```
!pip install transformers

import pandas as pd
from google.colab import drive
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from tensorflow.keras import datasets, layers, models
from transformers import AutoTokenizer
from sklearn.model_selection import train_test_split
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer
from nltk.corpus.reader import nltk
nltk.download('stopwords')
from sklearn.metrics import classification_report

drive.mount('/content/drive')

fake_df = pd.read_csv('/content/drive/MyDrive/HLT/Fake.csv')
true_df = pd.read_csv('/content/drive/MyDrive/HLT/True.csv')

fake_df = pd.DataFrame(fake_df).truncate(after=5000)
fake_df['value'] = 0
print(fake_df)
true_df = pd.DataFrame(true_df).truncate(after=5000)
true_df['value'] = 1

df = pd.concat([fake_df, true_df])
print(df)

stopwords = set(stopwords.words('english'))
vectorizer = TfidfVectorizer(stop_words = list(stopwords))

# Split test and train data using 25% of the dataset for validation purposes
x_train, x_test, y_train, y_test = train_test_split(df['text'],
                                                    df['value'], test_size=0.25, sh

tokenizer = AutoTokenizer.from_pretrained('bert-base-cased')
x_train_tokenized = tokenizer(list(x_train), return_tensors = 'np',
                              padding = True)['input_ids']
x_test_tokenized =  tokenizer(list(x_test), return_tensors = 'np',
                              padding = True)['input_ids']
```

```
print(x_train_tokenized)
dim = max(x_train_tokenized.max(), x_test_tokenized.max()) + 1

def vectorize_sequences(sequences, dimension=dim):
    # Create an all-zero matrix of shape (len(sequences), dimension)
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        results[i, sequence] = 1.  # set specific indices of results[i] to 1s
    return results

x_train = vectorize_sequences(x_train_tokenized)
x_test = vectorize_sequences(x_test_tokenized)

# # Our vectorized labels
y_train = np.asarray(y_train).astype('float32')
y_test = np.asarray(y_test).astype('float32')

# create a validation set
x_val_size = int(len(x_train) * 0.2)
partial_x_train = x_train[x_val_size:]
x_val = x_train[:x_val_size]

y_val_size = int(len(y_train) * 0.2)
partial_y_train = y_train[y_val_size:]
y_val = y_train[:y_val_size]

# build the model
model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(dim,)))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

# compile
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])

# train
history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data = (x_val, y_val))

# use sklearn evaluation
```

```
pred = model.predict(x_test)
pred = [1.0 if p>= 0.5 else 0.0 for p in pred]
print(classification_report(y_test, pred))

# use tf evaluation method
losses_and_metrics = model.evaluate(x_test, y_test, batch_size=128)
losses_and_metrics

# plot the training and validation loss
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(loss)+1)

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

# plot the training and validation accuracy
plt.clf()   # clear

acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

#CNN
max_features = 10000

model = models.Sequential()
model.add(layers.Embedding(max_features, 128, input_length = dim))
model.add(layers.Conv1D(32, 7, activation='relu'))
model.add(layers.MaxPooling1D(5))
model.add(layers.Conv1D(32, 7, activation='relu'))
model.add(layers.GlobalMaxPooling1D())
model.add(layers.Dense(1))
```

```
model.summary()
model.compile(optimizer=tf.keras.optimizers.RMSprop(lr=1e-4),
              loss='binary_crossentropy',
              metrics=['accuracy'])

with tf.device('/GPU:0'):
  history = model.fit(partial_x_train,
                      partial_x_train,
                      epochs = 3,
                      batch_size = 128,
                      validation_data = (x_val, y_val))

pred = model.predict(x_test)
pred = [1.0 if p>= 0.5 else 0.0 for p in pred]
print(classification_report(y_test, pred))

losses_and_metrics = model.evaluate(x_test, y_test, batch_size = 128)
losses_and_metrics

# With Embeddings
model = models.Sequential()
model.add(layers.Embedding(max_features, 8, input_length = dim))
model.add(layers.Flatten())
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
model.summary()

with tf.device('/GPU:0'):
  history = model.fit(partial_x_train,
                      partial_x_train,
                      epochs=3,
                      batch_size=32,
                      validation_split=0.2)

pred = model.predict(x_test)
pred = [1.0 if p>= 0.5 else 0.0 for p in pred]
print(classification_report(y_test, pred))

losses_and_metrics = model.evaluate(x_test, y_test, batch_size = 128)
losses_and_metrics
```

```
     embedding_1 (Embedding)     (None, 28206, 8)          80000

     flatten (Flatten)           (None, 225648)            0
```

```
 flatten (Flatten)          (None, 225648)            0

 dense_4 (Dense)            (None, 16)                3610384

 dense_5 (Dense)            (None, 1)                 17

=================================================================
Total params: 3,690,401
Trainable params: 3,690,401
Non-trainable params: 0
_________________________________________________________________
Epoch 1/3
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-1-88ceb8530bc0> in <cell line: 164>()
    163
    164 with tf.device('/GPU:0'):
--> 165   history = model.fit(partial_x_train,
    166                         partial_x_train,
    167                         epochs=3,

                                  1 frames
/usr/local/lib/python3.9/dist-packages/keras/engine/training.py in
tf__train_function(iterator)
     13                 try:
     14                     do_return = True
---> 15                     retval_ =
ag__.converted_call(ag__.ld(step_function), (ag__.ld(self),
ag__.ld(iterator)), None, fscope)
     16                 except:
     17                     do_return = False

ValueError: in user code:

    File "/usr/local/lib/python3.9/dist-packages/keras/engine/training.py",
line 1284, in train_function  *
        return step_function(self, iterator)
    File "/usr/local/lib/python3.9/dist-packages/keras/engine/training.py",
line 1268, in step_function  **
        outputs = model.distribute_strategy.run(run_step, args=(data,))
    File "/usr/local/lib/python3.9/dist-packages/keras/engine/training.py",
line 1249, in run_step  **
        outputs = model.train_step(data)
    File "/usr/local/lib/python3.9/dist-packages/keras/engine/training.py",
line 1051, in train_step
        loss = self.compute_loss(x, y, y_pred, sample_weight)
    File "/usr/local/lib/python3.9/dist-packages/keras/engine/training.py",
line 1109, in compute_loss
        return self.compiled_loss(
    File "/usr/local/lib/python3.9/dist-
packages/keras/engine/compile_utils.py", line 265, in __call__
```

```
packages/keras/engine/compile_utils.py", line 265, in __call__
        loss_value = loss_obj(y_t, y_p, sample_weight=sw)
    File "/usr/local/lib/python3.9/dist-packages/keras/losses.py", line 142,
in __call__
        losses = call_fn(y_true, y_pred)
    File "/usr/local/lib/python3.9/dist-packages/keras/losses.py", line 268,
```

Analysis of different architectures

We start off the classification with a simple sequential model which is able to get a pretty hgih accuracy in a small amount of time. We pass the text through a tokenizer and then vectorize it into arrays of numbers. This process doesn't take too long and the results are fairly clear to see. The second model is a CNN network which has many hidden layers and this takes a lot more time than the previous one and the accuracy is not that high. The last one is with embeddings by manipulating the text, this one took longer than the rest and had the lowest accuracy.