Homework 3 – Convolutional Neural Network (CNN)

***Anushka Nair, Bam Nakapakorn, and Madeline Luong***

*1. Study set*

This data was extracted from UCI machine learning Repository.

There are 5 Different font types which are ARIAL (CL1) ,CALIBRI (CL2), HANDPRINT (CL3), OCRA (CL4), and TAHOMA (CL5) with 400 features that are pixel positions. The goal is to implement a convolutional neural network to classify the 5 fonts by using the 20x20 matrix of pixel intensities. Then, all non- numerical and missing values have been discarded. The filtering is 0.4 or 0.7 strength and 0 or 1 for italic. We have used the datasets listed below.

Arial (CL1): 4789 x 401

Calibri (CL2):4768 x 401

Handprint (CL3):6872 x 401

Ocra (CL4): 3980 x 401

Tahoma(CL5): 3323 x 401

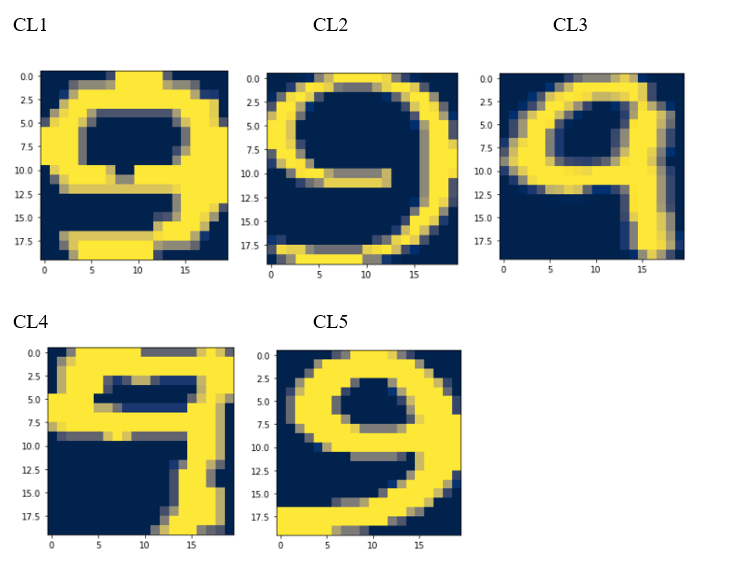
Features : 400 different pixel positions

Target: Classified font type by pixel intensities

Full data set : 23733 observations and instances and 401 features

*2. Preliminary Treatment*

Before implementing the CNN, preliminary treatment must be done. First, the data set must be flattened by changing the features 400 features to 20 x 20 matrices. The 20x20 matrices represent the pixel intensities. To do this we use the reshape function in python to change the shape from (23732 x 400) to (23732 x 20 x20). After reshaping the dataset, we are able to display the pixel intensities as an image. Below are the images for each class as a character 9.



*Figure 1 – Images of each font*

The graphic displays 5 different fonts, from left to right in order they are as following : Arial, Calibri, Handprint, Ocra and Tahoma. This image uses a 20x20 flatted matrix to plot the pixel densities.

*3. Convolutional Neural Network (CNN)*

Convolutional Neural Network (CNN) is a type of neural network typically use to detect pattern of images. It is like an MLP, but CNN has a layer called convolutional layer that helps with feature detection. CNN consists of input, convolution layers, pooling layers, fully-connected layers, and an output. The convolution layer is a filter over the image pixels to create a convolved feature map to see specific features. This layer produces linear activations that the network will then apply a nonlinear function RELU to. Next, a pooling layer is applied to have a translational invariance. Translational invariance means that the image will still be recognized even when it goes through several transformations such as rotation, translation, etc. The pooling layer reduces the number of parameters in the network that is needed to be processed. The 3 pooling functions we can use are max, min, and average pooling. In this assignment, we are using max pooling to summarize statistics of nearby outputs to helps with translational invariance. Then the network will flatten the max pooled output (pooled feature map) into a 1-dimensional array to become inputs of a fully-connected neural network. These “new inputs” will go through the hidden layers. Lastly, the output values of the hidden layer will go through softmax function to change the values into the probabilities of each font class. Softmax is a function that is used to normalize the output of a neural network to a probability distribution over predicted output class. Each value in the output is interpreted as the probability of membership for the given class. Convolutions improve machine learning system by sparse interactions, parameter sharing, and equivalent representations. There are many CNN architectures that we can use. For this assignment, the detail of CNN architecture we use is shown below.

Input -> Conv.1 -> maxpool.1 -> Conv.2 -> maxpool.2 -> flatten -> H -> output -> softmax -> probabilities

**Convolution1** : 16 channels ; window size = 5x5 ; stride =1;

Output of Conv1 = 16 images of size 16x16

**Maxpool1**: 16 channels ; window size = 2x2 ; stride =2;

Output of maxpool1 = 16 images of size 8x8

**Convolution2** : 16 channels ; window size = 3x3 ; stride =1;

Output of Conv.2 = 16 images of size 6x6

**Matchpool2** : 16 channels ; window size = 2x2 ; stride =2;

Output of Matchpool2 = 16 images of size 3x3

**Flatten** : replace the last 16x3x3 pixel intensities by a long vector Vn

Vn has dimension 144

**H** **= last hidden layer** of dimension h

The h neurons of H are fully linked to all the 144 neurons encoding Vn

**Flatten -> H -> outputs -> softmax -> probabilities**

dim (Flatten) = 144; dim (outputs) =5 ; dim (probabilities) = 5

RELU will be use as activation function for neuron responses.

*4. Compute weights and biases and compare to total number of infos from training data*

Through model summary, we can look at the information about the input layer, the parameters of each layer, the shape of the output layers, and the total count of parameters of the model. Each layer’s output becomes the input for the next layer. Below is the model summary of the three different CNNs corresponding to the three hidden layer values:

Table

Description automatically generatedTable

Description automatically generated

Table

Description automatically generated

The number of parameters of dense layers are calculated by *output size\*(input size + 1)*. The number of parameters of convolutional layers are calculated by *output channels\*(input channels\*window size + 1)*. From the model summary above, the input parameters is 16\*(25+1) = 416. Next, the number of parameters if layer 1 is (9\*16+1)\*16 = 2,320. Then the input (5,5,16) from pooling layer is flatten to 3\*3\*16 = 144 values. For dense layer h=90, the number of parameters is 90\*(144+1) = 13050. The calculations for h=150 and h=200 follow the same calculation resulting in number of parameters equal to 21700 and 29000 respectively. Lastly, the output of last dense layer for h=90 has number of parameters is 5\*(90+1) = 455. The number of parameters for h=150 and h=200 is calculated similarly and are equal to 755 and 1005 respectively. The total number of parameters for h=90,150 and 200 is 16241, 25241 and 32741 respectively. As the h value increases, the number of parameters increases as well.

Each training case brings 5 infos, corresponding to the 5 one-hot encoded fonts, which translates to 94925 infos for the whole training set of size 23732. For h=90, the number of weights and thresholds is 16241. Thus, we calculated the ratio of infos per parameter i.e 94925/16241 = 5.84 infos per parameter. For h=150, the number of weights and thresholds is 25241. Thus, the ratio of infos per parameter is 94925/25241 = 3.76 infos per parameter. For h=200, the number of weights and thresholds is 32741 which results in a ratio of infos per parameter of 94925/32741 = 2.89. The three models have good ratios since each parameter corresponds to at least 1 info. Having more infos than parameters is ideal since the estimations are more stable and robust. Typically, too many parameters can potentially lead to overfitting, since the ratio of infos per parameter would be smaller. On the other hand, having too little parameters could potentially lead to underfitting, with one parameter corresponding to too many infos.

*5. Launch training of the CNN*

We will implement automatic learning using TensorFlow. After creating the model, we will apply ADAM gradient descent. Gradient descent is an algorithm use to minimize function by repeating moves in the steepest descending direction. In automatic learning, we use this to update our weights to minimize our loss. The process work by going through each observation one by one and update gradient after each one. This will be cycled through many times according to our chosen epoch size. For ADAM gradient descent, we choose a learning rate that will affect the descent. High learning rate risk going over the lowest point. So, it is better to have very low learning rate. This will give us a more precise calculation. The learning rate applied is .001. To measures the performance of a classification model with output probability value between 0 and 1, we will train the model by cross entropy function. The higher the cross-entropy loss, the worse the probability prediction deviates from the actual label. The batch size is determined by calculating the sqrt(training set) = sqrt(18985) which gives us a batch size of 137. Our chosen epoch size is 100. We also implement early stopping with a patience value of 5, which means that the training will be stopped if there is no improvement after 5 epochs.

Overfitting is often a problem in the neural network. One way to reduce this problem is to use dropout technique in the neural network. Dropout technique allows the network to learn more robust features that are useful for the output. The model does this by randomly dropping certain neurons during every epoch or every minibatch. These neurons that are “ignored” will not be considered during the forward or backward pass in the model. Thus, the dropout technique results in a reduced network model. The dropout layer/ technique aids in overfitting and improves generalization error in the network model.

To decide which layer to implement the dropout method along with how many dropout layers to implement, we explore two different methods to add dropout layers. The first way is to add a dropout layer after each RELU activation resulting in total of 3 dropout layers. The second way is to only add one dropout layer after the hidden layer. To compare the performances of the two methods, we will look at the accuracy for both train and test sets. Ideally, the dropout should reduce the overfitting so the method with lower overfit show the best way to use dropout. To decide the best hidden layer (h\*) resulting in best performing model, we explore different h values (90,150,200). To understand which hidden layer gives the best model, we will look at the accuracy of both train and test sets. Below are the accuracies of 2 ways to implement dropout layers at different hidden layers.

*Model with three drop out layers*

Chart

Description automatically generatedChart, line chart

Description automatically generated

Chart, line chart

Description automatically generated

Time elapsed for h = 90 : 1.45 mins

Time elapsed for h = 150 : 1.38 mins

Time elapsed for h = 200 : 1.34 mins

*Figure 2 – Train and test accuracy plots for h = 90, 150 and 200 with three drop out layers*

The top left plot displays the accuracy for h = 90 and the top right plot displays the accuracy for h = 150. The bottom plot exhibits the values for h = 200. The Y axis on both plots represents the accuracy and the X axis represents the epochs. Further, the train values are in blue and the test values are in red.

Overall, you can see with 3 drop out layers, h = 90 produces the highest accuracy in results. We can also see that all three graphics display an immense amount of overfit. Beginning with h = 90, the overfits seems to be around 2-3% with a test set accuracy of 77% and a train set accuracy of 80%. We can also see that early stopping is implemented and the model halts around 40 epochs.

For h = 150, We can see the final accuracy for the test set is around 67% and the train set is around 73%. Thus, the percent over fit is around 5-6% over fit as well. When implementing early stopping, the model begins to stop around 25 epochs with a patience of 5.

In the h = 200 model, the accuracy for the test set is around 68% and the train accuracy is around 75%. The percentage over fit for this model is around 5-7%. The model begins to implement early stopping around 25 epochs.

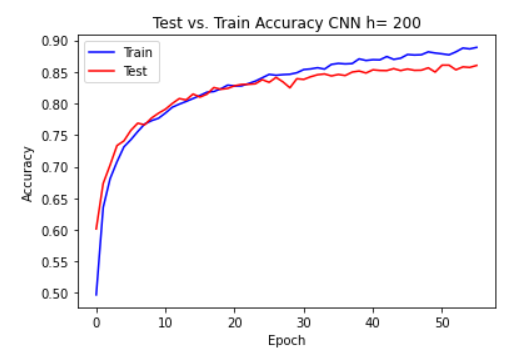
These graphics display that the validation loss is lower than the train loss.

*Model with one drop out layer*

Chart

Description automatically generatedChart

Description automatically generated



Time elapsed for h = 90 : 2.15 mins

Time elapsed for h = 150 : 1.79 mins

Time elapsed for h = 200 : 1.98 mins

*Figure 3 – Train and test accuracy plots for h = 90, 150 and 200 with one drop out layer*

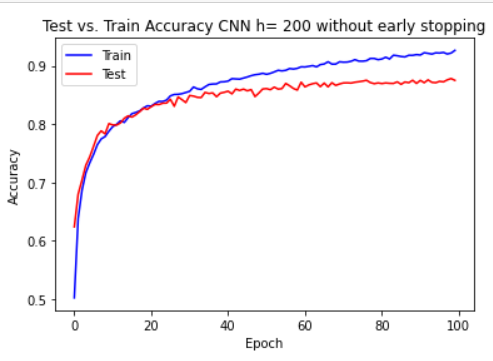
The top left plot displays the accuracy for h = 90 and the top right plot displays the accuracy for h = 150. The bottom plot exhibits the values for h = 200. The Y axis on all plots represents the accuracy and the X axis represents the epochs. Further, the train values are in blue and the test values are in red.

Comparing figure 2 to figure 3 we can see that figure 3 has better fit models. The overfit is not as large as figure 2. Overall, we can see that h = 200 has the highest accuracy out of all three models with a test accuracy of about 85% and a train accuracy of 87%. The model does begin to start over fitting around 30 epochs, although by only about 2-3%. Early stopping also begins around 50 epochs with a patience level of 5.

Where h = 90, we can see that the model is much more overfit compared to h=200 and h=150 and begins to overfit at the start of the model. We can see that the test accuracy stabilizes at 80% and the test accuracy is around 83%, thus the overfit is around 3%. Further, the model begins to stabilize and implement early stopping around 80 epochs with the patience level of 5.

The model displaying h = 150 has a good fit model, the train and test set follow and overlap well. Further, we can see that the test set accuracy stabilizes around 81% and the train set accuracy stabilizes around 84%. The model has a slight overfit of 3% around 40 epochs and begins to stabilize around 50 epochs where early stopping is implemented.

We can assume that the highest accuracy model is h= 200 out of all six models. We can also assume that implementing only one drop layer is beneficial to the accuracy. Adding the drop out after the last hidden layer is better because too many dropout layers leads to a decreased number of neurons and loss of information. To understand if early stopping was implemented correctly, we look figure 4 below.



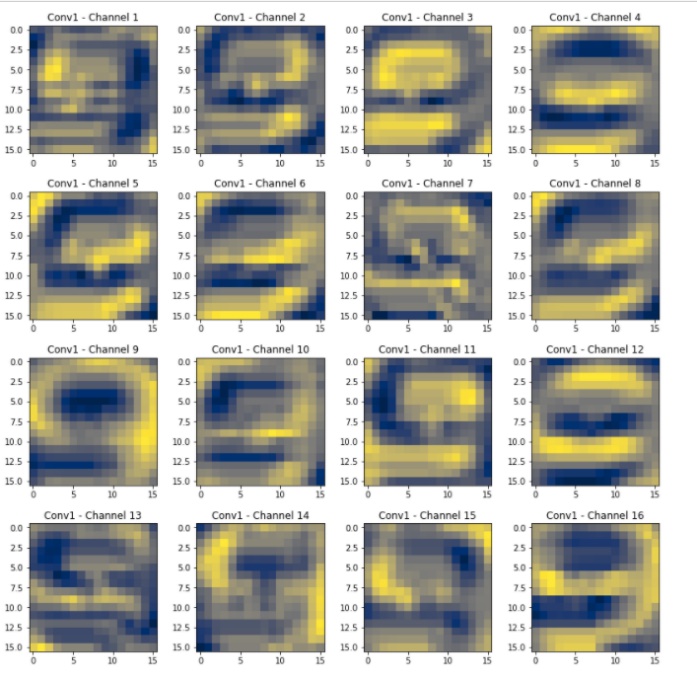
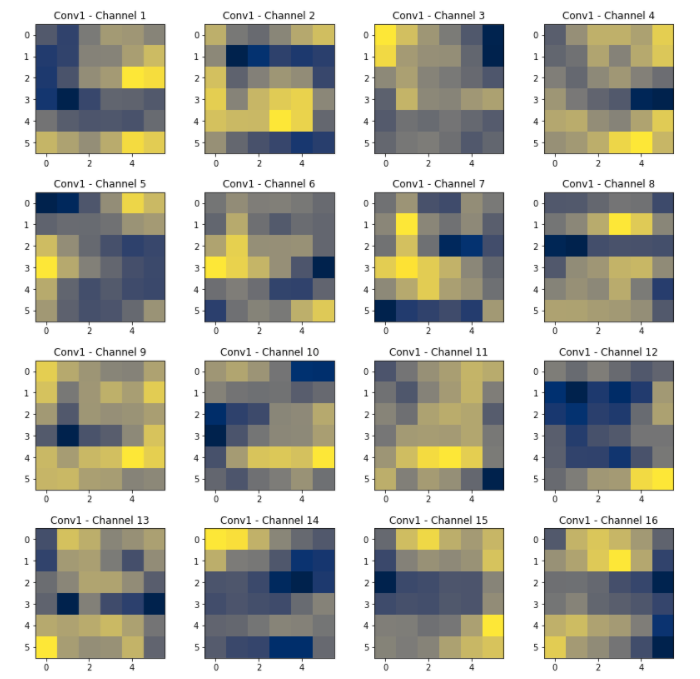
Time elapsed for h = 200 : 2.67 mins

*Figure 4- Accuracy plot for best H (200) without early stopping*

The Y axis represents the accuracy, and the X axis represents the epochs. Further, the train values are in blue and the test values are in red.

From figure 4 we can see that the model does in fact begin to stabilize around 50 epochs. It is safe to assume that having the patience level of 5 is value to stop. After 50 epochs, the model begins to overfit with the train accuracy continuing to increase while the test accuracy remains stagnant.

Next, we decided to look at the output of the convolutional layers for a deeper understanding of the model. We display the output of the two convolutional layers with the image of number 9 from font CL1.

The picture on the left are the 16 filters applied in the first convolutional layer, and the picture on the right are the filters on the second convolutional layer. Filters help identify and highlight different features in an image such as vertical lines, curves and edges. We can see that the 16 filters on the first convolutional layer pick up on different aspects of the number “9” . Some of the filters focus on the curves of the number 9, while some focus on the outline. The second convolutional layer is extremely pixelated because the image has been processed through a few layers including the maxpool. However, the model is able to extract extremely useful information from this layer. Stacking these convolutional layers allow for a hierarchical decomposition of the input image. The first layer filters extract a combination of lower-level features, and the deeper layers extract complex and more abstract features.

*6. Compute the 5x5 confusion matrices for each choice of h*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | **PREDICTED** | | | | |
|  | H = 90 | **CL1** | **CL2** | **CL3** | **CL4** | **CL5** |
| **TRUE** | **CL1** | 88% | 4% | 3% | 3% | 2% |
| **CL2** | 2% | 76% | 2% | 7% | 12% |
| **CL3** | 1% | 1% | 97% | 1% | 1% |
| **CL4** | 2% | 3% | 6% | 88% | 1% |
| **CL5** | 2% | 39% | 5% | 5% | 49% |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | **PREDICTED** | | | | |
|  | H = 150 | **CL1** | **CL2** | **CL3** | **CL4** | **CL5** |
| **TRUE** | **CL1** | 89% | 4% | 2% | 3% | 2% |
| **CL2** | 3% | 78% | 1% | 3% | 15% |
| **CL3** | 1% | 1% | 96% | 2% | 1% |
| **CL4** | 1% | 2% | 2% | 93% | 1% |
| **CL5** | 1% | 34% | 3% | 4% | 59% |

95% CI:(79.2% ,81.5%) 95% CI:(81.9% ,84.1%)

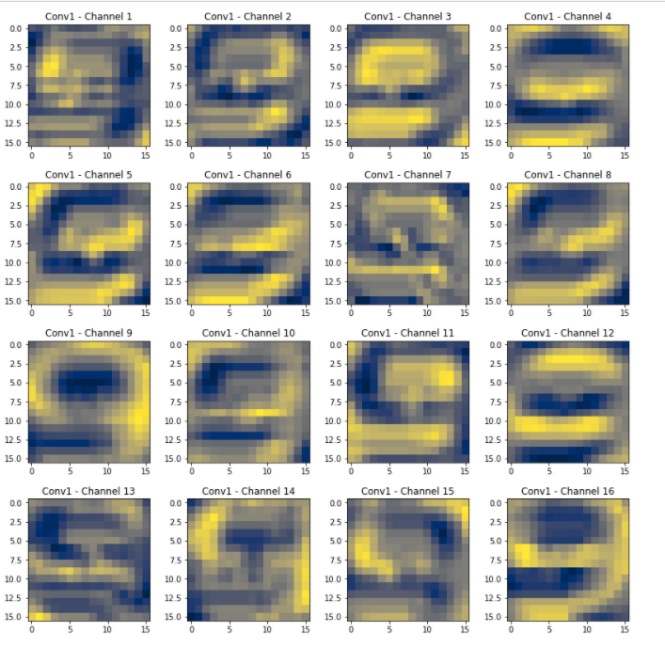
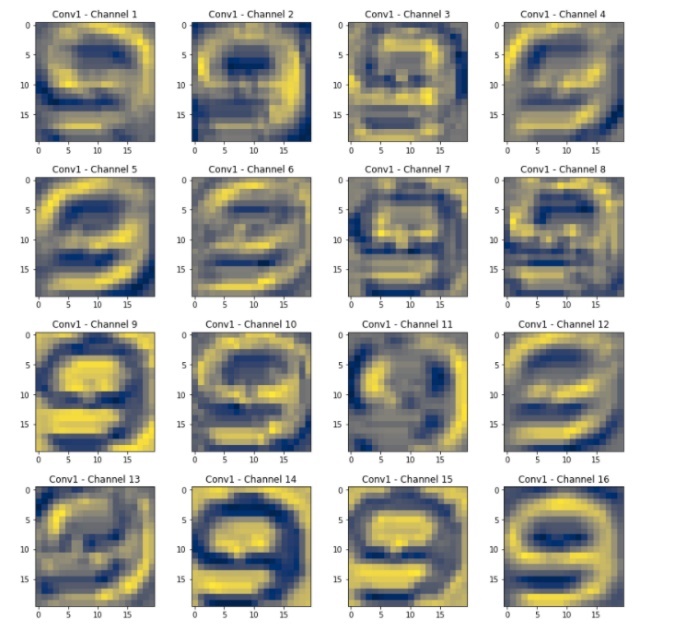
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | **PREDICTED** | | | | |
|  | H\*=200 | **CL1** | **CL2** | **CL3** | **CL4** | **CL5** |
| **TRUE** | **CL1** | 89% | 4% | 3% | 3% | 2% |
| **CL2** | 2% | 79% | 1% | 3% | 15% |
| **CL3** | 1% | 1% | 95% | 2% | 1% |
| **CL4** | 1% | 3% | 3% | 92% | 1% |
| **CL5** | 2% | 30% | 2% | 4% | 63% |

95% CI:( 84.01%,85.9%)

The first model with h=90 has a global accuracy of 80%. The second model with h=150 has a global accuracy of 83%. The model with H=200 has a global accuracy of 84%. The three models are equally good at predicting CL1 and CL3 with extremely high accuracies. For CL2, CL4 and CL5, as dim(H) increases, the accuracy seems to increase. The accuracy for CL5 goes up by 15% with best H=200 which is a significant increase. One significant observation is that all the three models misclassify approximately 35% of CL5 to be CL2. This could be because the fonts look slightly similar. There does not seem to be too much overlap between the three confidence intervals, thus we can conclude that the model with H=200 is the best model.

*7. Further exploration*

As the final step, we decided to further explore if any changes to the model would lead to better results. We tried several approaches including different h values and different filter sizes, however there was no improvement or change. Next, we decided to alter the padding parameter. Padding refers to the process of adding pixels to the input image when being processed by the Convolutional neural network. We set padding = “same”, which means the size of output feature-maps are the same as the input feature-maps when stride=1.



The picture on the left is the output of the first convolutional layer of a model trained with no padding implemented, while the picture on the right is with padding implemented. In the model with no padding, the information on the borders of the images are not preserved as well as the middle, with some portions of the character cut off. The second picture, however, seems to be able to preserve the information on the border of the images just as well as the middle. Thus, we decided to run a new model with the implementation of padding to see if it leads to improved results. Below are two confusion matrices comparing the model with and without padding.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | padding | **PREDICTED** | | | | |
|  | H=200 | **CL1** | **CL2** | **CL3** | **CL4** | **CL5** |
| **TRUE** | **CL1** | 94% | 3% | 1% | 1% | 1% |
| **CL2** | 2% | 83% | 1% | 1% | 13% |
| **CL3** | 1% | 0% | 98% | 1% | 0% |
| **CL4** | 2% | 2% | 2% | 93% | 1% |
| **CL5** | 2% | 25% | 2% | 2% | 70% |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | No padding | **PREDICTED** | | | | |
|  | H\*=200 | **CL1** | **CL2** | **CL3** | **CL4** | **CL5** |
| **TRUE** | **CL1** | 89% | 4% | 3% | 3% | 2% |
| **CL2** | 2% | 79% | 1% | 3% | 15% |
| **CL3** | 1% | 1% | 95% | 2% | 1% |
| **CL4** | 1% | 3% | 3% | 92% | 1% |
| **CL5** | 2% | 30% | 2% | 4% | 63% |

95% CI: (84.01%,85.9%) 95%CI: (86.6%,88.5%)

The model with no padding has a global accuracy of 84% , and the model with padding has a global accuracy of 88%. The two confidence intervals do not overlap, so we can conclude that the model with padding is significantly better than the model without padding. Every class has an improved accuracy rate when moving from the model with no padding to the model with padding. The most significant improvement is to CL5 with a 7% improvement. CL1, CL3 and CL4 have extremely high accuracies in the 90s, while CL2 and CL5 seem to be the lowest. The model with padding also seems to confuse CL2 and CL5, although at a lower rate than without padding. Adding more trainable data to these two fonts could potentially help this problem and lead to higher accuracy.

*Script:*

*# In[1]:*

*import pandas as pd*

*import seaborn as sns*

*from sklearn.preprocessing import StandardScaler*

*from sklearn.decomposition import PCA*

*from matplotlib import pyplot as plt*

*import numpy as np*

*import tensorflow as tf*

*from sklearn.model\_selection import train\_test\_split*

*from tensorflow import keras*

*from tensorflow.keras.layers import Dropout*

*from tensorflow.keras.callbacks import EarlyStopping*

*# In[2]:*

*X= pd.read\_excel("Data/fontx.xlsx")*

*x= np.array(X)*

*Y=pd.read\_excel("Data/fonty.xlsx")*

*Y=Y-1*

*y= np.array(Y)*

*# In[3]:*

*print(x.shape,y.shape)*

*# In[116]:*

*#Reshape the data*

*ReshapeData =x.reshape(23732,20,20)*

*# In[117]:*

*from sklearn.model\_selection import train\_test\_split*

*Ytrain,Ytest,Xtrain,Xtest = train\_test\_split(y,ReshapeData, train\_size=0.8, random\_state= 0)*

*# In[118]:*

*print(Xtrain.shape,Ytrain.shape,Xtest.shape,Ytest.shape)*

*# In[119]:*

*from tensorflow.keras.utils import to\_categorical*

*from tensorflow.keras.models import Sequential*

*from tensorflow.keras.layers import Dense, Activation, Flatten, Conv2D, MaxPooling2D*

*# In[120]:*

*es = EarlyStopping(monitor = 'val\_accuracy',mode='max',patience=5,restore\_best\_weights=True,verbose=1)*

*# In[121]:*

*y\_train = to\_categorical(Ytrain, 5)*

*y\_test = to\_categorical(Ytest, 5)*

*# In[122]:*

*xtrain = tf.expand\_dims(Xtrain, axis=-1)*

*# In[123]:*

*xtest = tf.expand\_dims(Xtest, axis=-1)*

*# In[124]:*

*xtrain.shape*

*# In[ ]:*

*https://www.machinecurve.com/index.php/2019/12/18/how-to-use-dropout-with-keras/*

*# In[125]:*

*xtrain.shape[1:]*

*# In[ ]:*

*#implimenting plot to see layers of models*

*# In[126]:*

*model = Sequential()*

*model.add(Conv2D(16,(5,5), strides=1, input\_shape=xtrain.shape[1:]))*

*model.add(Activation('relu'))*

*#model.add(Dropout(0.5))*

*model.add(MaxPooling2D(pool\_size=(2, 2), strides = 2))*

*#model.add(Dropout(0.5))*

*model.add(Conv2D(16, (3, 3),strides=1))*

*model.add(Activation('relu'))*

*model.add(MaxPooling2D(pool\_size=(2, 2),strides = 2))*

*#model.add(Dropout(0.5))*

*model.add(Flatten())*

*model.add(Dense(150, activation = 'relu'))*

*model.add(Dropout(0.5))*

*model.add(Dense(5,activation = 'softmax'))*

*# 90,150,200*

*# In[127]:*

*#https://stackoverflow.com/questions/63279168/valueerror-input-0-of-layer-sequential-is-incompatible-with-the-layer-expect*

*#https://stackoverflow.com/questions/43895750/keras-input-shape-for-conv2d-and-manually-loaded-images*

*#https://stackoverflow.com/questions/63760734/valueerror-input-0-of-layer-sequential-is-incompatible-with-the-layer-expect*

*# In[128]:*

*model.summary()*

*#https://machinelearningmastery.com/cnn-models-for-human-activity-recognition-time-series-classification/*

*# In[ ]:*

*ReshapeData=tf.expand\_dims(ReshapeData, axis=-1)*

*ReshapeData=ReshapeData/255*

*# In[ ]:*

*convlayer1=model.layers[0](ReshapeData)*

*fig,axes = plt.subplots(4, 4, figsize=(14, 14))*

*for i,x,y in zip(range(16),np.concatenate([([i]\*4) for i in [0,1,2,3]], axis=0),list(range(4))\*4):*

*axes[x,y].imshow(convlayer1[43,:,:,i],cmap='cividis', interpolation='nearest')*

*axes[x,y].set\_title(f'Conv1 - Channel {i+1}')*

*plt.subplots\_adjust(hspace=0.3)*

*plt.show()*

*# In[ ]:*

*# Running model*

*# In[129]:*

*from tensorflow.keras.optimizers import Adam*

*opt = Adam(lr=0.001, decay=1e-7)*

*model.compile(loss='categorical\_crossentropy',*

*optimizer=opt,*

*metrics=['accuracy'])*

*# In[130]:*

*x\_train=xtrain/255*

*x\_test=xtest/255*

*# In[131]:*

*from tensorflow.keras.callbacks import ModelCheckpoint*

*checkpointer = ModelCheckpoint(filepath='/content/weights.hdf5', monitor='val\_accuracy', save\_best\_only=True)*

*# In[132]:*

*Monitor = model.fit(x\_train, y\_train,*

*batch\_size=137,*

*epochs=100,*

*validation\_data=(x\_test, y\_test),*

*callbacks = [es],*

*shuffle = True)*

*#137 = sqrt(training set)*

*# In[133]:*

*plt.plot(Monitor.history['accuracy'],color='blue',label='Train')*

*plt.plot(Monitor.history['val\_accuracy'],color = 'red', label='Test')*

*plt.title('Test vs. Train Accuracy CNN h= 150')*

*plt.ylabel('Accuracy')*

*plt.xlabel('Epoch')*

*plt.legend()*

*plt.show()*

*# In[134]:*

*predictions=model.predict(x\_test)*

*predicted = tf.argmax(predictions, axis = 1)*

*true=tf.argmax(y\_test, axis = 1)*

*# In[135]:*

*## for 90*

*model1 = Sequential()*

*model1.add(Conv2D(16,(5,5), strides=1, input\_shape=xtrain.shape[1:]))*

*model1.add(Activation('relu'))*

*#model.add(Dropout(0.5))*

*model1.add(MaxPooling2D(pool\_size=(2, 2), strides = 2))*

*#model.add(Dropout(0.5))*

*model1.add(Conv2D(16, (3, 3),strides=1))*

*model1.add(Activation('relu'))*

*#model.add(Dropout(0.5))*

*model1.add(MaxPooling2D(pool\_size=(2, 2),strides = 2))*

*model1.add(Flatten())*

*model1.add(Dropout(0.5))*

*model1.add(Dense(90, activation = 'relu'))*

*model1.add(Dense(5,activation = 'softmax'))*

*# 90,150,200*

*# In[136]:*

*from tensorflow.keras.optimizers import Adam*

*opt = Adam(lr=0.001, decay=1e-7)*

*model1.compile(loss='categorical\_crossentropy',*

*optimizer=opt,*

*metrics=['accuracy'])*

*from tensorflow.keras.callbacks import ModelCheckpoint*

*checkpointer = ModelCheckpoint(filepath='/content/weights.hdf5', monitor='val\_accuracy', save\_best\_only=True)*

*# In[137]:*

*Monitor1 = model1.fit(x\_train, y\_train,*

*batch\_size=137,*

*epochs=100,*

*validation\_data=(x\_test, y\_test),*

*callbacks = [es],*

*shuffle = True)*

*#137 = sqrt(training set)*

*# In[138]:*

*plt.plot(Monitor1.history['accuracy'],color='red',label='Test')*

*plt.plot(Monitor1.history['val\_accuracy'],color = 'blue', label='Train')*

*plt.title('Test vs. Train Accuracy CNN h= 90')*

*plt.ylabel('Accuracy')*

*plt.xlabel('Epoch')*

*plt.legend()*

*plt.show()*

*# In[139]:*

*## for 200*

*model2 = Sequential()*

*model2.add(Conv2D(16,(5,5), strides=1, input\_shape=xtrain.shape[1:]))*

*model2.add(Activation('relu'))*

*#model.add(Dropout(0.5))*

*model2.add(MaxPooling2D(pool\_size=(2, 2), strides = 2))*

*#model2.add(Dropout(0.5))*

*model2.add(Conv2D(16, (3, 3),strides=1))*

*model2.add(Activation('relu'))*

*#model2.add(Dropout(0.5))*

*model2.add(MaxPooling2D(pool\_size=(2, 2),strides = 2))*

*model2.add(Flatten())*

*#model2.add(Dropout(0.5))*

*model2.add(Dense(200, activation = 'relu'))*

*model2.add(Dense(5,activation = 'softmax'))*

*# 90,150,200*

*# In[140]:*

*from tensorflow.keras.optimizers import Adam*

*opt = Adam(lr=0.001, decay=1e-7)*

*model2.compile(loss='categorical\_crossentropy',*

*optimizer=opt,*

*metrics=['accuracy'])*

*from tensorflow.keras.callbacks import ModelCheckpoint*

*checkpointer = ModelCheckpoint(filepath='/content/weights.hdf5', monitor='val\_accuracy', save\_best\_only=True)*

*# In[141]:*

*Monitor2 = model2.fit(x\_train, y\_train,*

*batch\_size=137,*

*epochs=100,*

*validation\_data=(x\_test, y\_test),*

*callbacks = [es],*

*shuffle = True)*

*#137 = sqrt(training set)*

*# In[142]:*

*plt.plot(Monitor2.history['accuracy'],color='blue',label='Train')*

*plt.plot(Monitor2.history['val\_accuracy'],color = 'red', label='Test')*

*plt.title('Test vs. Train Accuracy CNN h= 200')*

*plt.ylabel('Accuracy')*

*plt.xlabel('Epoch')*

*plt.legend()*

*plt.show()*

*# In[ ]:*

*#create a model without early stopping and without dropout for comparison*

*# In[149]:*

*predictions=model2.predict(x\_test)*

*predicted = tf.argmax(predictions, axis = 1)*

*true=tf.argmax(y\_test, axis = 1)*

*# In[150]:*

*from sklearn import metrics*

*print(metrics.confusion\_matrix(true, predicted, labels=[0,1, 2, 3,4]))*

*# Printing the precision and recall, among other metrics*

*print(metrics.classification\_report(true, predicted, labels=[0,1, 2, 3,4]))*