Dependencies

```
import numpy as np
import math
import re
import pandas as pd
from bs4 import BeautifulSoup
import random
import tensorflow as tf
from google.colab import drive
from transformers import BertTokenizer, TFBertModel

drive.mount("/content/drive")
```

```
Mounted at /content/drive
```

```
!pip install transformers
!pip install sentencepiece
```

```
Requirement already satisfied: transformers in /usr/local/lib/python3.10/dist-packages (4.42.4)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from transformers) (3.15.4)
Requirement already satisfied: huggingface-hub<1.0,>=0.23.2 in /usr/local/lib/python3.10/dist-packages (from transformers) (0.23.5)
Requirement already satisfied: numpy<2.0,>=1.17 in /usr/local/lib/python3.10/dist-packages (from transformers) (1.25.2)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from transformers) (24.1)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-packages (from transformers) (6.0.1)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.10/dist-packages (from transformers) (2024.5.15)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from transformers) (2.31.0)
Requirement already satisfied: safetensors>=0.4.1 in /usr/local/lib/python3.10/dist-packages (from transformers) (0.4.3)
Requirement already satisfied: tokenizers<0.20,>=0.19 in /usr/local/lib/python3.10/dist-packages (from transformers) (0.19.1)
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.10/dist-packages (from transformers) (4.66.4)
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.23.2->transformers) (2024.6.1)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.23.2->transformers) (4.12.2)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (2024.7.4)
Requirement already satisfied: sentencepiece in /usr/local/lib/python3.10/dist-packages (0.1.99)
```

Load Data

```
cols = ["sentiment", "id", "date", "query", "user", "text"]
data = pd.read_csv(
    "/content/drive/MyDrive/dataset/training.csv",
    header=None,
    names=cols,
    engine="python",
    encoding="latin1"
)
data.drop(["id", "date", "query", "user"], axis=1, inplace=True)
data.head(5)
```

	sentiment	text
0	0	@switchfoot http://twitpic.com/2y1zl - Awww, t...
1	0	is upset that he can't update his Facebook by ...
2	0	@Kenichan I dived many times for the ball. Man...
3	0	my whole body feels itchy and like its on fire
4	0	@nationwideclass no, it's not behaving at all...

Data Cleaning

```
def clean_tweet(tweet):
    tweet = BeautifulSoup(tweet, "lxml").get_text()
    tweet = re.sub(r"@[A-Za-z0-9]+", ' ', tweet)
    tweet = re.sub(r"https?://[A-Za-z0-9./]+", ' ', tweet)
    tweet = re.sub(r"[^a-zA-Z.!?']", ' ', tweet)
    tweet = re.sub(r" +", ' ', tweet)
    return tweet

data_clean = [clean_tweet(tweet) for tweet in data.text]
data_labels = data.sentiment.values
data_labels[data_labels == 4] = 1
```

```
<ipython-input-4-5c41a25d0dcc>:2: MarkupResemblesLocatorWarning: The input looks more like a filename than markup. You may want to open this file and pass the filehan
  tweet = BeautifulSoup(tweet, "lxml").get_text()
```

Tokenization and Encoding

```
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')

def encode_sentence(sent):
    return tokenizer.encode(sent, add_special_tokens=True)

data_inputs = [encode_sentence(sentence) for sentence in data_clean]
data_with_len = [[sent, data_labels[i], len(sent)] for i, sent in enumerate(data_inputs)]
random.shuffle(data_with_len)
data_with_len.sort(key=lambda x: x[2])
sorted_all = [(sent_lab[0], sent_lab[1]) for sent_lab in data_with_len if sent_lab[2] > 7]
```

```
/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_token.py:89: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secret in your Google Colab and res
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
```

```
tokenizer_config.json: 100%                                    48.0/48.0 [00:00<00:00, 2.74kB/s]

vocab.txt: 100%                                    232k/232k [00:00<00:00, 5.27MB/s]

tokenizer.json: 100%                                    466k/466k [00:00<00:00, 5.50MB/s]

config.json: 100%                                    570/570 [00:00<00:00, 28.6kB/s]
```

Creating Dataset

```
all_dataset = tf.data.Dataset.from_generator(lambda: sorted_all, output_types=(tf.int32, tf.int32))
BATCH_SIZE = 32
all_batched = all_dataset.padded_batch(BATCH_SIZE, padded_shapes=((None,), ()))
NB_BATCHES = math.ceil(len(sorted_all) / BATCH_SIZE)
NB_BATCHES_TEST = NB_BATCHES // 10
test_dataset = all_batched.take(NB_BATCHES_TEST)
train_dataset = all_batched.skip(NB_BATCHES_TEST)
```

```
WARNING:tensorflow:From <ipython-input-6-e4d4685df90e>:1: calling DatasetV2.from_generator (from tensorflow.python.data.ops.dataset_ops) with output_types is deprecat
Instructions for updating:
Use output_signature instead
```

```
import tensorflow as tf
from tensorflow.keras import layers
```

Define the Model

```
class DCNN(tf.keras.Model):
    def __init__(self, vocab_size, emb_dim=128, nb_filters=50, FFN_units=512, nb_classes=2, dropout_rate=0.1, name="dcnn"):
        super(DCNN, self).__init__(name=name)
        self.embedding = layers.Embedding(vocab_size, emb_dim)
        self.bigram = layers.Conv1D(filters=nb_filters, kernel_size=2, padding="valid", activation="relu")
        self.trigram = layers.Conv1D(filters=nb_filters, kernel_size=3, padding="valid", activation="relu")
        self.fourgram = layers.Conv1D(filters=nb_filters, kernel_size=4, padding="valid", activation="relu")
        self.pool = layers.GlobalMaxPooling1D()
        self.dense_1 = layers.Dense(units=FFN_units, activation="relu")
        self.dropout = layers.Dropout(rate=dropout_rate)
        self.last_dense = layers.Dense(units=nb_classes, activation="softmax")

    def call(self, inputs, training):
        x = self.embedding(inputs)
        x_1 = self.bigram(x)
        x_1 = self.pool(x_1)
        x_2 = self.trigram(x)
        x_2 = self.pool(x_2)
        x_3 = self.fourgram(x)
        x_3 = self.pool(x_3)
        merged = tf.concat([x_1, x_2, x_3], axis=-1)
        merged = self.dense_1(merged)
        merged = self.dropout(merged, training)
        output = self.last_dense(merged)
        return output

VOCAB_SIZE = len(tokenizer.vocab)
EMB_DIM = 200
NB_FILTERS = 100
FFN_UNITS = 256
NB_CLASSES = 2
DROPOUT_RATE = 0.2
NB_EPOCHS = 5

Dcnn = DCNN(vocab_size=VOCAB_SIZE, emb_dim=EMB_DIM, nb_filters=NB_FILTERS, FFN_units=FFN_UNITS, nb_classes=NB_CLASSES, dropout_rate=DROPOUT_RATE)

if NB_CLASSES == 2:
    Dcnn.compile(loss="binary_crossentropy", optimizer="adam", metrics=["accuracy"])
else:
    Dcnn.compile(loss="sparse_categorical_crossentropy", optimizer="adam", metrics=["sparse_categorical_accuracy"])
```

```
class DCNN(tf.keras.Model):
    def __init__(self, vocab_size, emb_dim=128, nb_filters=50, FFN_units=512, nb_classes=2, dropout_rate=0.1, name="dcnn"):
        super(DCNN, self).__init__(name=name)
        self.embedding = layers.Embedding(vocab_size, emb_dim)
        self.bigram = layers.Conv1D(filters=nb_filters, kernel_size=2, padding="valid", activation="relu")
        self.trigram = layers.Conv1D(filters=nb_filters, kernel_size=3, padding="valid", activation="relu")
        self.fourgram = layers.Conv1D(filters=nb_filters, kernel_size=4, padding="valid", activation="relu")
        self.pool = layers.GlobalMaxPooling1D()
        self.dense_1 = layers.Dense(units=FFN_units, activation="relu")
        self.dropout = layers.Dropout(rate=dropout_rate)
        self.last_dense = layers.Dense(units=1, activation="sigmoid")  # Single output with sigmoid

    def call(self, inputs, training):
        x = self.embedding(inputs)
        x_1 = self.bigram(x)
        x_1 = self.pool(x_1)
        x_2 = self.trigram(x)
        x_2 = self.pool(x_2)
        x_3 = self.fourgram(x)
        x_3 = self.pool(x_3)
        merged = tf.concat([x_1, x_2, x_3], axis=-1)
        merged = self.dense_1(merged)
        merged = self.dropout(merged, training)
        output = self.last_dense(merged)
        return output



# Define the checkpoint path
checkpoint_path = "/content/drive/MyDrive/dataset/"
ckpt = tf.train.Checkpoint(Dcnn=Dcnn)
ckpt_manager = tf.train.CheckpointManager(ckpt, checkpoint_path, max_to_keep=1)

# Restore the latest checkpoint if available
if ckpt_manager.latest_checkpoint:
    ckpt.restore(ckpt_manager.latest_checkpoint)
    print("Latest Checkpoint restored!")

# Custom callback to save model checkpoints
class MyCustomCallback(tf.keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs=None):
        ckpt_manager.save()
        print("Checkpoint saved at {}.".format(checkpoint_path))
```

```
Latest Checkpoint restored!
```

```
# Adjust the model class (DCNN) to use one output neuron with sigmoid activation
class DCNN(tf.keras.Model):
    def __init__(self,
                 vocab_size,
                 emb_dim=128,
                 nb_filters=50,
                 FFN_units=512,
                 nb_classes=2,  # Should be 1 for binary classification
                 dropout_rate=0.1,
                 name="dcnn"):
        super(DCNN, self).__init__(name=name)

        self.embedding = layers.Embedding(vocab_size, emb_dim)
        self.bigram = layers.Conv1D(filters=nb_filters, kernel_size=2, padding="valid", activation="relu")
        self.trigram = layers.Conv1D(filters=nb_filters, kernel_size=3, padding="valid", activation="relu")
        self.fourgram = layers.Conv1D(filters=nb_filters, kernel_size=4, padding="valid", activation="relu")

        self.pool = layers.GlobalMaxPooling1D()
        self.dense_1 = layers.Dense(units=FFN_units, activation="relu")
        self.dropout = layers.Dropout(rate=dropout_rate)

        self.last_dense = layers.Dense(units=1, activation="sigmoid")  # Output layer for binary classification

    def call(self, inputs, training):
        x = self.embedding(inputs)
        x_1 = self.bigram(x)
        x_1 = self.pool(x_1)

        x_2 = self.trigram(x)
        x_2 = self.pool(x_2)

        x_3 = self.fourgram(x)
        x_3 = self.pool(x_3)

        merged = tf.concat([x_1, x_2, x_3], axis=-1)
        merged = self.dense_1(merged)
        merged = self.dropout(merged, training)

        output = self.last_dense(merged)
        return output

# Compile the model
Dcnn = DCNN(vocab_size=VOCAB_SIZE,
           emb_dim=EMB_DIM,
           nb_filters=NB_FILTERS,
           FFN_units=FFN_UNITS,
           nb_classes=1,  # Should be 1 for binary classification
           dropout_rate=DROPOUT_RATE)

Dcnn.compile(loss="binary_crossentropy",
            optimizer="adam",
            metrics=["accuracy"])

# Train the model
Dcnn.fit(train_dataset, epochs=NB_EPOCHS, callbacks=[MyCustomCallback()])
```

```
Epoch 1/5
  40623/Unknown - 4297s 105ms/step - loss: 0.4155 - accuracy: 0.8105Checkpoint saved at /content/drive/MyDrive/dataset/.
40623/40623 [==============================] - 4297s 105ms/step - loss: 0.4155 - accuracy: 0.8105
Epoch 2/5
40623/40623 [==============================] - ETA: 0s - loss: 0.3717 - accuracy: 0.8357Checkpoint saved at /content/drive/MyDrive/dataset/.
40623/40623 [==============================] - 4431s 108ms/step - loss: 0.3717 - accuracy: 0.8357
Epoch 3/5
40623/40623 [==============================] - ETA: 0s - loss: 0.3367 - accuracy: 0.8535Checkpoint saved at /content/drive/MyDrive/dataset/.
40623/40623 [==============================] - 4194s 103ms/step - loss: 0.3367 - accuracy: 0.8535
Epoch 4/5
40623/40623 [==============================] - ETA: 0s - loss: 0.3009 - accuracy: 0.8710Checkpoint saved at /content/drive/MyDrive/dataset/.
40623/40623 [==============================] - 4132s 101ms/step - loss: 0.3009 - accuracy: 0.8710
Epoch 5/5
40623/40623 [==============================] - ETA: 0s - loss: 0.2673 - accuracy: 0.8866Checkpoint saved at /content/drive/MyDrive/dataset/.
40623/40623 [==============================] - 4201s 103ms/step - loss: 0.2673 - accuracy: 0.8866
<keras.src.callbacks.History at 0x7904d8668d60>
```

```
# Evaluate the model
results = Dcnn.evaluate(test_dataset)
print(f"Test Loss: {results[0]}, Test Accuracy: {results[1]}")
```

```
4513/4513 [==============================] - 48s 11ms/step - loss: 0.3943 - accuracy: 0.8317
Test Loss: 0.39425358176231384, Test Accuracy: 0.8316529989242554
```

```
def get_prediction(sentence):
    tokens = encode_sentence(sentence)
    inputs = tf.expand_dims(tokens, 0)
    output = Dcnn(inputs, training=False)
    sentiment = tf.argmax(output, axis=-1).numpy()[0]

    if sentiment == 0:
        print(f"Output of the model: {output}\nPredicted sentiment: negative.")
    elif sentiment == 1:
        print(f"Output of the model: {output}\nPredicted sentiment: positive.")
```

```
get_prediction("This movie was pretty interesting.")
```

```
Output of the model: [[0.9587253]]
Predicted sentiment: negative.
```

```
get_prediction("I'd rather not do that again.")
```

```
Output of the model: [[0.27174208]]
Predicted sentiment: negative.
```

```
get_prediction("I love algorithms")
```

```
Output of the model: [[0.8606998]]
Predicted sentiment: negative.
```

Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.