image-classification

April 20, 2023

Sanjana Jadhav (CS 4375.004)

1 Image Classification using Deep Learning

In this notebook, we will try to classify cards using the following three models: Sequential Model, Convolutional Neural Networks Model, & Pre-Trained (Transfer Learning) Model

Dataset: https://www.kaggle.com/datasets/gpiosenka/cards-image-datasetclassification

Other Sources: Slides, Textbook, Professor Mazidi's Github + other sources listed below

1.0.1 Data Processing + Visualization

import numpy, pandas, & tensorflow

- numpy: numerical computing library
- pandas: data analysis library
- tensorflow: used to build/train neural network models

```
[7]: import numpy as np
     import pandas as pd
     import tensorflow as tf
     import matplotlib.pyplot as plt
```

import train/test data

- use tf to import train/test data
- important parameters: image size (224 x 224), seed=1234 (for reproducibility), batch size is 32

```
[4]: train = tf.keras.preprocessing.image_dataset_from_directory(
         "/kaggle/input/cards-image-datasetclassification/train",
         labels="inferred",
         label_mode="categorical",
         color_mode="rgb",
         image_size=(224, 224),
    seed=1234,
    batch_size=32,
)

test = tf.keras.preprocessing.image_dataset_from_directory(
    "/kaggle/input/cards-image-datasetclassification/test",
    labels="inferred",
    label_mode="categorical",
    color_mode="rgb",
    image_size=(224, 224),
    seed=1234,
    batch_size=32,
)
```

```
Found 7624 files belonging to 53 classes.
Found 265 files belonging to 53 classes.
```

1.0.2 Print/Plot Class Distribution

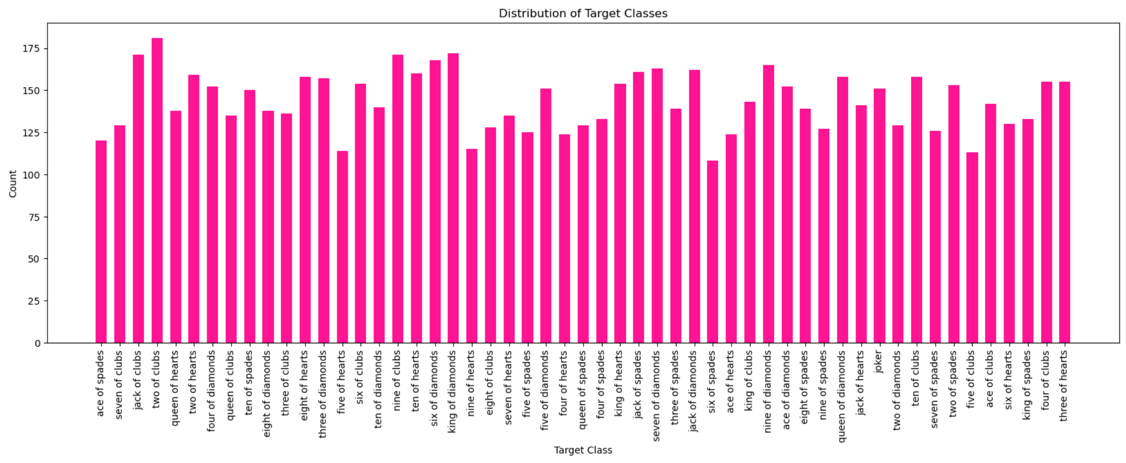
Sources Used:

- https://docs.python.org/3/library/os.html
- https://realpython.com/python-zip-function/
- https://stackoverflow.com/questions/28663856/how-do-i-count-the-occurrence-of-a-certain-item-in-an-ndarray
- https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.bar.html
- https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.figure.html
- https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.xticks.html

[4]:
```
print(train.class_names)

# set number of classes
num_classes = len(train.class_names)
```

```
['ace of clubs', 'ace of diamonds', 'ace of hearts', 'ace of spades', 'eight of
clubs', 'eight of diamonds', 'eight of hearts', 'eight of spades', 'five of
clubs', 'five of diamonds', 'five of hearts', 'five of spades', 'four of clubs',
'four of diamonds', 'four of hearts', 'four of spades', 'jack of clubs', 'jack
of diamonds', 'jack of hearts', 'jack of spades', 'joker', 'king of clubs',
'king of diamonds', 'king of hearts', 'king of spades', 'nine of clubs', 'nine
of diamonds', 'nine of hearts', 'nine of spades', 'queen of clubs', 'queen of
diamonds', 'queen of hearts', 'queen of spades', 'seven of clubs', 'seven of
diamonds', 'seven of hearts', 'seven of spades', 'six of clubs', 'six of
diamonds', 'six of hearts', 'six of spades', 'ten of clubs', 'ten of diamonds',
'ten of hearts', 'ten of spades', 'three of clubs', 'three of diamonds', 'three
of hearts', 'three of spades', 'two of clubs', 'two of diamonds', 'two of
hearts', 'two of spades']
```

[12]:
```python
import os

# Get the class names and indices
class_names = os.listdir("/kaggle/input/cards-image-datasetclassification/
 ↪train")
class_indices = dict(zip(class_names, range(len(class_names))))

# Get class counts
class_counts = np.zeros(len(class_names))
for images, labels in train:
    class_counts += np.sum(labels.numpy(), axis=0)

# Increase size of graph for readibility
plt.figure(figsize=(20, 6))

# Create a bar graph
plt.bar(class_names, class_counts, width=0.6, align='center', color='deeppink')
plt.title('Distribution of Target Classes')
plt.xlabel('Target Class')
plt.ylabel('Count')
plt.xticks(rotation=90)

# Print graph
plt.show()
```



1.0.3 Sequential Model

- input layer: image size is 224 x 224 and channels = 3
- rescale the image between [0,1]
- flattens tensor into 1d array
- create a dense layer with 256 neurons and relu activation

- drop (0.5) to prevent overfitting
- create another dense layer with 128 neurons and relu activation
- drop (0.5) to prevent overfitting
- create another a dense layer with 53 neurons (num_classes) and softmax activation (used for multi-class classification to output probabilities)

[5]:
```python
model = tf.keras.Sequential([
    tf.keras.layers.InputLayer(input_shape=(224, 224, 3)),
    tf.keras.layers.Rescaling(1./255),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(num_classes, activation='softmax')
])
```

Compile/Train Model

[6]:
```python
# Compile Model
# Loss: categorical_crossentropy (used for multi-class classification for␣
 ↪labels that must fit in one class)
# Optimizer: rmsprop
# Metrics: Accuracy
model.compile(loss='categorical_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])

# Train model
# Train model using the training data, 10 epochs, a batch size of 32, and use␣
 ↪the test data
history = model.fit(train,
                    epochs=10,
                    batch_size=32,
                    validation_data=test)
```

```
Epoch 1/10
239/239 [==============================] - 96s 397ms/step - loss: 13.4753 -
accuracy: 0.0195 - val_loss: 3.9795 - val_accuracy: 0.0189
Epoch 2/10
239/239 [==============================] - 94s 392ms/step - loss: 4.0556 -
accuracy: 0.0215 - val_loss: 3.9728 - val_accuracy: 0.0189
Epoch 3/10
239/239 [==============================] - 92s 386ms/step - loss: 3.9700 -
accuracy: 0.0218 - val_loss: 3.9742 - val_accuracy: 0.0189
Epoch 4/10
239/239 [==============================] - 93s 388ms/step - loss: 3.9658 -
accuracy: 0.0237 - val_loss: 3.9751 - val_accuracy: 0.0189
```

```
Epoch 5/10
239/239 [==============================] - 94s 394ms/step - loss: 3.9655 -
accuracy: 0.0241 - val_loss: 3.9760 - val_accuracy: 0.0189
Epoch 6/10
239/239 [==============================] - 94s 393ms/step - loss: 3.9643 -
accuracy: 0.0226 - val_loss: 3.9754 - val_accuracy: 0.0189
Epoch 7/10
239/239 [==============================] - 95s 396ms/step - loss: 3.9642 -
accuracy: 0.0239 - val_loss: 3.9756 - val_accuracy: 0.0189
Epoch 8/10
239/239 [==============================] - 94s 392ms/step - loss: 3.9643 -
accuracy: 0.0237 - val_loss: 3.9767 - val_accuracy: 0.0189
Epoch 9/10
239/239 [==============================] - 93s 390ms/step - loss: 3.9653 -
accuracy: 0.0236 - val_loss: 3.9789 - val_accuracy: 0.0189
Epoch 10/10
239/239 [==============================] - 95s 397ms/step - loss: 3.9633 -
accuracy: 0.0240 - val_loss: 3.9795 - val_accuracy: 0.0189
```

Summary

- rescale layer
- flatten layer
- dense layer (256 neurons and 38535424 trainable parameters)
- dropout layer
- dense layer (128 neurons and 32896 trainable parameters)
- dropout later
- dense layer (53 neurons and 6837 trainable parameters)

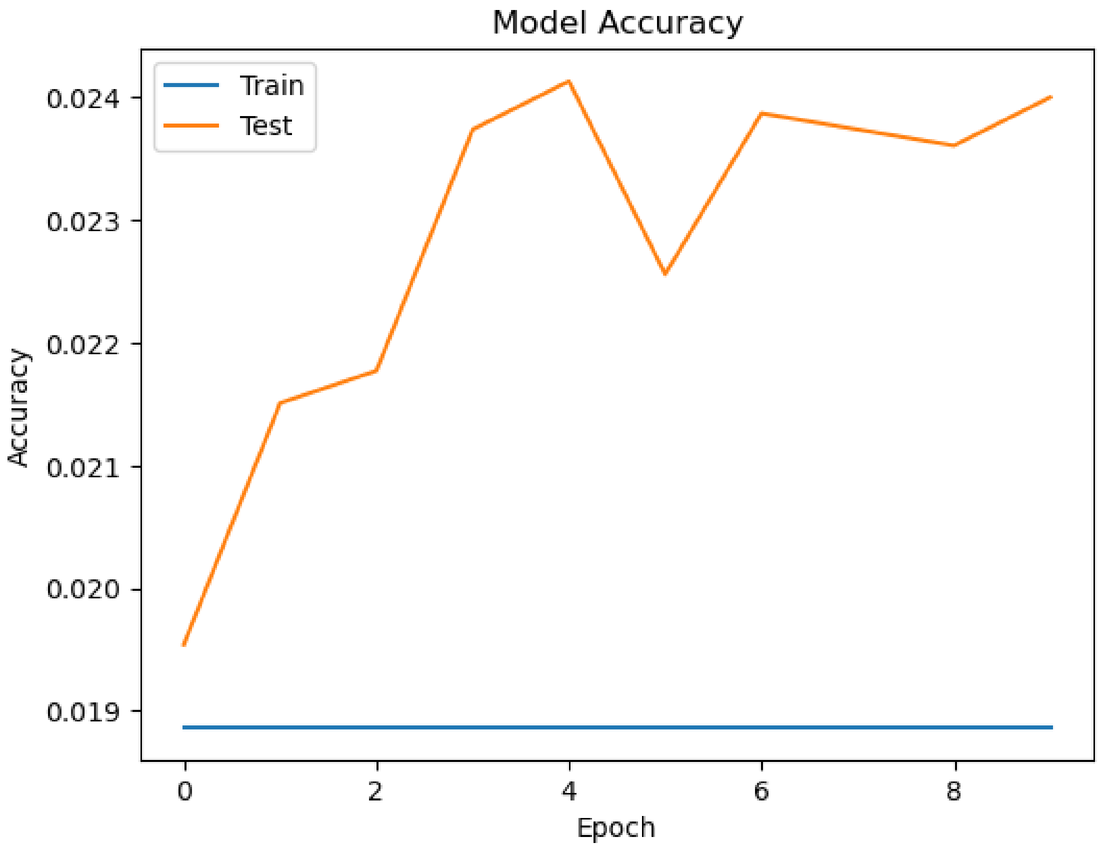
[17]:
```
model.summary()
```

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

 flatten (Flatten)           (None, 150528)            0

 dense (Dense)               (None, 256)               38535424

 dropout (Dropout)           (None, 256)               0

 dense_1 (Dense)             (None, 128)               32896

 dropout_1 (Dropout)         (None, 128)               0

 dense_2 (Dense)             (None, 53)                6837
```

```
=================================================================
Total params: 38,575,157
Trainable params: 38,575,157
Non-trainable params: 0
_________________________________________________________________
```

Plot the Model Metrics for each Epoch

[8]:
```
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(test, verbose=0)
print('Test Loss:', score[0])
print('Test Accuracy:', score[1])
```



```
Test Loss: 3.979480266571045
Test Accuracy: 0.018867924809455 87
```

Analysis Based on the results, it appears that the Sequential model has a low test accuracy of around 2%. This could be due to the fact that sequential models are typically designed for analyzing sequential data and may not be the most suitable choice for image analysis. Furthermore, a more complex architecture such as a CNN may be necessary to accurately extract features and classify the images in this dataset.

1.0.4 Convolutional Neural Network Model

- rescale image: normalize pixels values & image size is 224 x 224 and channels = 3
- create a convolutional layer with 32 filters, same padding and relu activation
- max pooling
- create another convolutional layer with 64 filters, same padding and relu activation
- max pooling
- create another convolutional layer with 128 filters, same padding and relu activation
- max pooling
- flattens tensor into 1d array
- drop (0.5) to prevent overfitting
- create a dense layer with 128 neurons and relu activation
- create a dense layer with 53 neurons and softmax activation (used for multi-class classification to output probabilities)

[9]:
```
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten,␣
 ↪Dense

model2 = tf.keras.Sequential([
    tf.keras.layers.Rescaling(1./255, input_shape=(224, 224, 3)),
    tf.keras.layers.Conv2D(32, 3, padding='same', activation='relu'),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Conv2D(64, 3, padding='same', activation='relu'),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Conv2D(128, 3, padding='same', activation='relu'),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(num_classes, activation="softmax"),
])
```

Compile/Train Model

[10]:
```
# Compile Model
# Loss: categorical_crossentropy (used for multi-class classification for␣
↪labels that must fit in one class)
# Optimizer: adam
# Metrics: Accuracy
model2.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

# Train model
# Train model using the training data, 5 epochs, a batch size of 32, and use␣
↪the test data
history = model2.fit(train,
                    epochs=5,
                    batch_size=32,
                    verbose=1,
                    validation_data=test)
```

```
Epoch 1/5
239/239 [==============================] - 514s 2s/step - loss: 2.8535 -
accuracy: 0.2812 - val_loss: 1.7789 - val_accuracy: 0.4981
Epoch 2/5
239/239 [==============================] - 517s 2s/step - loss: 1.5378 -
accuracy: 0.5815 - val_loss: 1.2408 - val_accuracy: 0.6830
Epoch 3/5
239/239 [==============================] - 511s 2s/step - loss: 0.8144 -
accuracy: 0.7745 - val_loss: 1.2549 - val_accuracy: 0.7472
Epoch 4/5
239/239 [==============================] - 505s 2s/step - loss: 0.4217 -
accuracy: 0.8797 - val_loss: 1.4080 - val_accuracy: 0.7774
Epoch 5/5
239/239 [==============================] - 512s 2s/step - loss: 0.2618 -
accuracy: 0.9256 - val_loss: 1.3314 - val_accuracy: 0.7774
```

Summary

- rescale layer
- 2d conv layer with 32 filters (896 trainable parameters)
- max pooling layer
- 2d conv layer with 64 filters (18496 trainable parameters)
- max pooling layer
- 2d conv layer with 64 filters (73856 trainable parameters)
- max pooling layer
- flatten layer
- dropout layer
- dense layer (128 neurons and 12845184 trainable parameters)
- dense layer (53 neurons and 6837 trainable parameters)

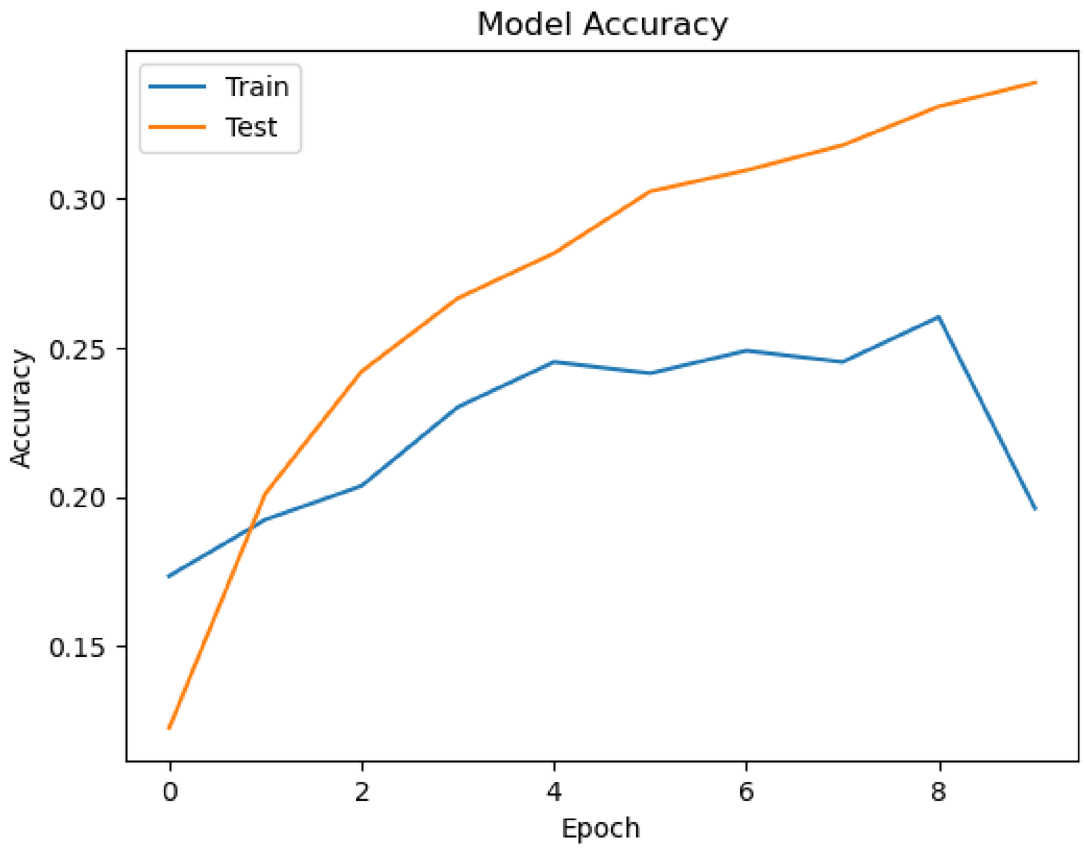
[19]:
```
model2.summary()
```

```
Model: "sequential_1"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_1 (Rescaling)     (None, 224, 224, 3)       0

 conv2d (Conv2D)             (None, 224, 224, 32)      896

 max_pooling2d (MaxPooling2D  (None, 112, 112, 32)     0
 )

 conv2d_1 (Conv2D)           (None, 112, 112, 64)      18496

 max_pooling2d_1 (MaxPooling  (None, 56, 56, 64)       0
 2D)

 conv2d_2 (Conv2D)           (None, 56, 56, 128)       73856

 max_pooling2d_2 (MaxPooling  (None, 28, 28, 128)      0
 2D)

 flatten_1 (Flatten)         (None, 100352)            0

 dropout_2 (Dropout)         (None, 100352)            0

 dense_3 (Dense)             (None, 128)               12845184

 dense_4 (Dense)             (None, 53)                6837

=================================================================
Total params: 12,945,269
Trainable params: 12,945,269
Non-trainable params: 0
_________________________________________________________________
```

Plot the Model Metrics for each Epoch

[21]:
```
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 = model2.evaluate(test, verbose=0)
```

```
print('Test Loss:', score[0])
print('Test Accuracy:', score[1])
```



```
Test Loss: 1.3314069509506226
Test Accuracy: 0.7773584723472595
```

Analysis Our Convolutional Neural Network (CNN) model has high accuracy (~78%) due to its ability to learn hierarchical representations, where each layer learns progressively more complex features. Moreover, CNNs reduce the input's dimensionality using convolutional and pooling layers, and control overfitting through methods like dropout. Additionally, having a large dataset is beneficial to predict better on unseen data.

1.0.5 Pre-Trained Model: MobileNet (V2)

Source: https://www.kaggle.com/code/paultimothymooney/mobilenetv2-with-tensorflow

Steps

- Resize the Data: 128 x 128
- Load the Pre-Trained Model

- Define a Sequenial Model
- Provide Build Parameters
- Compile the Model
- Train Model

[13]:
```python
import tensorflow_hub as hub

# Resize the Training Data to 128 x 128
train = tf.keras.preprocessing.image_dataset_from_directory(
    "/kaggle/input/cards-image-datasetclassification/train",
    labels="inferred",
    label_mode="categorical",
    color_mode="rgb",
    image_size=(128, 128),
    seed=1234,
    batch_size=32,
)

# Resize the Testing Data to 128 x 128
test = tf.keras.preprocessing.image_dataset_from_directory(
    "/kaggle/input/cards-image-datasetclassification/test",
    labels="inferred",
    label_mode="categorical",
    color_mode="rgb",
    image_size=(128, 128),
    seed=1234,
    batch_size=32,
)

# Load the Pre-Trained Model (1st layer)
keras_layer = hub.KerasLayer('https://kaggle.com/models/google/mobilenet-v2/
 ↪frameworks/TensorFlow2/variations/035-128-classification/versions/2')

# Define a Sequential Model:
# Neural Network with 53 neurons (2nd layer)
# Softmax Activation: used for multi-class classification to find probabilities␣
 ↪for each class (53 classes)
model3 = tf.keras.Sequential([
    keras_layer,
    tf.keras.layers.Dense(53, activation='softmax')
])

# Build Parameters:
# None: Batch size is not specified
# Shape of the Input: 128 x 128
# Channels: 3
model3.build([None, 128, 128, 3])
```

```
# Compile Model
# Loss: categorical_crossentropy (used for multi-class classification for␣
↪labels that must fit in one class)
# Optimizer: adam
# Metrics: Accuracy
model3.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

# Train model
# Train model using the training data, 10 epochs, a batch size of 32, and use␣
↪the test data
history = model3.fit(train,
                    epochs=10,
                    batch_size=32,
                    verbose=1,
                    validation_data=test)
```

```
Found 7624 files belonging to 53 classes.
Found 265 files belonging to 53 classes.
```

Summary

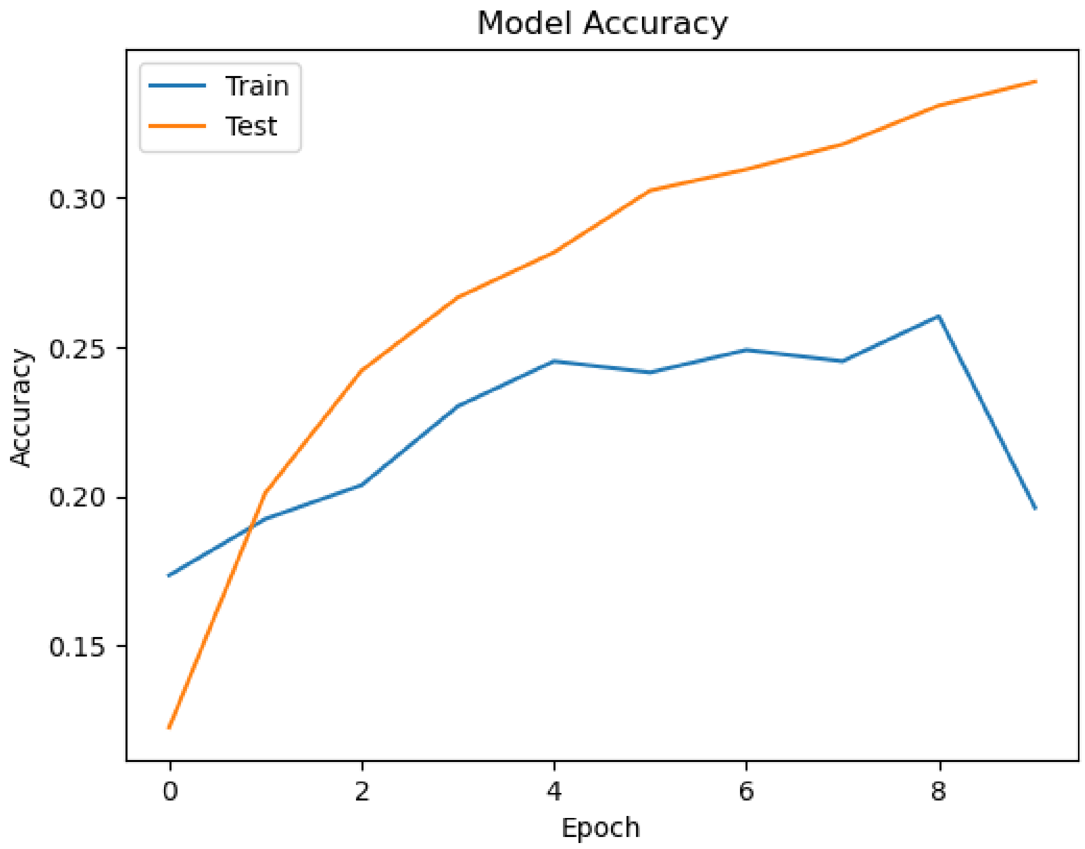
- Pre-trained MobileNet (V2) Model (1692489 trainable parameters)
- neural network with 53 classes (53,106 trainable parameters)

[14]:
```
model3.summary()
```

```
Model: "sequential_2"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 keras_layer (KerasLayer)    (None, 1001)              1692489

 dense_5 (Dense)             (None, 53)                53106

=================================================================
Total params: 1,745,595
Trainable params: 53,106
Non-trainable params: 1,692,489
_________________________________________________________________
```

Plot the Model Metrics for each Epoch

[15]:
```
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 = model3.evaluate(test, verbose=0)
print('Test Loss:', score[0])
print('Test Accuracy:', score[1])
```



```
Test Loss: 2.972522258758545
Test Accuracy: 0.19622641801834106
```

Analysis As observed, the test accuracy obtained through transfer learning using the MobileNet (V2) model is quite low (~20%). One possible explanation for this could be that the MobileNet (V2) model was originally trained using the massive ImageNet dataset and may not be able to generalize well on our dataset, which is much smaller in comparison. Specifically, the ImageNet dataset contains 14,197,122 images, whereas ours only has 7,624 images. Hence, it is likely that the MobileNet (V2) model is too complex to effectively learn from our limited amount of data.

1.1 The CNN Model yields the highest accuracy for our data!