Image Classification w/ DL

Lets import all the neccessary libraries and set some variables

In [30]:
```
#All Imports
import os
import numpy as np
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sb
import tensorflow as tf
import keras
from keras.applications import InceptionV3

# Unzip after uploading
# %reset
# !unzip '/content/cards/data.zip'
# !rm -rf '/content/__MACOSX'

#Constants:
batch_size = 40
image_size = (224,224)
num_classes = 4
epochs = 10
```

Create Sets

We are going to try to compare models in their ability to classify playing cards by suit

So we start by creating the train and test sets, and show a few examples of the types of images in the set.

In [26]:
```
#Create train and test sets
train_data, test_data = tf.keras.utils.image_dataset_from_directory('/conten

#Check the inferred classes
class_names = train_data.class_names
print(class_names)

#Found at [https://www.tensorflow.org/tutorials/images/transfer_learning]
plt.figure(figsize=(10, 10))
for images, labels in train_data.take(1):
  for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(images[i].numpy().astype("uint8"))
    #print(list((labels[i].numpy().astype('int'))).index(1))
    plt.title(class_names[list((labels[i].numpy().astype('int'))).index(1)])
    plt.axis("off")

train_data = train_data.cache().shuffle(1000).prefetch(buffer_size=tf.data.A
test_data = test_data.cache().prefetch(buffer_size=tf.data.AUTOTUNE)
```

```
Found 8024 files belonging to 4 classes.
Using 6420 files for training.
Using 1604 files for validation.
['Club', 'Diamond', 'Heart', 'Spade']
```

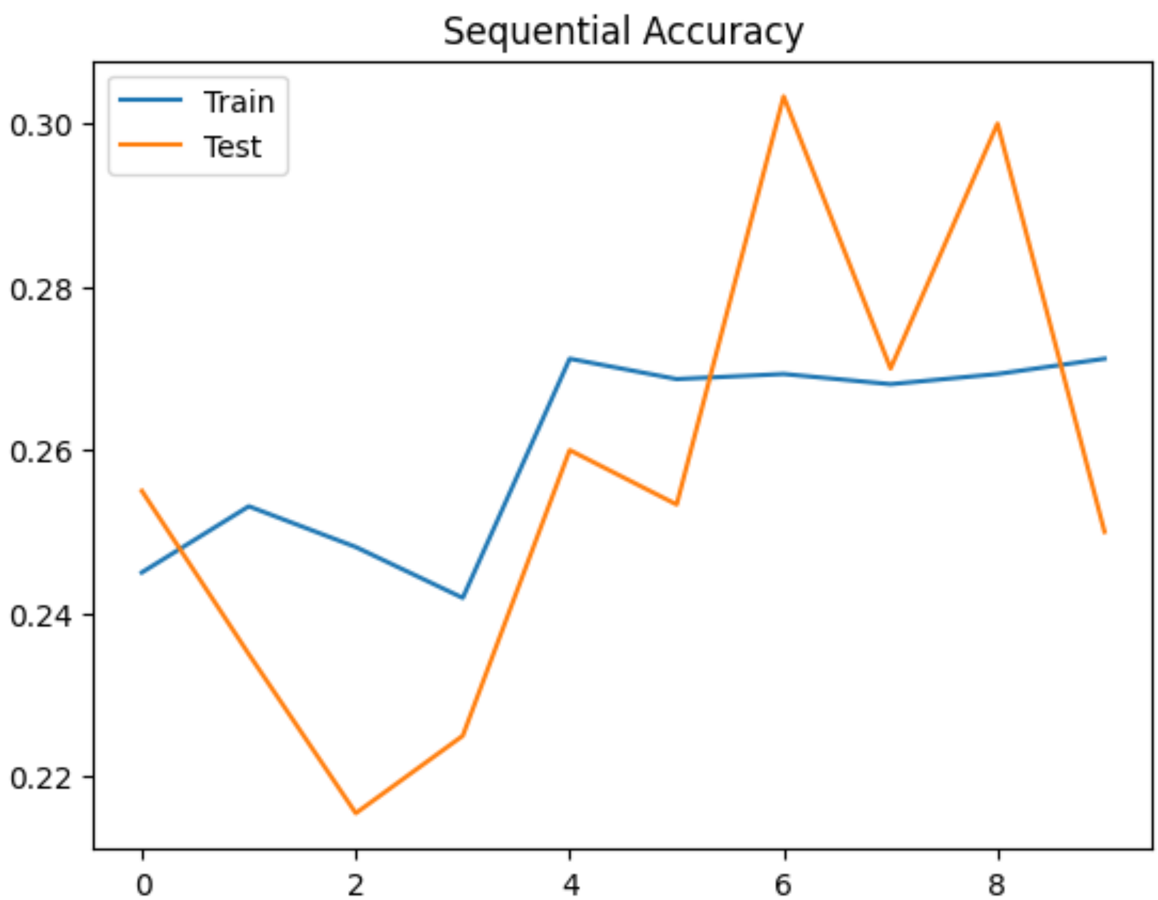
Sequential Model

This is the a normal Sequential Model

In [4]:
```
sequential = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(224,224,3)),
    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')
])

sequential.compile(
    loss = 'categorical_crossentropy',
    optimizer = 'rmsprop',
    metrics = ['accuracy']
)

sequential.summary()

history_sequential = sequential.fit(
    train_data,
    epochs = epochs,
    steps_per_epoch = 15,
    batch_size = batch_size,
    validation_data = test_data
)
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 flatten_1 (Flatten)         (None, 150528)            0         
                                                                 
 dense_3 (Dense)             (None, 512)               77070848  
                                                                 
 dropout_2 (Dropout)         (None, 512)               0         
                                                                 
 dense_4 (Dense)             (None, 512)               262656    
                                                                 
 dropout_3 (Dropout)         (None, 512)               0         
                                                                 
 dense_5 (Dense)             (None, 4)                 2052      
                                                                 
=================================================================
Total params: 77,335,556
Trainable params: 77,335,556
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
15/15 [==============================] - 25s 2s/step - loss: 69479.0234 - ac
curacy: 0.2550 - val_loss: 16944.3984 - val_accuracy: 0.2450
Epoch 2/10
15/15 [==============================] - 25s 2s/step - loss: 5222.2573 - acc
uracy: 0.2350 - val_loss: 13.5702 - val_accuracy: 0.2531
Epoch 3/10
15/15 [==============================] - 24s 2s/step - loss: 28.7262 - accur
acy: 0.2155 - val_loss: 3.1651 - val_accuracy: 0.2481
Epoch 4/10
15/15 [==============================] - 28s 2s/step - loss: 6.2252 - accura
cy: 0.2250 - val_loss: 2.9856 - val_accuracy: 0.2419
Epoch 5/10
15/15 [==============================] - 27s 2s/step - loss: 2.5171 - accura
cy: 0.2600 - val_loss: 4.1997 - val_accuracy: 0.2712
Epoch 6/10
15/15 [==============================] - 25s 2s/step - loss: 18.6030 - accur
acy: 0.2533 - val_loss: 4.5738 - val_accuracy: 0.2687
Epoch 7/10
15/15 [==============================] - 28s 2s/step - loss: 3.3126 - accura
cy: 0.3033 - val_loss: 2.8934 - val_accuracy: 0.2693
Epoch 8/10
15/15 [==============================] - 30s 2s/step - loss: 8.0784 - accura
cy: 0.2700 - val_loss: 2.4039 - val_accuracy: 0.2681
Epoch 9/10
15/15 [==============================] - 29s 2s/step - loss: 1.6194 - accura
cy: 0.3000 - val_loss: 2.3869 - val_accuracy: 0.2693
Epoch 10/10
15/15 [==============================] - 25s 2s/step - loss: 38.6142 - accur
acy: 0.2500 - val_loss: 1.4601 - val_accuracy: 0.2712
```

In [5]:
```
plt.plot(history_sequential.history['val_accuracy'])
plt.plot(history_sequential.history['accuracy'])
plt.title('Sequential Accuracy')
plt.legend(['Train', 'Test'])
plt.show()
```



The fact that accuracy goes up and down means the learning rate is too high.

CNN Model

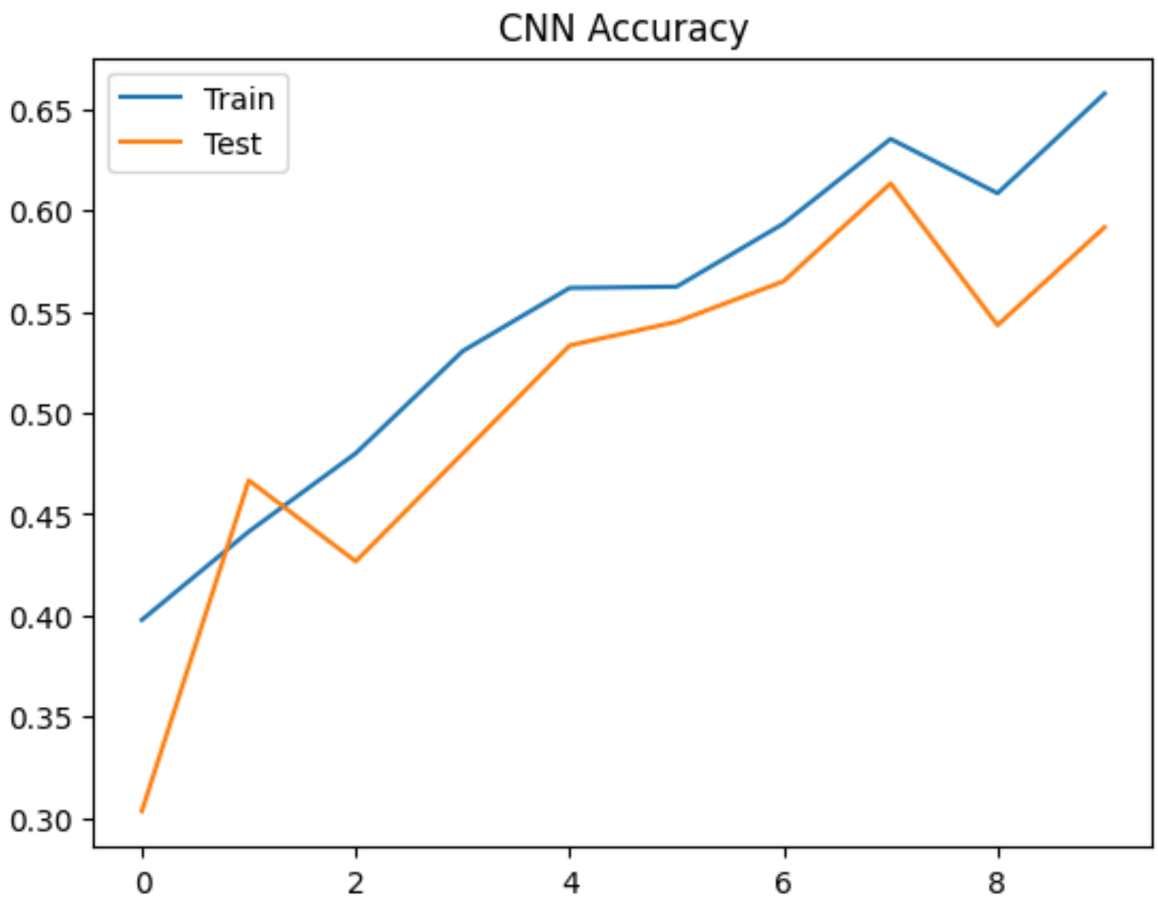
In [8]:
```
cnn = tf.keras.models.Sequential([
    tf.keras.Input(shape=(224,224,3)),
    tf.keras.layers.Conv2D(32, kernel_size=(3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(pool_size=(2,2)),
    tf.keras.layers.Conv2D(32, kernel_size=(3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(pool_size=(2,2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(num_classes, activation='softmax')
])

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

cnn.summary()

history_cnn = cnn.fit(
    train_data,
    epochs = epochs,
    steps_per_epoch = 15,
    batch_size = batch_size,
    validation_data = test_data
)
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_4 (Conv2D)           (None, 222, 222, 32)      896

 max_pooling2d_4 (MaxPooling  (None, 111, 111, 32)     0
 2D)

 conv2d_5 (Conv2D)           (None, 109, 109, 32)      9248

 max_pooling2d_5 (MaxPooling  (None, 54, 54, 32)       0
 2D)

 flatten_4 (Flatten)         (None, 93312)             0

 dropout_6 (Dropout)         (None, 93312)             0

 dense_8 (Dense)             (None, 4)                 373252

=================================================================
Total params: 383,396
Trainable params: 383,396
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
```

```
15/15 [==============================] - 86s 6s/step - loss: 299.4039 - accu
racy: 0.3033 - val_loss: 6.7360 - val_accuracy: 0.3978
Epoch 2/10
15/15 [==============================] - 87s 6s/step - loss: 4.1345 - accura
cy: 0.4667 - val_loss: 1.8878 - val_accuracy: 0.4414
Epoch 3/10
15/15 [==============================] - 85s 6s/step - loss: 1.8325 - accura
cy: 0.4267 - val_loss: 1.4062 - val_accuracy: 0.4800
Epoch 4/10
15/15 [==============================] - 77s 5s/step - loss: 1.4700 - accura
cy: 0.4800 - val_loss: 1.2492 - val_accuracy: 0.5305
Epoch 5/10
15/15 [==============================] - 87s 6s/step - loss: 1.2388 - accura
cy: 0.5333 - val_loss: 1.1473 - val_accuracy: 0.5617
Epoch 6/10
15/15 [==============================] - 86s 6s/step - loss: 1.1681 - accura
cy: 0.5450 - val_loss: 1.0635 - val_accuracy: 0.5623
Epoch 7/10
15/15 [==============================] - 76s 5s/step - loss: 1.1208 - accura
cy: 0.5650 - val_loss: 1.0169 - val_accuracy: 0.5935
Epoch 8/10
15/15 [==============================] - 89s 6s/step - loss: 1.0622 - accura
cy: 0.6133 - val_loss: 0.9411 - val_accuracy: 0.6353
Epoch 9/10
15/15 [==============================] - 84s 6s/step - loss: 1.1167 - accura
cy: 0.5433 - val_loss: 0.9737 - val_accuracy: 0.6085
Epoch 10/10
15/15 [==============================] - 89s 6s/step - loss: 0.9859 - accura
cy: 0.5917 - val_loss: 0.9077 - val_accuracy: 0.6577
```

In [36]:

```python
plt.plot(history_cnn.history['val_accuracy'])
plt.plot(history_cnn.history['accuracy'])
plt.title('CNN Accuracy')
plt.legend(['Train', 'Test'])
plt.show()
```



Pre-trained Model

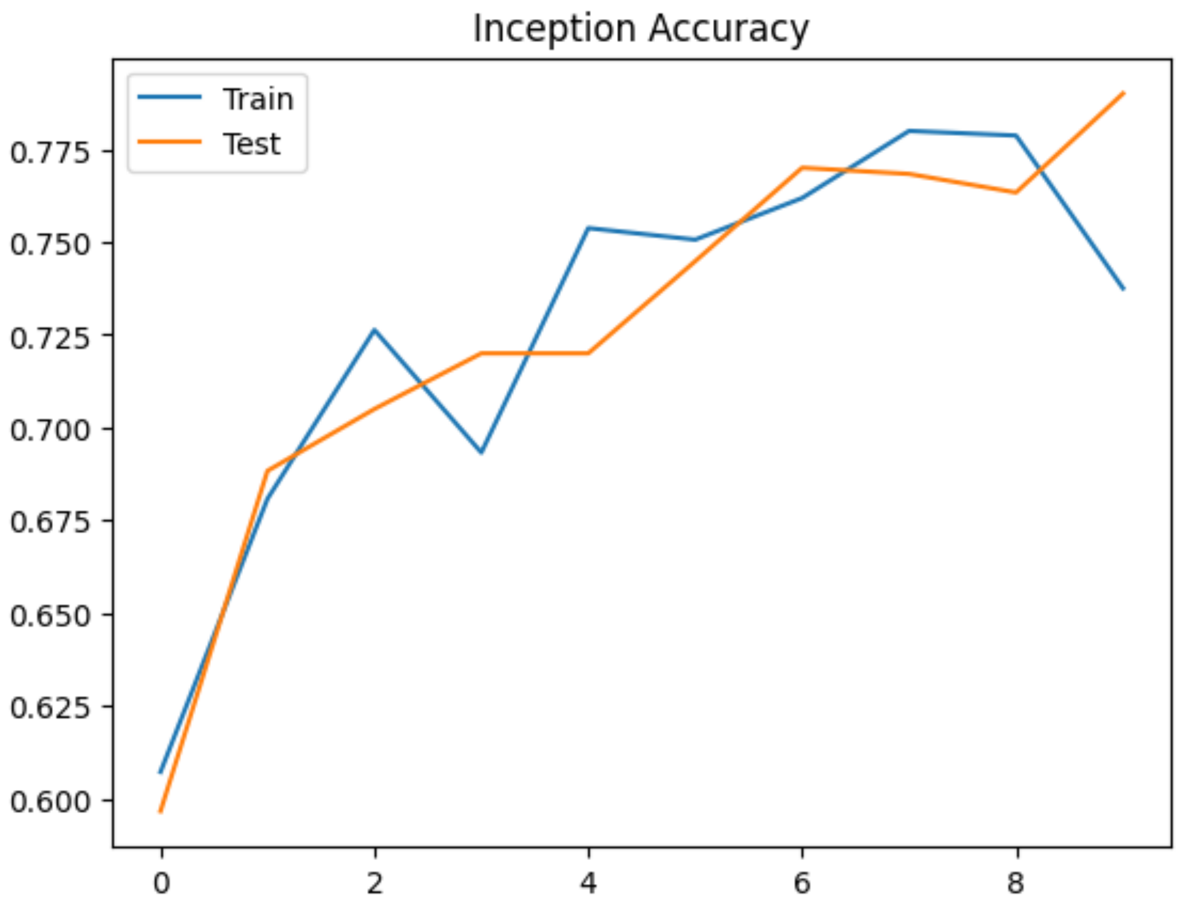
In [34]:
```
pre_trained = InceptionV3(weights = 'imagenet', classes = num_classes, inclu

for layer in pre_trained.layers:
  layer.trainable = False

inception = tf.keras.models.Sequential([
    tf.keras.layers.experimental.preprocessing.Rescaling(1./255, input_shape
    pre_trained,
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(num_classes, activation='softmax')
])

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

inception.summary()

history_inception = inception.fit(
    train_data,
    epochs = epochs,
    steps_per_epoch = 15,
    batch_size = batch_size,
    validation_data = test_data
)
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling (Rescaling)       (None, 224, 224, 3)       0

 inception_v3 (Functional)   (None, 5, 5, 2048)        21802784

 flatten_5 (Flatten)         (None, 51200)             0

 dense_9 (Dense)             (None, 512)               26214912

 batch_normalization_282 (Ba  (None, 512)              2048
 tchNormalization)

 dropout_7 (Dropout)         (None, 512)               0

 dense_10 (Dense)            (None, 4)                 2052

=================================================================
Total params: 48,021,796
Trainable params: 26,217,988
```

```
Non-trainable params: 21,803,808
_________________________________________________________________
Epoch 1/10
15/15 [==============================] - 303s 20s/step - loss: 1.6099 - accu
racy: 0.5967 - val_loss: 4.9819 - val_accuracy: 0.6072
Epoch 2/10
15/15 [==============================] - 284s 20s/step - loss: 1.0382 - accu
racy: 0.6883 - val_loss: 1.8306 - val_accuracy: 0.6808
Epoch 3/10
15/15 [==============================] - 286s 20s/step - loss: 0.7696 - accu
racy: 0.7050 - val_loss: 1.0974 - val_accuracy: 0.7263
Epoch 4/10
15/15 [==============================] - 285s 20s/step - loss: 0.6924 - accu
racy: 0.7200 - val_loss: 1.1951 - val_accuracy: 0.6933
Epoch 5/10
15/15 [==============================] - 285s 20s/step - loss: 0.7110 - accu
racy: 0.7200 - val_loss: 0.7085 - val_accuracy: 0.7537
Epoch 6/10
15/15 [==============================] - 281s 20s/step - loss: 0.6277 - accu
racy: 0.7448 - val_loss: 0.7235 - val_accuracy: 0.7506
Epoch 7/10
15/15 [==============================] - 285s 20s/step - loss: 0.5978 - accu
racy: 0.7700 - val_loss: 0.6816 - val_accuracy: 0.7618
Epoch 8/10
15/15 [==============================] - 284s 20s/step - loss: 0.5678 - accu
racy: 0.7683 - val_loss: 0.5949 - val_accuracy: 0.7799
Epoch 9/10
15/15 [==============================] - 284s 20s/step - loss: 0.6033 - accu
racy: 0.7633 - val_loss: 0.5736 - val_accuracy: 0.7787
Epoch 10/10
15/15 [==============================] - 281s 20s/step - loss: 0.5254 - accu
racy: 0.7900 - val_loss: 0.7932 - val_accuracy: 0.7375
```

In [35]:

```
plt.plot(history_inception.history['val_accuracy'])
plt.plot(history_inception.history['accuracy'])
plt.title('Inception Accuracy')
plt.legend(['Train', 'Test'])
plt.show()
```



I was going to take a picture of one of the playing cards I have at home, and see if the model would accurately classify it, but I ran out of time.

Analysis

The 3 models used have very different levels of accuracy:

- Normal Sequential: ~25% accuracy
- CNN: ~59% accuracy
- Pre-Trained Model: ~79% accuracy

So just by glance, it would seem the Pre-Trained Model is the best, especially getting that accurate with the relatively small dataset given. However it also took significantly longer to train compared to the others. In general it seems that the cost of getting a more accurate model, tends to be an increase in training time.