**ICP- 6**

**Tejendra konduru**

**700756561**

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”)](mailto:thing.@realDonaldTrump”)\)

!pip uninstall -y tensorflow keras pandas matplotlib scikit-learn

!pip install tensorflow==2.15.0 keras==2.15.0 pandas==2.0.3 matplotlib==3.9.1 scikit-learn==1.5.1

import pandas as pd # Basic packages for creating dataframes and loading dataset

import numpy as np

import matplotlib.pyplot as plt # Package for visualization

import re # importing package for Regular expression operations

from sklearn.model\_selection import train\_test\_split # Package for splitting the data

from sklearn.preprocessing import LabelEncoder # Package for conversion of categorical to Numerical

from tensorflow.keras.preprocessing.text import Tokenizer # Tokenization

from tensorflow.keras.preprocessing.sequence import pad\_sequences # Add zeros or crop based on the length

from tensorflow.keras.models import Sequential # Sequential Neural Network

from tensorflow.keras.layers import Dense, Embedding, LSTM, SpatialDropout1D # For layers in Neural Network

from tensorflow.keras.utils import to\_categorical

from tensorflow.keras.models import load\_model

# Load the dataset as a Pandas DataFrame

path\_to\_csv = '/content/sample\_data/Sentiment.csv'

dataset = pd.read\_csv(path\_to\_csv, header=0)

# Select only the necessary columns 'text' and 'sentiment'

mask = dataset.columns.isin(['text', 'sentiment'])

data = dataset.loc[:, mask]

# Keeping only the necessary columns

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', ' ') # Removing Retweets

max\_features = 2000

tokenizer = Tokenizer(num\_words=max\_features, split=' ') # Maximum words is 2000 to tokenize sentence

tokenizer.fit\_on\_texts(data['text'].values)

X = tokenizer.texts\_to\_sequences(data['text'].values) # Taking values to feature matrix

X = pad\_sequences(X) # Padding the feature matrix

embed\_dim = 128 # Dimension of the Embedded layer

lstm\_out = 196 # Long short-term memory (LSTM) layer neurons

def createmodel():

    model = Sequential() # Sequential Neural Network

    model.add(Embedding(max\_features, embed\_dim, input\_length = X.shape[1])) # input dimension 2000 Neurons, output dimension 128 Neurons

    model.add(LSTM(lstm\_out, dropout=0.2, recurrent\_dropout=0.2)) # Drop out 20%, 196 output Neurons, recurrent dropout 20%

    model.add(Dense(3, activation='softmax')) # 3 output neurons[positive, Neutral, Negative], softmax as activation

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

    return model

labelencoder = LabelEncoder() # Applying label Encoding on the label matrix

integer\_encoded = labelencoder.fit\_transform(data['sentiment']) # Fitting the model

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) # 67% training data, 33% test data split

batch\_size = 32 # Batch size 32

model = createmodel() # Function call to Sequential Neural Network

model.fit(X\_train, Y\_train, epochs=1, batch\_size=batch\_size, verbose=2) # verbose the higher, the more messages

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

print(score)

print(acc)

print(model.metrics\_names) # metrics of the model

print(integer\_encoded)

print(data['sentiment'])

# Predicting on the text data

sentence = ['A lot of good things are happening. We are respected again throughout the world, and that is a great thing.@realDonaldTrump']

sentence = tokenizer.texts\_to\_sequences(sentence) # Tokenizing the sentence

sentence = pad\_sequences(sentence, maxlen=X.shape[1], dtype='int32', value=0) # Padding the sentence

sentiment\_probs = model.predict(sentence, batch\_size=1, verbose=2)[0] # Predicting the sentence text

sentiment = np.argmax(sentiment\_probs)

print(sentiment\_probs)

if sentiment == 0:

    print("Neutral")

elif sentiment == 1:

    print("Negative")

else:

    print("Positive")

# Custom wrapper for Keras model

from sklearn.base import BaseEstimator, ClassifierMixin

class CustomKerasClassifier(BaseEstimator, ClassifierMixin):

    def \_\_init\_\_(self, build\_fn=None, epochs=1, batch\_size=32, verbose=1, \*\*sk\_params):

        self.build\_fn = build\_fn

        self.epochs = epochs

        self.batch\_size = batch\_size

        self.verbose = verbose

        self.sk\_params = sk\_params

        self.model = None

    def fit(self, X, y, \*\*kwargs):

        self.model = self.build\_fn()

        return self.model.fit(X, y, epochs=self.epochs, batch\_size=self.batch\_size, verbose=self.verbose, \*\*kwargs)

    def predict(self, X, \*\*kwargs):

        return self.model.predict(X, \*\*kwargs)

    def predict\_proba(self, X, \*\*kwargs):

        return self.model.predict(X, \*\*kwargs)

    def score(self, X, y, \*\*kwargs):

        \_, accuracy = self.model.evaluate(X, y, verbose=0)

        return accuracy

# Use the custom Keras classifier

model = CustomKerasClassifier(build\_fn=createmodel, verbose=2)

batch\_size = [10, 20, 40]

epochs = [1, 2]

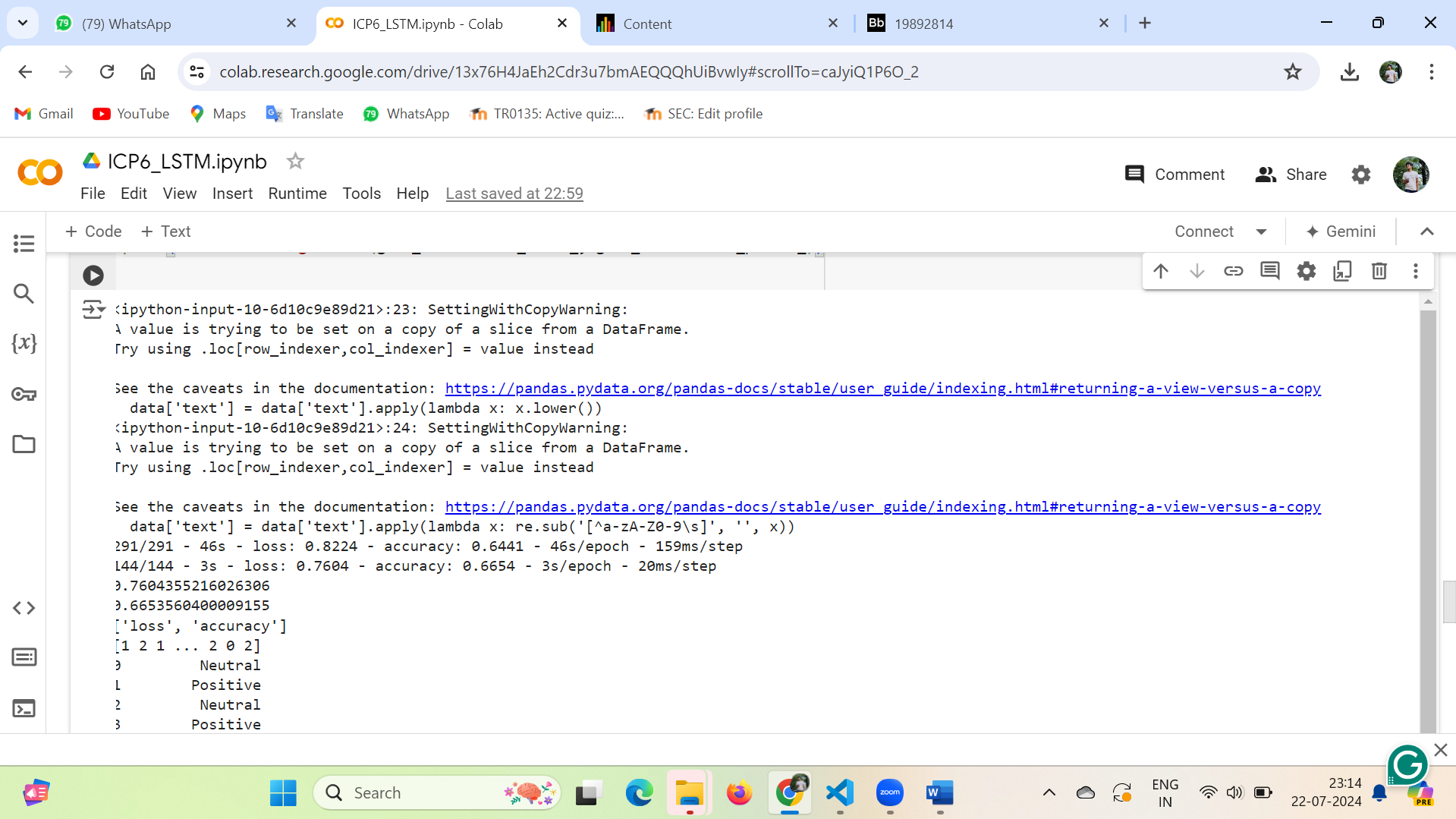
param\_grid = {'batch\_size': batch\_size, 'epochs': epochs}

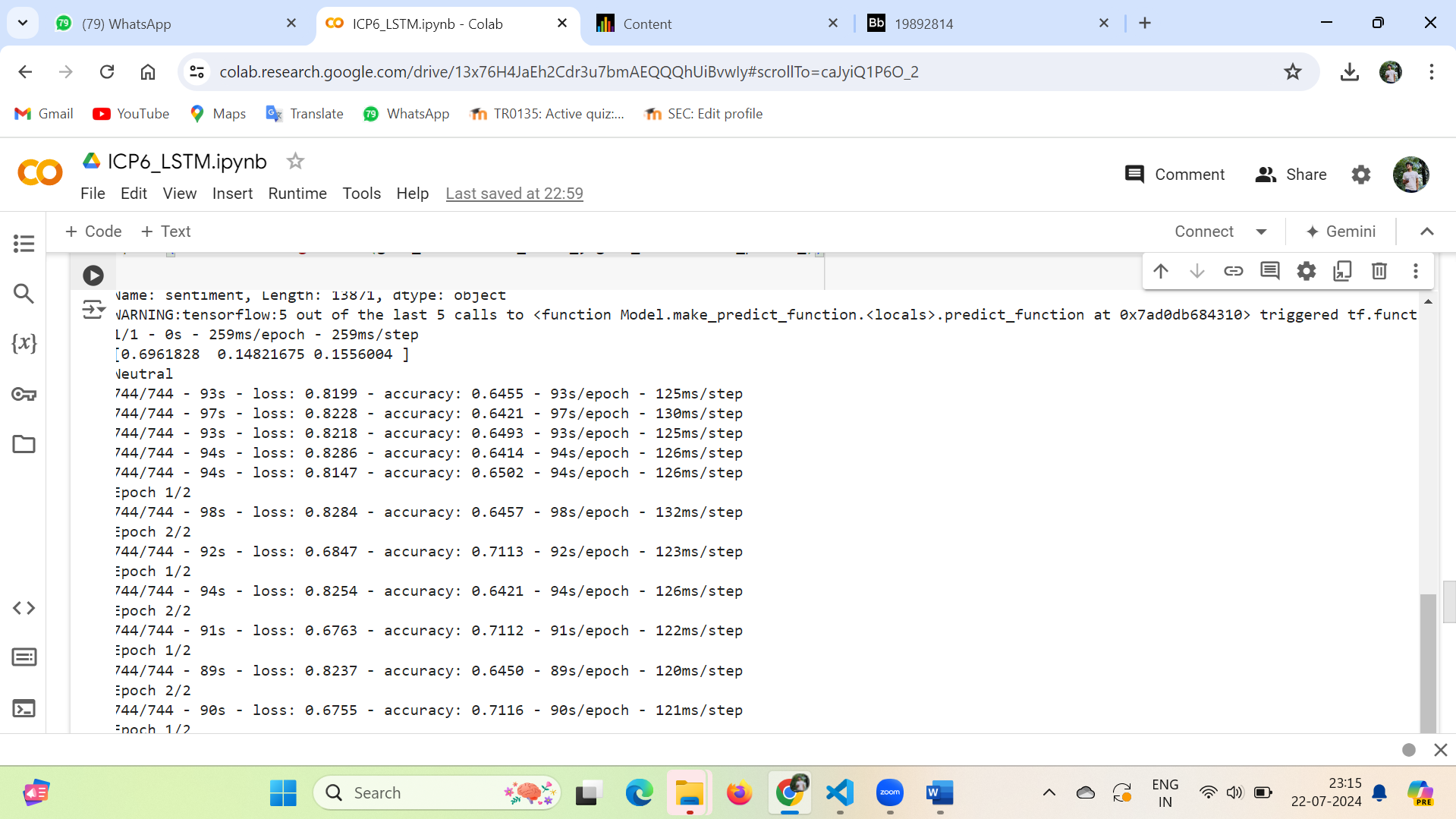
grid = GridSearchCV(estimator=model, param\_grid=param\_grid)

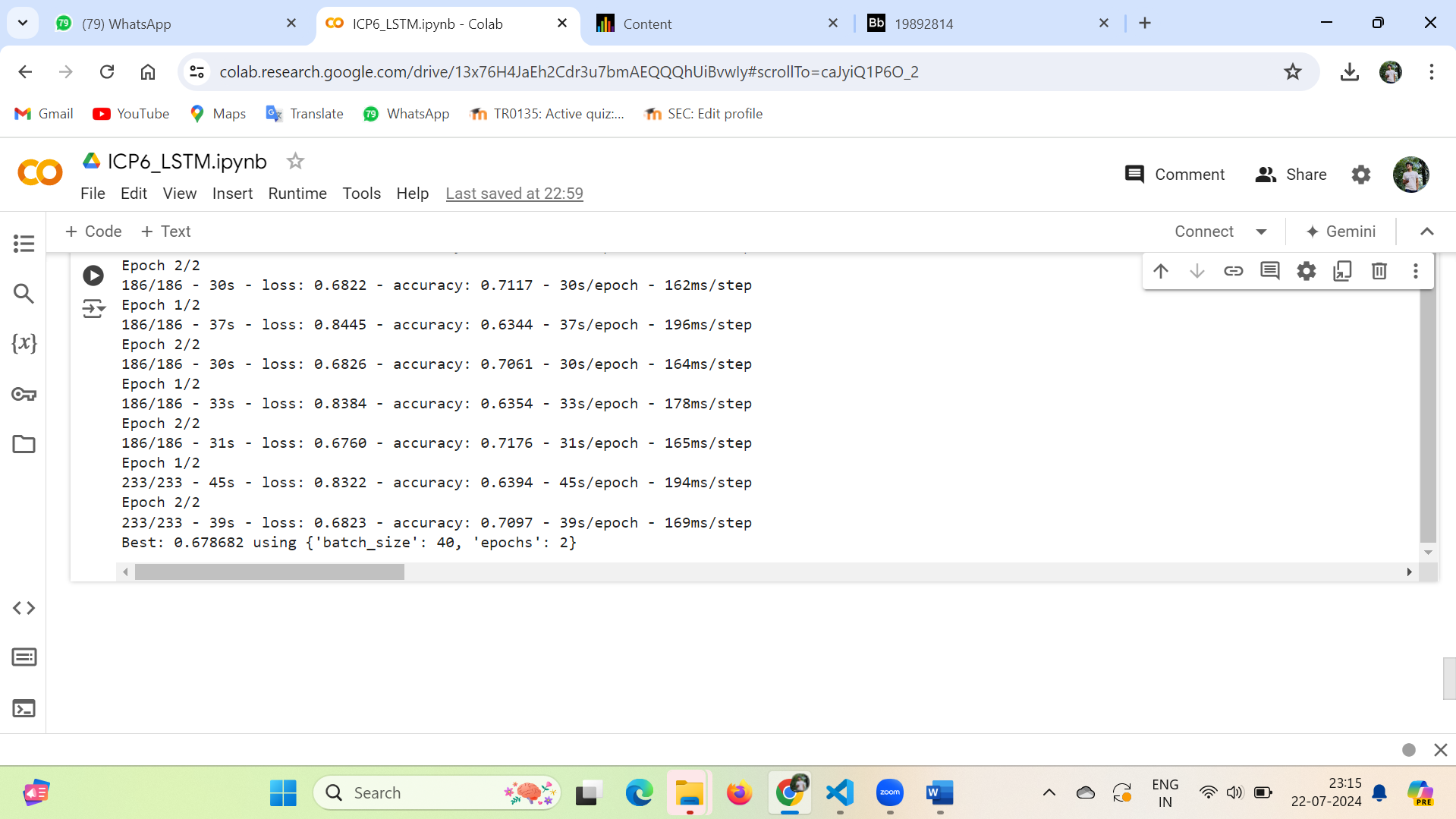
grid\_result = grid.fit(X\_train, Y\_train)

# Summarize results

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







GitHub:

<https://github.com/teja375/DNN/tree/main/ICP6>