# ASSIGNMENT-10

**Harish Polishetti - 700744427**

**GitHub Link:** [**https://github.com/HarishPolishetti/icp10**](https://github.com/HarishPolishetti/icp10)

import pandas as pd # data processing, CSV file I/O (e.g. pd.read\_csv) from keras.preprocessing.text import Tokenizer

from tensorflow.keras.preprocessing.sequence import pad\_sequences from keras.models import Sequential

from keras.layers import Dense, Embedding, LSTM, SpatialDropout1D from matplotlib import pyplot

from sklearn.model\_selection import train\_test\_split from keras.utils.np\_utils import to\_categorical import re

from sklearn.preprocessing import LabelEncoder

data = pd.read\_csv('Sentiment.csv')

# Keeping only the neccessary columns data = data[['text','sentiment']]

data['text'] = data['text'].apply(lambda x: x.lower())

data['text'] = data['text'].apply((lambda x: re.sub('[^a-zA-z0-9\s]', '', x)))

for idx, row in data.iterrows(): row[0] = row[0].replace('rt', ' ')

max\_fatures = 2000

tokenizer = Tokenizer(num\_words=max\_fatures, split=' ') tokenizer.fit\_on\_texts(data['text'].values)

X = tokenizer.texts\_to\_sequences(data['text'].values) X = pad\_sequences(X)

embed\_dim = 128

lstm\_out = 196

def createmodel():

model = Sequential()

model.add(Embedding(max\_fatures, embed\_dim,input\_length = X.shape[1]))

model.add(LSTM(lstm\_out, dropout=0.2, recurrent\_dropout=0.2)) model.add(Dense(3,activation='softmax'))

model.compile(loss = 'categorical\_crossentropy', optimizer='adam',metrics = ['accuracy'])

return model

# print(model.summary())

labelencoder = LabelEncoder()

integer\_encoded = labelencoder.fit\_transform(data['sentiment']) y = to\_categorical(integer\_encoded)

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X,y, test\_size = 0.33, random\_state = 42)

batch\_size = 32

model = createmodel()

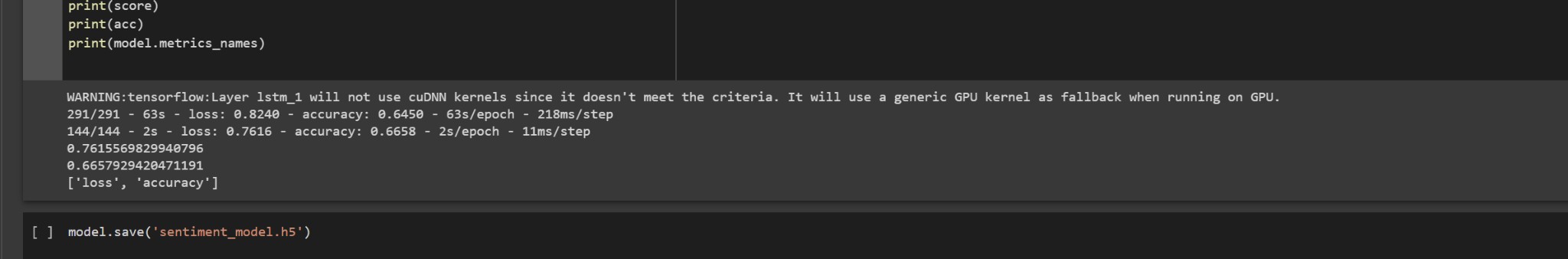
model.fit(X\_train, Y\_train, epochs = 1, batch\_size=batch\_size, verbose

= 2)

score,acc = model.evaluate(X\_test,Y\_test,verbose=2,batch\_size=batch\_size) print(score)

print(acc) print(model.metrics\_names)

Output:



model.save('sentiment\_model.h5')

This code loads the saved model using the load\_model function, and then preprocesses the new text data in the same way as the training data. The predict method is called on the loaded model to get the predicted

class probabilities for the new text data. The class with the highest probability is chosen as the predicted sentiment. The predicted sentiment and probabilities are then printed to the console.

To apply GridSearchCV on the provided source code, we can use the GridSearchCV class from sklearn to search for the best combination of hyperparameters for the LSTM model. The hyperparameters that can be tuned are the number of LSTM units, the dropout rate, and the learning rate of the optimizer.

from keras.models import load\_model import numpy as np

loaded\_model = load\_model('sentiment\_model.h5')

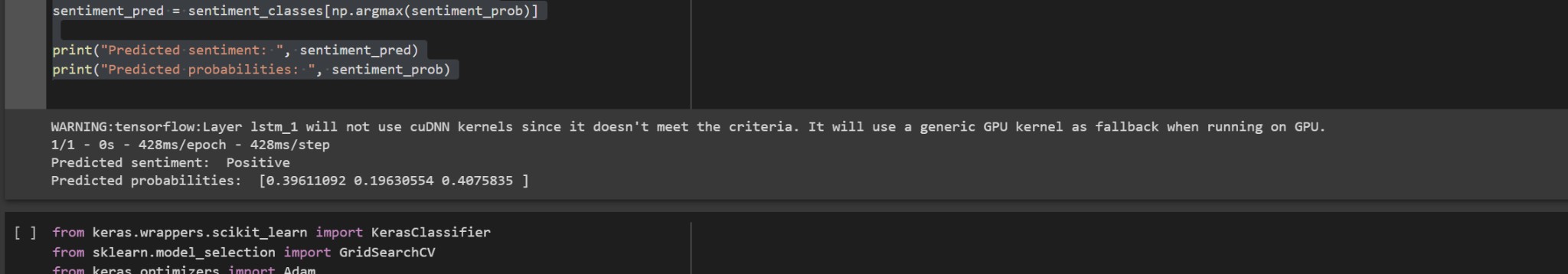
new\_text = ["A lot of good things are happening. We are respected again throughout the world, and that's a great thing.@realDonaldTrump"] new\_text = tokenizer.texts\_to\_sequences(new\_text)

new\_text = pad\_sequences(new\_text, maxlen=X.shape[1], dtype='int32', value=0)

sentiment\_prob = loaded\_model.predict(new\_text, batch\_size=1, verbose=2)[0]

sentiment\_classes = ['Negative', 'Neutral', 'Positive'] sentiment\_pred = sentiment\_classes[np.argmax(sentiment\_prob)]

print("Predicted sentiment: ", sentiment\_pred) print("Predicted probabilities: ", sentiment\_prob)



from keras.wrappers.scikit\_learn import KerasClassifier from sklearn.model\_selection import GridSearchCV

from keras.optimizers import Adam

def create\_model(units=196, dropout=0.2, learning\_rate=0.001): model = Sequential()

model.add(Embedding(max\_fatures, embed\_dim,input\_length = X.shape[1]))

model.add(LSTM(units, dropout=dropout, recurrent\_dropout=dropout)) model.add(Dense(3, activation='softmax'))

optimizer = Adam(lr=learning\_rate) model.compile(loss='categorical\_crossentropy', optimizer=optimizer,

metrics=['accuracy']) return model

model = KerasClassifier(build\_fn=create\_model, verbose=2)

units = [64, 128, 196]

dropout = [0.1, 0.2, 0.3]

learning\_rate = [0.001, 0.01, 0.1]

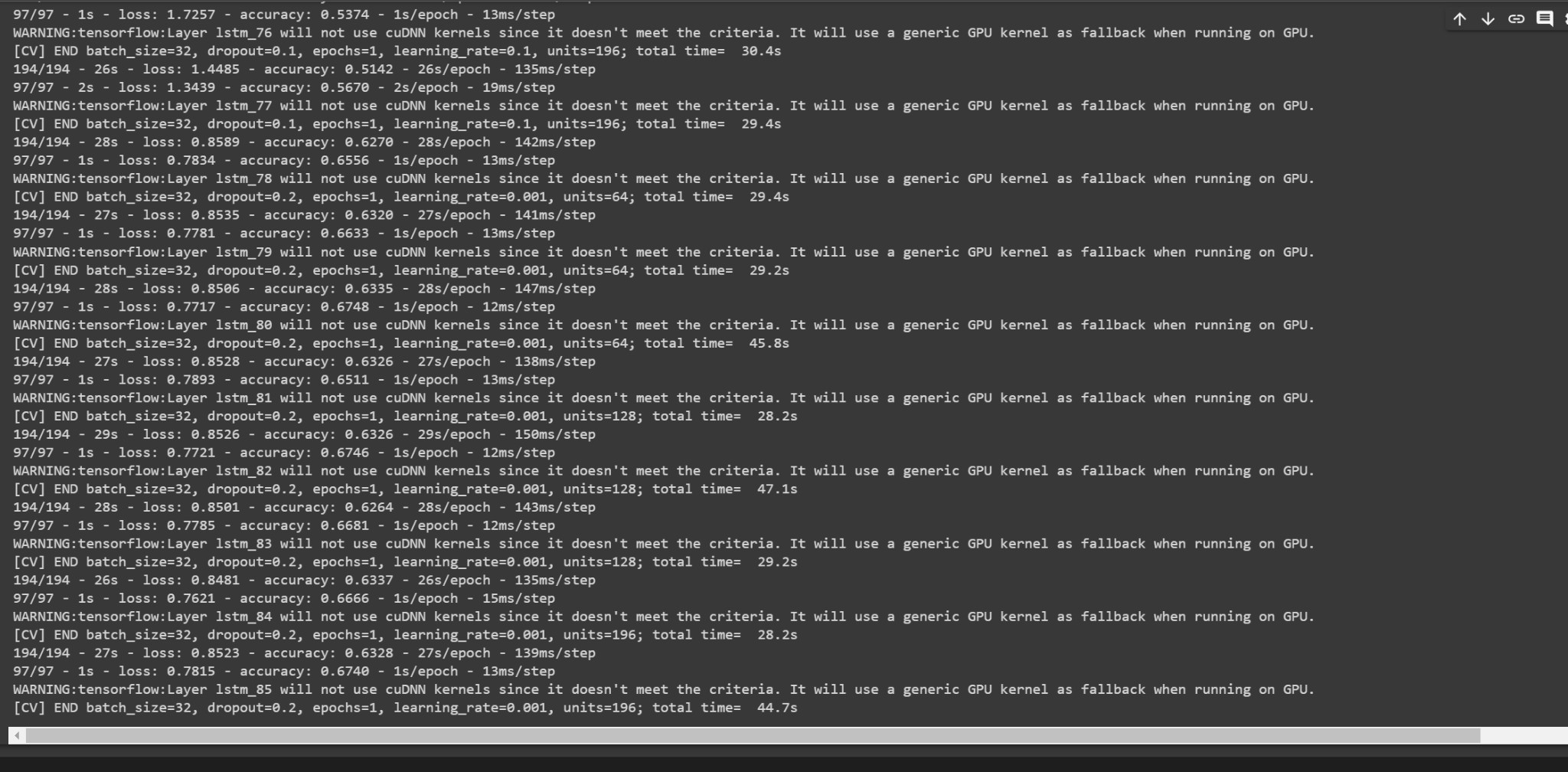
epochs = [1]

batch\_size = [32]

param\_grid = dict(units=units, dropout=dropout, learning\_rate=learning\_rate, epochs=epochs, batch\_size=batch\_size) grid = GridSearchCV(estimator=model, param\_grid=param\_grid, cv=3, verbose=2)

grid\_result = grid.fit(X\_train, Y\_train)

print("Best: %f using %s" % (grid\_result.best\_score\_, grid\_result.best\_params\_))



This code defines the create\_model function that returns a Keras model with the specified hyperparameters. The KerasClassifier class is used to create a wrapper for the create\_model function,

which can be used as an estimator for GridSearchCV. The hyperparameters to be tuned are defined in the param\_grid dictionary. GridSearchCV is then called with the KerasClassifier object, the param\_grid dictionary