**PROJECT REPORT:**

**TWITTER SENTINMENT ANALYSIS:**

**SYED AUN RAZA RIZVI**

**61559**

**BS. AI**

**Project: Sentiment Analysis with CNN Using Keras and TensorFlow**

**1. Introduction**

This project demonstrates how to use a Convolutional Neural Network (CNN) to perform sentiment analysis on tweets. The task is to classify tweets based on their sentiment (positive, negative, or neutral). The project uses a dataset of tweets and their corresponding sentiment labels, preprocesses the data, and then builds, trains, and evaluates a CNN model to predict the sentiment of tweets.

**2. Data Preprocessing**

python

Copy code

import pandas as pd

# Load the CSV file

input\_file = '/content/twitter\_training.csv' # Replace with your CSV file path

df = pd.read\_csv(input\_file)

# Rename columns and drop the second column

# Assuming columns are in the order: ['col1', 'col2', 'col3', 'col4']

# Renaming them to: ['id', 'sentiment', 'tweet']

df = df.rename(columns={

df.columns[0]: 'id', # Rename first column to 'id'

df.columns[2]: 'sentiment', # Rename third column to 'sentiment'

df.columns[3]: 'tweet' # Rename fourth column to 'tweet'

})

# Drop the second column (index 1)

df = df.drop(df.columns[1], axis=1)

# Save the modified DataFrame to a new CSV file

output\_file = 'output\_modified.csv' # Specify the output file path

df.to\_csv(output\_file, index=False)

print(f"New CSV file saved as '{output\_file}' with modified columns.")

**Explanation:**

* The CSV file containing tweets is loaded.
* Unnecessary columns are dropped, and remaining columns are renamed for clarity.
* The cleaned dataset is saved as output\_modified.csv.

**3. Preparing Data for CNN Model**

python

Copy code

import pandas as pd

import numpy as np

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

from sklearn.preprocessing import LabelEncoder

from tensorflow.keras.preprocessing.text import Tokenizer

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

# Load the modified CSV file

input\_file = 'output\_modified.csv'

df = pd.read\_csv(input\_file)

# Check if 'tweet' column exists

if 'tweet' not in df.columns:

raise ValueError("The 'tweet' column is not found in the CSV file.")

# Binary classification: Create a new column if necessary

df['sentiment\_binary'] = df['sentiment'].apply(lambda x: 1 if x == 'positive' else 0)

# Prepare text and label data

texts = df['tweet'].astype(str).values # Convert tweets to strings

labels = df['sentiment'].values # Sentiment labels

# Encode labels into integers

label\_encoder = LabelEncoder()

labels = label\_encoder.fit\_transform(labels)

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(texts, labels, test\_size=0.2, random\_state=42)

# Tokenize and pad the text data

max\_words = 10000

max\_len = 100

tokenizer = Tokenizer(num\_words=max\_words)

tokenizer.fit\_on\_texts(X\_train)

X\_train\_seq = tokenizer.texts\_to\_sequences(X\_train)

X\_test\_seq = tokenizer.texts\_to\_sequences(X\_test)

X\_train\_pad = pad\_sequences(X\_train\_seq, maxlen=max\_len)

X\_test\_pad = pad\_sequences(X\_test\_seq, maxlen=max\_len)

**Explanation:**

* The tweet texts are tokenized and padded into sequences so they can be used as input to the CNN.
* Labels are encoded as integers to work with the model.
* The dataset is split into training and testing sets.

**4. Building and Training the CNN Model**

python

Copy code

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Embedding, Conv1D, MaxPooling1D, Flatten, Dense

from tensorflow.keras.optimizers import Adam

# Build the CNN model

model = Sequential()

model.add(Embedding(input\_dim=max\_words, output\_dim=128, input\_length=max\_len)) # Embedding layer

model.add(Conv1D(filters=128, kernel\_size=5, activation='relu')) # Convolutional layer

model.add(MaxPooling1D(pool\_size=4)) # Max-pooling layer

model.add(Flatten()) # Flatten the 1D output

model.add(Dense(10, activation='relu')) # Fully connected layer

model.add(Dense(len(np.unique(labels)), activation='softmax')) # Output layer for multi-class classification

# Compile the model

model.compile(optimizer=Adam(), loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

# Train the model

history = model.fit(X\_train\_pad, y\_train, epochs=5, batch\_size=32, validation\_split=0.1)

**Explanation:**

* The model is built using an embedding layer to convert tokens to vectors.
* A 1D convolutional layer is used for feature extraction, followed by max-pooling and a fully connected dense layer.
* The model is compiled using the Adam optimizer and sparse categorical cross-entropy as the loss function.

**5. Evaluating the Model**

python

Copy code

# Evaluate the model on the test data

loss, accuracy = model.evaluate(X\_test\_pad, y\_test)

print(f"Test loss: {loss}")

print(f"Test accuracy: {accuracy}")

# Predict sentiment for the test set

predictions = model.predict(X\_test\_pad)

predicted\_labels = np.argmax(predictions, axis=1)

**Explanation:**

* The trained model is evaluated on the test set, and its loss and accuracy are printed.
* Predictions are made for the test set, and the predicted sentiment labels are generated by taking the class with the highest probability.

**6. Plotting Training and Validation Loss/Accuracy**

python

Copy code

import matplotlib.pyplot as plt

# Retrieve the training history

history\_dict = history.history

train\_loss = history\_dict['loss']

val\_loss = history\_dict['val\_loss']

train\_acc = history\_dict['accuracy']

val\_acc = history\_dict['val\_accuracy']

# Plot training & validation loss values

plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)

plt.plot(train\_loss, label='Training Loss')

plt.plot(val\_loss, label='Validation Loss')

plt.title('Model Loss')

plt.xlabel('Epoch')

plt.ylabel('Loss')

plt.legend()

# Plot training & validation accuracy values

plt.subplot(1, 2, 2)

plt.plot(train\_acc, label='Training Accuracy')

plt.plot(val\_acc, label='Validation Accuracy')

plt.title('Model Accuracy')

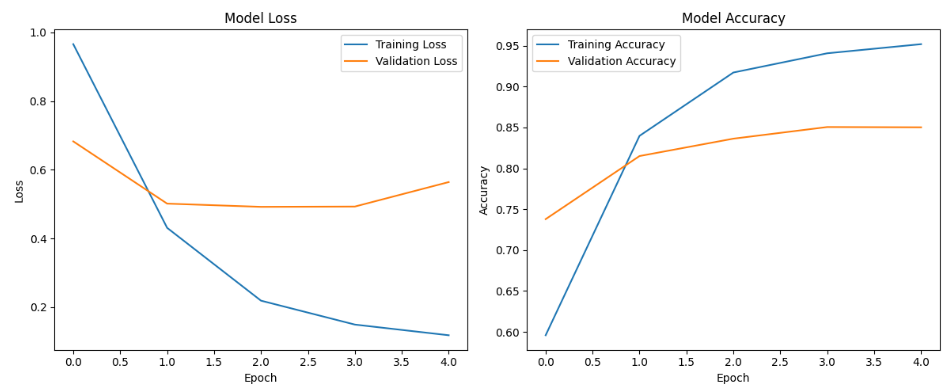
plt.xlabel('Epoch')

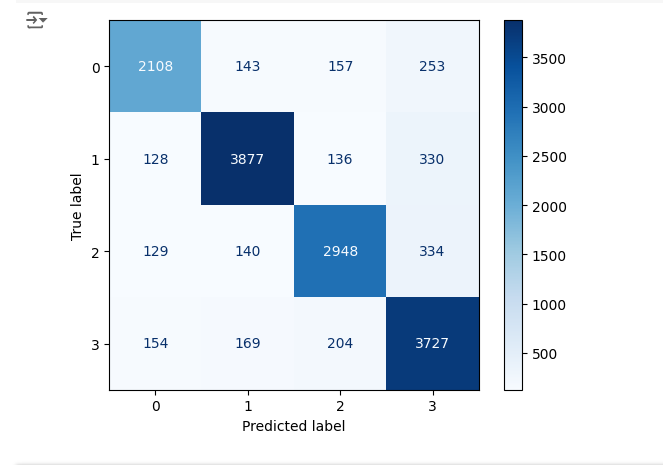
plt.ylabel('Accuracy')

plt.legend()

plt.tight\_layout()

plt.show()





**Explanation:**

* A visual representation of the training process is provided by plotting the training and validation loss and accuracy for each epoch.

**7. Confusion Matrix**

python

Copy code

from sklearn.metrics import confusion\_matrix, ConfusionMatrixDisplay

# Generate the confusion matrix

cm = confusion\_matrix(y\_test, predicted\_labels)

# Display the confusion matrix

disp = ConfusionMatrixDisplay(confusion\_matrix=cm)

disp.plot(cmap=plt.cm.Blues)

plt.show()

**Explanation:**

* The confusion matrix is generated to visualize the performance of the model in classifying each sentiment label correctly.

**8. Saving and Downloading the Model**

python

Copy code

# Save the model to disk

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

# Download the saved model in Colab

from google.colab import files

files.download('my\_model.h5')

**Explanation:**

* The trained model is saved as my\_model.h5, and then downloaded using Google Colab's file interface.

**9. Uploading a Model File**

python

Copy code

from google.colab import files

# Upload the model file

uploaded = files.upload()

# Once uploaded, the file will be available in the current directory

**Explanation:**

* This code allows you to upload a model file from your local machine to the Colab environment.

**Summary:**

This project demonstrates how to preprocess a text dataset, tokenize and pad sequences for input into a CNN model, and train and evaluate the CNN for sentiment classification. It also includes saving and loading the model, as well as generating a confusion matrix for performance evaluation.