Convolutional Neural Network Font Classification

4/14/2021

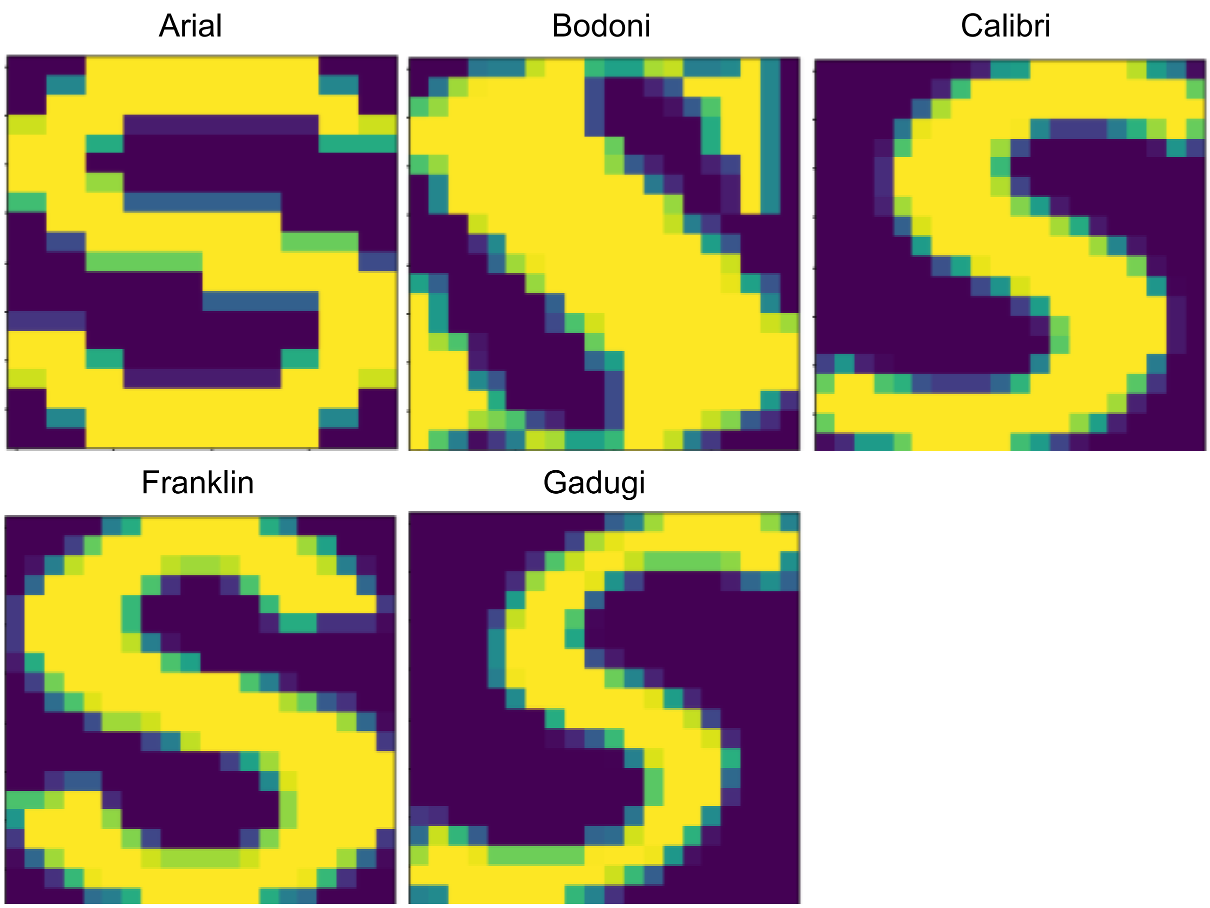
Raphael Suarez

Step 1:

Our data set consisted of five fonts from the Character Font Images Data Set from the UC Irvine Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets/Character+Font+Images). Our five fonts are Arial, Bodoni, Calibri, Franklin, and Gadugi. 3000 cases of each font were taken in order to build our complete set of 15000 cases. 2400 cases from each font were used as training data while the remaining 600 were used for testing.

Step 2:

Each case, containing 400 flattened pixel intensity features, was reshaped in order to reobtain their 20x20 image structure. An example of an “S” from each font can be seen below.



Note: All characters within our data sets were kept as they are. We considered the removal of italic characters however this would result in insufficiently large amounts of observations for fonts such as Bodoni or Gadugi.

Step 3:

Three initial convolutional neural networks were constructed of form:
$input \Rightarrow Conv.\ 1 \Rightarrow Max\ Pooling\ 1 \Rightarrow Conv.\ 2 \Rightarrow Max\ Pooling\ 2 \Rightarrow flatten \Rightarrow hidden \Rightarrow output \Rightarrow softmax \Rightarrow probabilities$

<u>input</u> = Input layer
<u>Conv. 1</u> = Convolutional layer with 16 channels, 5x5 window size, and a stride of 1 which outputs 16 16x16 images
<u>Max Pooling 1</u> = Pooling layer with 16 channels, 2x2 window size, and a stride of 2 that results in 16 8x8 images
<u>Conv. 2</u> = Convolutional layer with 16 channels, 3x3 window size, and a stride of 1 which outputs 16 6x6 images
<u>Max Pooling 2</u> = Pooling layer with 16 channels, 2x2 window size, and a stride of 2 that results in 16 3x3 images
<u>flatten</u> = Transformation of the 16 3x3 images into a long vector of dimension 144
<u>hidden</u> = Final hidden layer of dimension ‘h’ fully connected to the previous 144 neurons. ‘h’ of 90, 150,and 200 respectively between the three models
<u>output</u> = Output of “hidden” of dimension 5
<u>softmax</u> = Softmax function applied to “output” in order to obtain “probabilities” for classification of the fonts
<u>probabilities</u> = Result of “softmax”. Vector of dimension 5 of probabilities of each font. Allows for classification of the fonts based on highest probability.

The total amount of weights and biases from the network is 39,281 for h = 90, 63,641 for h = 150, or 83,941 for h = 200.

The total number of infos brought by the training set is 60,000.

The ratio obtained by our number of infos per model and parameters is then 1.53 for h = 90, 0.94 for h = 150, and 0.71 for h = 200. These higher ratios signal we can generally expect higher performance stability.

Step 4:

We launch three separate CNN models, one for each dimension of the hidden layer (90,150,200).

The dropout technique will be used, which chooses random neurons to ignore by random. These neurons will not be considered when the model is undergoing forward or backward pass. In our case, the probability that each neuron is “ignored” will be 0.5. Dropout technique is mostly used to prevent overfitting of the data. More specifically, it prevents neurons from developing co-dependency on each other during training.

Our loss function for each model will be cross entropy and the neuron response function will be RELU (rectified linear activation function).

Finally, the size of the batches will be approximately the square root of the number of cases in our training set - 110. The number of epochs will be 25 in order for the computation time to not be extreme and to give the model time to stabilize.

h=90

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_10 (Conv2D)           (None, 20, 20, 16)        416
_________________________________________________________________
activation_10 (Activation)   (None, 20, 20, 16)        0
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 10, 10, 16)        0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 10, 10, 16)        2320
_________________________________________________________________
activation_11 (Activation)   (None, 10, 10, 16)        0
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 5, 5, 16)          0
_________________________________________________________________
flatten_5 (Flatten)          (None, 400)               0
_________________________________________________________________
dense_10 (Dense)             (None, 90)                36090
_________________________________________________________________
dropout_5 (Dropout)          (None, 90)                0
_________________________________________________________________
dense_11 (Dense)             (None, 5)                 455
=================================================================
Total params: 39,281
Trainable params: 39,281
Non-trainable params: 0
_________________________________________________________________
```

h=150

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_12 (Conv2D)           (None, 20, 20, 16)        416
_________________________________________________________________
activation_12 (Activation)   (None, 20, 20, 16)        0
_________________________________________________________________
max_pooling2d_12 (MaxPooling (None, 10, 10, 16)        0
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 10, 10, 16)        2320
_________________________________________________________________
activation_13 (Activation)   (None, 10, 10, 16)        0
_________________________________________________________________
max_pooling2d_13 (MaxPooling (None, 5, 5, 16)          0
_________________________________________________________________
flatten_6 (Flatten)          (None, 400)               0
_________________________________________________________________
dense_12 (Dense)             (None, 150)               60150
_________________________________________________________________
dropout_6 (Dropout)          (None, 150)               0
_________________________________________________________________
dense_13 (Dense)             (None, 5)                 755
=================================================================
Total params: 63,641
Trainable params: 63,641
Non-trainable params: 0
_________________________________________________________________
```

h=200

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_14 (Conv2D)           (None, 20, 20, 16)        416
_________________________________________________________________
activation_14 (Activation)   (None, 20, 20, 16)        0
_________________________________________________________________
max_pooling2d_14 (MaxPooling (None, 10, 10, 16)        0
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 10, 10, 16)        2320
_________________________________________________________________
activation_15 (Activation)   (None, 10, 10, 16)        0
_________________________________________________________________
max_pooling2d_15 (MaxPooling (None, 5, 5, 16)          0
_________________________________________________________________
flatten_7 (Flatten)          (None, 400)               0
_________________________________________________________________
dense_14 (Dense)             (None, 200)               80200
_________________________________________________________________
dropout_7 (Dropout)          (None, 200)               0
_________________________________________________________________
dense_15 (Dense)             (None, 5)                 1005
=================================================================
Total params: 83,941
Trainable params: 83,941
Non-trainable params: 0
_________________________________________________________________
```

The model summaries above give a brief insight into the structure of each layer. More specifically, we can see the difference in the number of parameters when we increase the size of the hidden layer (h).

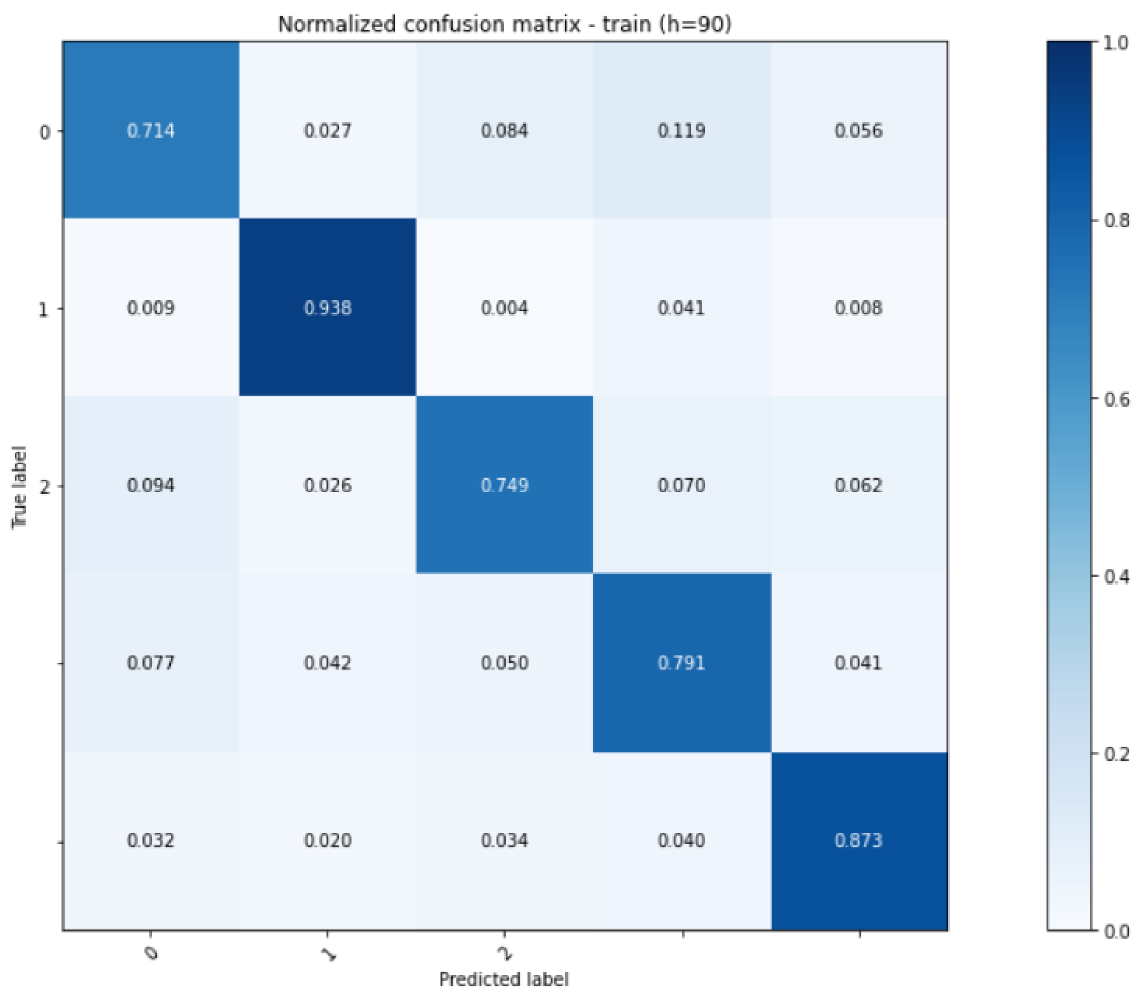
Step 5:

In order to compare the performance of the different model structures, we saved the best performing model of each h value using a checkpoint function. This checkpoint looks at validation accuracy (test set accuracy) and saves a model we can later recall.

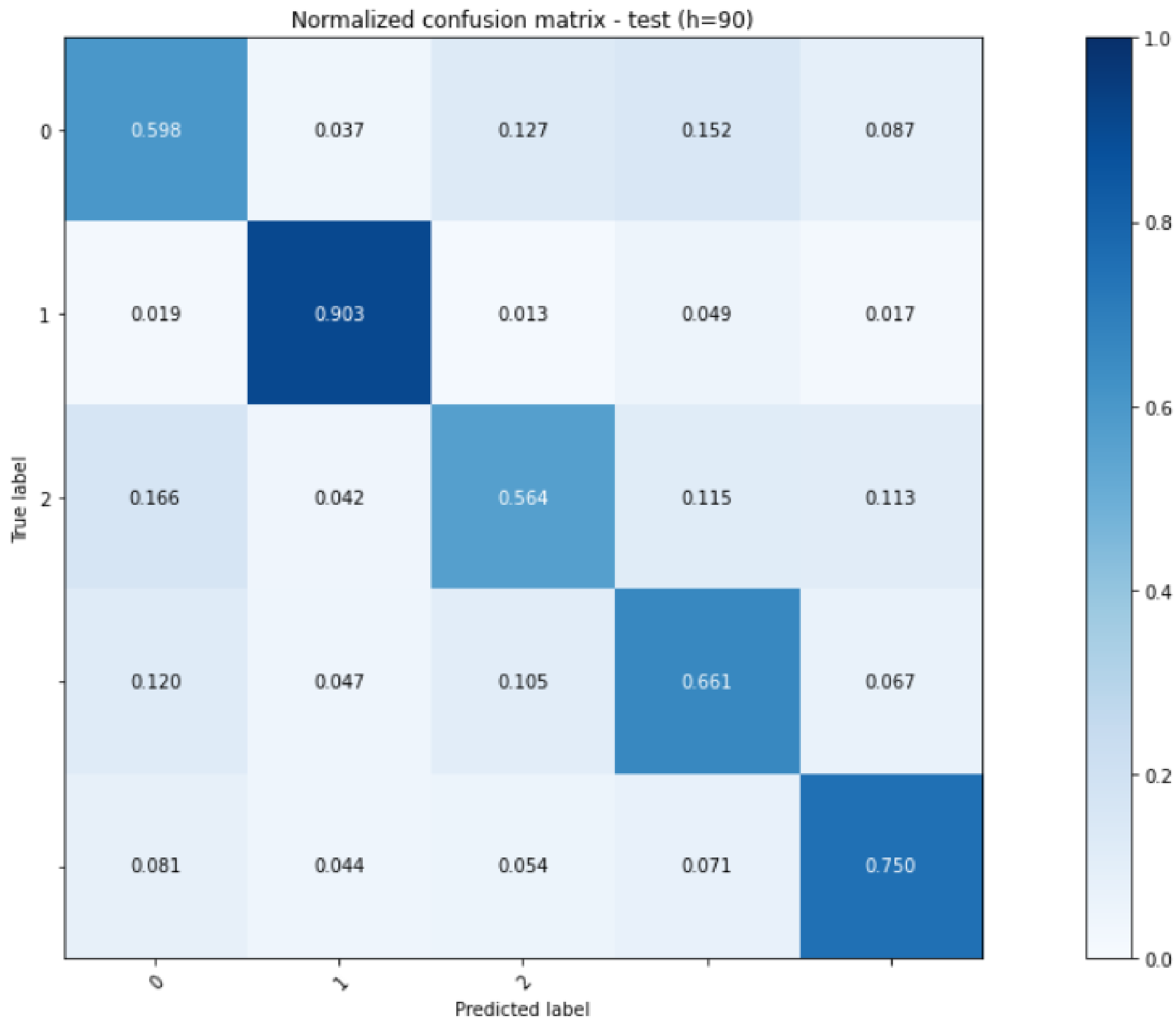
Then, we calculated the confusion matrices for both TRAIN and TEST sets along with the overall accuracy and compared.

h=90

Overall accuracy of TRAIN = 81.2%
TRAIN confusion matrix:



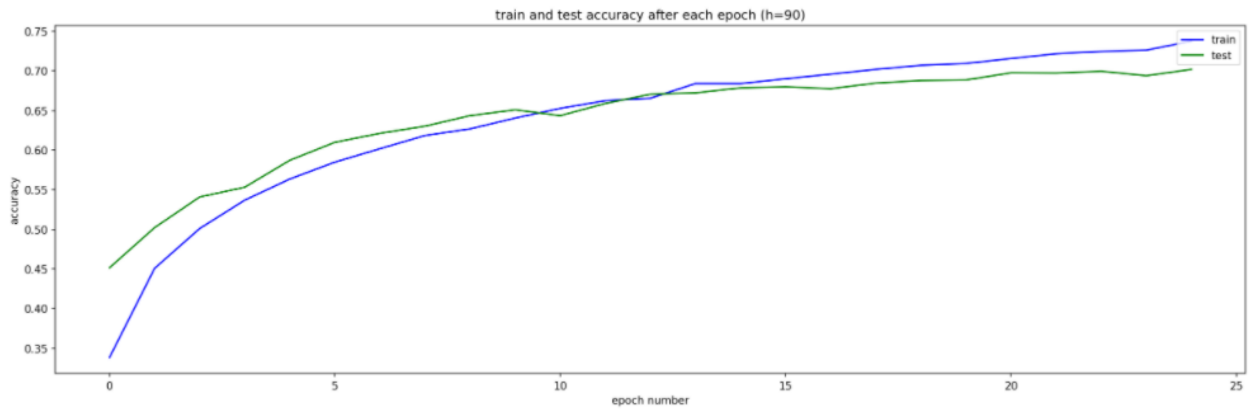
Overall accuracy of TEST = 69.9%
TEST confusion matrix:



As we can see from the confusion matrices above, the prediction trend remained almost the same in both TRAIN and TEST sets. Our model did a great job correctly assigning class 1 cases, with a 90.3% accuracy, and a good job predicting class 4 cases, with a 75.0% accuracy. For both of these fonts, when the model predicted wrong, there wasn't a specific class which received the classification, it was pretty balanced overall.

However, for class 0 cases, the model had a 59.8% accuracy while wrongly predicting 12.7% of the cases to be class 2 and 15.2% of the cases to be class 3.

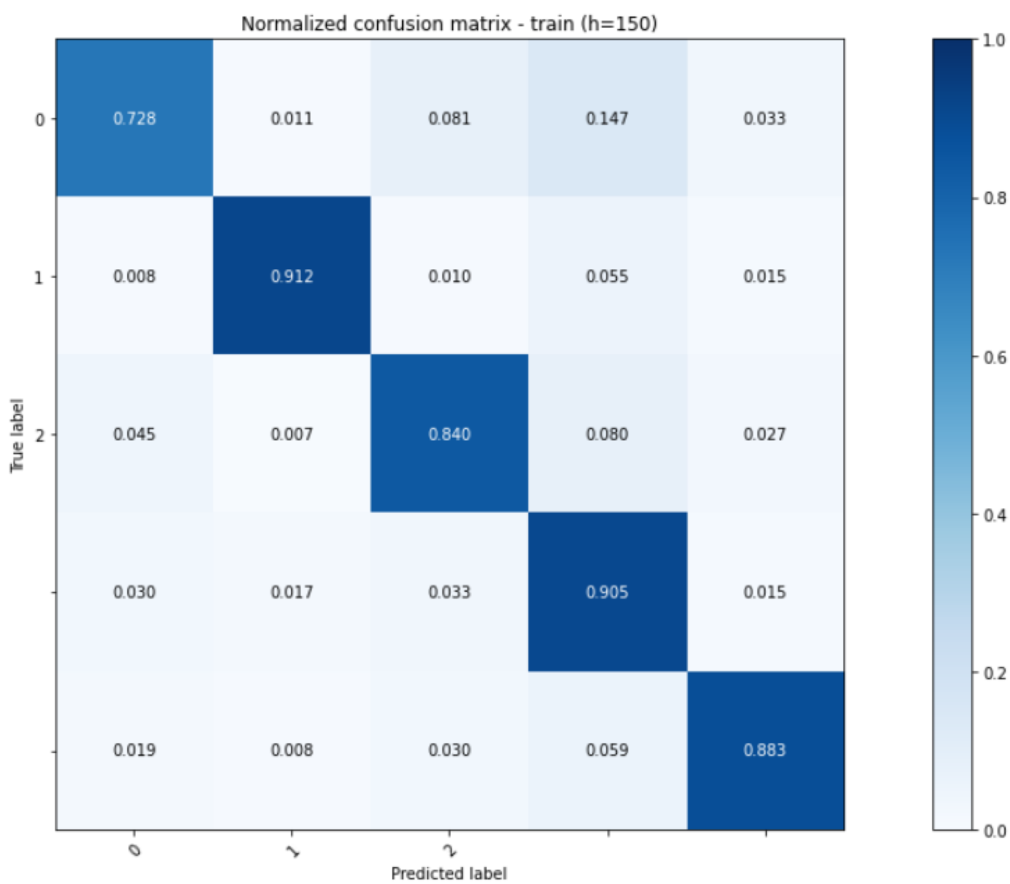
For class 2 cases, the model had 56.4% accuracy, which was the lowest. It predicted 16.6% of the cases to be in class 0.



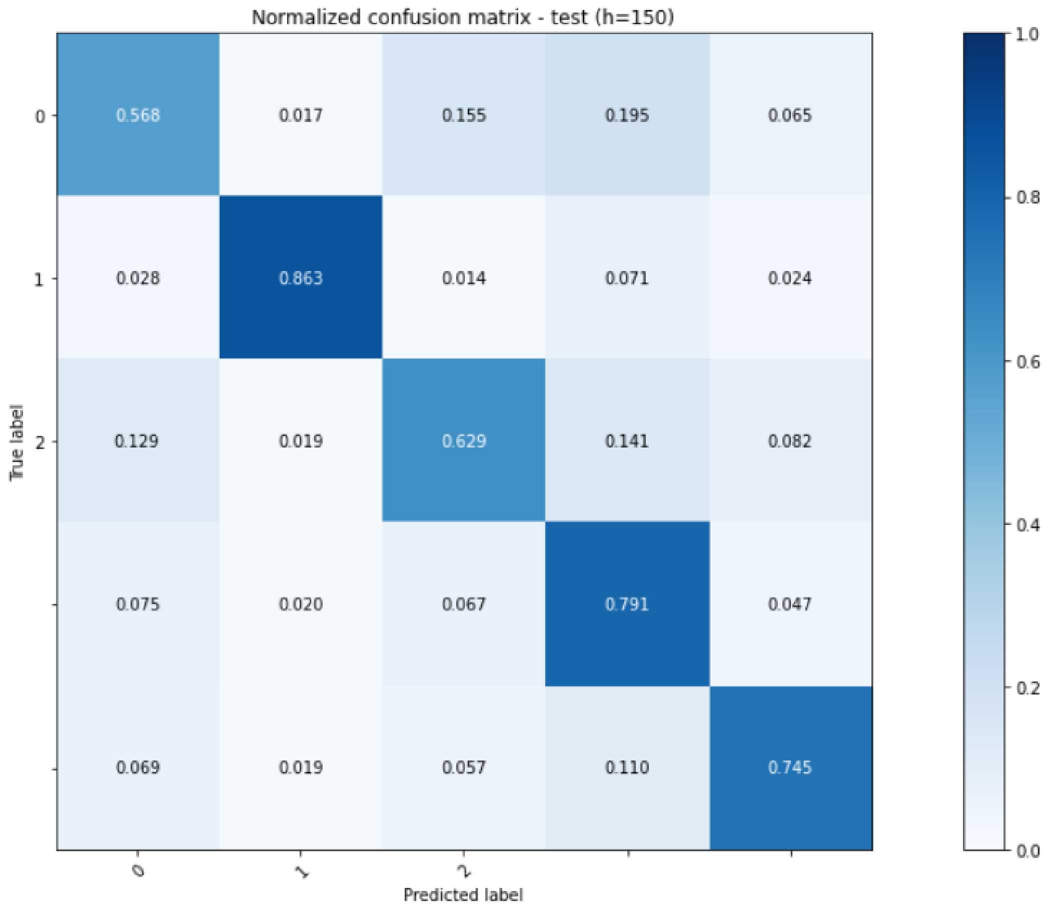
From the above plot we can see the h=90 model's performance on the train and test set over time. Train and test accuracy overlap several times around epochs 9 through 13 and the train and test set improve with relatively similar intensity until around epoch 23.

h=150

Overall accuracy of TRAIN = 85.3%
TRAIN confusion matrix:

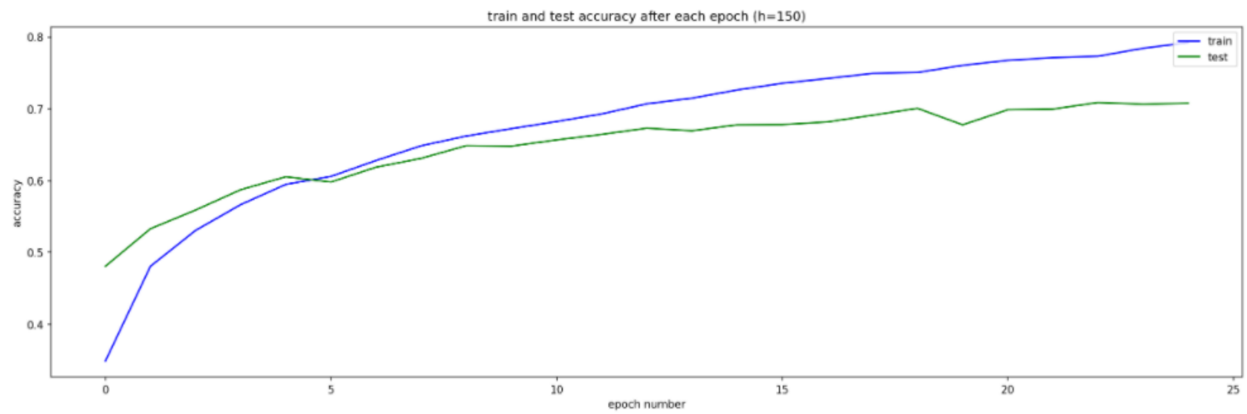


Overall accuracy of TEST = 72.2%
TEST confusion matrix:



For the h=150 model, the TRAIN accuracy improved by 4.1% while the TEST accuracy improved by 2.3%.

It's interesting to notice that while the class 1 and class 4 accuracies decreased (which were the highest performing classes), the overall accuracy increased. This obviously means that the model classified the other classes a bit better, giving more balance to the predictions.
For example, the class 2 accuracy which was previously 56.4% improved to 62.9%. The class 3 accuracy, which was 66.1% improved to 79.1%, which is a very significant change.

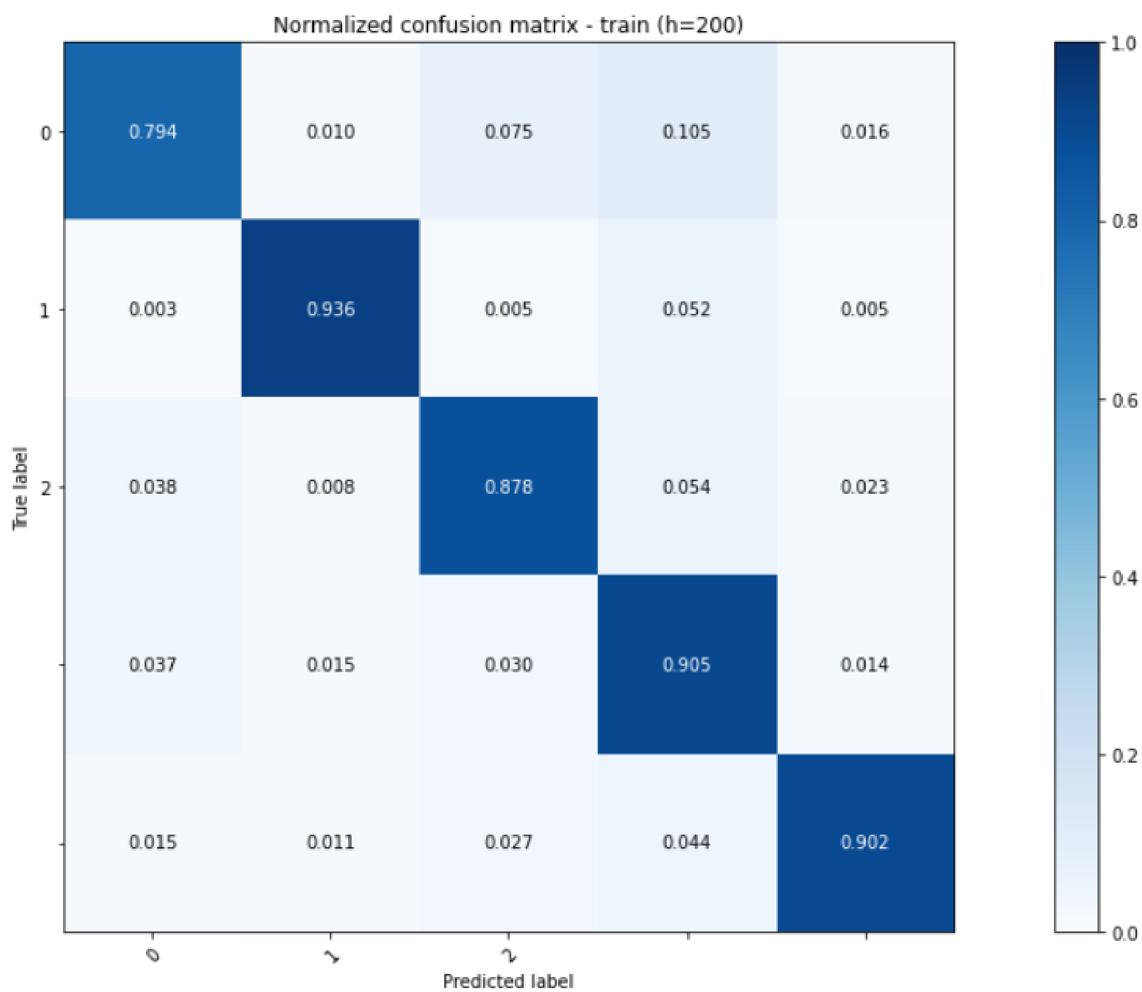


From the above plot we can see the h=150 model's performance on the train and test set over time. Train and test accuracy overlap at around epoch 4 and the train accuracy increases significantly more than test accuracy beginning at around epoch 12.

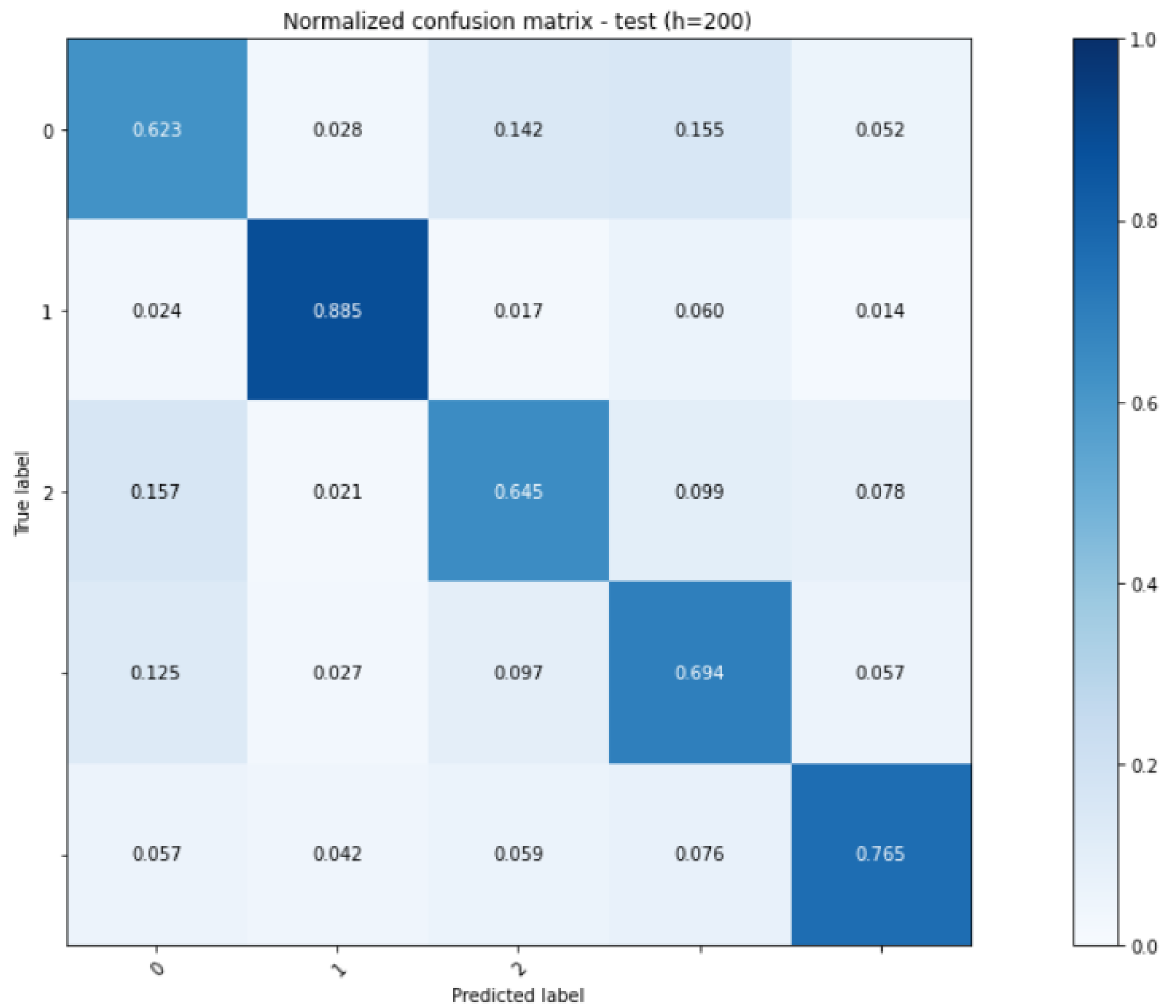
h=200

Overall accuracy of TRAIN = 88.3%

TRAIN confusion matrix:



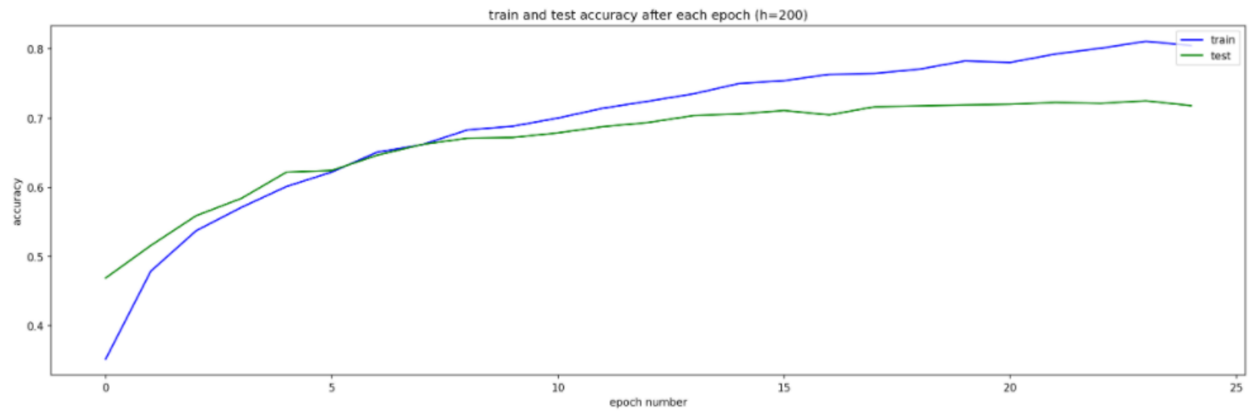
Overall accuracy of TEST = 72.5%
TEST confusion matrix:



Once again, the overall accuracies for the h=200 model improved, although not by great margins.

The TRAIN accuracy increased by 3.0% and the TEST accuracy increased by only 0.3%.

However, the individual prediction accuracies for each class changed quite a bit. Class 1 is still the most accurate, but class 3 accuracy decreased by almost 10%, getting closer to the h=90 model. Regardless, every other class accuracy increased by a couple of percentage points, making the model classification even more balanced.



From the above plot we can see the h=200 model's performance on the train and test set over time. The train and test set overlap around epochs 5 through 7 and start to diverge significantly without much increase in test accuracy at around epoch 17.

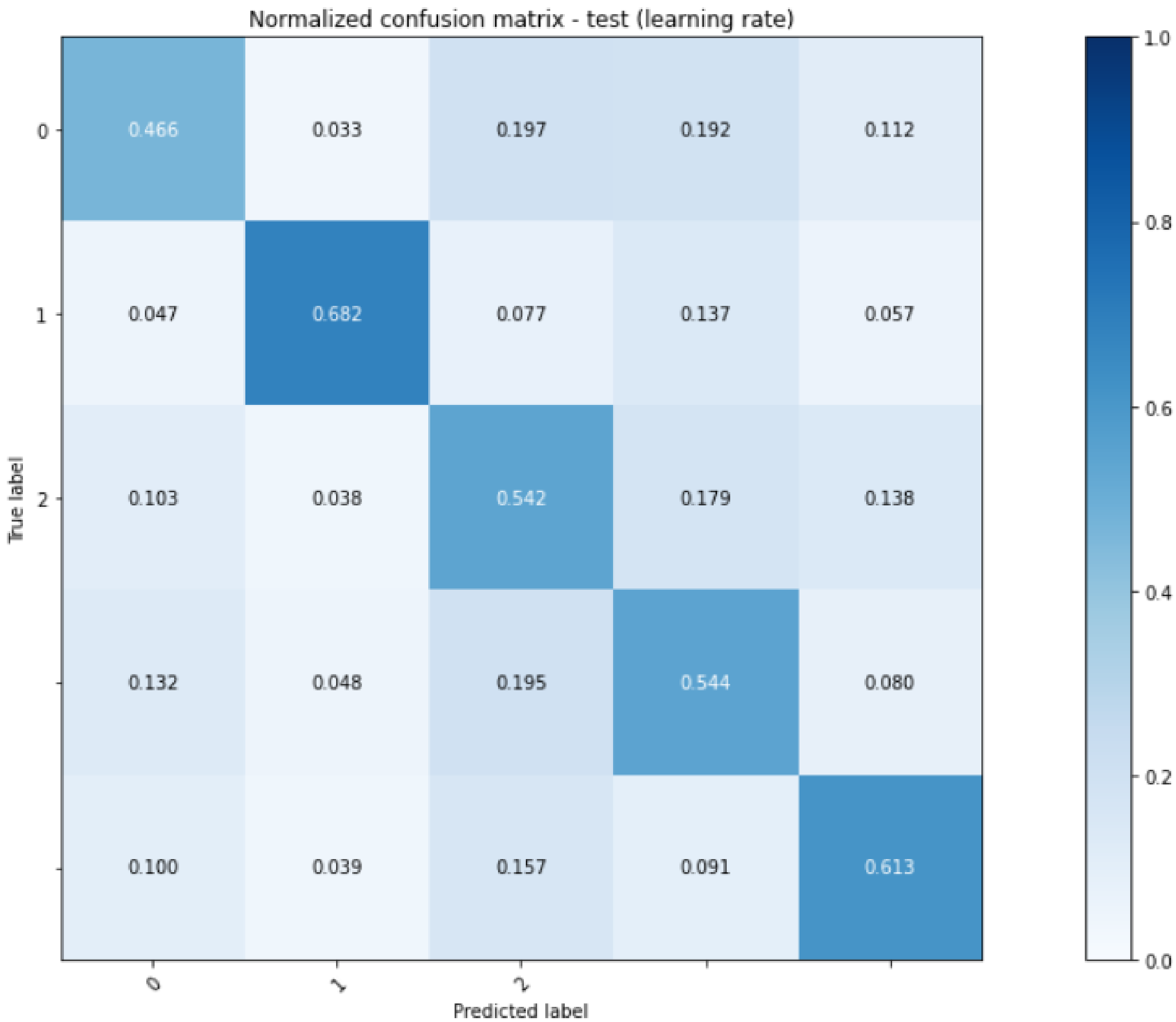
Conclusion/Additional Changes

After comparing the performances of all three model structures, we can conclude that the h=200 model not only had the highest prediction accuracies, but also had the most balanced results. Since we know exactly which fonts underperformed, we can take a closer look into the hidden layer neurons and analyze their activity levels as well as their weights. It's evident that the size of the hidden layer has a significant effect on the performance of the model and it would be interesting to experiment with more h-values.

We used our best model (h=200) to also experiment with the learning rate parameter as well as the batch size. Originally, we used a learning rate of 0.001 and a batch size of 110. However, we ran the model again, once with a learning rate of 0.01 and another time with a smaller batch size of 50 and compared the results.

Learning rate = 0.01

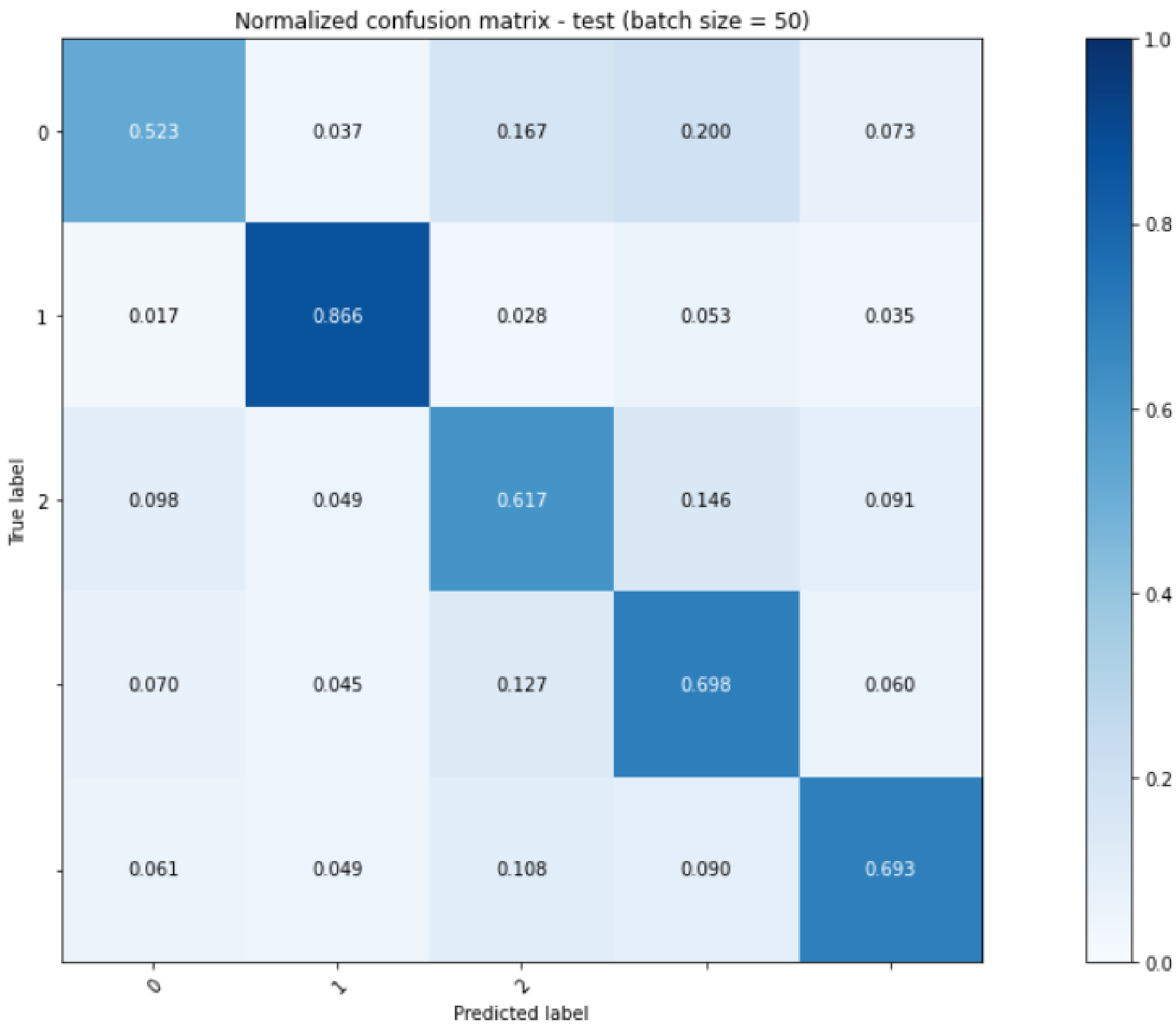
TRAIN set accuracy = 64.7%
TEST set accuracy = 57.1%



We can see from the accuracies and the confusion matrix that when increasing the learning rate, the performance decreases significantly. The overall accuracies decrease as well as the individual class accuracies. Therefore, we will keep the original learning rate of 0.001.

Batch size of 50

TRAIN set accuracy = 79.2%
TEST set accuracy = 68.2%



When we decrease the batch size, the performance of the model still decreases but not as significantly as when we changed the learning rate. The TEST accuracy decreased by 4.3% and the pattern of the individual class accuracies remained about the same, with the highest accuracies belonging to class 1,3, and 4.

In conclusion, we assume that a lower batch size and a higher learning rate do not improve our best model.

CODE

HW3_6373
April 14, 2021
Access to individual fonts - https://www.kaggle.com/aniruddhamandal/uci-font-dataset
Potential fonts -

- Arial: 26.2k cases
- Bodoni: 3964 cases
- Calibri: 19.1k cases
- Franklin: 15.7k cases
- Gadugi: 4164 cases

[ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation, Flatten, Conv2D, ␣
,→MaxPooling2D, Dropout
```
[ ]:
```
from google.colab import drive
drive.mount('/content/gdrive')
```
Mounted at /content/gdrive

[ ]:
```
arial = pd.read_csv('/content/gdrive/MyDrive/Colab Notebooks/MATH 6373 -␣
,→Azencott/HW3/Fonts/ARIAL.csv')
bodoni = pd.read_csv('/content/gdrive/MyDrive/Colab Notebooks/MATH 6373 -␣
,→Azencott/HW3/Fonts/BODONI.csv')
calibri = pd.read_csv('/content/gdrive/MyDrive/Colab Notebooks/MATH 6373 -␣
,→Azencott/HW3/Fonts/CALIBRI.csv')
franklin = pd.read_csv('/content/gdrive/MyDrive/Colab Notebooks/MATH 6373 -␣
,→Azencott/HW3/Fonts/FRANKLIN.csv')
gadugi = pd.read_csv('/content/gdrive/MyDrive/Colab Notebooks/MATH 6373 -␣
,→Azencott/HW3/Fonts/GADUGI.csv')
```
1

[ ]:
```
# For Rafa's Drive
arial = pd.read_csv('/content/gdrive/MyDrive/ARIAL.csv')
bodoni = pd.read_csv('/content/gdrive/MyDrive/BODONI.csv')
calibri = pd.read_csv('/content/gdrive/MyDrive/CALIBRI.csv')
franklin = pd.read_csv('/content/gdrive/MyDrive/FRANKLIN.csv')
gadugi = pd.read_csv('/content/gdrive/MyDrive/GADUGI.csv')
```

```
[ ]: from google.colab import drive
drive.mount('/content/drive')
Mounted at /content/drive
[ ]: a = arial.iloc[:,12:].to_numpy()
b = bodoni.iloc[:,12:].to_numpy()
c = calibri.iloc[:,12:].to_numpy()
f = franklin.iloc[:,12:].to_numpy()
g = gadugi.iloc[:,12:].to_numpy()
[ ]: # Arial "S"
plt.imshow(a[2,].reshape((20,20)))
[ ]: <matplotlib.image.AxesImage at 0x7fe5f5043410>
[ ]: # Bodoni "S"
plt.imshow(b[173,].reshape((20,20)))
```

2

```
[ ]: <matplotlib.image.AxesImage at 0x7fe5f4b13ed0>
[ ]: # Calibri "S"
plt.imshow(c[19017,].reshape((20,20)))
[ ]: <matplotlib.image.AxesImage at 0x7fe5f4b070d0>
```

3

```
[ ]: # Franklin "S"
plt.imshow(f[572,].reshape((20,20)))
[ ]: <matplotlib.image.AxesImage at 0x7fe5f4a66cd0>
[ ]: # Gadugi "S"
plt.imshow(g[1999,].reshape((20,20)))
[ ]: <matplotlib.image.AxesImage at 0x7fe5f49ce8d0>
```

4

1 Data Prep

```
[ ]: # Sample 3000 observations from each font
arial_sample = arial.sample(3000, random_state=42)
bodoni_sample = bodoni.sample(3000, random_state=42)
calibri_sample = calibri.sample(3000, random_state=42)
franklin_sample = franklin.sample(3000, random_state=42)
gadugi_sample = gadugi.sample(3000, random_state=42)
# Combine the samples into one 15000 observation set
data = pd.concat([arial_sample, bodoni_sample, calibri_sample, franklin_sample,␣
,→gadugi_sample])
data.reset_index(inplace=True, drop=True) # Reset the index to avoid duplicate␣
,→index values
# Obtain just the 400 feature values
data_r = data.iloc[:,12:].copy()
```

```
cols = data_r.columns
idx = data_r.index
# Standardize the feature values
data_r_scaled = preprocessing.scale(data_r)
data_r_scaled_df = pd.DataFrame(data_r_scaled, columns=cols, index = idx)
data_r_scaled_reshaped = data_r_scaled.reshape((15000,20,20)) # Reshape␣
,→standardized feature data from (15000, 400) to (15000, 20, 20)
```

5

```
#Obtain the 400 font labels
data_y = data.iloc[:,0].copy()
data_y_num = data_y.copy()
data_y_num = pd.DataFrame(data_y_num).iloc[:,0].astype('category').cat.codes #␣
,→Convert string font names to integers
X_train, X_test, y_train, y_test = train_test_split(data_r_scaled_reshaped,␣
,→data_y_num, train_size=0.80, random_state=42)
```

```
[ ]: data.head()
```

```
[ ]: font fontVariant m_label strength … r19c16 r19c17 r19c18 r19c19
0 ARIAL scanned 51 0.4 … 53 53 44 44
1 ARIAL scanned 52 0.4 … 201 216 173 152
2 ARIAL scanned 48 0.4 … 1 1 1 1
3 ARIAL ARIAL 8022 0.4 … 29 1 1 1
4 ARIAL ARIAL 65243 0.7 … 255 255 255 255
[5 rows x 412 columns]
```

```
[ ]: data_r.head()
```

```
[ ]: r0c0 r0c1 r0c2 r0c3 r0c4 … r19c15 r19c16 r19c17 r19c18 r19c19
0 30 30 44 44 184 … 180 53 53 44 44
1 1 1 1 1 1 … 193 201 216 173 152
2 1 1 1 40 255 … 99 1 1 1 1
3 1 1 4 113 172 … 88 29 1 1 1
4 1 1 1 1 1 … 255 255 255 255 255
[5 rows x 400 columns]
```

Arial = 0 Bodoni = 1 Calibri = 2 Franklin = 3 Gadugi = 4

```
[ ]: # Save y index values as converting to categorical removes them
y_train_idx = y_train.index
y_test_idx = y_test.index
# Transform y values for keras
y_train = to_categorical(y_train, 5)
y_test = to_categorical(y_test, 5)
```

2 Step 3

```
[ ]: X_train.shape[1:]
```

```
[ ]: (20, 20)
[ ]: X_train.shape
[ ]: (12000, 20, 20)
[ ]: X_train2 = X_train.reshape(12000, 20, 20, 1)
[ ]: X_test.shape
[ ]: (3000, 20, 20)
[ ]: X_test2 = X_test.reshape(3000, 20, 20, 1)
```

2.0.1 h=90

```
[ ]: model = Sequential()
model.add(Conv2D(16, (5, 5), padding='same', input_shape=(20,20,1)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(16, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(90,activation='relu'))
model.add(Dropout(0.5))
num_classes = 5
model.add(Dense(num_classes,activation='softmax'))
[ ]: model.summary()
Model: "sequential"
_________________________________________________________________
Layer (type) Output Shape Param #
=================================================================
conv2d (Conv2D) (None, 20, 20, 16) 416
_________________________________________________________________
activation (Activation) (None, 20, 20, 16) 0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 10, 10, 16) 0
_________________________________________________________________
conv2d_1 (Conv2D) (None, 10, 10, 16) 2320
_________________________________________________________________
```

7

```
activation_1 (Activation) (None, 10, 10, 16) 0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 5, 5, 16) 0
_________________________________________________________________
flatten (Flatten) (None, 400) 0
```

dense (Dense) (None, 90) 36090

---

dropout (Dropout) (None, 90) 0

---

dense_1 (Dense) (None, 5) 455
===
Total params: 39,281
Trainable params: 39,281
Non-trainable params: 0

[ ]: from keras.utils.vis_utils import plot_model
plot_model(model, show_shapes=True, show_layer_names=True)
[ ]:
8
9
[ ]: from tensorflow.keras.optimizers import Adam
opt = Adam(lr=0.001, decay=1e-7)
model.compile(loss='categorical_crossentropy',
optimizer=opt,
metrics=['accuracy'])
[ ]: from tensorflow.keras.callbacks import ModelCheckpoint
checkpointer = ModelCheckpoint(filepath='/content/weights_90.hdf5', ␣
,→monitor='val_accuracy', save_best_only=True)
[ ]: Monitor = model.fit(X_train2, y_train,
batch_size=110,
epochs=25,
validation_data=(X_test2, y_test),
callbacks = [checkpointer],
shuffle = True)
Epoch 1/25
110/110 [==============================] - 6s 49ms/step - loss: 1.5723 -
accuracy: 0.2823 - val_loss: 1.3921 - val_accuracy: 0.4513
Epoch 2/25
110/110 [==============================] - 5s 44ms/step - loss: 1.3844 -
accuracy: 0.4332 - val_loss: 1.2662 - val_accuracy: 0.5067
Epoch 3/25
110/110 [==============================] - 5s 46ms/step - loss: 1.2724 -
accuracy: 0.4928 - val_loss: 1.1544 - val_accuracy: 0.5477
Epoch 4/25

```
110/110 [==============================] - 5s 44ms/step - loss: 1.1736 -
accuracy: 0.5367 - val_loss: 1.0920 - val_accuracy: 0.5683
Epoch 5/25
110/110 [==============================] - 5s 46ms/step - loss: 1.1190 -
accuracy: 0.5554 - val_loss: 1.0430 - val_accuracy: 0.5843
Epoch 6/25
110/110 [==============================] - 5s 45ms/step - loss: 1.0651 -
accuracy: 0.5743 - val_loss: 1.0170 - val_accuracy: 0.6010
Epoch 7/25
110/110 [==============================] - 5s 46ms/step - loss: 1.0182 -
accuracy: 0.5909 - val_loss: 0.9900 - val_accuracy: 0.6070
Epoch 8/25
110/110 [==============================] - 5s 43ms/step - loss: 0.9748 -
accuracy: 0.6198 - val_loss: 0.9666 - val_accuracy: 0.6133
Epoch 9/25
110/110 [==============================] - 5s 45ms/step - loss: 0.9477 -
10
accuracy: 0.6233 - val_loss: 0.9714 - val_accuracy: 0.6147
Epoch 10/25
110/110 [==============================] - 5s 43ms/step - loss: 0.9199 -
accuracy: 0.6360 - val_loss: 0.9166 - val_accuracy: 0.6423
Epoch 11/25
110/110 [==============================] - 5s 43ms/step - loss: 0.9042 -
accuracy: 0.6415 - val_loss: 0.9159 - val_accuracy: 0.6357
Epoch 12/25
110/110 [==============================] - 5s 44ms/step - loss: 0.8690 -
accuracy: 0.6530 - val_loss: 0.8969 - val_accuracy: 0.6440
Epoch 13/25
110/110 [==============================] - 5s 44ms/step - loss: 0.8601 -
accuracy: 0.6547 - val_loss: 0.8812 - val_accuracy: 0.6497
Epoch 14/25
110/110 [==============================] - 5s 43ms/step - loss: 0.8293 -
accuracy: 0.6694 - val_loss: 0.8637 - val_accuracy: 0.6633
Epoch 15/25
110/110 [==============================] - 5s 44ms/step - loss: 0.8115 -
accuracy: 0.6831 - val_loss: 0.8549 - val_accuracy: 0.6627
Epoch 16/25
110/110 [==============================] - 5s 43ms/step - loss: 0.7869 -
accuracy: 0.6905 - val_loss: 0.8405 - val_accuracy: 0.6723
Epoch 17/25
```

```
110/110 [==============================] - 5s 44ms/step - loss: 0.7699 -
accuracy: 0.6933 - val_loss: 0.8429 - val_accuracy: 0.6637
Epoch 18/25
110/110 [==============================] - 5s 45ms/step - loss: 0.7539 -
accuracy: 0.6982 - val_loss: 0.8321 - val_accuracy: 0.6760
Epoch 19/25
110/110 [==============================] - 5s 45ms/step - loss: 0.7464 -
accuracy: 0.7035 - val_loss: 0.8286 - val_accuracy: 0.6783
Epoch 20/25
110/110 [==============================] - 5s 44ms/step - loss: 0.7282 -
accuracy: 0.7071 - val_loss: 0.8297 - val_accuracy: 0.6830
Epoch 21/25
110/110 [==============================] - 5s 44ms/step - loss: 0.7118 -
accuracy: 0.7210 - val_loss: 0.8131 - val_accuracy: 0.6897
Epoch 22/25
110/110 [==============================] - 5s 43ms/step - loss: 0.6994 -
accuracy: 0.7194 - val_loss: 0.8199 - val_accuracy: 0.6843
Epoch 23/25
110/110 [==============================] - 5s 44ms/step - loss: 0.6993 -
accuracy: 0.7163 - val_loss: 0.8150 - val_accuracy: 0.6897
Epoch 24/25
110/110 [==============================] - 5s 45ms/step - loss: 0.6611 -
accuracy: 0.7365 - val_loss: 0.8220 - val_accuracy: 0.6817
Epoch 25/25
110/110 [==============================] - 5s 45ms/step - loss: 0.6723 -
11
accuracy: 0.7327 - val_loss: 0.8093 - val_accuracy: 0.6880
[ ]: import tensorflow as tf
bestModel_90 = tf.keras.models.load_model('weights_90.hdf5')
[ ]: y_pred_train_90 = bestModel_90.predict(X_train2)
y_pred_test_90 = bestModel_90.predict(X_test2)
predlabel_train_90 = np.argmax(y_pred_train_90,axis=1)
predlabel_test_90 = np.argmax(y_pred_test_90,axis=1)
[ ]: val_accuracy_90 = Monitor.history['val_accuracy']
train_accuracy_90 = Monitor.history['accuracy']
```

2.0.2 h = 150

```
[ ]: model = Sequential()
model.add(Conv2D(16, (5, 5), padding='same', input_shape=(20,20,1)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
```

```
model.add(Conv2D(16, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(150,activation='relu'))
model.add(Dropout(0.5))
num_classes = 5
model.add(Dense(num_classes,activation='softmax'))
[ ]: model.summary()
Model: "sequential_1"
_________________________________________________________________
Layer (type) Output Shape Param #
=================================================================
conv2d_2 (Conv2D) (None, 20, 20, 16) 416
_________________________________________________________________
activation_2 (Activation) (None, 20, 20, 16) 0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 10, 10, 16) 0
_________________________________________________________________
conv2d_3 (Conv2D) (None, 10, 10, 16) 2320
_________________________________________________________________
12
activation_3 (Activation) (None, 10, 10, 16) 0
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 5, 5, 16) 0
_________________________________________________________________
flatten_1 (Flatten) (None, 400) 0
_________________________________________________________________
dense_2 (Dense) (None, 150) 60150
_________________________________________________________________
dropout_1 (Dropout) (None, 150) 0
_________________________________________________________________
dense_3 (Dense) (None, 5) 755
=================================================================
Total params: 63,641
Trainable params: 63,641
Non-trainable params: 0
_________________________________________________________________
[ ]: plot_model(model, show_shapes=True, show_layer_names=True)
[ ]:
```

```
[ ]: opt = Adam(lr=0.001, decay=1e-7)
model.compile(loss='categorical_crossentropy',
optimizer=opt,
metrics=['accuracy'])
[ ]: checkpointer = ModelCheckpoint(filepath='/content/weights_150.hdf5', ␣
,→monitor='val_accuracy', save_best_only=True)
[ ]: Monitor = model.fit(X_train2, y_train,
batch_size=110,
epochs=25,
validation_data=(X_test2, y_test),
callbacks = [checkpointer],
shuffle = True)
Epoch 1/25
110/110 [==============================] - 5s 43ms/step - loss: 1.5732 -
accuracy: 0.2748 - val_loss: 1.3638 - val_accuracy: 0.4730
Epoch 2/25
110/110 [==============================] - 4s 41ms/step - loss: 1.3403 -
accuracy: 0.4544 - val_loss: 1.2288 - val_accuracy: 0.5117
Epoch 3/25
110/110 [==============================] - 5s 41ms/step - loss: 1.2108 -
accuracy: 0.5168 - val_loss: 1.1273 - val_accuracy: 0.5570
Epoch 4/25
110/110 [==============================] - 4s 41ms/step - loss: 1.1160 -
accuracy: 0.5593 - val_loss: 1.0581 - val_accuracy: 0.5870
Epoch 5/25
110/110 [==============================] - 4s 41ms/step - loss: 1.0528 -
accuracy: 0.5823 - val_loss: 0.9922 - val_accuracy: 0.6020
Epoch 6/25
110/110 [==============================] - 5s 41ms/step - loss: 0.9968 -
accuracy: 0.6048 - val_loss: 0.9660 - val_accuracy: 0.6033
Epoch 7/25
110/110 [==============================] - 4s 41ms/step - loss: 0.9380 -
accuracy: 0.6345 - val_loss: 0.9296 - val_accuracy: 0.6230
Epoch 8/25
110/110 [==============================] - 5s 41ms/step - loss: 0.9018 -
accuracy: 0.6489 - val_loss: 0.8872 - val_accuracy: 0.6340
Epoch 9/25
110/110 [==============================] - 4s 41ms/step - loss: 0.8528 -
```

```
accuracy: 0.6677 - val_loss: 0.8705 - val_accuracy: 0.6550
Epoch 10/25
110/110 [==============================] - 5s 41ms/step - loss: 0.8421 -
15
accuracy: 0.6740 - val_loss: 0.8549 - val_accuracy: 0.6560
Epoch 11/25
110/110 [==============================] - 4s 41ms/step - loss: 0.8118 -
accuracy: 0.6822 - val_loss: 0.8462 - val_accuracy: 0.6653
Epoch 12/25
110/110 [==============================] - 4s 41ms/step - loss: 0.7849 -
accuracy: 0.6893 - val_loss: 0.8291 - val_accuracy: 0.6727
Epoch 13/25
110/110 [==============================] - 4s 41ms/step - loss: 0.7577 -
accuracy: 0.7063 - val_loss: 0.8170 - val_accuracy: 0.6820
Epoch 14/25
110/110 [==============================] - 4s 41ms/step - loss: 0.7202 -
accuracy: 0.7166 - val_loss: 0.8014 - val_accuracy: 0.6857
Epoch 15/25
110/110 [==============================] - 5s 41ms/step - loss: 0.7183 -
accuracy: 0.7194 - val_loss: 0.7865 - val_accuracy: 0.6890
Epoch 16/25
110/110 [==============================] - 4s 41ms/step - loss: 0.6866 -
accuracy: 0.7316 - val_loss: 0.7831 - val_accuracy: 0.6893
Epoch 17/25
110/110 [==============================] - 4s 41ms/step - loss: 0.6597 -
accuracy: 0.7470 - val_loss: 0.7851 - val_accuracy: 0.6950
Epoch 18/25
110/110 [==============================] - 5s 41ms/step - loss: 0.6624 -
accuracy: 0.7478 - val_loss: 0.7782 - val_accuracy: 0.6940
Epoch 19/25
110/110 [==============================] - 4s 41ms/step - loss: 0.6269 -
accuracy: 0.7503 - val_loss: 0.7762 - val_accuracy: 0.6993
Epoch 20/25
110/110 [==============================] - 5s 41ms/step - loss: 0.6162 -
accuracy: 0.7639 - val_loss: 0.7696 - val_accuracy: 0.7023
Epoch 21/25
110/110 [==============================] - 4s 41ms/step - loss: 0.5905 -
accuracy: 0.7710 - val_loss: 0.7686 - val_accuracy: 0.7080
Epoch 22/25
110/110 [==============================] - 5s 41ms/step - loss: 0.6011 -
```

accuracy: 0.7705 - val_loss: 0.7757 - val_accuracy: 0.7083
Epoch 23/25
110/110 [==============================] - 5s 41ms/step - loss: 0.5964 -
accuracy: 0.7700 - val_loss: 0.7651 - val_accuracy: 0.7157
Epoch 24/25
110/110 [==============================] - 5s 41ms/step - loss: 0.5726 -
accuracy: 0.7734 - val_loss: 0.7771 - val_accuracy: 0.7083
Epoch 25/25
110/110 [==============================] - 4s 41ms/step - loss: 0.5495 -
accuracy: 0.7892 - val_loss: 0.7617 - val_accuracy: 0.7127
16

```
[ ]: bestModel_150 = tf.keras.models.load_model('weights_150.hdf5')
[ ]: y_pred_train_150 = bestModel_150.predict(X_train2)
y_pred_test_150 = bestModel_150.predict(X_test2)
predlabel_train_150 = np.argmax(y_pred_train_150,axis=1)
predlabel_test_150 = np.argmax(y_pred_test_150,axis=1)
[ ]: val_accuracy_150 = Monitor.history['val_accuracy']
train_accuracy_150 = Monitor.history['accuracy']
```

2.0.3 h = 200

```
[ ]: model = Sequential()
model.add(Conv2D(16, (5, 5), padding='same', input_shape=(20,20,1)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(16, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(200,activation='relu'))
model.add(Dropout(0.5))
num_classes = 5
model.add(Dense(num_classes,activation='softmax'))
[ ]: model.summary()
```

Model: "sequential_2"

Layer (type) Output Shape Param #
===
conv2d_4 (Conv2D) (None, 20, 20, 16) 416

---

activation_4 (Activation) (None, 20, 20, 16) 0

---

max_pooling2d_4 (MaxPooling2 (None, 10, 10, 16) 0

---

conv2d_5 (Conv2D) (None, 10, 10, 16) 2320

---

activation_5 (Activation) (None, 10, 10, 16) 0

---

max_pooling2d_5 (MaxPooling2 (None, 5, 5, 16) 0
17

---

flatten_2 (Flatten) (None, 400) 0

---

dense_4 (Dense) (None, 200) 80200

---

dropout_2 (Dropout) (None, 200) 0

---

dense_5 (Dense) (None, 5) 1005
===
Total params: 83,941
Trainable params: 83,941
Non-trainable params: 0

```
[ ]: plot_model(model, show_shapes=True, show_layer_names=True)
[ ]:
```
18
19
```
[ ]: opt = Adam(lr=0.001, decay=1e-7)
model.compile(loss='categorical_crossentropy',
optimizer=opt,
metrics=['accuracy'])
[ ]: checkpointer = ModelCheckpoint(filepath='/content/weights_200.hdf5', ␣
,→monitor='val_accuracy', save_best_only=True)
[ ]: Monitor = model.fit(X_train2, y_train,
batch_size=110,
epochs=25,
validation_data=(X_test2, y_test),
callbacks = [checkpointer],
shuffle = True)
Epoch 1/25
110/110 [==============================] - 5s 44ms/step - loss: 1.5326 -
accuracy: 0.3167 - val_loss: 1.3390 - val_accuracy: 0.4830
```

Epoch 2/25
110/110 [==============================] - 5s 41ms/step - loss: 1.3191 -
accuracy: 0.4603 - val_loss: 1.2029 - val_accuracy: 0.5400
Epoch 3/25
110/110 [==============================] - 5s 41ms/step - loss: 1.1931 -
accuracy: 0.5298 - val_loss: 1.1085 - val_accuracy: 0.5593
Epoch 4/25
110/110 [==============================] - 5s 41ms/step - loss: 1.1150 -
accuracy: 0.5607 - val_loss: 1.0426 - val_accuracy: 0.5920
Epoch 5/25
110/110 [==============================] - 5s 41ms/step - loss: 1.0340 -
accuracy: 0.6019 - val_loss: 0.9814 - val_accuracy: 0.6103
Epoch 6/25
110/110 [==============================] - 5s 41ms/step - loss: 0.9804 -
accuracy: 0.6150 - val_loss: 0.9446 - val_accuracy: 0.6337
Epoch 7/25
110/110 [==============================] - 5s 41ms/step - loss: 0.9331 -
accuracy: 0.6338 - val_loss: 0.9118 - val_accuracy: 0.6400
Epoch 8/25
110/110 [==============================] - 5s 41ms/step - loss: 0.8718 -
accuracy: 0.6582 - val_loss: 0.8931 - val_accuracy: 0.6483
Epoch 9/25
110/110 [==============================] - 5s 41ms/step - loss: 0.8491 -
accuracy: 0.6700 - val_loss: 0.8659 - val_accuracy: 0.6600
Epoch 10/25
110/110 [==============================] - 5s 41ms/step - loss: 0.8303 -

accuracy: 0.6795 - val_loss: 0.8563 - val_accuracy: 0.6540
Epoch 11/25
110/110 [==============================] - 5s 41ms/step - loss: 0.7774 -
accuracy: 0.6973 - val_loss: 0.8253 - val_accuracy: 0.6727
Epoch 12/25
110/110 [==============================] - 5s 41ms/step - loss: 0.7482 -
accuracy: 0.7104 - val_loss: 0.7989 - val_accuracy: 0.6910
Epoch 13/25
110/110 [==============================] - 5s 41ms/step - loss: 0.7116 -
accuracy: 0.7201 - val_loss: 0.7942 - val_accuracy: 0.6900
Epoch 14/25
110/110 [==============================] - 5s 42ms/step - loss: 0.6863 -
accuracy: 0.7364 - val_loss: 0.7849 - val_accuracy: 0.6890

Epoch 15/25
110/110 [==============================] - 5s 41ms/step - loss: 0.6541 - accuracy: 0.7513 - val_loss: 0.7726 - val_accuracy: 0.6973
Epoch 16/25
110/110 [==============================] - 5s 42ms/step - loss: 0.6707 - accuracy: 0.7448 - val_loss: 0.7667 - val_accuracy: 0.7013
Epoch 17/25
110/110 [==============================] - 5s 41ms/step - loss: 0.6696 - accuracy: 0.7344 - val_loss: 0.7733 - val_accuracy: 0.6960
Epoch 18/25
110/110 [==============================] - 5s 41ms/step - loss: 0.6075 - accuracy: 0.7643 - val_loss: 0.7531 - val_accuracy: 0.7063
Epoch 19/25
110/110 [==============================] - 5s 42ms/step - loss: 0.5869 - accuracy: 0.7744 - val_loss: 0.7489 - val_accuracy: 0.7127
Epoch 20/25
110/110 [==============================] - 5s 42ms/step - loss: 0.5640 - accuracy: 0.7822 - val_loss: 0.7681 - val_accuracy: 0.7153

[ ]: bestModel_200 = tf.keras.models.load_model('weights_200.hdf5')

[ ]: y_pred_train_200 = bestModel_200.predict(X_train2)
y_pred_test_200 = bestModel_200.predict(X_test2)
predlabel_train_200 = np.argmax(y_pred_train_200,axis=1)
predlabel_test_200 = np.argmax(y_pred_test_200,axis=1)

[ ]: val_accuracy_200 = Monitor.history['val_accuracy']
train_accuracy_200 = Monitor.history['accuracy']

2.0.4 Total number of weights and biases for each model:

h = 90 model:
39,281 weights and biases

21

h = 150 model:
63,641 weights and biases

h = 200 model:
83,941 weights and biases

[ ]: X_train2.shape

[ ]: 12000 * 5

There are 60,000 infos brought by the whole training set
5 per case, 12000 cases

3 Step 5

[ ]: # Function to make confusion matrix
import numpy as np

```
import matplotlib.pyplot as plt
from sklearn import svm, datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.utils.multiclass import unique_labels
def plot_confusion_matrix(y_true, y_pred, classes,
normalize=False,
title=None,
cmap=plt.cm.Blues):
"""
This function prints and plots the confusion matrix.
Normalization can be applied by setting `normalize=True`.
"""
if not title:
if normalize:
title = 'Normalized confusion matrix'
else:
title = 'Confusion matrix, without normalization'
# Compute confusion matrix
cm = confusion_matrix(y_true, y_pred)
# Only use the labels that appear in the data
#################################### HAD TO HARD CODE THE LABELS AS
[0, 1, 2]
classes = np.array([0,1,2])
if normalize:
cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
print("Normalized confusion matrix")
22
else:
print('Confusion matrix, without normalization')
print(cm)
fig, ax = plt.subplots(figsize=(16,8))
im = ax.imshow(cm, interpolation='nearest', cmap=cmap,vmin=0,vmax=1)
ax.figure.colorbar(im, ax=ax)
# We want to show all ticks...
ax.set(xticks=np.arange(cm.shape[1]),
yticks=np.arange(cm.shape[0]),
# ... and label them with the respective list entries
xticklabels=classes, yticklabels=classes,
title=title,
```

```
ylabel='True label',
xlabel='Predicted label')
# Rotate the tick labels and set their alignment.
plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
rotation_mode="anchor")
# Loop over data dimensions and create text annotations.
fmt = '.3f' if normalize else 'd'
thresh = cm.max() / 2.
for i in range(cm.shape[0]):
for j in range(cm.shape[1]):
ax.text(j, i, format(cm[i, j], fmt),
ha="center", va="center",
color="white" if cm[i, j] > thresh else "black")
fig.tight_layout()
return ax
```

3.0.1 Analysis of h = 90

```
[ ]: y_train = y_train.astype(int)
y_train_labels = list(range(12000))
[ ]: for i in range(12000):
if y_train[i][0] == 1:
y_train_labels[i] = 0
elif y_train[i][1] == 1:
y_train_labels[i] = 1
elif y_train[i][2] == 1:
y_train_labels[i] = 2
elif y_train[i][3] == 1:
y_train_labels[i] = 3
elif y_train[i][4] == 1:
```



```
y_train_labels[i] = 4
[ ]: y_train_labels
[ ]: from sklearn import metrics
results = metrics.accuracy_score(y_train_labels,predlabel_train_90)
print('The overall accuracy for the h = 90 Train set is:')
print('{0:.3f}'.format(results))
[ ]: classes = np.array([0,1,2,3,4])
plot_confusion_matrix(y_train_labels, predlabel_train_90, classes=classes,␣
,→normalize=True,
title='Normalized confusion matrix - train (h=90)')
plt.show()
```

```
[ ]: y_test = y_test.astype(int)
y_test_labels = list(range(3000))
[ ]: for i in range(3000):
if y_test[i][0] == 1:
y_test_labels[i] = 0
elif y_test[i][1] == 1:
y_test_labels[i] = 1
elif y_test[i][2] == 1:
y_test_labels[i] = 2
elif y_test[i][3] == 1:
y_test_labels[i] = 3
elif y_test[i][4] == 1:
y_test_labels[i] = 4
[ ]: results = metrics.accuracy_score(y_test_labels,predlabel_test_90)
print('The overall accuracy for the h = 90 Test set is:')
print('{0:.3f}'.format(results))
[ ]: classes = np.array([0,1,2,3,4])
plot_confusion_matrix(y_test_labels, predlabel_test_90, classes=classes, ␣
,→normalize=True,
title='Normalized confusion matrix - test (h=90)')
plt.show()
Normalized confusion matrix
[[0.53923205 0.03839733 0.14858097 0.1836394 0.09015025]
[0.0172956 0.88207547 0.01415094 0.06761006 0.01886792]
[0.11672474 0.04355401 0.61324042 0.13066202 0.09581882]
```



```
[0.09682805 0.04507513 0.10350584 0.67946578 0.07512521]
[0.0625 0.03716216 0.06756757 0.08952703 0.74324324]]
```

Accuracy over time

```
[ ]: x = range(25)
plt.rcParams["figure.figsize"] = (20,6)
plt.figure(dpi=150)
plt.plot(x,train_accuracy_90,'b',label='trainMSE')
plt.plot(x,val_accuracy_90,'g',label='testMSE')
plt.title('trainMSE and testMSE after each epoch (h=90)')
plt.xlabel('epoch number')
plt.ylabel('MSE')
plt.legend(loc='upper right')
[ ]: <matplotlib.legend.Legend at 0x7f76008c9e10>
```

3.0.2 Analysis of h = 150

```
[ ]: results = metrics.accuracy_score(y_train_labels,predlabel_train_150)
print('The overall accuracy for the h = 150 Train set is:')
print('{0:.3f}'.format(results))
```

```
The overall accuracy for the h = 150 Train set is:
0.864
```

```
[ ]: classes = np.array([0,1,2,3,4])
plot_confusion_matrix(y_train_labels, predlabel_train_150, classes=classes, ␣
,→normalize=True,
title='Normalized confusion matrix - train (h=150)')
plt.show()
```

```
Normalized confusion matrix
[[0.74302374 0.02165764 0.10662224 0.091212 0.03748438]
[0.00296108 0.94754653 0.00592217 0.03976311 0.00380711]
[0.03380049 0.01690025 0.88499588 0.0362737 0.02802968]
[0.04414827 0.03248646 0.05914202 0.83840067 0.02582257]
[0.01827243 0.01785714 0.02906977 0.02782392 0.90697674]]
```



```
[ ]: results = metrics.accuracy_score(y_test_labels,predlabel_test_150)
print('The overall accuracy for the h = 150 Test set is:')
print('{0:.3f}'.format(results))
```

```
The overall accuracy for the h = 150 Test set is:
0.716
```

```
[ ]: classes = np.array([0,1,2,3,4])
plot_confusion_matrix(y_test_labels, predlabel_test_150, classes=classes, ␣
,→normalize=True,
title='Normalized confusion matrix - test (h=150)')
plt.show()
```

```
Normalized confusion matrix
[[0.56594324 0.03171953 0.1769616 0.13856427 0.08681135]
[0.01572327 0.89622642 0.02672956 0.04874214 0.01257862]
```



```
[0.10278746 0.03310105 0.70731707 0.07142857 0.08536585]
[0.10350584 0.05008347 0.12520868 0.6427379 0.07846411]
[0.06081081 0.04560811 0.08783784 0.05067568 0.75506757]]
```

```
[ ]: x = range(25)
plt.rcParams["figure.figsize"] = (20,6)
plt.figure(dpi=150)
plt.plot(x,train_accuracy_150,'b',label='trainMSE')
plt.plot(x,val_accuracy_150,'g',label='testMSE')
```

plt.title('trainMSE and testMSE after each epoch (h=150)')
plt.xlabel('epoch number')
plt.ylabel('MSE')
plt.legend(loc='upper right')
[ ]: <matplotlib.legend.Legend at 0x7f760067ed50>
28

3.0.3 Analysis of h = 200

[ ]: results = metrics.accuracy_score(y_train_labels,predlabel_train_200)
print('The overall accuracy for the h = 200 Train set is:')
print('{0:.3f}'.format(results))
The overall accuracy for the h = 200 Train set is:
0.886
[ ]: classes = np.array([0,1,2,3,4])
plot_confusion_matrix(y_train_labels, predlabel_train_200, classes=classes, ␣
,→normalize=True,
title='Normalized confusion matrix - train (h=200)')
plt.show()
Normalized confusion matrix
[[0.77842566 0.01124531 0.091212 0.08246564 0.0366514]
[0.00296108 0.93443316 0.00634518 0.04483926 0.01142132]
[0.02184666 0.00494641 0.92291838 0.03132729 0.01896125]
[0.03456893 0.01749271 0.05372761 0.87213661 0.02207414]
[0.01328904 0.00539867 0.02948505 0.02699336 0.92483389]]
29
[ ]: results = metrics.accuracy_score(y_test_labels,predlabel_test_200)
print('The overall accuracy for the h = 200 Test set is:')
print('{0:.3f}'.format(results))
The overall accuracy for the h = 200 Test set is:
0.725
[ ]: classes = np.array([0,1,2,3,4])
plot_confusion_matrix(y_test_labels, predlabel_test_200, classes=classes, ␣
,→normalize=True,
title='Normalized confusion matrix - test (h=200)')
plt.show()
Normalized confusion matrix
[[0.56093489 0.01836394 0.1869783 0.13188648 0.10183639]
[0.0172956 0.87735849 0.03144654 0.04716981 0.02672956]
30
[0.10627178 0.02264808 0.71602787 0.08536585 0.06968641]
[0.11185309 0.02337229 0.10851419 0.68280467 0.07345576]

[0.05236486 0.02702703 0.08614865 0.05405405 0.78040541]]

[ ]: x = range(25)

```
plt.rcParams["figure.figsize"] = (20,6)
plt.figure(dpi=150)
plt.plot(x,train_accuracy_200,'b',label='trainMSE')
plt.plot(x,val_accuracy_200,'g',label='testMSE')
plt.title('trainMSE and testMSE after each epoch (h=200)')
plt.xlabel('epoch number')
plt.ylabel('MSE')
plt.legend(loc='upper right')
```

[ ]: <matplotlib.legend.Legend at 0x7f7600414dd0>

4 Learning rate and batch size

4.0.1 learning rate = 0.01

[ ]: model = Sequential()

```
model.add(Conv2D(16, (5, 5), padding='valid', input_shape=X_train2.shape[1:
,→],activation = 'relu'))
model.add( MaxPooling2D(pool_size=(2, 2)) )
model.add(Conv2D(16, (3, 3), padding='valid',activation = 'relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
h = 200
model.add(Dense(h,activation='relu'))
model.add(Dropout(0.5))
num_classes = 5
model.add(Dense(num_classes,activation='softmax'))
```

[ ]: model.summary()

Model: "sequential_2"

Layer (type)	Output Shape	Param #
conv2d_4 (Conv2D)	(None, 16, 16, 16)	416
max_pooling2d_4 (MaxPooling2	(None, 8, 8, 16)	0
conv2d_5 (Conv2D)	(None, 6, 6, 16)	2320

Layer (type)	Output Shape	Param #
max_pooling2d_5 (MaxPooling2	(None, 3, 3, 16)	0

```
flatten_2 (Flatten) (None, 144) 0
_________________________________________________________________
dense_4 (Dense) (None, 200) 29000
_________________________________________________________________
dropout_2 (Dropout) (None, 200) 0
_________________________________________________________________
dense_5 (Dense) (None, 5) 1005
=================================================================
Total params: 32,741
Trainable params: 32,741
Non-trainable params: 0
_________________________________________________________________
```

```
[ ]: opt = Adam(lr=0.01, decay=1e-7)
model.compile(loss='categorical_crossentropy',
optimizer=opt,
metrics=['accuracy'])
[ ]: checkpointer = ModelCheckpoint(filepath='/content/weights_lr.hdf5', ␣
,→monitor='val_accuracy', save_best_only=True)
[ ]: trainSetSize = 12000
Monitor = model.fit(X_train2, y_train,
batch_size=int(np.sqrt(trainSetSize)),
epochs=25,
validation_data=(X_test2, y_test),
callbacks = [checkpointer],
shuffle = True)
Epoch 1/25
111/111 [==============================] - 4s 29ms/step - loss: 1.5431 -
accuracy: 0.3053 - val_loss: 1.4435 - val_accuracy: 0.4090
Epoch 2/25
111/111 [==============================] - 3s 25ms/step - loss: 1.4080 -
accuracy: 0.4073 - val_loss: 1.3537 - val_accuracy: 0.4547
Epoch 3/25
111/111 [==============================] - 3s 25ms/step - loss: 1.3511 -
accuracy: 0.4416 - val_loss: 1.2939 - val_accuracy: 0.4787
Epoch 4/25
111/111 [==============================] - 3s 26ms/step - loss: 1.2669 -
accuracy: 0.4871 - val_loss: 1.2460 - val_accuracy: 0.4983
Epoch 5/25
111/111 [==============================] - 3s 26ms/step - loss: 1.2832 -
```

accuracy: 0.4777 - val_loss: 1.1936 - val_accuracy: 0.5210
Epoch 6/25
111/111 [==============================] - 3s 26ms/step - loss: 1.2501 - accuracy: 0.4820 - val_loss: 1.2046 - val_accuracy: 0.5137
Epoch 7/25
111/111 [==============================] - 3s 25ms/step - loss: 1.2008 - accuracy: 0.5060 - val_loss: 1.1658 - val_accuracy: 0.5283
Epoch 8/25
111/111 [==============================] - 3s 25ms/step - loss: 1.1832 - accuracy: 0.5131 - val_loss: 1.1536 - val_accuracy: 0.5377
Epoch 9/25
111/111 [==============================] - 3s 25ms/step - loss: 1.1953 - accuracy: 0.5104 - val_loss: 1.1481 - val_accuracy: 0.5457
Epoch 10/25
111/111 [==============================] - 3s 25ms/step - loss: 1.1599 - accuracy: 0.5279 - val_loss: 1.2156 - val_accuracy: 0.4957
Epoch 11/25
111/111 [==============================] - 3s 26ms/step - loss: 1.1506 - accuracy: 0.5325 - val_loss: 1.1244 - val_accuracy: 0.5467
Epoch 12/25
111/111 [==============================] - 3s 27ms/step - loss: 1.1212 - accuracy: 0.5499 - val_loss: 1.1212 - val_accuracy: 0.5417
Epoch 13/25
111/111 [==============================] - 3s 25ms/step - loss: 1.1060 - accuracy: 0.5563 - val_loss: 1.1035 - val_accuracy: 0.5563
Epoch 14/25
111/111 [==============================] - 3s 25ms/step - loss: 1.0792 - accuracy: 0.5570 - val_loss: 1.0991 - val_accuracy: 0.5623
Epoch 15/25
111/111 [==============================] - 3s 25ms/step - loss: 1.0805 - accuracy: 0.5665 - val_loss: 1.0875 - val_accuracy: 0.5617
Epoch 16/25
111/111 [==============================] - 3s 26ms/step - loss: 1.1284 - accuracy: 0.5398 - val_loss: 1.1079 - val_accuracy: 0.5690
Epoch 17/25
111/111 [==============================] - 3s 26ms/step - loss: 1.0994 - accuracy: 0.5520 - val_loss: 1.1202 - val_accuracy: 0.5493
Epoch 18/25
111/111 [==============================] - 3s 26ms/step - loss: 1.0954 - accuracy: 0.5654 - val_loss: 1.1071 - val_accuracy: 0.5493

Epoch 19/25
111/111 [==============================] - 3s 25ms/step - loss: 1.0505 -
accuracy: 0.5736 - val_loss: 1.1362 - val_accuracy: 0.5447
Epoch 20/25
111/111 [==============================] - 3s 26ms/step - loss: 1.0646 -
accuracy: 0.5690 - val_loss: 1.1326 - val_accuracy: 0.5683
Epoch 21/25
111/111 [==============================] - 3s 26ms/step - loss: 1.0668 -

accuracy: 0.5608 - val_loss: 1.0985 - val_accuracy: 0.5550
Epoch 22/25
111/111 [==============================] - 3s 28ms/step - loss: 1.0456 -
accuracy: 0.5712 - val_loss: 1.0878 - val_accuracy: 0.5707
Epoch 23/25
111/111 [==============================] - 3s 26ms/step - loss: 1.0682 -
accuracy: 0.5661 - val_loss: 1.1082 - val_accuracy: 0.5493
Epoch 24/25
111/111 [==============================] - 3s 26ms/step - loss: 1.0639 -
accuracy: 0.5734 - val_loss: 1.0659 - val_accuracy: 0.5697
Epoch 25/25
111/111 [==============================] - 3s 26ms/step - loss: 1.0056 -
accuracy: 0.5862 - val_loss: 1.1071 - val_accuracy: 0.5710
[ ]: from tensorflow.keras.models import load_model
bestModel_lr = load_model('weights_lr.hdf5')
[ ]: y_pred_train_lr = bestModel_lr.predict(X_train2)
y_pred_test_lr = bestModel_lr.predict(X_test2)
predlabel_train_lr = np.argmax(y_pred_train_lr,axis=1)
predlabel_test_lr = np.argmax(y_pred_test_lr,axis=1)
[ ]: from sklearn import metrics
results = metrics.accuracy_score(y_train_labels,predlabel_train_lr)
print('The overall accuracy for new learning rate model Train set is:')
print('{0:.3f}'.format(results))
The overall accuracy for new learning rate model Train set is:
0.647
[ ]: results = metrics.accuracy_score(y_test_labels,predlabel_test_lr)
print('The overall accuracy for the new learning rate model Test set is:')
print('{0:.3f}'.format(results))
The overall accuracy for the new learning rate model Test set is:
0.571
[ ]: classes = np.array([0,1,2,3,4])

```
plot_confusion_matrix(y_test_labels, predlabel_test_lr, classes=classes, ␣
,→normalize=True,
title='Normalized confusion matrix - test (learning ␣
,→rate)')
plt.show()
```

Normalized confusion matrix

```
[[0.46577629 0.03338898 0.19699499 0.19198664 0.11185309]
```



```
[0.04716981 0.68238994 0.07704403 0.13679245 0.05660377]
[0.10278746 0.03832753 0.54181185 0.17944251 0.13763066]
[0.13188648 0.04841402 0.19532554 0.5442404 0.08013356]
[0.09966216 0.03885135 0.15709459 0.09121622 0.61317568]]
```

4.0.2 batch size = 50

```
[ ]: model = Sequential()
model.add(Conv2D(16, (5, 5), padding='valid', input_shape=X_train2.shape[1:
,→],activation = 'relu'))
model.add( MaxPooling2D(pool_size=(2, 2)) )
model.add(Conv2D(16, (3, 3), padding='valid',activation = 'relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
```



```
h = 200
model.add(Dense(h,activation='relu'))
model.add(Dropout(0.5))
num_classes = 5
model.add(Dense(num_classes,activation='softmax'))
[ ]: opt = Adam(lr=0.001, decay=1e-7)
model.compile(loss='categorical_crossentropy',
optimizer=opt,
metrics=['accuracy'])
[ ]: checkpointer = ModelCheckpoint(filepath='/content/weights_batch50.hdf5', ␣
,→monitor='val_accuracy', save_best_only=True)
[ ]: trainSetSize = 12000
Monitor = model.fit(X_train2, y_train,
batch_size=50,
epochs=25,
validation_data=(X_test2, y_test),
callbacks = [checkpointer],
shuffle = True)
```

Epoch 1/25

```
240/240 [==============================] - 4s 16ms/step - loss: 1.5514 -
accuracy: 0.2933 - val_loss: 1.3990 - val_accuracy: 0.4397
Epoch 2/25
240/240 [==============================] - 4s 15ms/step - loss: 1.3782 -
accuracy: 0.4293 - val_loss: 1.2789 - val_accuracy: 0.5077
Epoch 3/25
240/240 [==============================] - 3s 14ms/step - loss: 1.2728 -
accuracy: 0.4816 - val_loss: 1.2035 - val_accuracy: 0.5203
Epoch 4/25
240/240 [==============================] - 4s 16ms/step - loss: 1.1944 -
accuracy: 0.5172 - val_loss: 1.1283 - val_accuracy: 0.5513
Epoch 5/25
240/240 [==============================] - 4s 16ms/step - loss: 1.1125 -
accuracy: 0.5544 - val_loss: 1.1027 - val_accuracy: 0.5643
Epoch 6/25
240/240 [==============================] - 4s 17ms/step - loss: 1.0661 -
accuracy: 0.5778 - val_loss: 1.0678 - val_accuracy: 0.5763
Epoch 7/25
240/240 [==============================] - 4s 17ms/step - loss: 1.0269 -
accuracy: 0.5894 - val_loss: 1.0367 - val_accuracy: 0.5890
37
Epoch 8/25
240/240 [==============================] - 4s 17ms/step - loss: 0.9837 -
accuracy: 0.6188 - val_loss: 1.0091 - val_accuracy: 0.6030
Epoch 9/25
240/240 [==============================] - 4s 16ms/step - loss: 0.9673 -
accuracy: 0.6189 - val_loss: 0.9850 - val_accuracy: 0.6057
Epoch 10/25
240/240 [==============================] - 4s 16ms/step - loss: 0.9233 -
accuracy: 0.6454 - val_loss: 0.9994 - val_accuracy: 0.6097
Epoch 11/25
240/240 [==============================] - 4s 17ms/step - loss: 0.8991 -
accuracy: 0.6497 - val_loss: 0.9526 - val_accuracy: 0.6300
Epoch 12/25
240/240 [==============================] - 4s 16ms/step - loss: 0.8807 -
accuracy: 0.6560 - val_loss: 0.9432 - val_accuracy: 0.6217
Epoch 13/25
240/240 [==============================] - 4s 17ms/step - loss: 0.8526 -
accuracy: 0.6676 - val_loss: 0.9347 - val_accuracy: 0.6413
Epoch 14/25
```

240/240 [==============================] - 4s 18ms/step - loss: 0.8437 -
accuracy: 0.6752 - val_loss: 0.9143 - val_accuracy: 0.6410
Epoch 15/25
240/240 [==============================] - 4s 17ms/step - loss: 0.8309 -
accuracy: 0.6790 - val_loss: 0.9011 - val_accuracy: 0.6467
Epoch 16/25
240/240 [==============================] - 4s 16ms/step - loss: 0.8025 -
accuracy: 0.6867 - val_loss: 0.9037 - val_accuracy: 0.6517
Epoch 17/25
240/240 [==============================] - 4s 16ms/step - loss: 0.7877 -
accuracy: 0.6978 - val_loss: 0.9043 - val_accuracy: 0.6467
Epoch 18/25
240/240 [==============================] - 4s 15ms/step - loss: 0.7635 -
accuracy: 0.7029 - val_loss: 0.8888 - val_accuracy: 0.6493
Epoch 19/25
240/240 [==============================] - 4s 15ms/step - loss: 0.7678 -
accuracy: 0.6992 - val_loss: 0.8770 - val_accuracy: 0.6657
Epoch 20/25
240/240 [==============================] - 4s 16ms/step - loss: 0.7511 -
accuracy: 0.7146 - val_loss: 0.8669 - val_accuracy: 0.6650
Epoch 21/25
240/240 [==============================] - 4s 15ms/step - loss: 0.7337 -
accuracy: 0.7124 - val_loss: 0.8782 - val_accuracy: 0.6603
Epoch 22/25
240/240 [==============================] - 4s 17ms/step - loss: 0.7179 -
accuracy: 0.7218 - val_loss: 0.8493 - val_accuracy: 0.6743
Epoch 23/25
240/240 [==============================] - 4s 18ms/step - loss: 0.7042 -
accuracy: 0.7260 - val_loss: 0.8562 - val_accuracy: 0.6820
38
Epoch 24/25
240/240 [==============================] - 4s 15ms/step - loss: 0.6939 -
accuracy: 0.7350 - val_loss: 0.8607 - val_accuracy: 0.6727
Epoch 25/25
240/240 [==============================] - 4s 16ms/step - loss: 0.6705 -
accuracy: 0.7397 - val_loss: 0.8583 - val_accuracy: 0.6750
[ ]: bestModel_batch50 = load_model('weights_batch50.hdf5')
y_pred_train_batch50 = bestModel_batch50.predict(X_train2)
y_pred_test_batch50 = bestModel_batch50.predict(X_test2)
predlabel_train_batch50 = np.argmax(y_pred_train_batch50,axis=1)

```
predlabel_test_batch50 = np.argmax(y_pred_test_batch50,axis=1)
[ ]: results = metrics.accuracy_score(y_train_labels,predlabel_train_batch50)
print('The overall accuracy for batch size = 50 model Train set is:')
print('{0:.3f}'.format(results))
The overall accuracy for batch size = 50 model Train set is:
0.792
[ ]: results = metrics.accuracy_score(y_test_labels,predlabel_test_batch50)
print('The overall accuracy for the batch size = 50 model Test set is:')
print('{0:.3f}'.format(results))
The overall accuracy for the batch size = 50 model Test set is:
0.682
[ ]: classes = np.array([0,1,2,3,4])
plot_confusion_matrix(y_test_labels, predlabel_test_batch50, classes=classes, ␣
,→normalize=True,
title='Normalized confusion matrix - test (batch size = ␣
,→50)')
plt.show()
Normalized confusion matrix
[[0.52253756 0.03672788 0.16694491 0.20033389 0.07345576]
[0.0172956 0.8663522 0.02830189 0.05345912 0.03459119]
[0.09756098 0.04878049 0.61672474 0.14634146 0.09059233]
[0.07011686 0.04507513 0.12687813 0.69782972 0.06010017]
[0.06081081 0.04898649 0.10810811 0.08952703 0.69256757]]
39
40
```