```python
### Sai Gonuguntla, Haniyyah Hamid
# Image Classification with Keras

# using Keras in TensorFlow

import tensorflow as tf
import numpy as np
batch_size = 128
num_classes = 10
epochs = 20

from google.colab import files
uploaded = files.upload()
uploaded2 = files.upload()
```

```
<IPython.core.display.HTML object>

Saving fashion-mnist_train.csv to fashion-mnist_train.csv

<IPython.core.display.HTML object>

Saving fashion-mnist_test.csv to fashion-mnist_test.csv
```

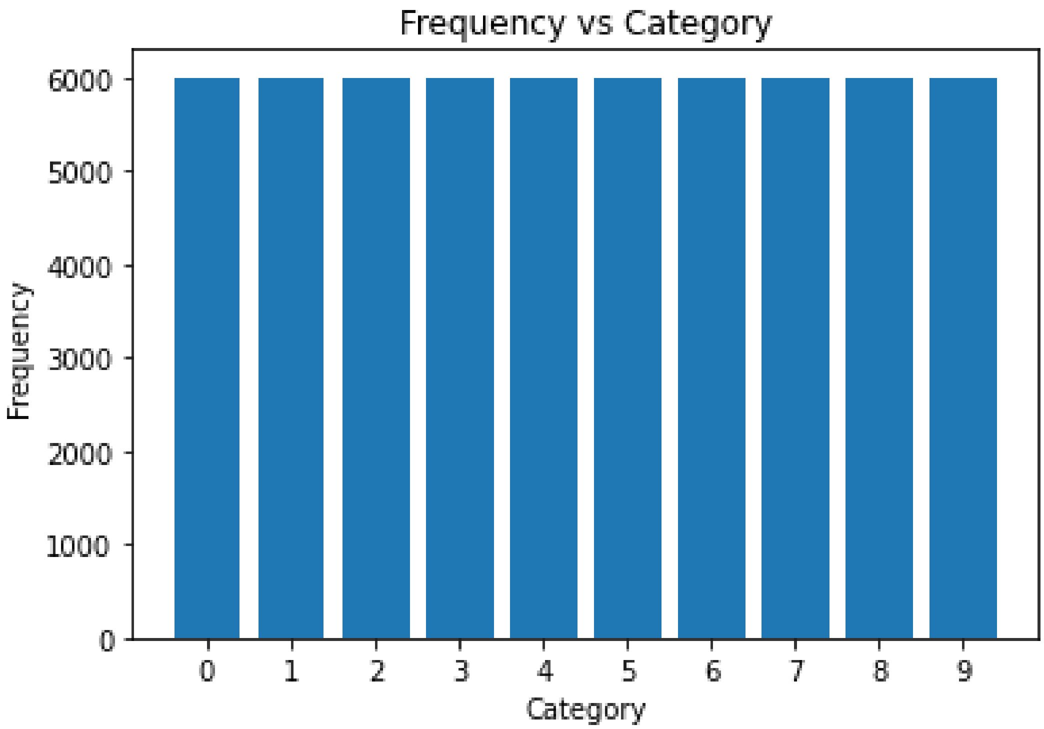
```python
### load the data
import pandas as pd

train_df = pd.read_csv('fashion-mnist_train.csv')
test_df  = pd.read_csv('fashion-mnist_test.csv')

###x_train, x_test = x_train / 255.0, x_test / 255.0

#print(x_train.shape, 'train samples')
#print(x_test.shape, 'test samples')

train_data = np.array(train_df)
test_data = np.array(test_df)

# divides into labels and pixels data
x_train = train_data[:, 1:]
y_train = train_data[:, 0]

x_test = test_data[:, 1:]
y_test = test_data[:, 0]

x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255

x_train = x_train.reshape(60000, 28, 28, 1)
```

```
x_test = x_test.reshape(10000, 28, 28, 1)

print(x_train.shape)

print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')
```

```
(60000, 28, 28, 1)
60000 train samples
10000 test samples
```

```
##catplot
import seaborn as sb
from sklearn import datasets

import pandas as pd


import matplotlib.pyplot as plt






import matplotlib.pyplot as plt

import numpy as np


unique, counts = np.unique(y_train, return_counts=True)
plt.bar(unique, counts)
plt.xticks(unique)
plt.xlabel("Category")
plt.ylabel("Frequency")
plt.title("Frequency vs Category")

#the dataset contains images of items of clothing such as bags, sandals, and sneakers.
#The model should be able to predict what clothing item each category(0-9) represents.
```

```
Text(0.5, 1.0, 'Frequency vs Category')
```




```
# convert lables to categorical
y_train = tf.keras.utils.to_categorical(y_train, num_classes)
y_test = tf.keras.utils.to_categorical(y_test, num_classes)

num_filters = 8
filter_size = 3
pool_size = 2
```

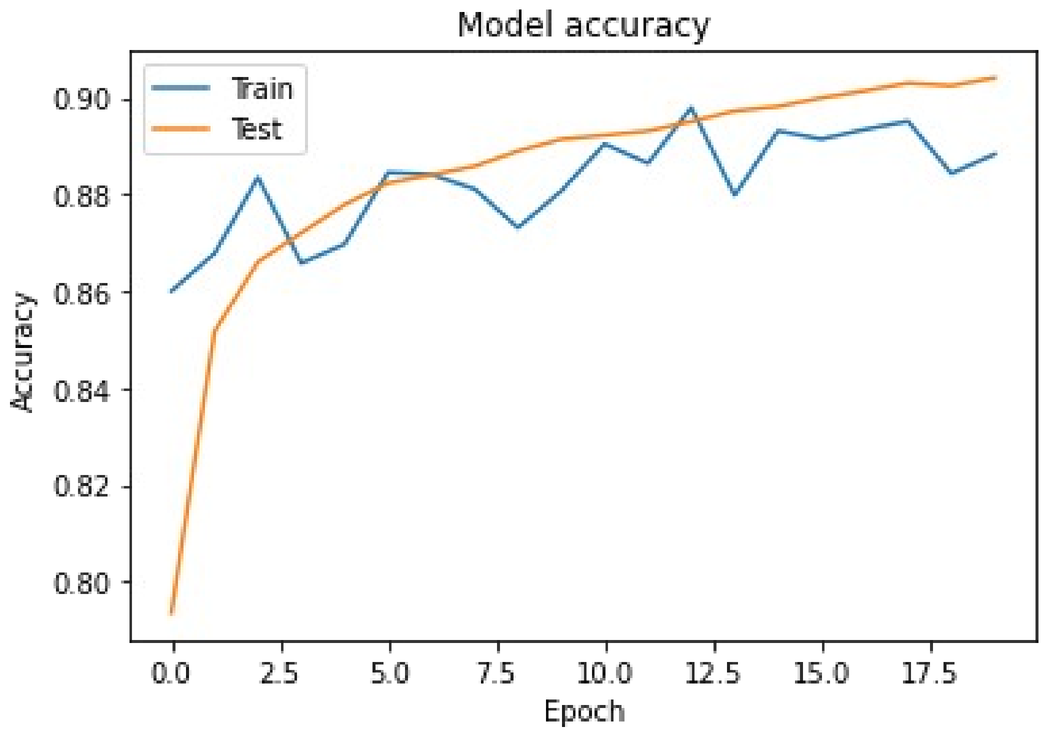
Sequential Model

```
model = tf.keras.models.Sequential([
tf.keras.layers.Flatten(input_shape=(28, 28)),
tf.keras.layers.Dense(512, activation='relu'),
tf.keras.layers.Dropout(0.2),
tf.keras.layers.Dense(512, activation='relu'),
tf.keras.layers.Dropout(0.2),
tf.keras.layers.Dense(num_classes, activation='softmax'),
])
model.summary()
model.compile(loss='categorical_crossentropy',
optimizer='rmsprop',
metrics=['accuracy'])

history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_data=(x_test, y_test))
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 784)               0

 dense (Dense)               (None, 512)               401920

 dropout (Dropout)           (None, 512)               0

 dense_1 (Dense)             (None, 512)               262656

 dropout_1 (Dropout)         (None, 512)               0

 dense_2 (Dense)             (None, 10)                5130

=================================================================
Total params: 669,706
Trainable params: 669,706
Non-trainable params: 0
_________________________________________________________________
Epoch 1/20
469/469 [==============================] - 11s 22ms/step - loss:
0.5625 - accuracy: 0.7936 - val_loss: 0.3913 - val_accuracy: 0.8600
Epoch 2/20
469/469 [==============================] - 10s 21ms/step - loss:
0.4086 - accuracy: 0.8517 - val_loss: 0.3786 - val_accuracy: 0.8678
Epoch 3/20
469/469 [==============================] - 10s 22ms/step - loss:
0.3701 - accuracy: 0.8661 - val_loss: 0.3278 - val_accuracy: 0.8835
Epoch 4/20
469/469 [==============================] - 10s 21ms/step - loss:
0.3549 - accuracy: 0.8720 - val_loss: 0.3657 - val_accuracy: 0.8658
Epoch 5/20
469/469 [==============================] - 10s 21ms/step - loss:
0.3419 - accuracy: 0.8779 - val_loss: 0.3655 - val_accuracy: 0.8697
Epoch 6/20
469/469 [==============================] - 10s 21ms/step - loss:
0.3291 - accuracy: 0.8823 - val_loss: 0.3346 - val_accuracy: 0.8845
Epoch 7/20
469/469 [==============================] - 10s 21ms/step - loss:
0.3228 - accuracy: 0.8840 - val_loss: 0.3256 - val_accuracy: 0.8841
Epoch 8/20
469/469 [==============================] - 10s 21ms/step - loss:
0.3161 - accuracy: 0.8858 - val_loss: 0.3492 - val_accuracy: 0.8811
Epoch 9/20
469/469 [==============================] - 10s 21ms/step - loss:
0.3121 - accuracy: 0.8889 - val_loss: 0.4060 - val_accuracy: 0.8731
Epoch 10/20
469/469 [==============================] - 10s 22ms/step - loss:
```

```
0.3063 - accuracy: 0.8914 - val_loss: 0.3857 - val_accuracy: 0.8807
Epoch 11/20
469/469 [==============================] - 10s 22ms/step - loss:
0.3007 - accuracy: 0.8922 - val_loss: 0.3353 - val_accuracy: 0.8904
Epoch 12/20
469/469 [==============================] - 10s 21ms/step - loss:
0.3024 - accuracy: 0.8931 - val_loss: 0.3534 - val_accuracy: 0.8864
Epoch 13/20
469/469 [==============================] - 11s 24ms/step - loss:
0.2953 - accuracy: 0.8950 - val_loss: 0.3215 - val_accuracy: 0.8979
Epoch 14/20
469/469 [==============================] - 11s 24ms/step - loss:
0.2906 - accuracy: 0.8972 - val_loss: 0.3627 - val_accuracy: 0.8798
Epoch 15/20
469/469 [==============================] - 10s 22ms/step - loss:
0.2850 - accuracy: 0.8981 - val_loss: 0.3719 - val_accuracy: 0.8931
Epoch 16/20
469/469 [==============================] - 10s 22ms/step - loss:
0.2819 - accuracy: 0.8999 - val_loss: 0.3667 - val_accuracy: 0.8914
Epoch 17/20
469/469 [==============================] - 10s 21ms/step - loss:
0.2853 - accuracy: 0.9014 - val_loss: 0.3720 - val_accuracy: 0.8934
Epoch 18/20
469/469 [==============================] - 10s 21ms/step - loss:
0.2785 - accuracy: 0.9030 - val_loss: 0.3587 - val_accuracy: 0.8951
Epoch 19/20
469/469 [==============================] - 10s 21ms/step - loss:
0.2772 - accuracy: 0.9024 - val_loss: 0.3981 - val_accuracy: 0.8843
Epoch 20/20
469/469 [==============================] - 10s 21ms/step - loss:
0.2757 - accuracy: 0.9041 - val_loss: 0.4129 - val_accuracy: 0.8883
```

```python
import matplotlib.pyplot as plt

# Plot training & validation accuracy values
plt.plot(history.history['val_accuracy'])
plt.plot(history.history['accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```




```
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
Test loss: 0.4128842353820801
Test accuracy: 0.8883000016212463
```

CNN

```
model = tf.keras.models.Sequential(
    [
        tf.keras.Input(shape=(28,28,1)),
        tf.keras.layers.Conv2D(32, kernel_size=(3, 3),
activation="relu"),
        tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
        tf.keras.layers.Conv2D(64, kernel_size=(3, 3),
activation="relu"),
        tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dropout(0.5),
        tf.keras.layers.Dense(num_classes, activation="softmax"),
    ]
)

model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
```

```
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320

 max_pooling2d (MaxPooling2D  (None, 13, 13, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 11, 11, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 5, 5, 64)         0
 2D)

 flatten_1 (Flatten)         (None, 1600)              0

 dropout_2 (Dropout)         (None, 1600)              0

 dense_3 (Dense)             (None, 10)                16010

=================================================================
Total params: 34,826
Trainable params: 34,826
Non-trainable params: 0
_________________________________________________________________
```

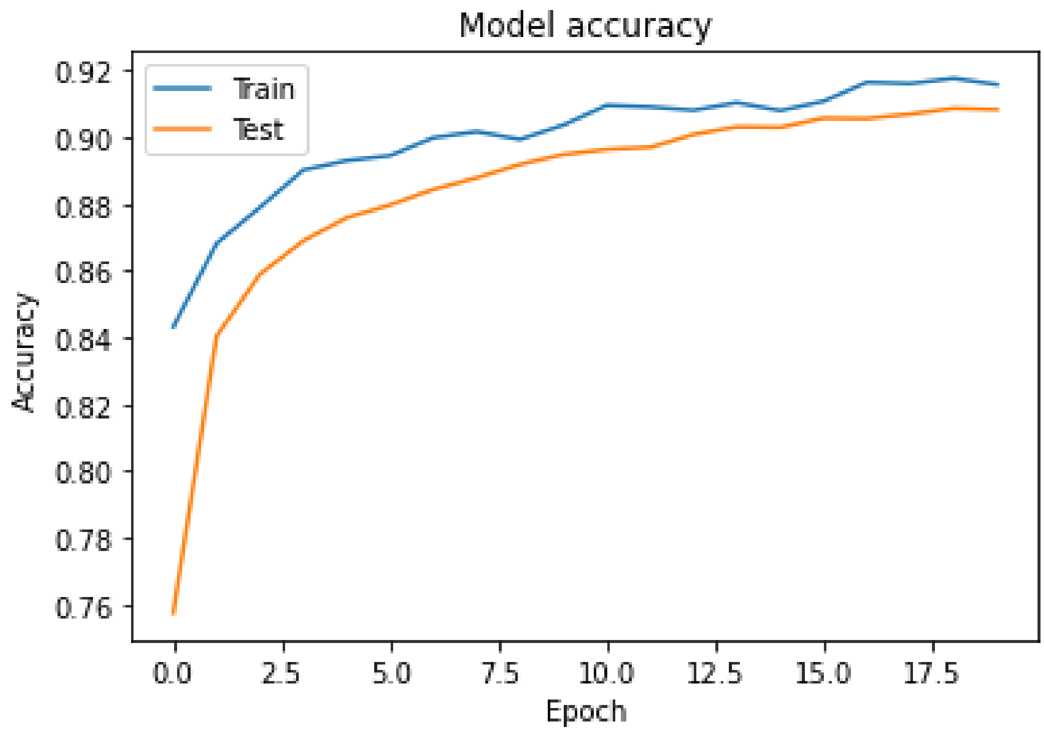
```
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_data=(x_test, y_test))
```

```
Epoch 1/20
469/469 [==============================] - 57s 121ms/step - loss:
0.6720 - accuracy: 0.7575 - val_loss: 0.4370 - val_accuracy: 0.8432
Epoch 2/20
469/469 [==============================] - 54s 115ms/step - loss:
0.4387 - accuracy: 0.8404 - val_loss: 0.3702 - val_accuracy: 0.8682
Epoch 3/20
469/469 [==============================] - 54s 114ms/step - loss:
0.3934 - accuracy: 0.8590 - val_loss: 0.3349 - val_accuracy: 0.8790
Epoch 4/20
469/469 [==============================] - 56s 120ms/step - loss:
0.3649 - accuracy: 0.8690 - val_loss: 0.3121 - val_accuracy: 0.8902
Epoch 5/20
469/469 [==============================] - 54s 114ms/step - loss:
0.3459 - accuracy: 0.8759 - val_loss: 0.3014 - val_accuracy: 0.8930
Epoch 6/20
469/469 [==============================] - 55s 118ms/step - loss:
```

```
0.3335 - accuracy: 0.8797 - val_loss: 0.2894 - val_accuracy: 0.8945
Epoch 7/20
469/469 [==============================] - 55s 117ms/step - loss:
0.3168 - accuracy: 0.8844 - val_loss: 0.2774 - val_accuracy: 0.8999
Epoch 8/20
469/469 [==============================] - 53s 114ms/step - loss:
0.3082 - accuracy: 0.8879 - val_loss: 0.2710 - val_accuracy: 0.9017
Epoch 9/20
469/469 [==============================] - 56s 120ms/step - loss:
0.2989 - accuracy: 0.8919 - val_loss: 0.2703 - val_accuracy: 0.8994
Epoch 10/20
469/469 [==============================] - 54s 115ms/step - loss:
0.2902 - accuracy: 0.8949 - val_loss: 0.2610 - val_accuracy: 0.9037
Epoch 11/20
469/469 [==============================] - 54s 114ms/step - loss:
0.2854 - accuracy: 0.8964 - val_loss: 0.2509 - val_accuracy: 0.9096
Epoch 12/20
469/469 [==============================] - 56s 119ms/step - loss:
0.2813 - accuracy: 0.8970 - val_loss: 0.2486 - val_accuracy: 0.9091
Epoch 13/20
469/469 [==============================] - 54s 114ms/step - loss:
0.2720 - accuracy: 0.9009 - val_loss: 0.2516 - val_accuracy: 0.9081
Epoch 14/20
469/469 [==============================] - 53s 114ms/step - loss:
0.2678 - accuracy: 0.9032 - val_loss: 0.2445 - val_accuracy: 0.9104
Epoch 15/20
469/469 [==============================] - 56s 119ms/step - loss:
0.2657 - accuracy: 0.9030 - val_loss: 0.2467 - val_accuracy: 0.9080
Epoch 16/20
469/469 [==============================] - 53s 113ms/step - loss:
0.2601 - accuracy: 0.9057 - val_loss: 0.2393 - val_accuracy: 0.9108
Epoch 17/20
469/469 [==============================] - 53s 113ms/step - loss:
0.2582 - accuracy: 0.9056 - val_loss: 0.2312 - val_accuracy: 0.9165
Epoch 18/20
469/469 [==============================] - 53s 113ms/step - loss:
0.2541 - accuracy: 0.9069 - val_loss: 0.2322 - val_accuracy: 0.9161
Epoch 19/20
469/469 [==============================] - 56s 120ms/step - loss:
0.2502 - accuracy: 0.9087 - val_loss: 0.2289 - val_accuracy: 0.9177
Epoch 20/20
469/469 [==============================] - 53s 114ms/step - loss:
0.2499 - accuracy: 0.9083 - val_loss: 0.2342 - val_accuracy: 0.9158
```

```
import matplotlib.pyplot as plt

# Plot training & validation accuracy values
plt.plot(history.history['val_accuracy'])
plt.plot(history.history['accuracy'])
plt.title('Model accuracy')
```

```
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```




```
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
Test loss: 0.2341943383216858
Test accuracy: 0.9157999753952026
```

Simple RNN

Processing single data points

```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models, preprocessing

max_features = 10000
maxlen = 500
batch_size = 128

# load the data
train_df = pd.read_csv('fashion-mnist_train.csv')
test_df  = pd.read_csv('fashion-mnist_test.csv')

train_data = np.array(train_df)
```

```
test_data = np.array(test_df)

# divides into labels and pixels data
x_train = train_data[:, 1:] #train_data
y_train = train_data[:, 0]  #train_labels

x_test = test_data[:, 1:]   #test_data
y_test = test_data[:, 0]    #test_labels

x_train = x_train.astype('float32')
x_test = x_test.astype('float32')

x_train /= 255
x_test /= 255


x_train = x_train.reshape(60000, 28, 28, 1)
x_test = x_test.reshape(10000, 28, 28, 1)


print(x_train.shape)
print(y_train.shape)

print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')

# pad the data to maxlen
x_train = preprocessing.sequence.pad_sequences(train_data,
maxlen=maxlen)
x_test = preprocessing.sequence.pad_sequences(test_data,
maxlen=maxlen)

y_train = tf.keras.utils.to_categorical(y_train, num_classes)
y_test = tf.keras.utils.to_categorical(y_test, num_classes)
```

```
(60000, 28, 28, 1)
(60000,)
60000 train samples
10000 test samples
```

```
# build a Sequential model with Embedding and SimpleRNN layers

model = models.Sequential()
model.add(layers.Embedding(max_features, 32))
model.add(layers.SimpleRNN(32))
model.add(layers.Dense(10, activation='softmax'))

model.summary()
```

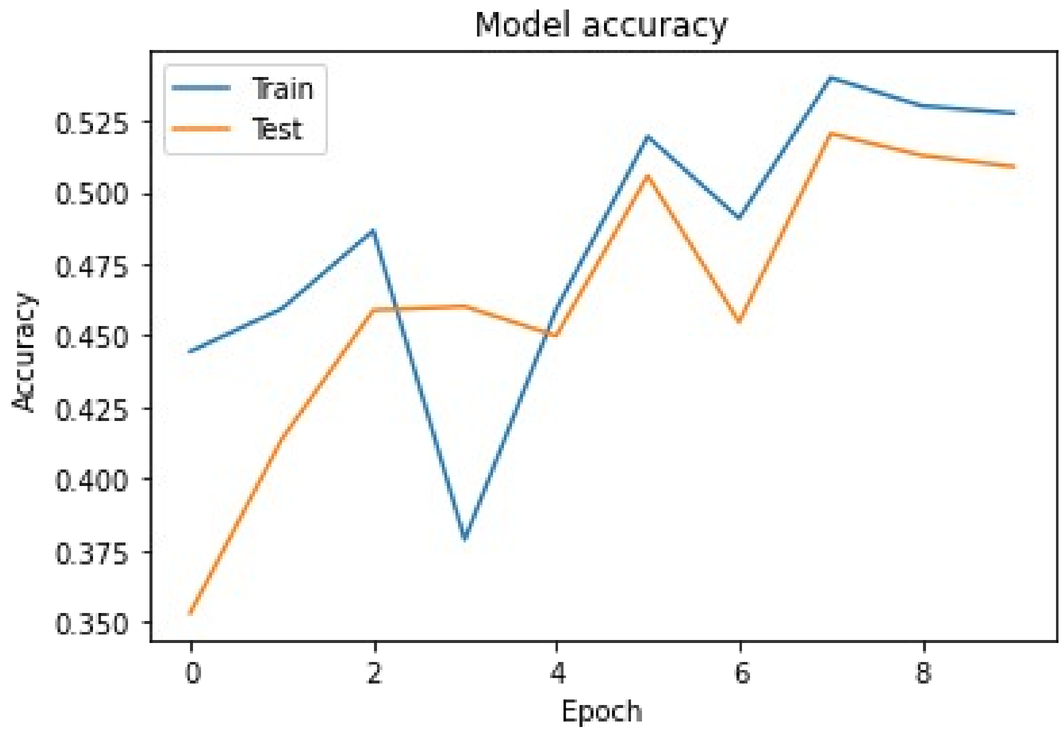
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, None, 32)          320000

 simple_rnn (SimpleRNN)      (None, 32)                2080

 dense_18 (Dense)            (None, 10)                330

=================================================================
Total params: 322,410
Trainable params: 322,410
Non-trainable params: 0
_________________________________________________________________
```

```
# compile

from keras.optimizers import SGD
opt = SGD(learning_rate=0.01)

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

# train

history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=10,
                    verbose=1,
                    validation_split=0.2)
```

```
Epoch 1/10
375/375 [==============================] - 57s 147ms/step - loss:
1.7707 - accuracy: 0.3532 - val_loss: 1.4689 - val_accuracy: 0.4444
Epoch 2/10
375/375 [==============================] - 56s 150ms/step - loss:
1.5236 - accuracy: 0.4138 - val_loss: 1.4249 - val_accuracy: 0.4593
Epoch 3/10
375/375 [==============================] - 57s 153ms/step - loss:
1.3928 - accuracy: 0.4588 - val_loss: 1.3457 - val_accuracy: 0.4866
Epoch 4/10
375/375 [==============================] - 55s 147ms/step - loss:
1.3786 - accuracy: 0.4601 - val_loss: 1.5892 - val_accuracy: 0.3787
Epoch 5/10
375/375 [==============================] - 55s 147ms/step - loss:
1.3834 - accuracy: 0.4499 - val_loss: 1.3922 - val_accuracy: 0.4592
Epoch 6/10
375/375 [==============================] - 56s 150ms/step - loss:
```

```
1.2604 - accuracy: 0.5058 - val_loss: 1.2228 - val_accuracy: 0.5194
Epoch 7/10
375/375 [==============================] - 65s 172ms/step - loss:
1.3992 - accuracy: 0.4546 - val_loss: 1.3087 - val_accuracy: 0.4909
Epoch 8/10
375/375 [==============================] - 56s 149ms/step - loss:
1.2216 - accuracy: 0.5205 - val_loss: 1.1937 - val_accuracy: 0.5399
Epoch 9/10
375/375 [==============================] - 56s 150ms/step - loss:
1.2368 - accuracy: 0.5128 - val_loss: 1.1871 - val_accuracy: 0.5301
Epoch 10/10
375/375 [==============================] - 56s 149ms/step - loss:
1.2561 - accuracy: 0.5089 - val_loss: 1.1865 - val_accuracy: 0.5276
```

```python
# Plot training & validation accuracy values
import matplotlib.pyplot as plt

plt.plot(history.history['val_accuracy'])
plt.plot(history.history['accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()

score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```




```
Test loss: 1.187773346900094
Test accuracy: 0.531000018119812
```

LSTM

```
# the data, split between train and test sets
# build a model with LSTM
model = models.Sequential()
model.add(layers.Embedding(max_features, 32))
model.add(layers.LSTM(32))
model.add(layers.Dense(10, activation='softmax'))

model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, None, 32)          320000

 lstm (LSTM)                 (None, 32)                8320

 dense_1 (Dense)             (None, 10)                330

=================================================================
Total params: 328,650
Trainable params: 328,650
```

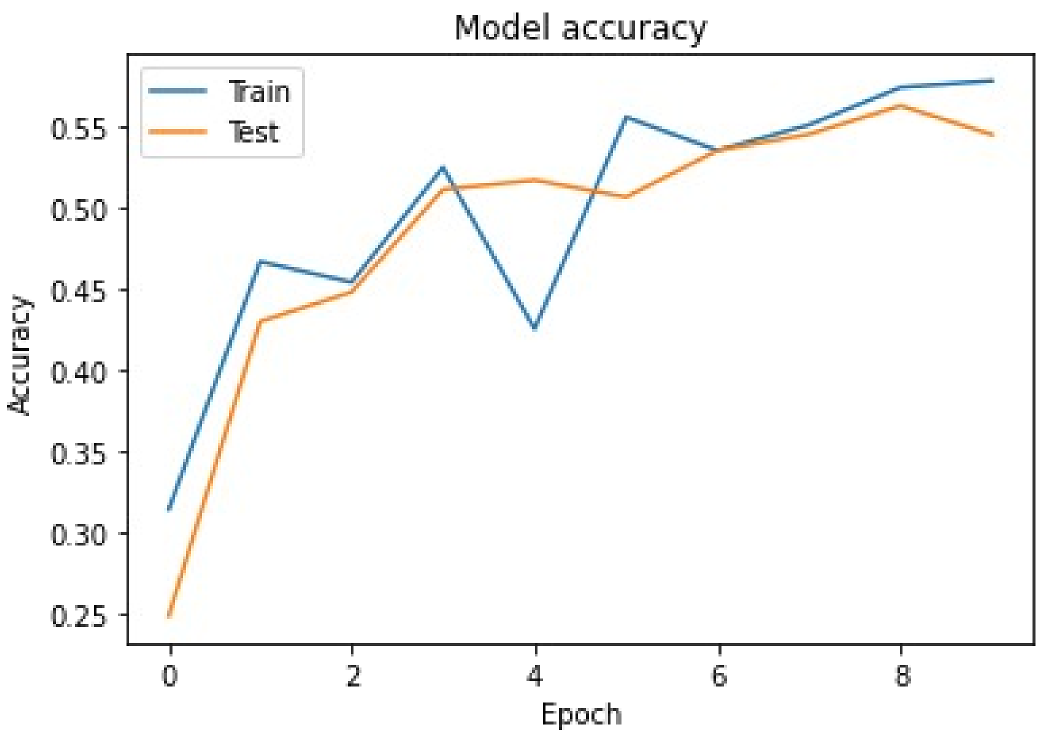
```
Non-trainable params: 0
_________________________________________________________________
```

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

# train
history = model.fit(x_train, y_train,
                    batch_size=128,
                    epochs=10,
                    verbose=1,
                    validation_split=0.2)
```

```
Epoch 1/10
375/375 [==============================] - 111s 292ms/step - loss:
1.8677 - accuracy: 0.2487 - val_loss: 1.6532 - val_accuracy: 0.3143
Epoch 2/10
375/375 [==============================] - 106s 284ms/step - loss:
1.4734 - accuracy: 0.4302 - val_loss: 1.3933 - val_accuracy: 0.4672
Epoch 3/10
375/375 [==============================] - 108s 288ms/step - loss:
1.3739 - accuracy: 0.4484 - val_loss: 1.3132 - val_accuracy: 0.4544
Epoch 4/10
375/375 [==============================] - 107s 286ms/step - loss:
1.2483 - accuracy: 0.5116 - val_loss: 1.1982 - val_accuracy: 0.5253
Epoch 5/10
375/375 [==============================] - 108s 287ms/step - loss:
1.2345 - accuracy: 0.5174 - val_loss: 1.5030 - val_accuracy: 0.4257
Epoch 6/10
375/375 [==============================] - 108s 287ms/step - loss:
1.2848 - accuracy: 0.5070 - val_loss: 1.1642 - val_accuracy: 0.5562
Epoch 7/10
375/375 [==============================] - 107s 285ms/step - loss:
1.2012 - accuracy: 0.5357 - val_loss: 1.2032 - val_accuracy: 0.5357
Epoch 8/10
375/375 [==============================] - 106s 283ms/step - loss:
1.1686 - accuracy: 0.5458 - val_loss: 1.1685 - val_accuracy: 0.5515
Epoch 9/10
375/375 [==============================] - 107s 286ms/step - loss:
1.1266 - accuracy: 0.5633 - val_loss: 1.1172 - val_accuracy: 0.5746
Epoch 10/10
375/375 [==============================] - 108s 289ms/step - loss:
1.1676 - accuracy: 0.5455 - val_loss: 1.0949 - val_accuracy: 0.5787
```

```python
# Plot training & validation accuracy values
import matplotlib.pyplot as plt
```

```
plt.plot(history.history['val_accuracy'])
plt.plot(history.history['accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()

score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```




```
Test loss: 1.1042006015777588
Test accuracy: 0.5716000199317932
```

LSTM has less test loss than the simple RNN (1.1042006015777588 vs 1.545217514038086) and LSTM has better accuracy than the simple RNN (~0.57 vs ~0.42)

Pretrained model and Transfer Learning

```
import matplotlib.pyplot as plt
import numpy as np
import os
import tensorflow as tf

# load dataset
import tensorflow as tf
from tensorflow.keras import datasets, layers, models, preprocessing
```

```
max_features = 10000
maxlen = 500
batch_size = 128

# load the data
train_df = pd.read_csv('fashion-mnist_train.csv')
test_df  = pd.read_csv('fashion-mnist_test.csv')

train_data = np.array(train_df)
test_data = np.array(test_df)

# divides into labels and pixels data
x_train = train_data[:, 1:] #train_data
y_train = train_data[:, 0]  #train_labels

x_test = test_data[:, 1:]   #test_data
y_test = test_data[:, 0]    #test_labels

x_train = x_train.astype('float32')
x_test = x_test.astype('float32')

x_train /= 255
x_test /= 255


x_train = x_train.reshape(60000, 28, 28, 1)
x_test = x_test.reshape(10000, 28, 28, 1)


print(x_train.shape)
print(y_train.shape)

print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')

# pad the data to maxlen
#x_train = preprocessing.sequence.pad_sequences(train_data,
maxlen=maxlen)
#x_test = preprocessing.sequence.pad_sequences(test_data,
maxlen=maxlen)

y_train = tf.keras.utils.to_categorical(y_train, num_classes)
y_test = tf.keras.utils.to_categorical(y_test, num_classes)
```

```
(60000, 28, 28, 1)
(60000,)
60000 train samples
10000 test samples
```

```
# creating base model

base_model = tf.keras.applications.Xception(
    weights=None,  # Load weights pre-trained on ImageNet.
    input_shape=(150, 150, 1),
    include_top=False)

# freeze model
base_model.trainable = False

# create new model on top

inputs = tf.keras.Input(shape=(150, 150, 1))
# We make sure that the base_model is running in inference mode here,
# by passing `training=False`. This is important for fine-tuning, as
you will
# learn in a few paragraphs.
x = base_model(inputs, training=False)
# Convert features of shape `base_model.output_shape[1:]` to vectors
x = tf.keras.layers.GlobalAveragePooling2D()(x)
# A Dense classifier with a single unit (binary classification)
outputs = tf.keras.layers.Dense(1)(x)
model = tf.keras.Model(inputs, outputs)

# train on our dataset

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

# train
history = model.fit(x_train, y_train,
                    batch_size=128,
                    epochs=10,
                    verbose=1,
                    validation_data=(x_test, y_test))
                    #validation_split=0.2)

##model.fit(new_dataset, epochs=20, callbacks=...,
validation_data=...)
```

```
Epoch 1/10

----------------------------------------------------------------------
-----
ValueError                                Traceback (most recent call
last)
<ipython-input-89-5afa80f0e0c1> in <module>
      6
      7 # train
```

```
----> 8 history = model.fit(x_train, y_train,
      9                     batch_size=128,
     10                     epochs=10,

/usr/local/lib/python3.8/dist-packages/keras/utils/traceback_utils.py
in error_handler(*args, **kwargs)
     65     except Exception as e:  # pylint: disable=broad-except
     66       filtered_tb = _process_traceback_frames(e.__traceback__)
---> 67       raise e.with_traceback(filtered_tb) from None
     68     finally:
     69       del filtered_tb

/usr/local/lib/python3.8/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/python3.8/dist-packages/keras/engine/training.py",
line 1051, in train_function  *
        return step_function(self, iterator)
    File
"/usr/local/lib/python3.8/dist-packages/keras/engine/training.py",
line 1040, in step_function  **
        outputs = model.distribute_strategy.run(run_step,
args=(data,))
    File
"/usr/local/lib/python3.8/dist-packages/keras/engine/training.py",
line 1030, in run_step  **
        outputs = model.train_step(data)
    File
"/usr/local/lib/python3.8/dist-packages/keras/engine/training.py",
line 889, in train_step
        y_pred = self(x, training=True)
    File
"/usr/local/lib/python3.8/dist-packages/keras/utils/traceback_utils.py
", line 67, in error_handler
        raise e.with_traceback(filtered_tb) from None
    File
"/usr/local/lib/python3.8/dist-packages/keras/engine/input_spec.py",
line 264, in assert_input_compatibility
        raise ValueError(f'Input {input_index} of layer "{layer_name}"
is '
```

```
    ValueError: Input 0 of layer "model_15" is incompatible with the layer: expected shape=(None, 150, 150, 1), found shape=(None, 28, 28, 1)
```

```
#the sequential model has a loss of .4129 and an accuracy of .888. CNN has  a loss of .234 and an accuracy of .9158
```

Analysis

- Sequential Model: The model had a loss of about 0.41 (higher than CNN), and an accuracy of 0.888, which is close to 1 but lower than the accuracy achieved by CNN.
- CNN: Overall, gave a loss value closer to 0, and an accuracy close to 1, making this model the most suitable for training with the fashion dataset. The
- Simple RNN: The model gave us a loss value > 1, which could mean that the sample size used to train the data was too small to account for the whole dataset.
- LSTM: Long short-term memory. Similar to the simple RNN in regards of the loss value and could be remedied the same way by increasing sample/batch size.
- We find that CNN was the most accurate with the least amount of loss out of these 3 architectures, while the simple RNN was the least accurate and had the most amount of loss. CNN was expected to perform well as it is a powerful architecture for image datasets and the results using this dataset prove this.