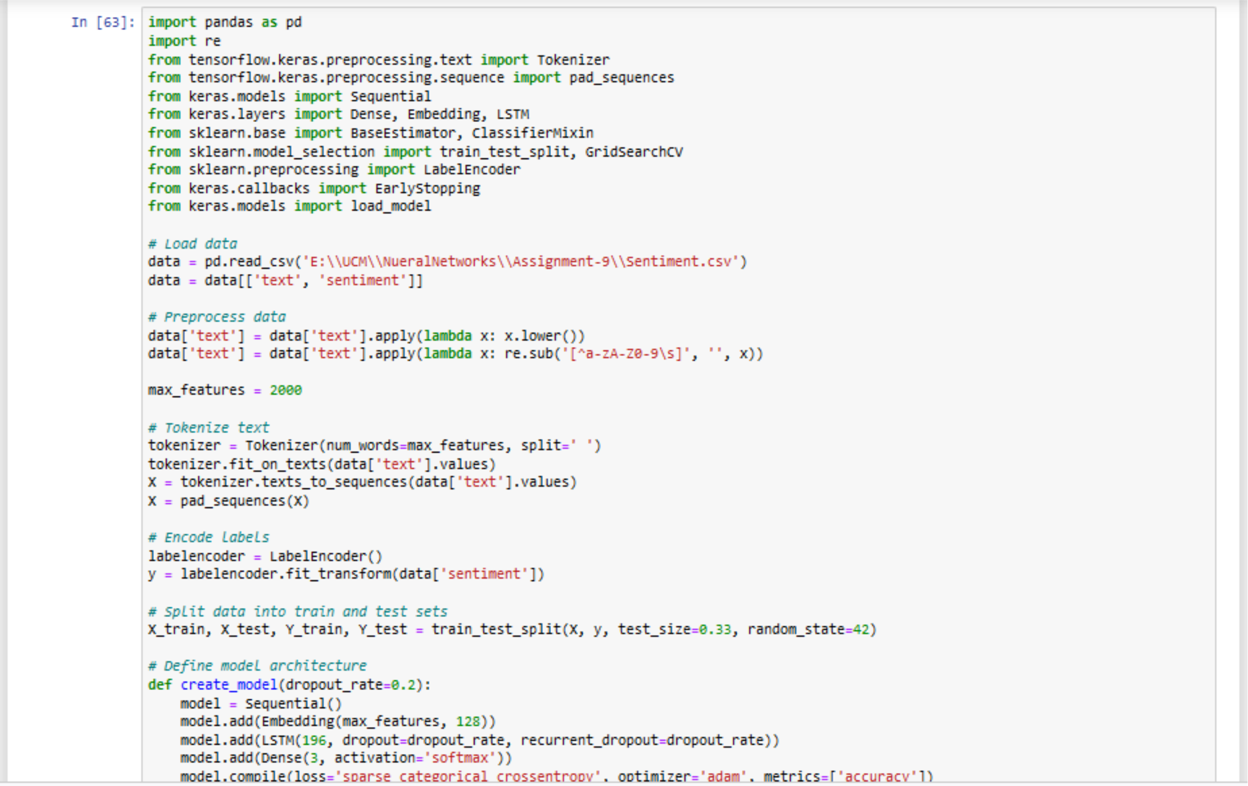
Name: Juhaiza Khan

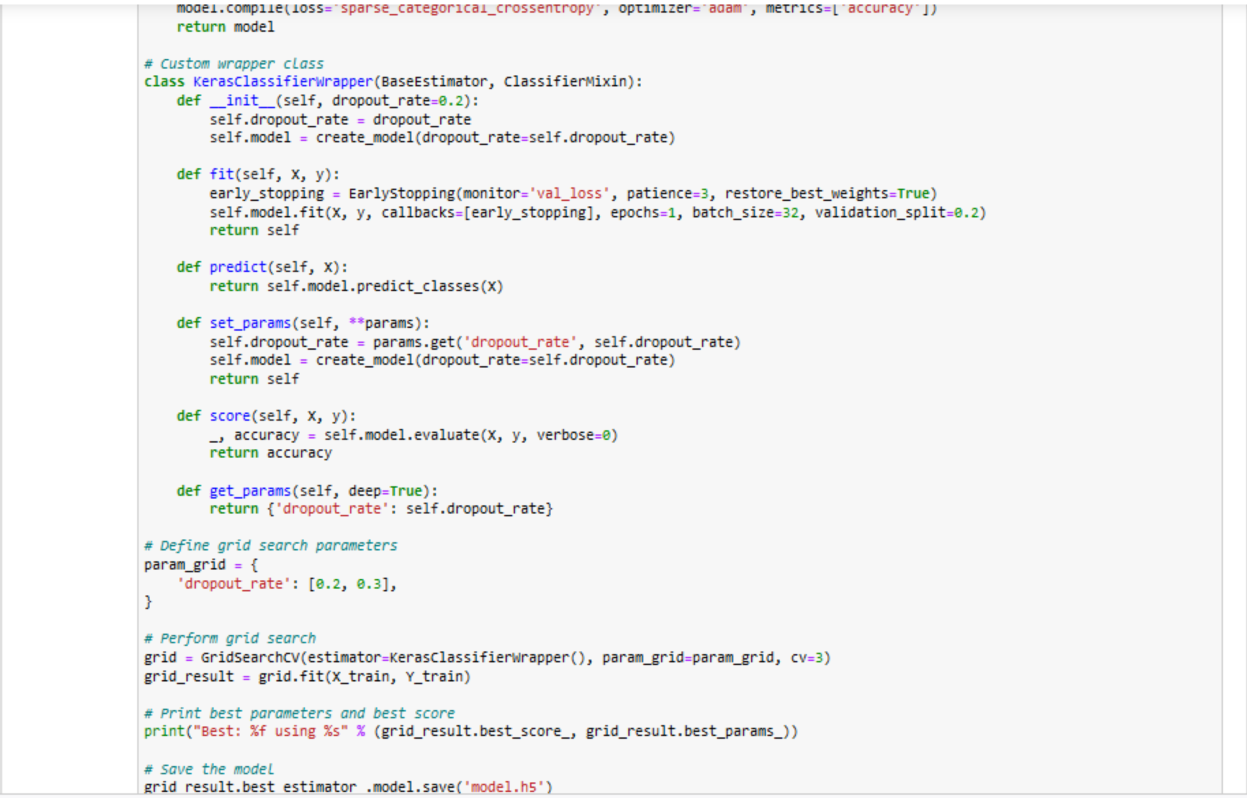
Student No.: 700757258

GitHub --> https://github.com/Juhaizakhan/icp10

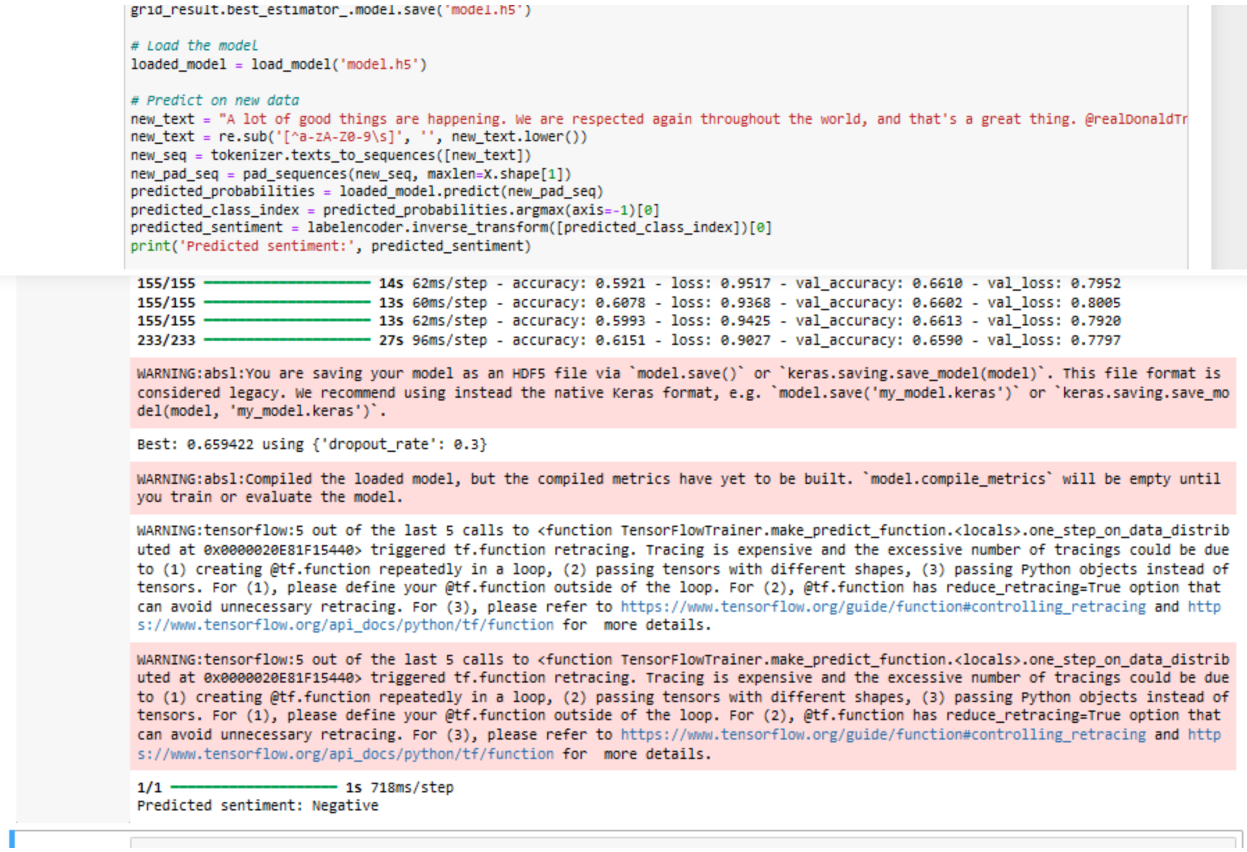
1. Save the model and use the saved model to predict on new text data (ex, “A lot of good things are happening. We are respected again throughout the world, and that's a great thing.@realDonaldTrump”)
2. Apply GridSearchCV on the source code provided in the class



```
In [63]: import pandas as pd
         import re
         from tensorflow.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
         from sklearn.base import BaseEstimator, ClassifierMixin
         from sklearn.model_selection import train_test_split, GridSearchCV
         from sklearn.preprocessing import LabelEncoder
         from keras.callbacks import EarlyStopping
         from keras.models import load_model

         # Load data
         data = pd.read_csv('E:\\UCM\\NueralNetworks\\Assignment-9\\Sentiment.csv')
         data = data[['text', 'sentiment']]

         # Preprocess data
         data['text'] = data['text'].apply(lambda x: x.lower())
         data['text'] = data['text'].apply(lambda x: re.sub('[^a-zA-Z0-9\s]', '', x))

         max_features = 2000

         # Tokenize text
         tokenizer = Tokenizer(num_words=max_features, split=' ')
         tokenizer.fit_on_texts(data['text'].values)
         X = tokenizer.texts_to_sequences(data['text'].values)
         X = pad_sequences(X)

         # Encode labels
         labelencoder = LabelEncoder()
         y = labelencoder.fit_transform(data['sentiment'])

         # Split data into train and test sets
         X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size=0.33, random_state=42)

         # Define model architecture
         def create_model(dropout_rate=0.2):
             model = Sequential()
             model.add(Embedding(max_features, 128))
             model.add(LSTM(196, dropout=dropout_rate, recurrent_dropout=dropout_rate))
             model.add(Dense(3, activation='softmax'))
             model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```



```
    model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model

# Custom wrapper class
class KerasClassifierWrapper(BaseEstimator, ClassifierMixin):
    def __init__(self, dropout_rate=0.2):
        self.dropout_rate = dropout_rate
        self.model = create_model(dropout_rate=self.dropout_rate)

    def fit(self, X, y):
        early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)
        self.model.fit(X, y, callbacks=[early_stopping], epochs=1, batch_size=32, validation_split=0.2)
        return self

    def predict(self, X):
        return self.model.predict_classes(X)

    def set_params(self, **params):
        self.dropout_rate = params.get('dropout_rate', self.dropout_rate)
        self.model = create_model(dropout_rate=self.dropout_rate)
        return self

    def score(self, X, y):
        _, accuracy = self.model.evaluate(X, y, verbose=0)
        return accuracy

    def get_params(self, deep=True):
        return {'dropout_rate': self.dropout_rate}

# Define grid search parameters
param_grid = {
    'dropout_rate': [0.2, 0.3],
}

# Perform grid search
grid = GridSearchCV(estimator=KerasClassifierWrapper(), param_grid=param_grid, cv=3)
grid_result = grid.fit(X_train, Y_train)

# Print best parameters and best score
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))

# Save the model
grid_result.best_estimator_.model.save('model.h5')
```



grid_result.best_estimator_.model.save('model.h5')

```
# Load the model
loaded_model = load_model('model.h5')

# Predict on new data
new_text = "A lot of good things are happening. We are respected again throughout the world, and that's a great thing. @realDonaldTr
new_text = re.sub('[^a-zA-Z0-9\s]', '', new_text.lower())
new_seq = tokenizer.texts_to_sequences([new_text])
new_pad_seq = pad_sequences(new_seq, maxlen=X.shape[1])
predicted_probabilities = loaded_model.predict(new_pad_seq)
predicted_class_index = predicted_probabilities.argmax(axis=-1)[0]
predicted_sentiment = labelencoder.inverse_transform([predicted_class_index])[0]
print('Predicted sentiment:', predicted_sentiment)
```

```
155/155 ━━━━━━━━━━━━━━━━━━━━ 14s 62ms/step - accuracy: 0.5921 - loss: 0.9517 - val_accuracy: 0.6610 - val_loss: 0.7952
155/155 ━━━━━━━━━━━━━━━━━━━━ 13s 60ms/step - accuracy: 0.6078 - loss: 0.9368 - val_accuracy: 0.6602 - val_loss: 0.8005
155/155 ━━━━━━━━━━━━━━━━━━━━ 13s 62ms/step - accuracy: 0.5993 - loss: 0.9425 - val_accuracy: 0.6613 - val_loss: 0.7920
233/233 ━━━━━━━━━━━━━━━━━━━━ 27s 96ms/step - accuracy: 0.6151 - loss: 0.9027 - val_accuracy: 0.6590 - val_loss: 0.7797
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is
considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_mo
del(model, 'my_model.keras')`.
```

```
Best: 0.659422 using {'dropout_rate': 0.3}
```

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until
you train or evaluate the model.
```

```
WARNING:tensorflow:5 out of the last 5 calls to <function TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distrib
uted at 0x0000020E81F15440> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due
to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects instead of
tensors. For (1), please define your @tf.function outside of the loop. For (2), @tf.function has reduce_retracing=True option that
can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/guide/function#controlling_retracing and http
s://www.tensorflow.org/api_docs/python/tf/function for  more details.
```

```
WARNING:tensorflow:5 out of the last 5 calls to <function TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distrib
uted at 0x0000020E81F15440> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due
to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects instead of
tensors. For (1), please define your @tf.function outside of the loop. For (2), @tf.function has reduce_retracing=True option that
can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/guide/function#controlling_retracing and http
s://www.tensorflow.org/api_docs/python/tf/function for  more details.
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 718ms/step
Predicted sentiment: Negative
```