For this assignment I use the dataset of dastfood.csv which can be found here:

https://www.kaggle.com/datasets/ulrikthygepedersen/fastfood-nutrition

```
from google.colab import files
#Uploading data
uploaded = files.upload()
```

Choose Files fastfood.csv

- **fastfood.csv**(text/csv) - 43604 bytes, last modified: 4/8/2023 - 100% done

```
Saving fastfood.csv to fastfood.csv
```

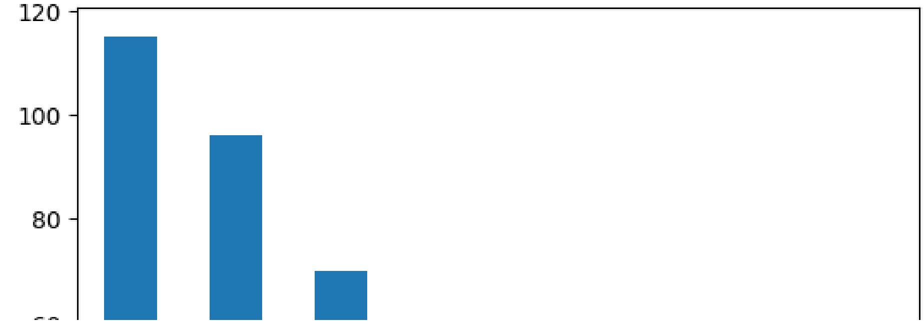
```
import pandas as pd
df = pd.read_csv('fastfood.csv', encoding='latin-1')
df = df[['restaurant', 'item']]
display(df)
```

	restaurant	item
0	Mcdonalds	Artisan Grilled Chicken Sandwich
1	Mcdonalds	Single Bacon Smokehouse Burger
2	Mcdonalds	Double Bacon Smokehouse Burger
3	Mcdonalds	Grilled Bacon Smokehouse Chicken Sandwich
4	Mcdonalds	Crispy Bacon Smokehouse Chicken Sandwich
...	...	...
510	Taco Bell	Spicy Triple Double Crunchwrap
511	Taco Bell	Express Taco Salad w/ Chips
512	Taco Bell	Fiesta Taco Salad-Beef
513	Taco Bell	Fiesta Taco Salad-Chicken
514	Taco Bell	Fiesta Taco Salad-Steak

515 rows × 2 columns

```
print(df['restaurant'].value_counts().plot(kind='bar'))
```

```
Axes(0.125,0.11;0.775x0.77)
```



I choose a dataset that distributes the number of items on the menu from each fast food place. The model should be able to predict the item from the menu from any specific fast food resturant.

```
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras import layers, models, preprocessing
from sklearn.metrics import classification_report
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
import numpy as np

# Set seed for reproducibility
np.random.seed(1234)


i = np.random.rand(len(df)) < 0.8
train = df[i]
test = df[~i]
print("Train data size: ", train.shape)
print("Test data size: ", test.shape)
```

```
Train data size:  (405, 2)
Test data size:  (110, 2)
```

```
# Fit the tokenizer on the training data
tokenizer = Tokenizer()
tokenizer.fit_on_texts(train.item)

X_train = tokenizer.texts_to_matrix(train.item, mode='tfidf')
X_test = tokenizer.texts_to_matrix(test.item, mode='tfidf')

encoder = LabelEncoder()
encoder.fit(train.restaurant)
Y_train = encoder.transform(train.restaurant)
Y_test = encoder.transform(test.restaurant)

Y_train = tf.keras.utils.to_categorical(Y_train, 10)
Y_test = tf.keras.utils.to_categorical(Y_test, 10)

# check shape
print("train shapes:", X_train.shape, Y_train.shape)
print("test shapes:", X_test.shape, Y_test.shape)

scale = StandardScaler()
sc = scale.fit(X_train)
X_train_sc = sc.transform(X_train)
X_test_sc = sc.transform(X_test)
```

```
train shapes: (405, 295) (405, 10)
test shapes: (110, 295) (110, 10)
```

```
# Sequential
model = models.Sequential()
model.add(layers.Dense(30, input_dim=295, kernel_initializer='normal',
                       activation='relu'))
model.add(layers.Dense(20, input_dim=295, kernel_initializer='normal',
                       activation='sigmoid'))
model.add(layers.Dense(10, kernel_initializer='normal', activation='softmax'))

model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

model.summary()
history = model.fit(X_train, Y_train, batch_size=132, epochs=30)
```

```
Model: "sequential_31"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_50 (Dense)            (None, 30)                8880

 dense_51 (Dense)            (None, 20)                620

 dense_52 (Dense)            (None, 10)                210

=================================================================
Total params: 9,710
Trainable params: 9,710
Non-trainable params: 0
_________________________________________________________________
Epoch 1/30
4/4 [==============================] - 1s 6ms/step - loss: 2.2922 - accuracy: 0.1160
```

```
Epoch 2/30
4/4 [==============================] - 0s 6ms/step - loss: 2.2701 - accuracy: 0.1160
Epoch 3/30
4/4 [==============================] - 0s 6ms/step - loss: 2.2503 - accuracy: 0.1679
Epoch 4/30
4/4 [==============================] - 0s 5ms/step - loss: 2.2328 - accuracy: 0.1506
Epoch 5/30
4/4 [==============================] - 0s 5ms/step - loss: 2.2164 - accuracy: 0.2469
Epoch 6/30
4/4 [==============================] - 0s 5ms/step - loss: 2.2001 - accuracy: 0.1852
Epoch 7/30
4/4 [==============================] - 0s 5ms/step - loss: 2.1854 - accuracy: 0.1778
Epoch 8/30
4/4 [==============================] - 0s 7ms/step - loss: 2.1715 - accuracy: 0.1778
Epoch 9/30
4/4 [==============================] - 0s 5ms/step - loss: 2.1582 - accuracy: 0.1778
Epoch 10/30
4/4 [==============================] - 0s 6ms/step - loss: 2.1460 - accuracy: 0.1877
Epoch 11/30
4/4 [==============================] - 0s 5ms/step - loss: 2.1340 - accuracy: 0.2346
Epoch 12/30
4/4 [==============================] - 0s 7ms/step - loss: 2.1220 - accuracy: 0.3358
Epoch 13/30
4/4 [==============================] - 0s 5ms/step - loss: 2.1097 - accuracy: 0.3531
Epoch 14/30
4/4 [==============================] - 0s 5ms/step - loss: 2.0975 - accuracy: 0.3012
Epoch 15/30
4/4 [==============================] - 0s 5ms/step - loss: 2.0851 - accuracy: 0.2272
Epoch 16/30
4/4 [==============================] - 0s 6ms/step - loss: 2.0735 - accuracy: 0.2222
Epoch 17/30
4/4 [==============================] - 0s 7ms/step - loss: 2.0615 - accuracy: 0.2222
Epoch 18/30
4/4 [==============================] - 0s 4ms/step - loss: 2.0499 - accuracy: 0.2222
Epoch 19/30
4/4 [==============================] - 0s 4ms/step - loss: 2.0379 - accuracy: 0.2222
Epoch 20/30
4/4 [==============================] - 0s 7ms/step - loss: 2.0264 - accuracy: 0.2222
Epoch 21/30
4/4 [==============================] - 0s 4ms/step - loss: 2.0148 - accuracy: 0.2222
```

```
# Sequential evaluation of test data
result = model.evaluate(X_test, Y_test, batch_size=132)
print('Accuracy is ', result[1])
```

```
1/1 [==============================] - 1s 532ms/step - loss: 1.9040 - accuracy: 0.2273
Accuracy is  0.22727273404598236
```

```
# CNN
model = models.Sequential()
model.add(layers.Embedding(50000, 64, input_length=295))
model.add(layers.Conv1D(30, 10, activation='relu'))
model.add(layers.MaxPooling1D(5))
model.add(layers.Conv1D(30, 10, activation='relu'))
model.add(layers.GlobalMaxPooling1D())
model.add(layers.Dense(10))

model.compile(optimizer=tf.keras.optimizers.RMSprop(lr=1e-4),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
model.summary()
history = model.fit(X_train, Y_train, epochs=5, batch_size=132, validation_split=0.2)
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimiz
Model: "sequential_32"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_16 (Embedding)    (None, 295, 64)           3200000

 conv1d_24 (Conv1D)          (None, 286, 30)           19230

 max_pooling1d_12 (MaxPoolin  (None, 57, 30)           0
 g1D)

 conv1d_25 (Conv1D)          (None, 48, 30)            9030

 global_max_pooling1d_10 (Gl  (None, 30)               0
 obalMaxPooling1D)

 dense_53 (Dense)            (None, 10)                310

=================================================================
Total params: 3,228,570
Trainable params: 3,228,570
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
3/3 [==============================] - 2s 337ms/step - loss: 6.7084 - accuracy: 0.1327 - val_loss: 16.11
Epoch 2/5
3/3 [==============================] - 1s 243ms/step - loss: 7.4621 - accuracy: 0.1296 - val_loss: 16.11
Epoch 3/5
3/3 [==============================] - 1s 269ms/step - loss: 7.4621 - accuracy: 0.1296 - val_loss: 16.11
Epoch 4/5
3/3 [==============================] - 1s 233ms/step - loss: 7.4621 - accuracy: 0.1296 - val_loss: 16.11
Epoch 5/5
3/3 [==============================] - 1s 253ms/step - loss: 7.4621 - accuracy: 0.1296 - val_loss: 16.11
```

```
# CNN evaluation of test data
result = model.evaluate(X_test, Y_test, batch_size=132)
print('Accuracy is ', result[1])
```

```
1/1 [==============================] - 0s 127ms/step - loss: 9.5243 - accuracy: 0.1000
Accuracy is  0.10000000149011612
```

```
# Embedding
model = models.Sequential()
model.add(layers.Embedding(50000, 16, input_length=295))
model.add(layers.Flatten())
model.add(layers.Dense(20, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))

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

history = model.fit(X_train, Y_train, epochs=10, batch_size=132)
```

```
Model: "sequential_33"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_17 (Embedding)    (None, 295, 16)           800000
```

```
 flatten_8 (Flatten)         (None, 4720)              0

 dense_54 (Dense)            (None, 20)                94420

 dense_55 (Dense)            (None, 10)                210

=================================================================
Total params: 894,630
Trainable params: 894,630
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
4/4 [==============================] - 1s 14ms/step - loss: 0.6502 - accuracy: 0.0790
Epoch 2/10
4/4 [==============================] - 0s 20ms/step - loss: 0.5037 - accuracy: 0.1160
Epoch 3/10
4/4 [==============================] - 0s 14ms/step - loss: 0.4239 - accuracy: 0.1086
Epoch 4/10
4/4 [==============================] - 0s 17ms/step - loss: 0.3817 - accuracy: 0.1160
Epoch 5/10
4/4 [==============================] - 0s 13ms/step - loss: 0.3595 - accuracy: 0.1160
Epoch 6/10
4/4 [==============================] - 0s 13ms/step - loss: 0.3471 - accuracy: 0.1160
Epoch 7/10
4/4 [==============================] - 0s 13ms/step - loss: 0.3404 - accuracy: 0.1160
Epoch 8/10
4/4 [==============================] - 0s 14ms/step - loss: 0.3325 - accuracy: 0.1160
Epoch 9/10
4/4 [==============================] - 0s 17ms/step - loss: 0.3269 - accuracy: 0.1012
Epoch 10/10
4/4 [==============================] - 0s 11ms/step - loss: 0.3215 - accuracy: 0.1160
```

```
# First embedding evaluation of test data
result = model.evaluate(X_test, Y_test, batch_size=132)
print('Accuracy is ', result[1])
```

```
1/1 [==============================] - 0s 189ms/step - loss: 0.3193 - accuracy: 0.0909
Accuracy is  0.09090909361839294
```

Overall the best accuracy out of CNN, sequential and embedding is sequential. Sequential has the highest of 0.22 compared to the other one with 0.09 and 0.01.

After looking through all the different model when I was doing this assignment I notice that some of the models needed a larger dataset to be more accurate. In other words, it needed a bigger testing and training set to make have a higher accuracy, otherwise the accuracy would be very low.

The dataset I use was quite small for the use of this assignment. It only had about 550 data within the file.

✓ 0s completed at 10:53 PM