Machine Learning Lab Assignment - 5b

Name: Sri Sai Vijaya Aditya Nittala

Roll No.: 177163

Section: A

Handwritten digit recognition using Convolutional Neural Networks

```
## MOUNTING GOOGLE DRIVE
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mour
```

```
## IMPORTING RELEVANT LIBRARIES
import pandas as pd
import numpy as np
from sklearn import preprocessing
import json
import matplotlib.pyplot as plt
from sklearn import model_selection
```

```
## IMPORTING RELEVANT PACKAGES FOR THE DATASET AS WELL AS THE LAYERS FOR THE CNN MODEL
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import Flatten
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import MaxPooling2D
from keras.utils import np_utils
```

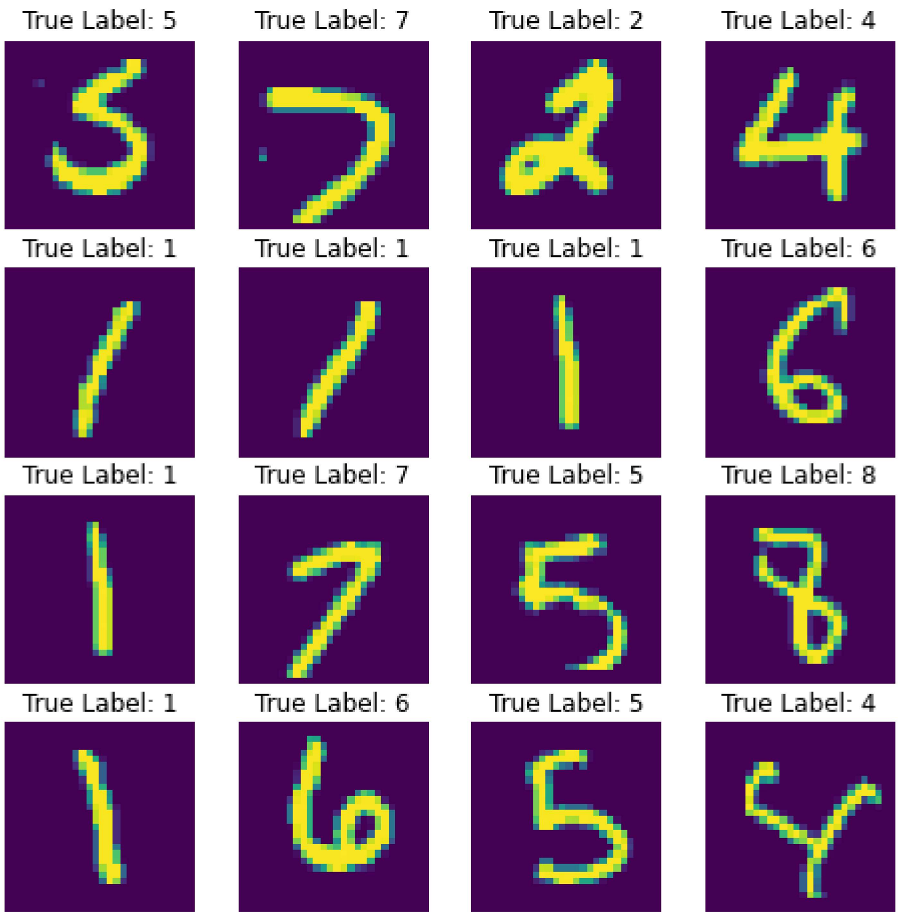
```
## LOADING TRAINING DATA
data = pd.read_csv('/content/drive/MyDrive/mnist.csv')
y = np.array(data['label'])
X = np.array(data.drop(['label'], axis = 1))

## SPLITTING MAIN DATASET INTO TRAINING AND TESTING DATASETS
X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y, test_size = 0.2, ra

## PRINTING IMAGES FROM THE MNIST DATASET
print("Sample of images from the MNIST Dataset : ")
plt.figure(figsize=(8, 8))
for i in range(16):
```

```
    plt.subplot(4, 4, i+1)
    plt.axis('off')
    r = np.random.randint(X_train.shape[0])   ## PICK A RANDON IMAGE TO SHOW
    plt.title('True Label: '+ str(y_train[r])) ## PRINT LABEL
    plt.imshow(X_train[r].reshape(28, 28))     ## PRINT IMAGE
plt.show()
```

```
Sample of images from the MNIST Dataset :
```




```
## NORMALIZING THE PIXEL VALUES FOR TRAINING DATA
X_train = X_train / 255

## CONVERTING TARGET VALUE TO ONE-HOT ENCODED FORM
X_train = np.array(X_train)
b = np.zeros((y_train.size, y_train.max()+1))
b[np.arange(y_train.size), y_train] = 1
y_train = np.array(b)

X_train = X_train.reshape(X_train.shape[0], 28, 28, 1).astype('float32')
print("y_train shape (after one-hot encoding) : {}".format(y_train.shape))
```

```
y_train shape (after one-hot encoding) : (33600, 10)
```

```
## NORMALIZING THE PIXEL VALUES FOR TESTING DATA
X_test = X_test / 255
```

```
## CONVERTING TARGET VALUE TO ONE-HOT ENCODED FORM
X_test = np.array(X_test)
b1 = np.zeros((y_test.size, 10))
b1[np.arange(y_test.size), y_test] = 1
y_test = np.array(b1)

X_test = X_test.reshape(X_test.shape[0], 28, 28, 1).astype('float32')
print("y_test shape (after one-hot encoding) : {}".format(y_test.shape))
```

```
y_test shape (after one-hot encoding) : (8400, 10)
```

```
num_classes = y_train.shape[1]
print("Number of classes : {}".format(num_classes))
```

```
Number of classes : 10
```

```
## CREATE MODEL AND COMPILE
def baseline_model():
    ## CREATE MODEL
    model = Sequential()
    model.add(Conv2D(32, (5, 5), input_shape=(28, 28, 1), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.2))
    model.add(Flatten())
    model.add(Dense(128, activation='relu'))
    model.add(Dense(num_classes, activation='softmax'))
    ## COMPILE MODEL
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
```

```
## SPLITTING INTO VALIDATION SET
X_test, X_val, y_test, y_val = model_selection.train_test_split(X_test, y_test, test_size=0.1
```

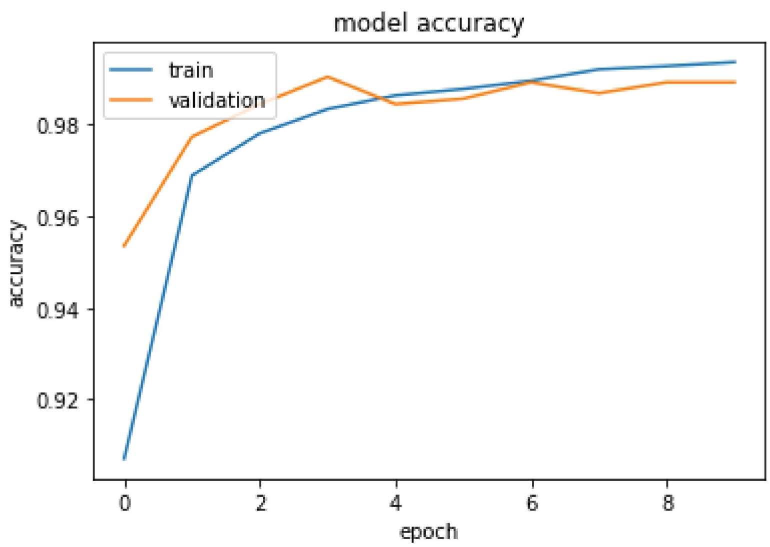
```
## BUILDING THE MODEL
model = baseline_model()
## TRAINING THE MODEL
history = model.fit(X_train, y_train, validation_data=(X_val, y_val), epochs=10, batch_size=2
```

```
Epoch 1/10
168/168 [==============================] - 16s 96ms/step - loss: 0.3296 - accuracy: 0.9
Epoch 2/10
168/168 [==============================] - 16s 95ms/step - loss: 0.1043 - accuracy: 0.9
Epoch 3/10
168/168 [==============================] - 16s 94ms/step - loss: 0.0714 - accuracy: 0.9
Epoch 4/10
168/168 [==============================] - 16s 93ms/step - loss: 0.0528 - accuracy: 0.9
Epoch 5/10
168/168 [==============================] - 16s 93ms/step - loss: 0.0434 - accuracy: 0.9
Epoch 6/10
```

```
168/168 [==============================] - 16s 93ms/step - loss: 0.0369 - accuracy: 0.98
Epoch 7/10
168/168 [==============================] - 16s 93ms/step - loss: 0.0327 - accuracy: 0.98
Epoch 8/10
168/168 [==============================] - 16s 94ms/step - loss: 0.0253 - accuracy: 0.99
Epoch 9/10
168/168 [==============================] - 16s 94ms/step - loss: 0.0231 - accuracy: 0.99
Epoch 10/10
168/168 [==============================] - 16s 94ms/step - loss: 0.0202 - accuracy: 0.99
```

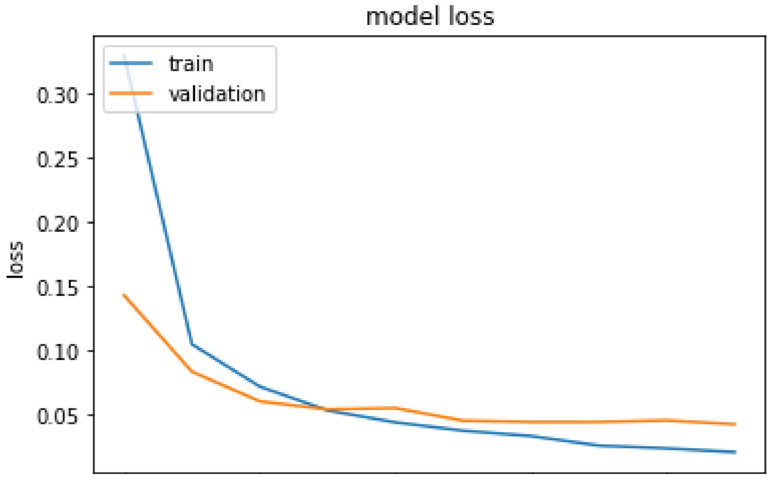
```
print("Training and Cross-validation accuracy with respect to epochs : ")
## ACCURACY VS ITERATIONS GRAPH
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```

```
Training and Cross-validation accuracy with respect to epochs :
```




```
print("Model loss with respect to epochs : ")
## LOSS VS ITERATIONS GRAPH
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```

```
Model loss with respect to epochs :
```




```
## EVALUATING THE MODEL USING TEST DATA
print("Evaluate on test data")
results = model.evaluate(X_test, y_test, batch_size=200)
print("\nTest loss : {}\tTest Accuracy : {}".format(results[0], results[1]))
## PRINTING WEIGHTS
print("\nWeights : {}".format(model.get_weights()))
```

```
Evaluate on test data
38/38 [==============================] - 1s 27ms/step - loss: 0.0471 - accuracy: 0.

Test loss : 0.04712556675076485 Test Accuracy : 0.9865079522132874

Weights : [array([[[[ 8.04903209e-02,  3.58311199e-02, -1.21079206e-01,
          -8.20335746e-02,  5.00785187e-02, -1.40752092e-01,
           6.75003156e-02, -7.89242536e-02, -8.68643522e-02,
          -2.46536791e-01,  1.87083445e-02,  1.09414859e-02,
          -2.45625034e-01, -1.19007915e-01, -1.41105533e-01,
          -2.93295234e-01,  8.91771689e-02, -1.07493453e-01,
          -2.83506244e-01,  8.89354497e-02, -1.54868662e-01,
           7.94681087e-02,  7.12729096e-02,  1.09676853e-01,
           1.22253887e-01,  1.70896381e-01,  1.38106704e-01,
           2.59836018e-02,  4.48092669e-02,  7.59988185e-03,
           4.38960902e-02,  9.46552306e-03]],

        [[-6.34383559e-02,  4.13288064e-02, -3.89649644e-02,
          -1.52956426e-01, -1.68809965e-01, -1.67483941e-01,
           1.36104614e-01, -1.54731795e-01,  5.80176599e-02,
          -1.58353567e-01,  4.69537340e-02, -1.06091596e-01,
          -1.09763727e-01,  3.16098668e-02, -2.20561653e-01,
          -1.98681280e-01,  7.05098584e-02, -5.46483658e-02,
          -2.57186741e-01,  1.10060602e-01, -2.68823504e-01,
           1.64977647e-02,  9.14800465e-02,  1.74672812e-01,
           2.15367246e-02,  1.01895235e-01,  1.74915150e-01,
           8.32996070e-02, -6.32474124e-02, -4.93028713e-03,
           1.03501268e-01, -6.02688603e-02]],

        [[-1.31863207e-02,  1.43554613e-01,  3.28485332e-02,
           9.27556190e-04, -2.58454919e-01, -4.15211683e-03,
           1.54285699e-01, -1.93343565e-01,  1.64745018e-01,
          -4.70422208e-02,  2.70640329e-02, -1.17916875e-01,
```

```
         1.11306675e-01,  7.69583210e-02, -2.30617091e-01,
        -2.64651805e-01,  1.09932832e-01,  6.75769523e-02,
        -1.37332648e-01, -1.20769411e-01, -2.18119249e-01,
         2.66152974e-02,  1.00792550e-01,  1.17125422e-01,
         7.69513547e-02, -5.80309331e-02,  1.47676036e-01,
         7.78525844e-02,  9.81035680e-02,  6.65683523e-02,
        -1.71693228e-02,  8.07870999e-02]],

       [[-4.76860255e-02,  4.95413467e-02,  1.26870200e-01,
         1.00691676e-01, -1.88199729e-01,  9.17144790e-02,
         1.00773238e-01, -1.40660107e-01,  4.45043594e-02,
         1.95903808e-01, -1.56834409e-01, -7.13862628e-02,
         1.41680837e-01,  3.90980728e-02, -1.78589627e-01,
        -1.96402952e-01,  4.70315218e-02,  5.62758930e-02,
        -8.15761387e-02, -1.11445501e-01, -1.96918119e-02,
         8.08384269e-04,  1.31162433e-02,  5.17936237e-02,
        -1.00141190e-01, -1.23945147e-01,  8.43953118e-02,
         4.60981466e-02,  9.73313972e-02, -1.21067025e-01,
         9.88864675e-02,  8.70501548e-02]],

       [[-1.18350191e-02,  1.56594947e-01,  2.91501861e-02,
        -5.33142909e-02, -2.44026817e-02,  9.02718157e-02,
        -3.22613232e-02, -9.91581976e-02, -1.74967617e-01,
         3.49674188e-02,  4.93341964e-03,  2.89866198e-02
```