1. .mat = matlab file
2. Split data into training and validation set using train\_test\_split
   1. Output has 10 classes, 1 = label 1, … , 0 = label 10, but we transform every 10 and make it into a 0, so we start from 0 to 9
3. Convert images to grey scale
4. Normalize all pixels to have values between 0 and 1
5. Keras.utils.to\_categorical converts a class vector to binary class matrix

I. INTRODUCTION

For this assignment, we have implemented a program that takes as input a training data set and a testing dataset in .mat format, creates a deep convolutional neural network, and saves a trained model as model.h5. We used SciKit Learn libraries, numpy, pandas, matplotlib, h5py, and the Keras functional API from Tensorflow to generate the trained Convolutional Neural Network(CNN) model. To test and validate the model, several different types of layers were manually tested until an accuracy of 90% was achieved. To run the program and generate a new model, the following line must be executed

from the command line in Linux:

$python3 A4.py [train\_32x32.mat] [test\_32x32.mat]

II. DATA PREPROCESSING

The primary task at hand was to classify house numbers given in pictures, which makes the colours of the image almost irrelevant as a red five or a blue five would still be classified as a 5. Based on this reasoning, we decided to convert all the training and testing data sets to grey scale. This drastically reduces the initial input of the CNN as we deal with with a single value per pixel, rather than the three values obtained from the RGB format. Since the colours of each pixel will always range from 0 to 255, we decided to apply a simple normalization step by dividing the value of each pixel by 255.

III. MODEL PARAMETERS

The model parameters were all selected and tested by hand, as there is not a defined method for selecting the number of layers, type of layers, activation functions, amongst other features, other than by trial and error. We started with the default values of the examples found in the Tensorflow documentation, and then proceeded to modify various parameters. Since the program was being executed locally in our computers, we did not have the computational power of servers generally used for this kind of tests as that of IBM’s Watson, or Google’s Alpha Go projects.

i. Activation Function

Only two activation functions were tested: ReLU and **XXXXXXX**. ReLU is generally considered to a good default function for any ML model, and hence, was the first function we decided to test. For testing purposes, we selected **XXXXXXX**, as we wanted to compare the performance of at least 2 activation functions.

ii. Optimizer

The first optimizer tested was ADAM as it is the default keras optimizer. Since we are more familiar with stochastic gradient descent, we also tested SGD, and obtained **BETTER/WORSE** results. Our final selection was **ADAM/SGD**.

iii. Number of epochs and batch size

Although, in the literature and in tutorials, values ranging from 10 to extremely large numbers may be selected, we decided to use an epoch value of 10, as our computational power was limited, and we wanted to avoid overfitting. Likewise, our batch parameter was set to 100 to keep our running as low as possible without compromising too much on accuracy.

iv. Metrics

Our primary metric was accuracy of the results.

from tensorflow import keras

from tensorflow.keras.datasets import mnist

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Dropout

from tensorflow.keras.optimizers import RMSprop

import numpy as np

import pandas as pd

import scipy.io as sio

import matplotlib.pyplot as plt

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

from sklearn.preprocessing import OneHotEncoder

import h5py

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Dropout, Conv2D, MaxPooling2D, Flatten

from tensorflow.keras.optimizers import RMSprop

from tensorflow.keras import backend as K

def display\_sample(num,X\_train\_gray):

    #Print the one-hot array of this sample's label

    print(train\_labels[num])

    #Print the label converted back to a number

    label = train\_labels[num].argmax(axis=0)

    #Reshape the 768 values to a 28x28 image

    image = X\_train\_gray[num].reshape([32,32])

    plt.title('Sample: %d  Label: %d' % (num, label))

    plt.imshow(image, cmap=plt.get\_cmap('gray\_r'))

    plt.show()

path='/home/mun/Dropbox/MachineLearning/MachineLearning\_Shared/Assignments/Assignment4/'

training\_data = sio.loadmat(path+'train\_32x32.mat')

print ( training\_data)

image\_ind =6

x\_train = training\_data['X']

y\_train = training\_data['y']

# plt.imshow(x\_train[:,:,:,image\_ind])

# plt.show()

test\_data= sio.loadmat(path+'test\_32x32.mat')

x\_test = test\_data['X']

y\_test = test\_data['y']

print (y\_train[image\_ind])

print ( x\_train.shape)

#====================================

# extract and reshape Xtrain and y train

#( Num\_observations, Dimentions , channels)

X\_train, y\_train = x\_train.transpose((3,0,1,2)), y\_train[:,0]

print( X\_train.shape)

# extract and reshape Xtest and Ytest

#( Num\_observations, Dimentions , channels)

X\_test, y\_test = x\_test.transpose((3,0,1,2)), y\_test[:,0]

print( X\_test.shape)

print('')

##################################

#using validation set approachs

X\_train, X\_val, y\_train, y\_val = train\_test\_split(X\_train, y\_train, test\_size=0.12, random\_state=7, stratify = y\_train)

y\_train[y\_train == 10] = 0

y\_test[y\_test == 10] = 0

y\_val[y\_val == 10] = 0

#X\_val =X\_train

###########

# Convert all to  gray scall to save calcultaions

X\_train\_gray= (np.dot(X\_train, [0.2990, 0.5870, 0.1140])).astype('float32')

X\_val\_gray= (np.dot(X\_val, [0.2990, 0.5870, 0.1140])).astype('float32')

X\_test\_gray= (np.dot(X\_test, [0.2990, 0.5870, 0.1140])).astype('float32')

X\_train\_gray =X\_train\_gray/255

X\_val\_gray = X\_val\_gray/255

X\_test\_gray =X\_test\_gray/255

#input ("X\_train\_gray")

train\_labels = keras.utils.to\_categorical(y\_train, 10)

test\_labels = keras.utils.to\_categorical(y\_test, 10)

y\_val\_labels = keras.utils.to\_categorical(y\_val, 10)

display\_sample(1234,X\_train\_gray)

if K.image\_data\_format() == 'channels\_first':

    X\_train\_gray = X\_train\_gray.reshape(X\_train\_gray.shape[0], 1, 32, 32)

    X\_val\_gray = X\_val\_gray.reshape(X\_val\_gray.shape[0], 1, 32, 32)

    X\_test = X\_test.reshape(X\_test.shape[0], 1, 32, 32)

    input\_shape = (1, 32, 32)

else:

    X\_train\_gray = X\_train\_gray.reshape(X\_train\_gray.shape[0], 32, 32, 1)

    X\_test\_gray = X\_test\_gray.reshape(X\_test.shape[0], 32, 32, 1)

    X\_val\_gray = X\_val\_gray.reshape(X\_val\_gray.shape[0], 32, 32, 1)

    input\_shape = (32, 32, 1)

model = Sequential()

model.add(Conv2D(32, kernel\_size=(3, 3),

                 activation='relu',

                 input\_shape=input\_shape))

# 64 3x3 kernels

model.add(Conv2D(64, (3, 3), activation='relu'))

model.add(Conv2D(64, (3, 3), activation='relu'))

model.add(Conv2D(64, (3, 3), activation='relu'))

# Reduce by taking the max of each 2x2 block

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

# Dropout to avoid overfitting

model.add(Dropout(0.25))

# Flatten the results to one dimension for passing into our final layer

model.add(Flatten())

# A hidden layer to learn with

model.add(Dense(128, activation='relu'))

# Another dropout

#model.add(Dropout(0.5))

# Final categorization from 0-9 with softmax

model.add(Dense(10, activation='softmax'))

###################original#############

model.summary()

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

              optimizer='adam',

              metrics=['accuracy'])

history = model.fit(X\_train\_gray, train\_labels,

                    batch\_size=100,

                    epochs=10,

                    verbose=2,

                    validation\_data=(X\_val\_gray, y\_val\_labels))

score = model.evaluate(X\_test\_gray, test\_labels, verbose=0)

print('Test loss:', score[0])

print('Test accuracy:', score[1])

model.save(path+'my\_model.h5')