0 0 1 0], and [0 0 0 0 1] one-hot encoding

```
X = np.concatenate([X_CL1, X_CL2, X_CL3, X_CL4, X_CL5]).reshape(-1, 20, 20, 1)
print(X.shape)
```

```
(17616, 20, 20, 1)
```

```
y = pd.concat([CL1, CL2, CL3, CL4, CL5], axis = 0).loc[:, 'font']
y
```

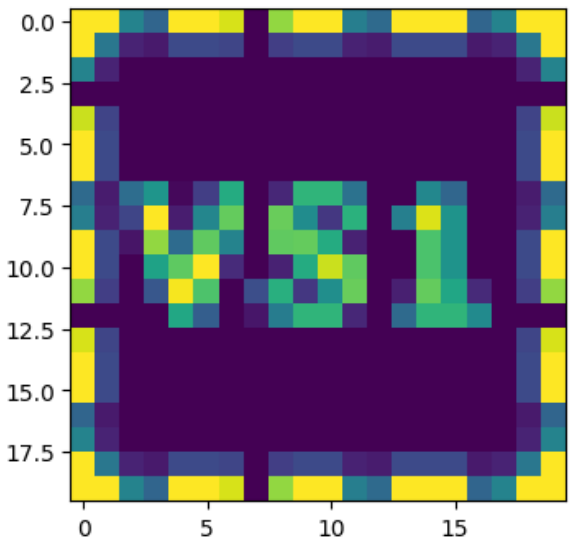
```
0       CAMBRIA
1       CAMBRIA
2       CAMBRIA
3       CAMBRIA
4       CAMBRIA
         ...
3318     TAHOMA
3319     TAHOMA
3320     TAHOMA
3321     TAHOMA
3322     TAHOMA
Name: font, Length: 17616, dtype: object
```

```
Y = np.concatenate([Y_CL1, Y_CL2, Y_CL3, Y_CL4, Y_CL5])
print(len(Y))
Y
```

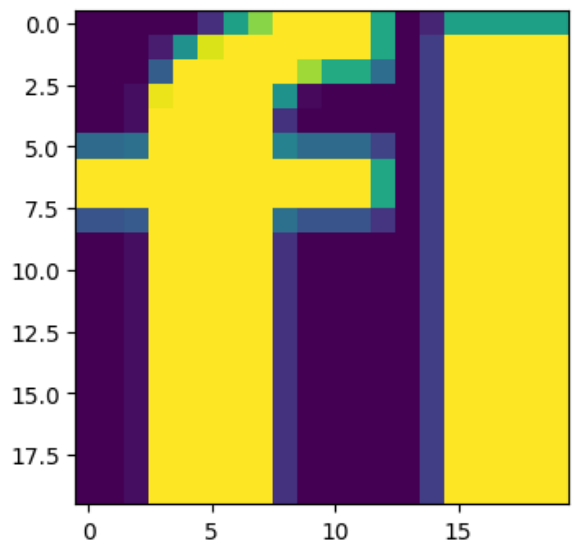
```
17616
array([[1., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0.],
       ...,
       [0., 0., 0., 0., 1.],
       [0., 0., 0., 0., 1.],
       [0., 0., 0., 0., 1.]], dtype=float32)
```

One character is selected and printed from each reshaped font to get the following 5 images:

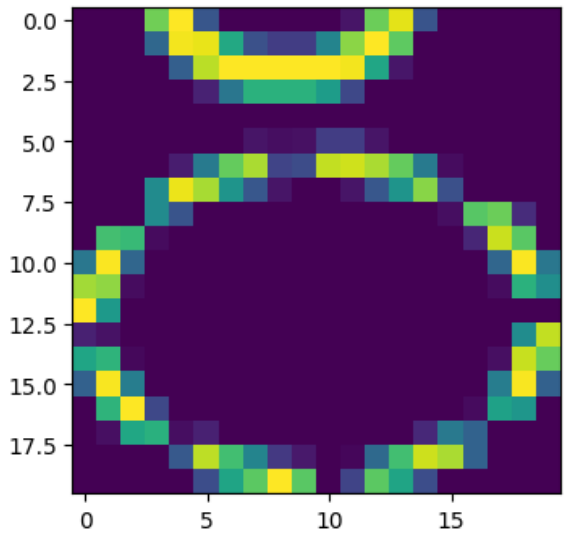
```
plt.figure(dpi = 100)
plt.imshow(image_1)
plt.show()
```



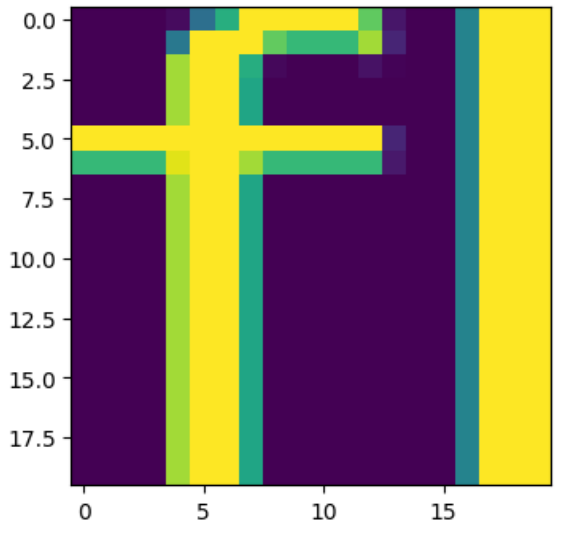

```
plt.figure(dpi = 100)
plt.imshow(image_2)
plt.show()
```



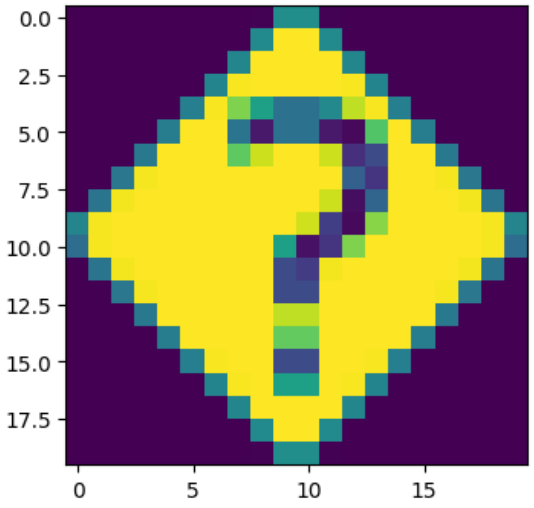

```
plt.figure(dpi = 100)
plt.imshow(image_3)
plt.show()
```




```
plt.figure(dpi = 100)
plt.imshow(image_4)
plt.show()
```



```
plt.figure(dpi = 100)
plt.imshow(image_5)
plt.show()
```



From these 5 images, we can see that our image consists of 20x20 pixels with three color channels. These channels are known as Red-Green-Blue or RGB values. The combination of these three colors allows us to produce all possible color pallets. With the help of the RGB values, we can observe additional colors in our images.

STEP 3: CNN Architecture

Convolutional Neural Networks also called CNN, is a machine learning model that allows one to extract higher representations for image content. CNN takes an image's raw pixel data, trains the model , then extracts the features automatically for a better classification prediction. The implementation of a typical CNN architecture is as follows:

Input → Dropout1 → **Conv.1** → **Maxpool.1** → Dropout2 → **Conv.2** → **Maxpool.2** → Dropout3 → **Flatten (Maxpool2)** → **Hidden H** → Dropout4 → **Output By Softmax** → **Probability**

Starting with the first Convolution layer, **Conv1**, we have a filter/channel of 32 and a window (5 x 5) that moves with a default of stride = 1. This gives us an output equal to 32 images of size (20 x 20). Notice this is reduced from our original input of 20x20. Note that stride is a component of convolutional neural networks, or neural networks tuned for the compression of images and video data. Stride is a parameter of the neural network's filter that modifies the amount of movement over the image or video. For example, if a neural network's stride is set to 1, the filter will move one pixel, or unit, at a time. The size of the filter affects the encoded output volume, so stride is often set to a whole integer, rather than a fraction or decimal.

Next we have a **Maxpool1**, 32 filters/channels and a window = (2 x 2) with a stride = 2 (meaning that move the filters to 2 pixels at a time), we get the output of 16 images of size (8 x 8). Another Convolution layer is done, **Conv2**, with 64 filters/channels and since we have small images now, the window is also smaller: (3 x 3)

along with a stride = 1 to get the output of 64 images of size (10 x 10). Finally, the **Matchpool2** is applied on top of this at 64 channels and the window of (2 x 2) with a stride = 2 and output = 64 images of size (5 x 5).

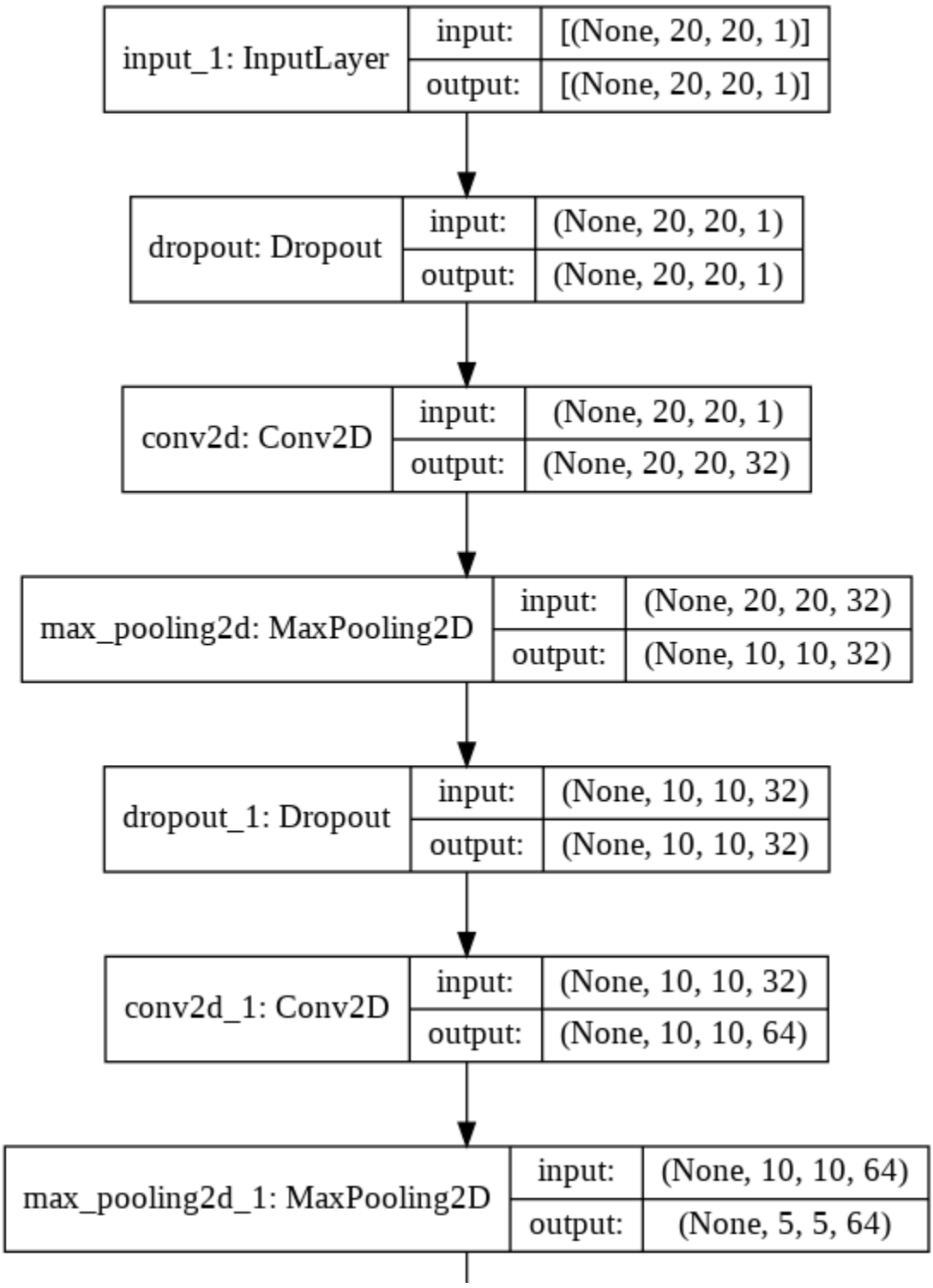
After these steps are taken, we need to condense the images by flattening them to get long vector of 64 channels * (5 x 5) window = 1600 dimension. These 1600 neurons encoding are linked to all neurons of H, where H is the last hidden layer of selected values h = 90, 150, and 200. The final two steps are the output by softmax having a dimension of 5 for the five fonts and the five probability of predicting the correct font class.

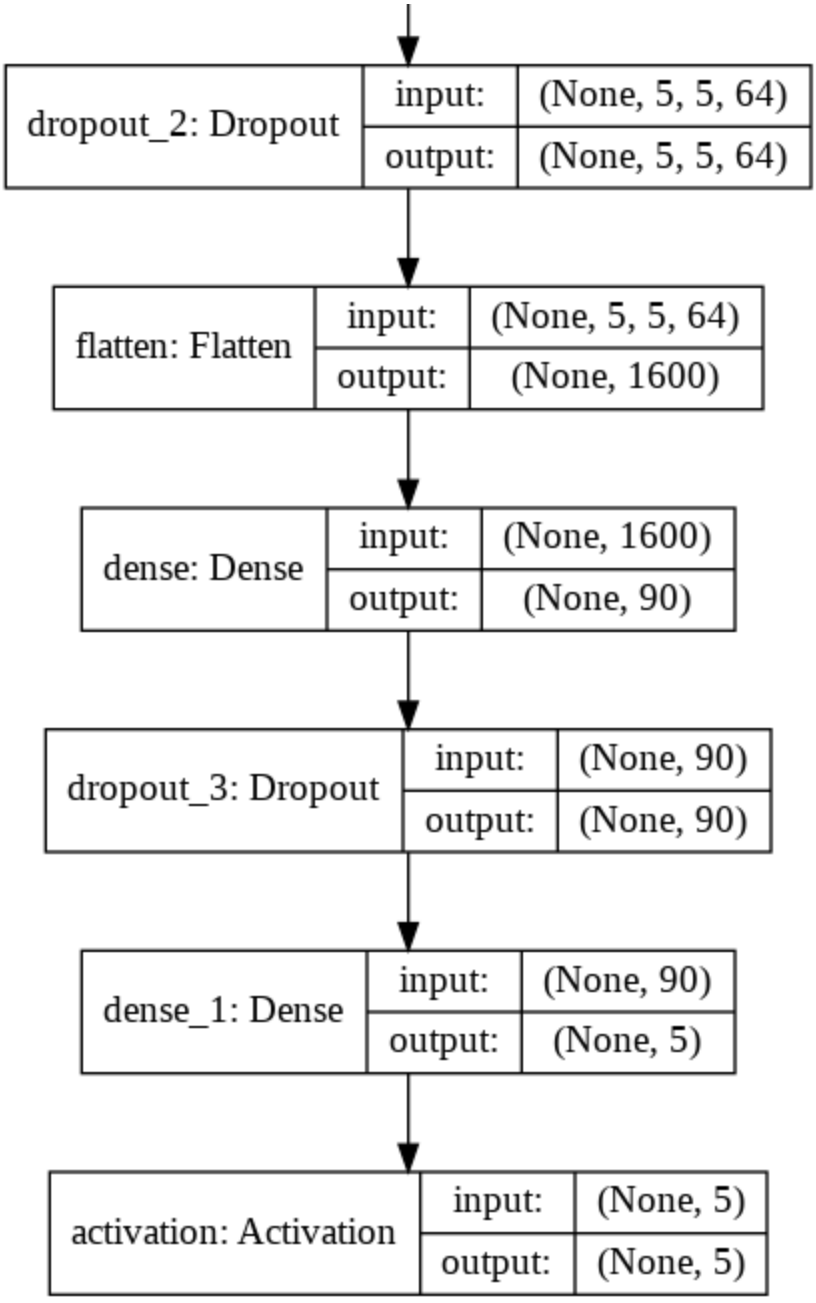
Since CNN is computationally intensive, in our report we used a Hardware Accelerator that boosted the training of the model in the Google Collab program. Using the tab Runtime → change runtime type → select GPU (graphic processing unit).

The following outputs are the model summary and plot model of the hidden layer **h = 90** that show the total number of weights and biases in the network.

```
trained_model_1.summary() # for n_hidden = 90
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dropout (Dropout)            (None, 20, 20, 1)         0
_________________________________________________________________
conv2d (Conv2D)              (None, 20, 20, 32)        832
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 10, 10, 32)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 10, 10, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 10, 10, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
dropout_2 (Dropout)          (None, 5, 5, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 1600)              0
_________________________________________________________________
dense (Dense)                (None, 90)                144090
_________________________________________________________________
dropout_3 (Dropout)          (None, 90)                0
_________________________________________________________________
dense_1 (Dense)              (None, 5)                 455
_________________________________________________________________
activation (Activation)      (None, 5)                 0
=================================================================
Total params: 163,873
Trainable params: 163,873
Non-trainable params: 0
_________________________________________________________________
```

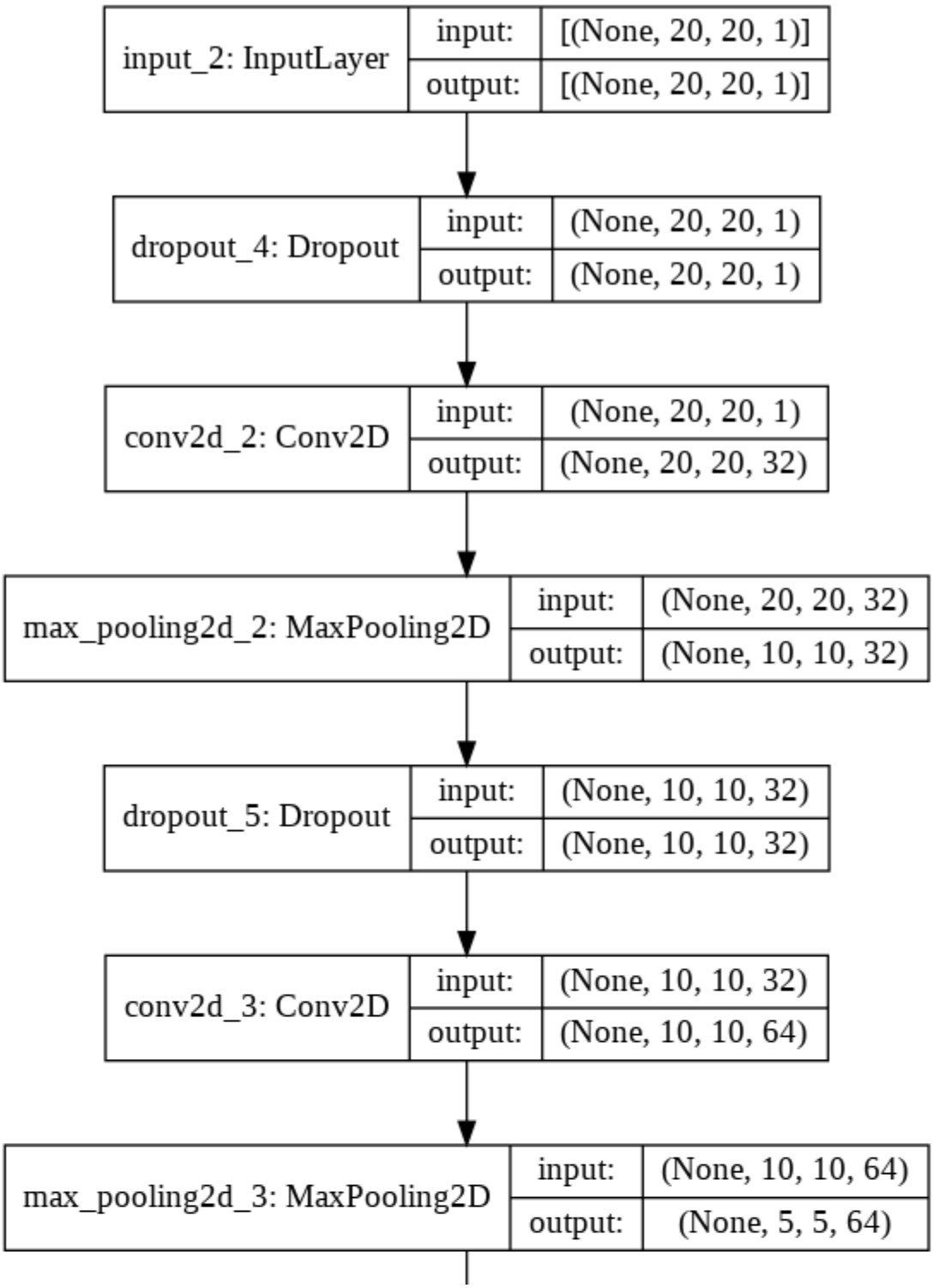

input_1: InputLayer
input:
[(None, 20, 20, 1)]
output:
[(None, 20, 20, 1)]
dropout: Dropout
input:
(None, 20, 20, 1)
output:
(None, 20, 20, 1)
conv2d: Conv2D
input:
(None, 20, 20, 1)
output:
(None, 20, 20, 32)
max_pooling2d: MaxPooling2D
input:
(None, 20, 20, 32)
output:
(None, 10, 10, 32)
dropout_1: Dropout
input:
(None, 10, 10, 32)
output:
(None, 10, 10, 32)
conv2d_1: Conv2D
input:
(None, 10, 10, 32)
output:
(None, 10, 10, 64)
max_pooling2d_1: MaxPooling2D
input:
(None, 10, 10, 64)
output:
(None, 5, 5, 64)

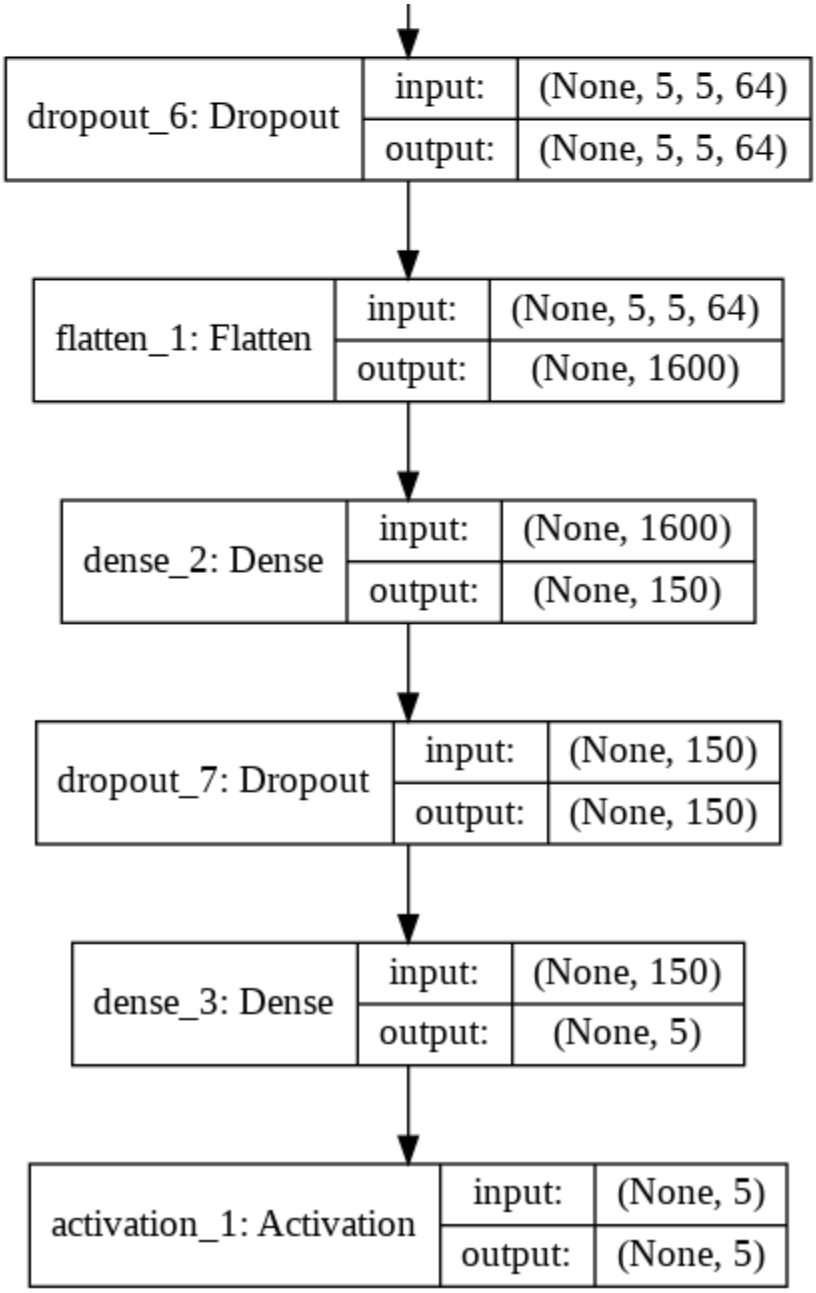

dropout_2: Dropout
input: (None, 5, 5, 64)
output: (None, 5, 5, 64)
flatten: Flatten
input: (None, 5, 5, 64)
output: (None, 1600)
dense: Dense
input: (None, 1600)
output: (None, 90)
dropout_3: Dropout
input: (None, 90)
output: (None, 90)
dense_1: Dense
input: (None, 90)
output: (None, 5)
activation: Activation
input: (None, 5)
output: (None, 5)

The following outputs are the model summary and plot model for the using the hidden layer **h = 150**

```
trained_model_2.summary() # for n_hidden = 150
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dropout_4 (Dropout)          (None, 20, 20, 1)         0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 20, 20, 32)        832
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 10, 10, 32)        0
_________________________________________________________________
dropout_5 (Dropout)          (None, 10, 10, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 10, 10, 64)        18496
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
dropout_6 (Dropout)          (None, 5, 5, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1600)              0
_________________________________________________________________
dense_2 (Dense)              (None, 150)               240150
_________________________________________________________________
dropout_7 (Dropout)          (None, 150)               0
_________________________________________________________________
dense_3 (Dense)              (None, 5)                 755
_________________________________________________________________
activation_1 (Activation)    (None, 5)                 0
=================================================================
Total params: 260,233
Trainable params: 260,233
Non-trainable params: 0
_________________________________________________________________
```

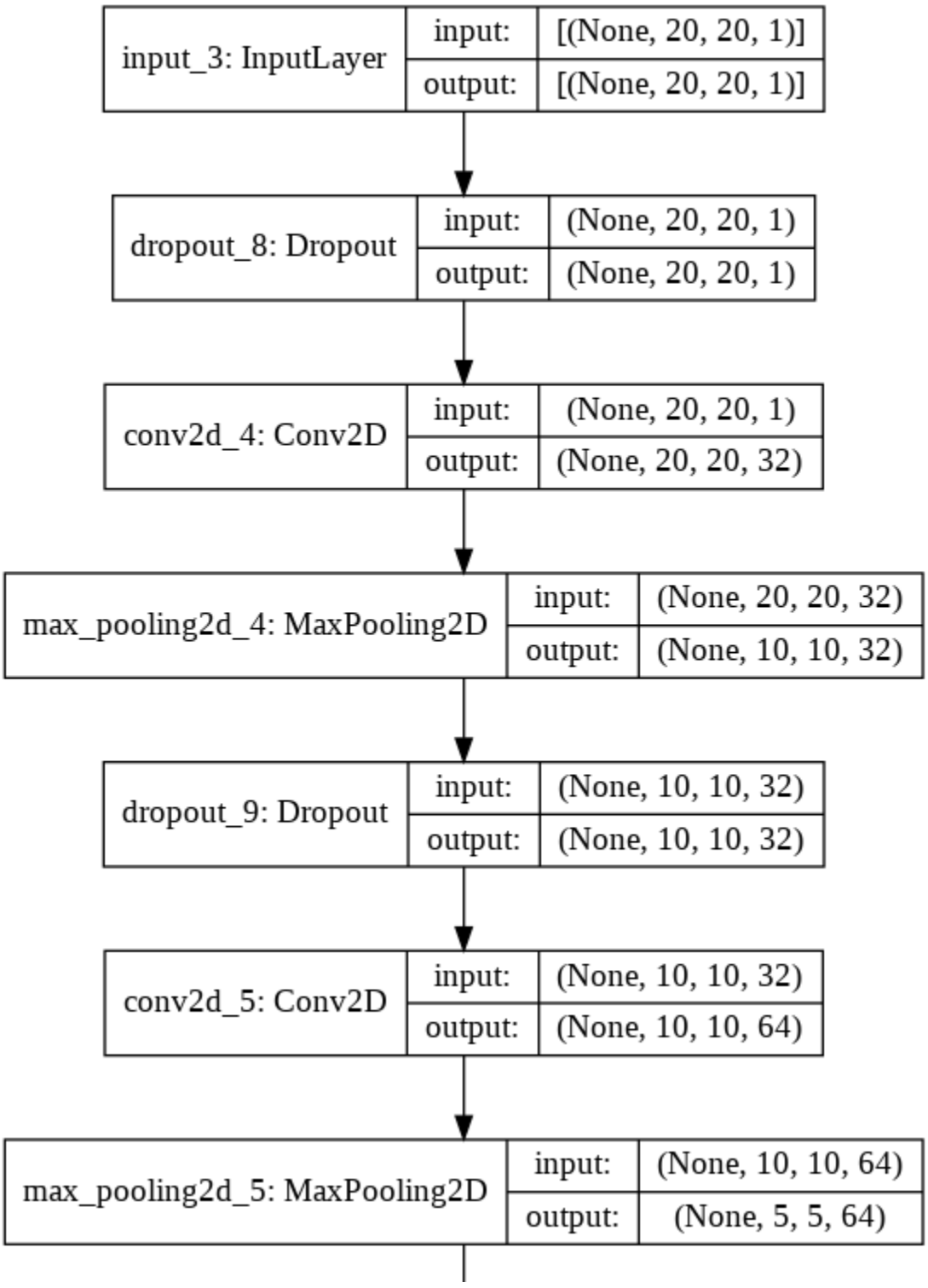

input_2: InputLayer
input:
[(None, 20, 20, 1)]
output:
[(None, 20, 20, 1)]
dropout_4: Dropout
input:
(None, 20, 20, 1)
output:
(None, 20, 20, 1)
conv2d_2: Conv2D
input:
(None, 20, 20, 1)
output:
(None, 20, 20, 32)
max_pooling2d_2: MaxPooling2D
input:
(None, 20, 20, 32)
output:
(None, 10, 10, 32)
dropout_5: Dropout
input:
(None, 10, 10, 32)
output:
(None, 10, 10, 32)
conv2d_3: Conv2D
input:
(None, 10, 10, 32)
output:
(None, 10, 10, 64)
max_pooling2d_3: MaxPooling2D
input:
(None, 10, 10, 64)
output:
(None, 5, 5, 64)

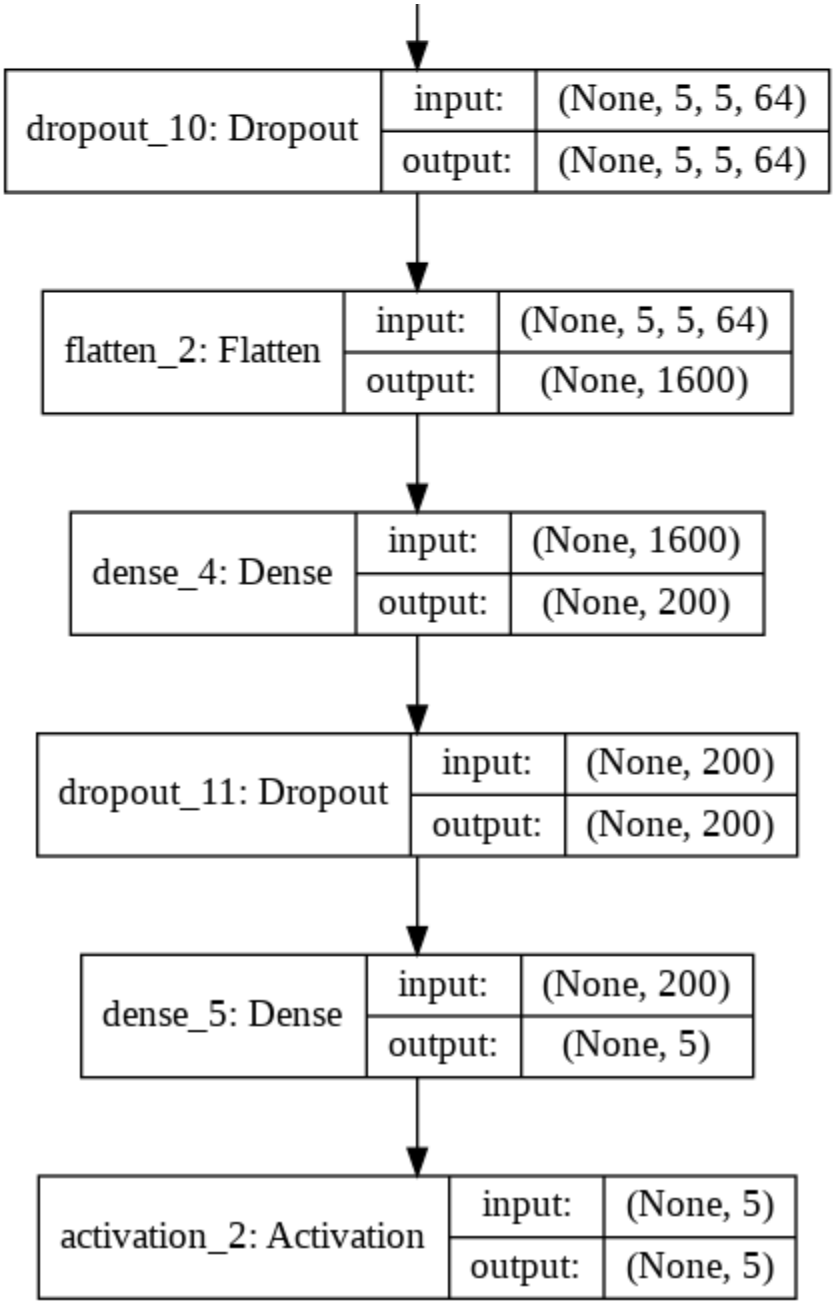

dropout_6: Dropout
input: (None, 5, 5, 64)
output: (None, 5, 5, 64)
flatten_1: Flatten
input: (None, 5, 5, 64)
output: (None, 1600)
dense_2: Dense
input: (None, 1600)
output: (None, 150)
dropout_7: Dropout
input: (None, 150)
output: (None, 150)
dense_3: Dense
input: (None, 150)
output: (None, 5)
activation_1: Activation
input: (None, 5)
output: (None, 5)

And lastly, the following outputs are the model summary and plot model for the using the hidden layer **h = 200**

```
trained_model_3.summary() # for n_hidden = 200
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dropout_8 (Dropout)          (None, 20, 20, 1)         0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 20, 20, 32)        832
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 10, 10, 32)        0
_________________________________________________________________
dropout_9 (Dropout)          (None, 10, 10, 32)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 10, 10, 64)        18496
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
dropout_10 (Dropout)         (None, 5, 5, 64)          0
_________________________________________________________________
flatten_2 (Flatten)          (None, 1600)              0
_________________________________________________________________
dense_4 (Dense)              (None, 200)               320200
_________________________________________________________________
dropout_11 (Dropout)         (None, 200)               0
_________________________________________________________________
dense_5 (Dense)              (None, 5)                 1005
_________________________________________________________________
activation_2 (Activation)    (None, 5)                 0
=================================================================
Total params: 340,533
Trainable params: 340,533
Non-trainable params: 0
_________________________________________________________________
```

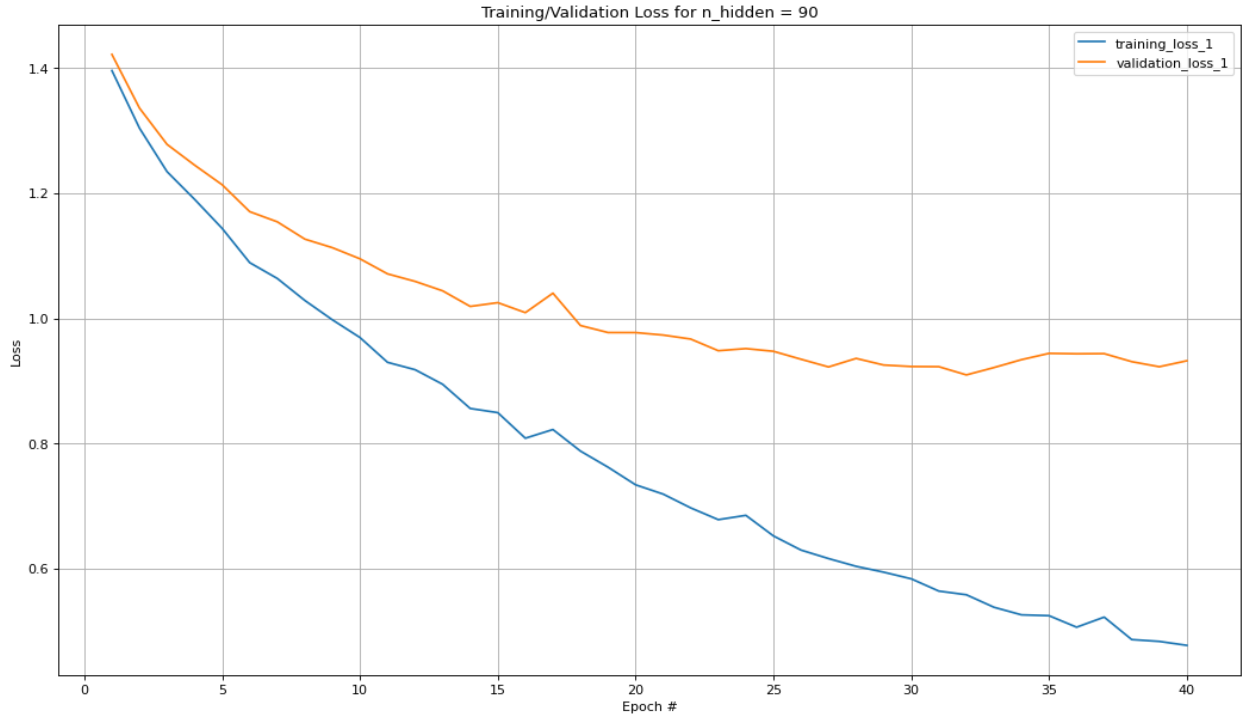

input_3: InputLayer
input: [(None, 20, 20, 1)]
output: [(None, 20, 20, 1)]
dropout_8: Dropout
input: (None, 20, 20, 1)
output: (None, 20, 20, 1)
conv2d_4: Conv2D
input: (None, 20, 20, 1)
output: (None, 20, 20, 32)
max_pooling2d_4: MaxPooling2D
input: (None, 20, 20, 32)
output: (None, 10, 10, 32)
dropout_9: Dropout
input: (None, 10, 10, 32)
output: (None, 10, 10, 32)
conv2d_5: Conv2D
input: (None, 10, 10, 32)
output: (None, 10, 10, 64)
max_pooling2d_5: MaxPooling2D
input: (None, 10, 10, 64)
output: (None, 5, 5, 64)

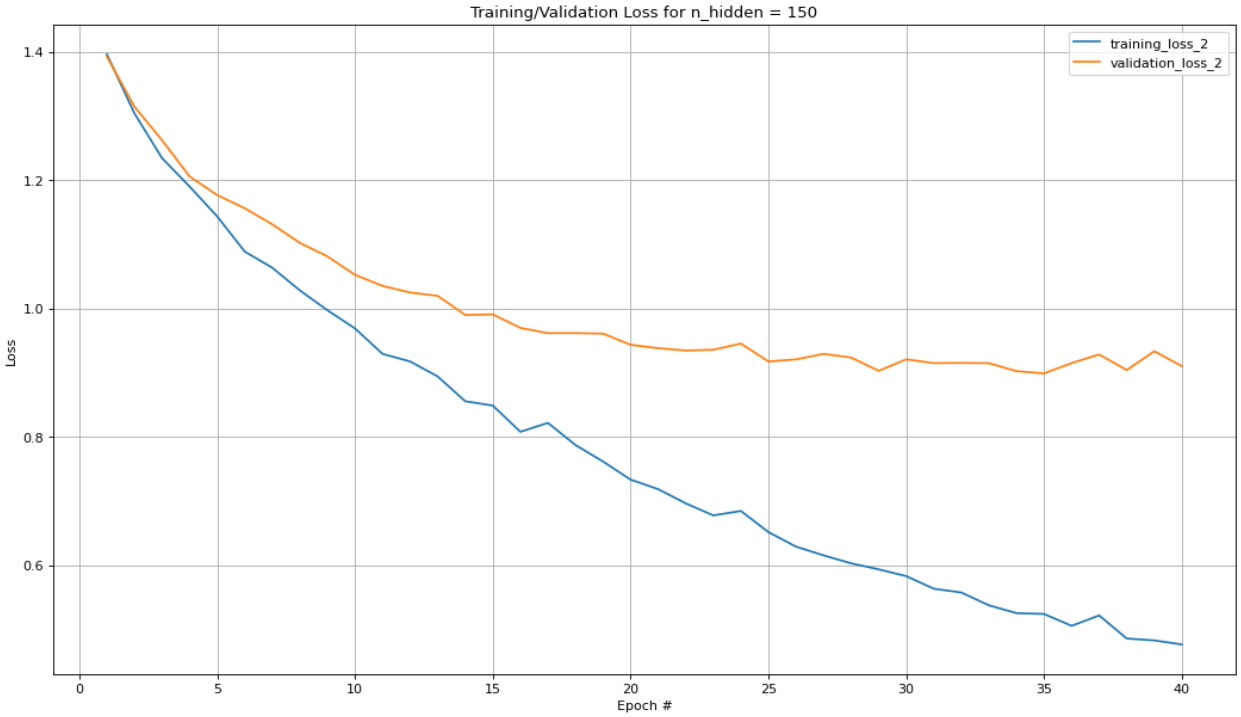

dropout_10: Dropout
input: (None, 5, 5, 64)
output: (None, 5, 5, 64)
flatten_2: Flatten
input: (None, 5, 5, 64)
output: (None, 1600)
dense_4: Dense
input: (None, 1600)
output: (None, 200)
dropout_11: Dropout
input: (None, 200)
output: (None, 200)
dense_5: Dense
input: (None, 200)
output: (None, 5)
activation_2: Activation
input: (None, 5)
output: (None, 5)

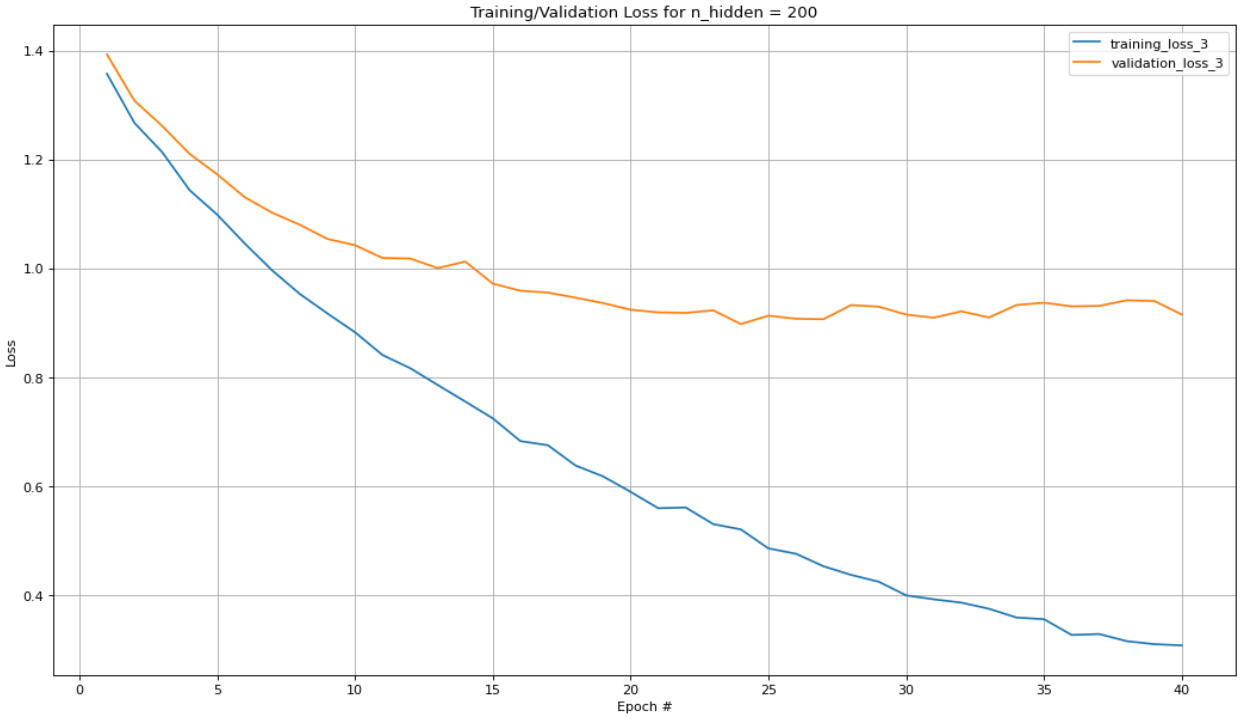
STEP 4: Training CNN

To launch the training of the CNN, we used the categorical cross entropy as a loss function, ReLu for the neuron activation response function and Softmax for the Output. Our batch size = sqrt(training set) = sqrt(14092) = 118. Epochs = 40.

In the graphs of Training/Validation Loss below, we discover that training loss is greater than validation loss in all graphs, which means the model is able to predict on the training data really well, but is not able to generalize correctly to the unseen data. so the models of 3 have overfitting issues.



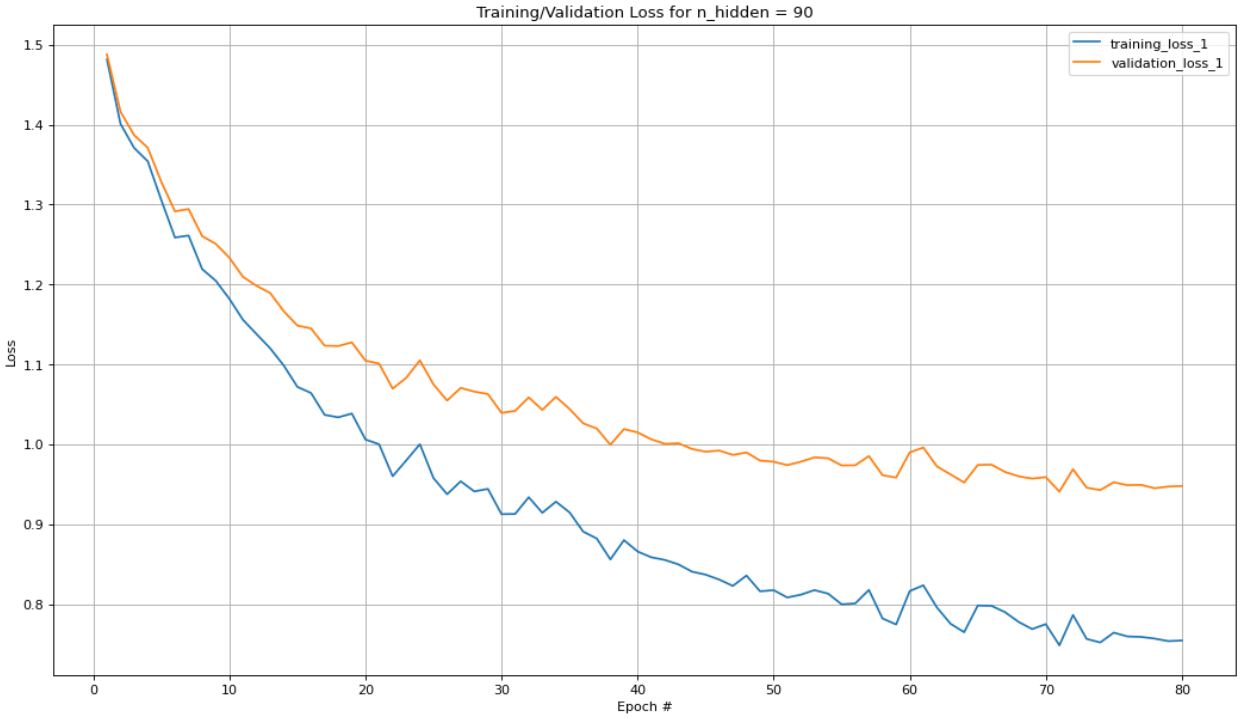


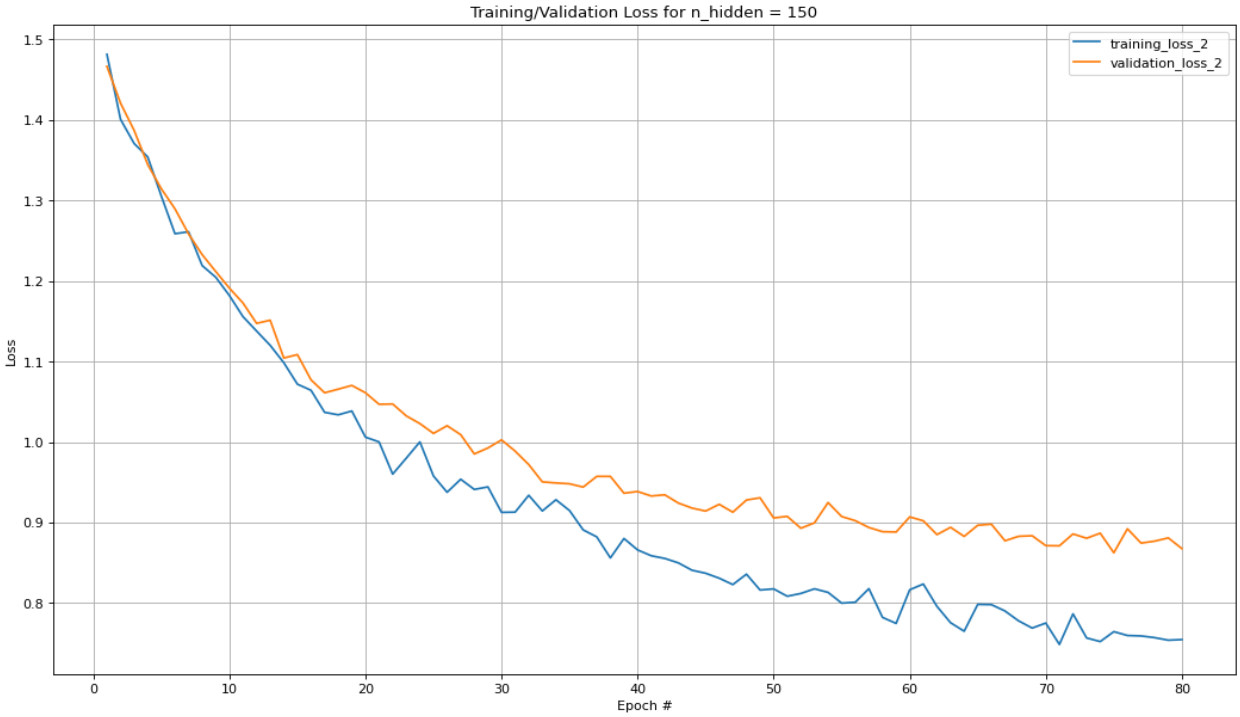


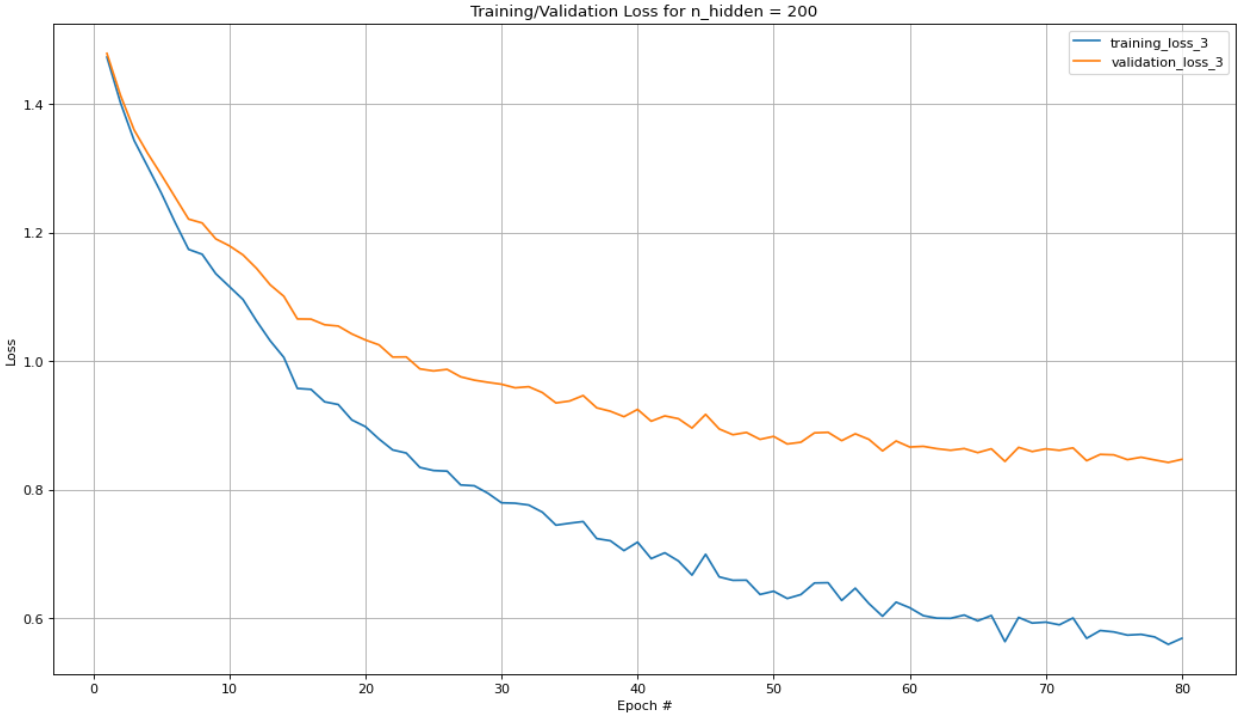
The solution of reducing overfitting in a convolutional neural network is to apply a simple but powerful regularization technique called Dropout to all input layers, hidden

layers and output layers to drop some nodes of the network in order to temporarily deactivate or ignore neurons. The Dropout regularization technique helps reduce any effects of overfitting, an error that can happen when a network is too closely fit to the input samples by ignoring a random subset of units in a layer while setting their weights to zero during the phase of training. We choose 0.4 as the dropout rate for the input and hidden layers and 0.2 as the dropout rate for the output layers.

After dropout regularization techniques are added to all the layers, we re-launch the training of the CNN, and increase the epoch size to 80.（Technically, we should train the model more than 80 epochs for obtaining higher test accuracy, but the RAM will crash with GPU accelerator in google colab) However, the values of both training loss and test loss are getting closer based on the new graphs of Training/Validation Loss below, and we are good to go.




Training/Validation Loss for n_hidden = 150
training_loss_2
validation_loss_2
Loss
Epoch #
1.5
1.4
1.3
1.2
1.1
1.0
0.9
0.8
0
10
20
30
40
50
60
70
80


Training/Validation Loss for n_hidden = 200
training_loss_3
validation_loss_3
Loss
Epoch #
1.4
1.2
1.0
0.8
0.6
0
10
20
30
40
50
60
70
80

STEP 5: Performance Analysis

We observe the following train and test performances for the 3 selected hidden layer values h = 90, 150, and 200:

```
print(trainperf_1)
print(trainperf_2)
print(trainperf_3)
```

```
0.8040732330400228
0.8578626170877094
0.8685779165483962
```

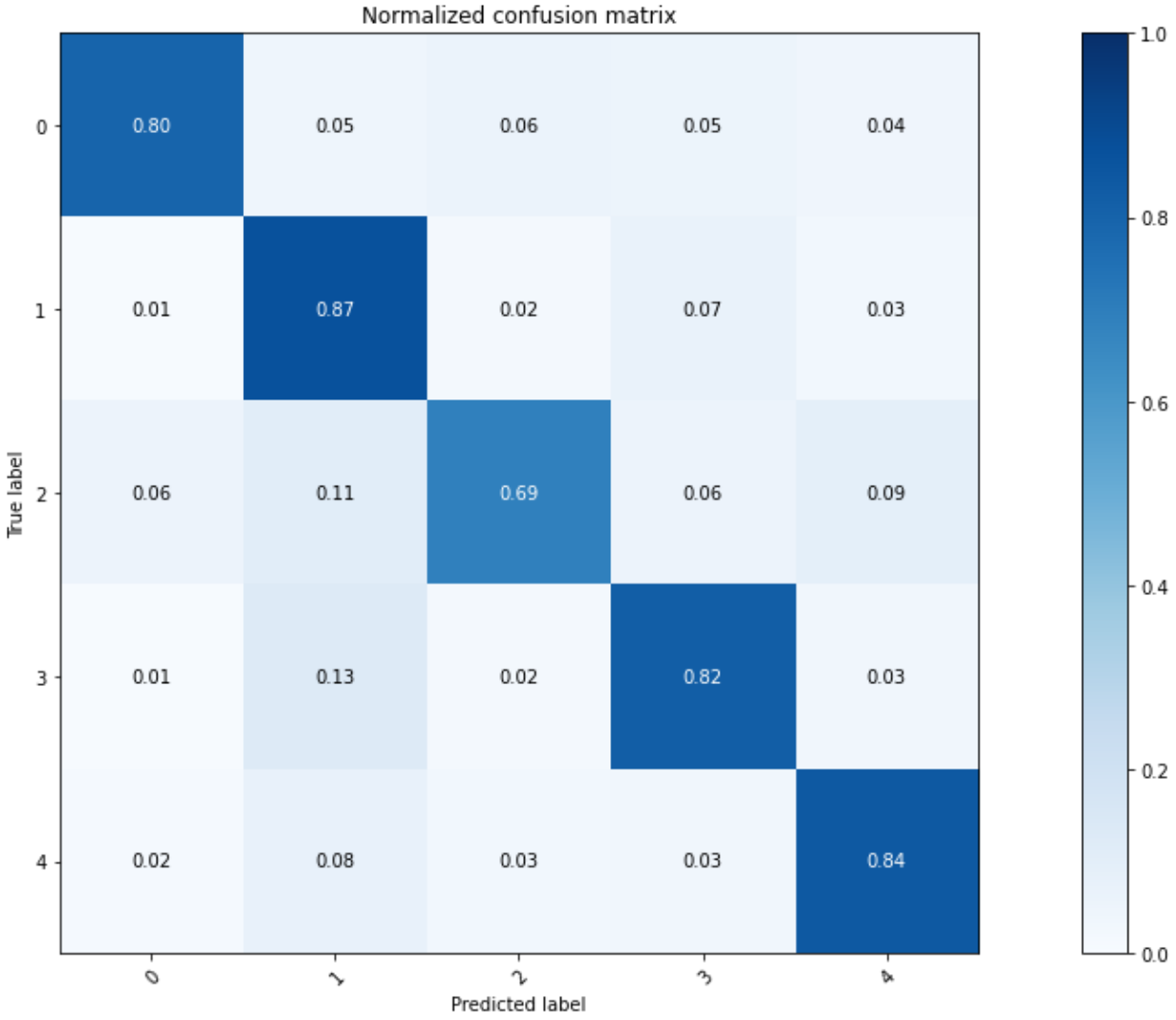
```
print(testperf_1)
print(testperf_2)
print(testperf_3)
```

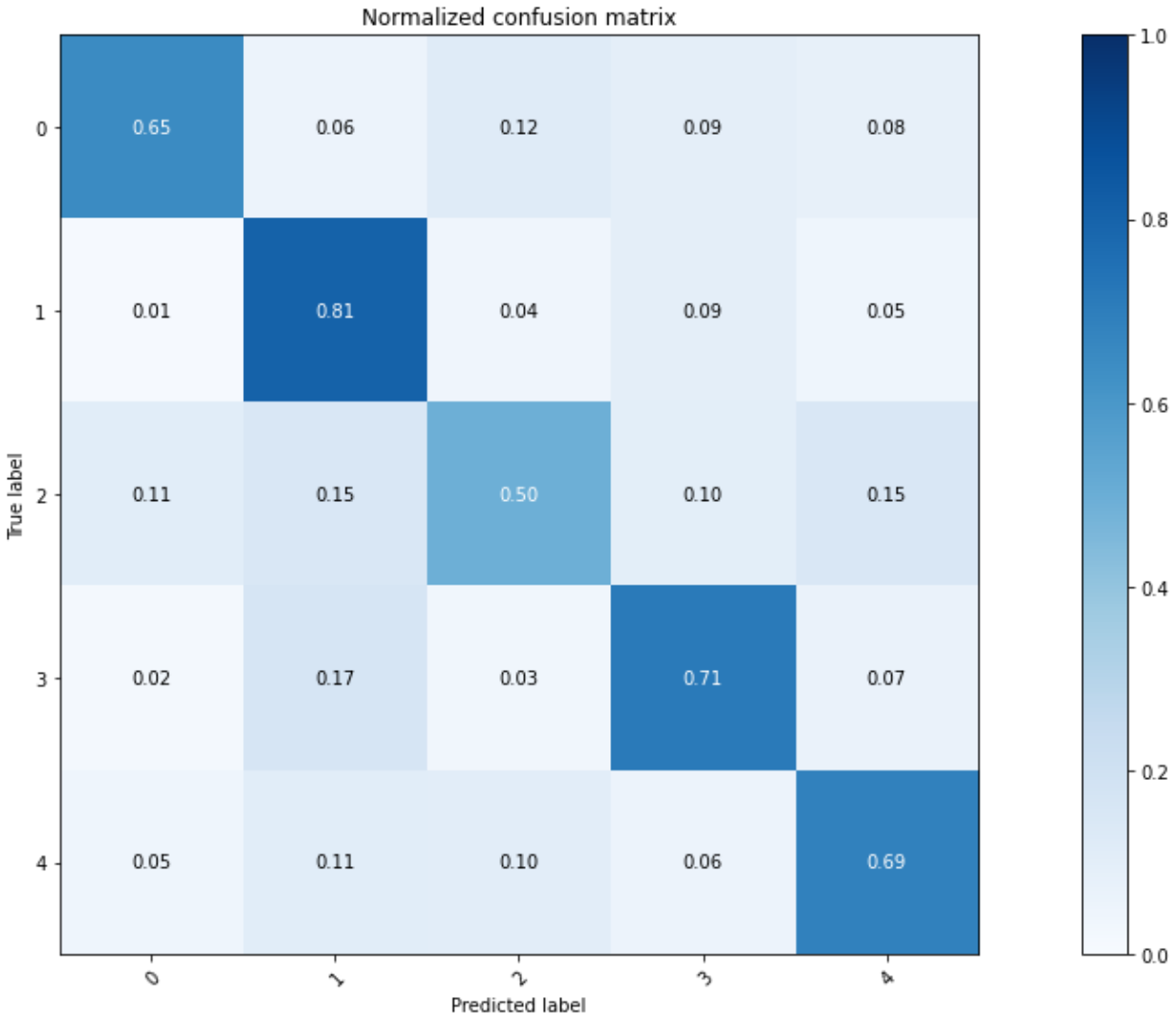
```
0.6702610669693531
0.6932463110102156
0.7017593643586834
```

We observe that both the training and test show to have improved performance as the selected hidden layers increased. Overall our training performance is higher than the test. This indicates we are moving in the right direction.

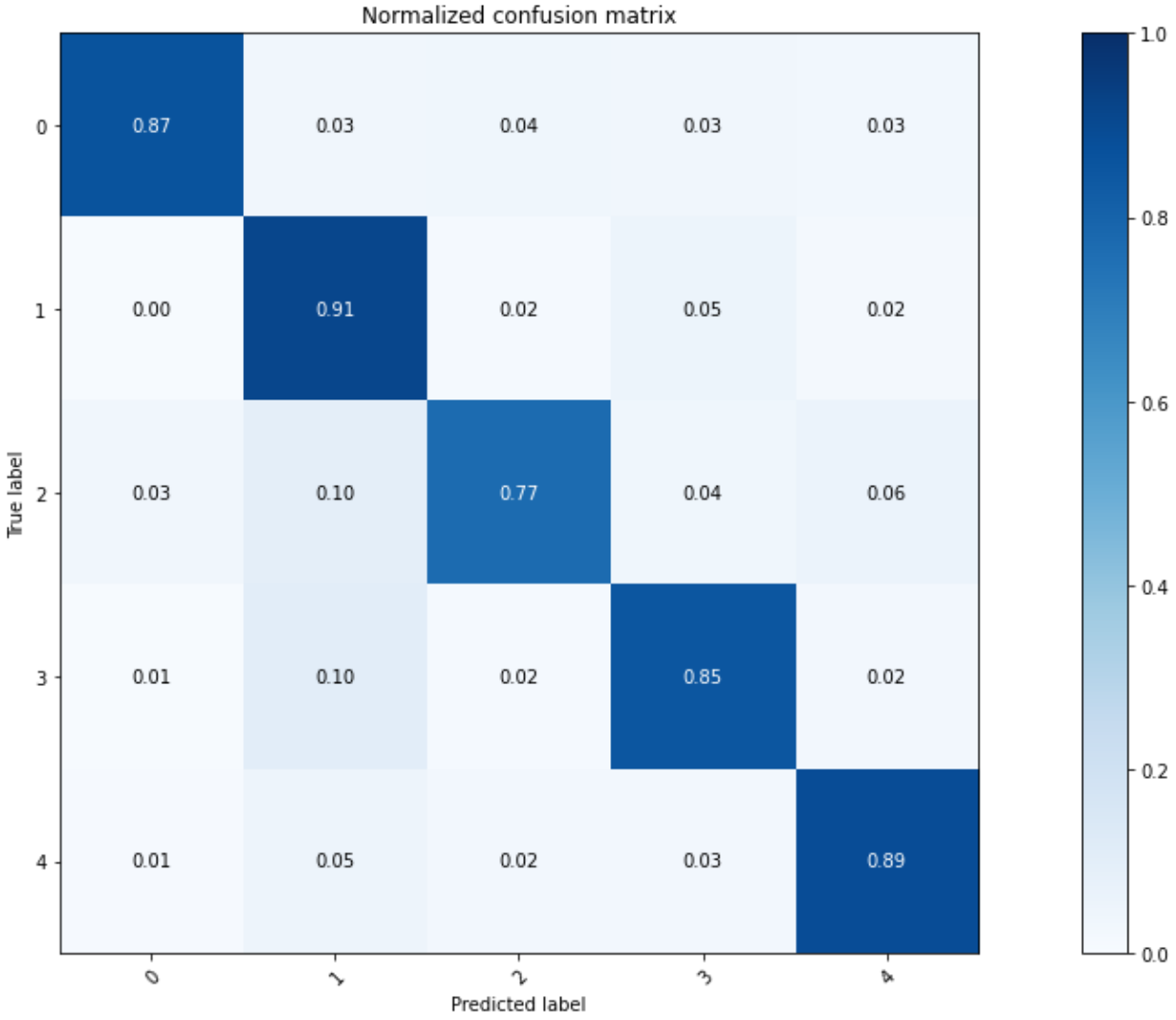
For these three performances, we get the following (5 x 5) confusion matrices of the selected hidden h:

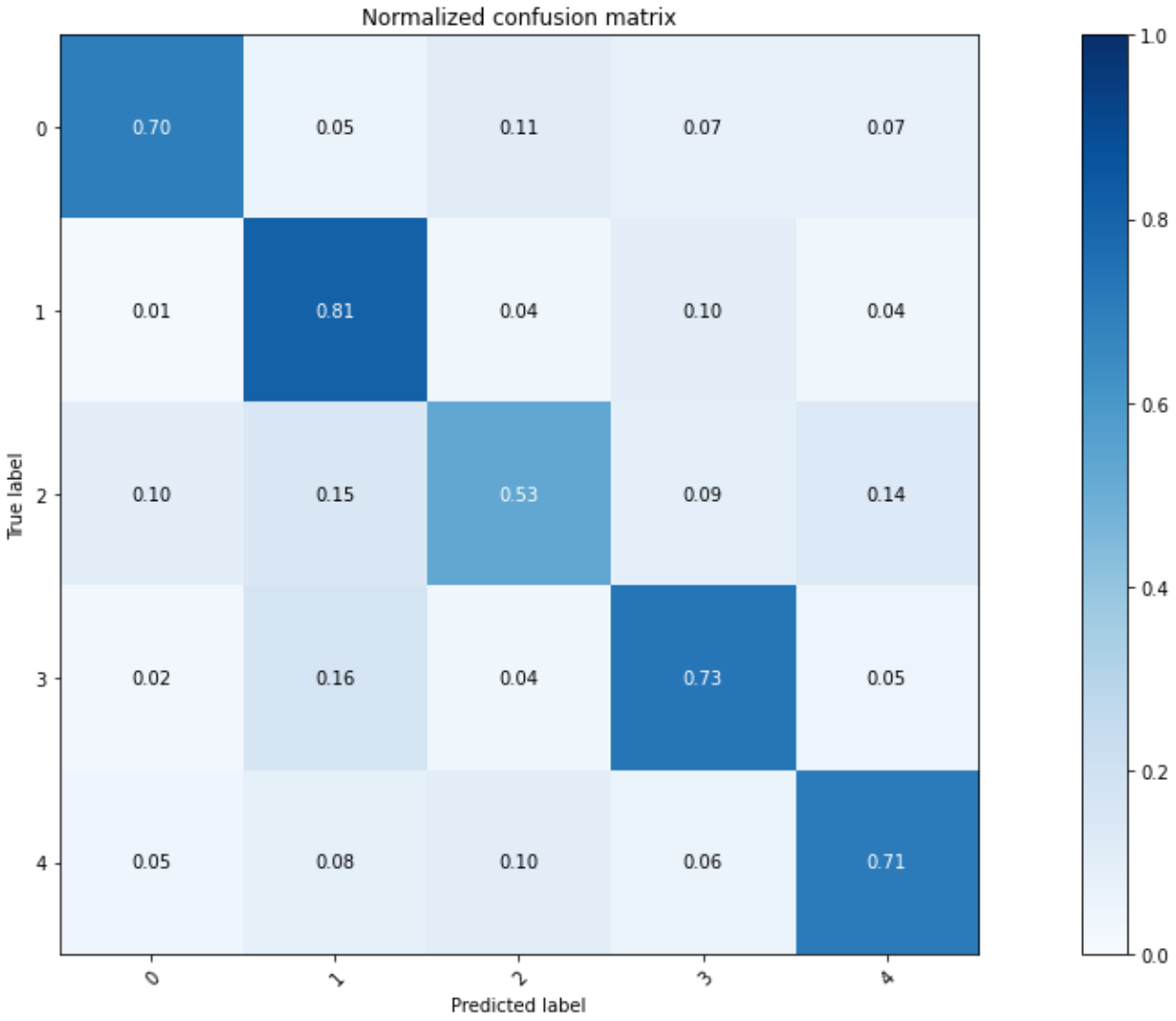
Confusion matrix training and testing set respectfully for the hidden layer **h = 90**. We can observe that font class 3: LUCIDA, appears to have the lowest training performance at 69% while class 2: FRANKLIN has the highest at 87%. Similar pattern is shown in the test performance.




Normalized confusion matrix
0.65 0.06 0.12 0.09 0.08
0.01 0.81 0.04 0.09 0.05
0.11 0.15 0.50 0.10 0.15
0.02 0.17 0.03 0.71 0.07
0.05 0.11 0.10 0.06 0.69
True label
Predicted label
0
1
2
3
4
1.0
0.8
0.6
0.4
0.2
0.0

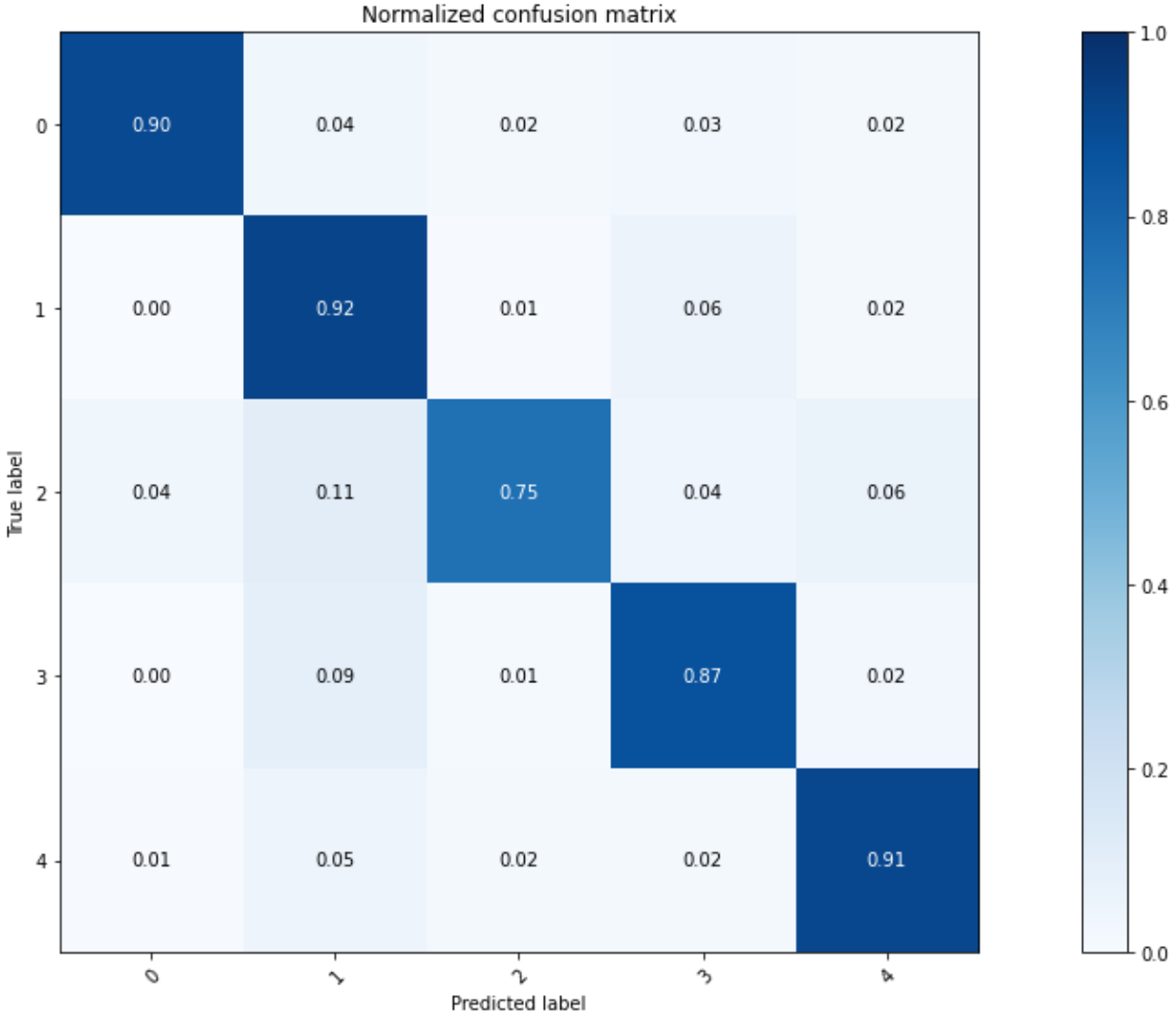
Confusion matrix training and testing set respectfully for the hidden layer **h = 150.** We can observe a similar pattern for this hidden layer as well. Font class 3: LUCIDA appears to have the lowest training performance at 77% while class 2: FRANKLIN has the highest at 91%. Similar pattern is shown in the test performance.

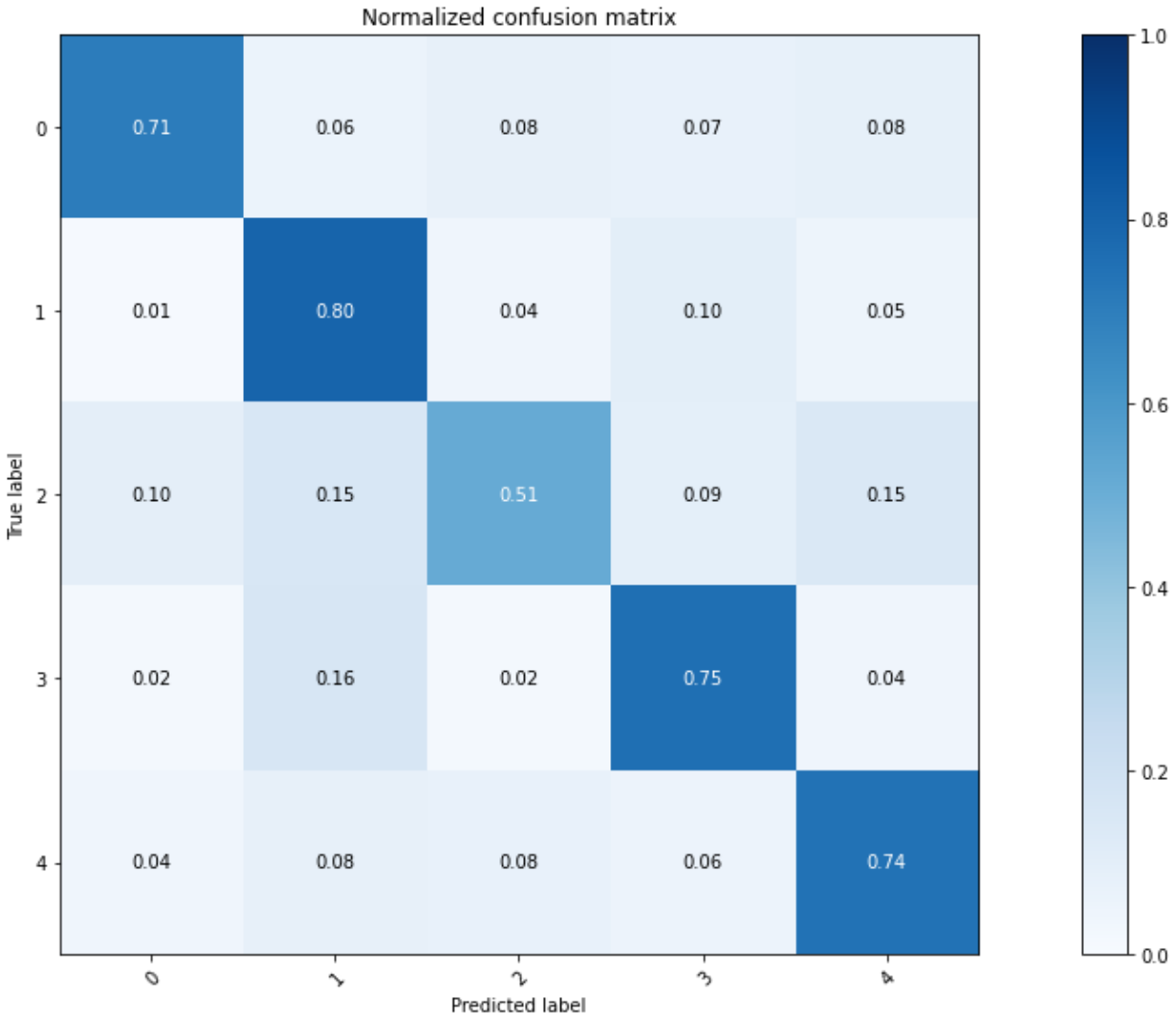



Normalized confusion matrix
0.70 0.05 0.11 0.07 0.07
0.01 0.81 0.04 0.10 0.04
0.10 0.15 0.53 0.09 0.14
0.02 0.16 0.04 0.73 0.05
0.05 0.08 0.10 0.06 0.71
True label
Predicted label
0
1
2
3
4
1.0
0.8
0.6
0.4
0.2
0.0

Confusion matrix training and testing set respectfully for the hidden layer **h = 200.**

Lastly, we can observe that font class 3: LUCIDA, appears to have the lowest training performance at 75% while class 2: FRANKLIN has the highest at 92%. Overall, we see an improvement in performance as we increase our hidden layers, therefore know we are headed in the right direction. Similar pattern is shown in the test performance.




Normalized confusion matrix
0.71 0.06 0.08 0.07 0.08
0.01 0.80 0.04 0.10 0.05
0.10 0.15 0.51 0.09 0.15
0.02 0.16 0.02 0.75 0.04
0.04 0.08 0.08 0.06 0.74
True label
Predicted label
0
1
2
3
4
1.0
0.8
0.6
0.4
0.2
0.0

Recall the training and test set performance in STEP 5, **h = 90, 150, and 200**:

```
print(trainperf_1)
print(trainperf_2)
print(trainperf_3)
```

```
0.8040732330400228
0.8578626170877094
0.8685779165483962
```

```
print(testperf_1)
print(testperf_2)
print(testperf_3)
```

```
0.6702610669693531
0.6932463110102156
0.7017593643586834
```

We tested the training and rest performance using higher hidden layer values of **h = 200, 300, and 400** and observe that there is indeed an improvement in the performances if we use higher h values. Therefore it would be interesting to explore the model with these higher hidden h layers.

```
] print(trainperf_1)
  print(trainperf_2)
  print(trainperf_3)
```

```
  0.8780158955435708
  0.8944791370990633
  0.9100908316775476
```

```
] print(testperf_1)
  print(testperf_2)
  print(testperf_3)
```

```
  0.7014755959137344
  0.7114074914869466
  0.7202043132803633
```

Conclusion

In this report we explored Convolutional Neural Networks also called CNN. It is a machine learning model that allows one to extract higher representations for image content. After taking a 2D input image and processing it through two convolution layers, two max-pooling, flattening into 1D and using a Dropout technique, we managed to build a model with high prediction performances of the five font classes. As mentioned, as we increased the hidden layer value up to h = 400, our performance for the training and test set continued to show improvement; it would be interesting to explore how high of a h value we can use before it would no longer have a positive effect on the performance.