Dmitrii Obideiko, Matt McCoy

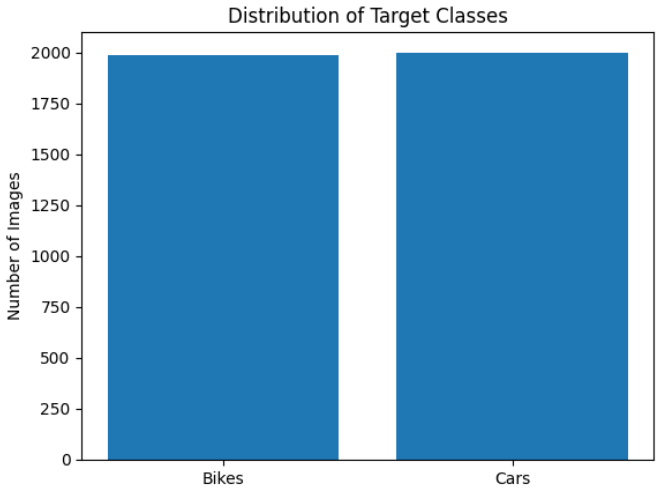
Load the Dataset

```
import io
import imghdr
import zipfile
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, GlobalAveragePooling2D
from keras.utils import to_categorical
from keras.applications.mobilenet_v2 import MobileNetV2, preprocess_input

num_classes = 2
num_channels = 3

image_height = 32
image_width = 32

def resize_images(images, size):
    return np.stack([np.array(image.convert('RGB').resize(size)) / 255.0 for image in images])

# Load the dataset
zip_path = "Car-Bike-Dataset.zip"
folders = ["Bike", "Car"]
bike_images = []
car_images = []

with zipfile.ZipFile(zip_path, "r") as zip_file:
    for folder in folders:
        folder_path = folder + "/"

        for filename in zip_file.namelist():
            if folder_path in filename and imghdr.what(None, zip_file.read(filename)) is not None:
                image_data = zip_file.read(filename)
                image = Image.open(io.BytesIO(image_data))

                if folder == "Bike":
                    bike_images.append(image)
                else:
                    car_images.append(image)

all_images = bike_images + car_images
bike_labels = [0] * len(bike_images)
car_labels = [1] * len(car_images)
all_labels = bike_labels + car_labels

# Divide the data into training and testing sets
train_images, test_images, train_labels, test_labels = train_test_split(all_images, all_labels, test_size = 0.2, random_state = 4

# Define the image size for resizing
image_size = (image_height, image_width)
train_images = resize_images(train_images, image_size)
test_images = resize_images(test_images, image_size)

train_labels = np.array(train_labels)
test_labels = np.array(test_labels)

train_labels = to_categorical(train_labels, num_classes)
test_labels = to_categorical(test_labels, num_classes)
```

```
/usr/local/lib/python3.9/dist-packages/PIL/Image.py:975: UserWarning: Palette images with Transparency expressed in bytes sh
  warnings.warn(
```

Distribution of the Target Classes

```
# Display the distribution of the target classes
labels = ['Bikes', 'Cars']
counts = [len(bike_images), len(car_images)]

plt.bar(labels, counts)
plt.xlabel('Classes')
plt.ylabel('Number of Images')
plt.title('Distribution of Target Classes')
plt.show()
```



Description of the Data and the Purpose of the Model

It looks like the number of bike images is the same as the number of car images as the bars for both taget classes look about the same in height, which indicates that the distribution is balanced. This can lead to good performce of the model as a blanced distribution of the model tells us that the model is not biasd towards a particular target class. The model is supposed to take an image of a car and a bike and then guess wheather it is a car or a bike with a fairly high accuracy. Cars and bikes that are in pictures can be different in some way or another (for example they can me different models or have different color), and the model should be able to find common characteristics for making accurate predictions despite those minor differences.

Sequential Model

```
# Create Sequential model
simple_model = Sequential([
    Flatten(input_shape=(image_height, image_width, num_channels)),
    Dense(64, activation='relu'),
    Dense(num_classes, activation='softmax')
])

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

simple_model.fit(train_images, train_labels, batch_size=32, epochs=10, validation_data=(test_images, test_labels))

simple_test_loss, seq_test_acc = simple_model.evaluate(test_images, test_labels)
print('Simple Sequential Model Test accuracy:', seq_test_acc)
print("Test Accuracy:", seq_test_acc)
```

```
Epoch 1/10
100/100 [==============================] - 1s 9ms/step - loss: 0.6641 - accuracy: 0.6632 - val_loss: 0.5496 - val_accuracy:
Epoch 2/10
100/100 [==============================] - 1s 7ms/step - loss: 0.5316 - accuracy: 0.7458 - val_loss: 0.5306 - val_accuracy:
Epoch 3/10
100/100 [==============================] - 1s 7ms/step - loss: 0.5198 - accuracy: 0.7492 - val_loss: 0.5303 - val_accuracy:
Epoch 4/10
100/100 [==============================] - 1s 7ms/step - loss: 0.5163 - accuracy: 0.7508 - val_loss: 0.5128 - val_accuracy:
Epoch 5/10
```

```
100/100 [==============================] - 1s 7ms/step - loss: 0.4810 - accuracy: 0.7668 - val_loss: 0.5119 - val_accuracy:
Epoch 6/10
100/100 [==============================] - 1s 6ms/step - loss: 0.4643 - accuracy: 0.7803 - val_loss: 0.4993 - val_accuracy:
Epoch 7/10
100/100 [==============================] - 1s 7ms/step - loss: 0.4663 - accuracy: 0.7859 - val_loss: 0.4956 - val_accuracy:
Epoch 8/10
100/100 [==============================] - 1s 7ms/step - loss: 0.4502 - accuracy: 0.7859 - val_loss: 0.5582 - val_accuracy:
Epoch 9/10
100/100 [==============================] - 1s 7ms/step - loss: 0.4310 - accuracy: 0.8035 - val_loss: 0.4919 - val_accuracy:
Epoch 10/10
100/100 [==============================] - 1s 6ms/step - loss: 0.4194 - accuracy: 0.8104 - val_loss: 0.4649 - val_accuracy:
25/25 [==============================] - 0s 3ms/step - loss: 0.4649 - accuracy: 0.7804
Simple Sequential Model Test accuracy: 0.7804266214370728
Test Accuracy: 0.7804266214370728
```

The Sequential model received an accuracy of 0.78, which shows that the model perfromed fairly well. This means that the data that was used to trian the model has a storng sequential pattern as sequential models were designed to campute relationships between elements in a sequence.

Custom CNN Model

```
cnn_model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(image_height, image_width, num_channels)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    Flatten(),
    Dense(64, activation='relu'),
    Dense(num_classes, activation='softmax')
])

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

cnn_model.fit(train_images, train_labels, batch_size=32, epochs=10, validation_data=(test_images, test_labels))

cnn_test_loss, cnn_test_acc = cnn_model.evaluate(test_images, test_labels)
print('Custom CNN Test accuracy:', cnn_test_acc)
```

```
Epoch 1/10
100/100 [==============================] - 7s 59ms/step - loss: 0.5517 - accuracy: 0.7062 - val_loss: 0.4714 - val_accuracy:
Epoch 2/10
100/100 [==============================] - 5s 47ms/step - loss: 0.2990 - accuracy: 0.8741 - val_loss: 0.2805 - val_accuracy:
Epoch 3/10
100/100 [==============================] - 6s 55ms/step - loss: 0.2438 - accuracy: 0.9043 - val_loss: 0.2679 - val_accuracy:
Epoch 4/10
100/100 [==============================] - 5s 46ms/step - loss: 0.2028 - accuracy: 0.9168 - val_loss: 0.2626 - val_accuracy:
Epoch 5/10
100/100 [==============================] - 5s 49ms/step - loss: 0.1820 - accuracy: 0.9209 - val_loss: 0.2546 - val_accuracy:
Epoch 6/10
100/100 [==============================] - 5s 52ms/step - loss: 0.1554 - accuracy: 0.9360 - val_loss: 0.2668 - val_accuracy:
Epoch 7/10
100/100 [==============================] - 5s 49ms/step - loss: 0.1373 - accuracy: 0.9432 - val_loss: 0.3272 - val_accuracy:
Epoch 8/10
100/100 [==============================] - 6s 59ms/step - loss: 0.1200 - accuracy: 0.9579 - val_loss: 0.2738 - val_accuracy:
Epoch 9/10
100/100 [==============================] - 5s 46ms/step - loss: 0.0922 - accuracy: 0.9642 - val_loss: 0.2671 - val_accuracy:
Epoch 10/10
100/100 [==============================] - 6s 58ms/step - loss: 0.0699 - accuracy: 0.9749 - val_loss: 0.3027 - val_accuracy:
25/25 [==============================] - 0s 14ms/step - loss: 0.3027 - accuracy: 0.9046
Custom CNN Test accuracy: 0.9046424031257629
```

The CNN model perfomred exceptionally well, with an accuracy of 0.905. This tells us that the data that was used to train this model has a strong spatial or local correlations as the CNN model tends to be effective when it comes to capturing those patterns.

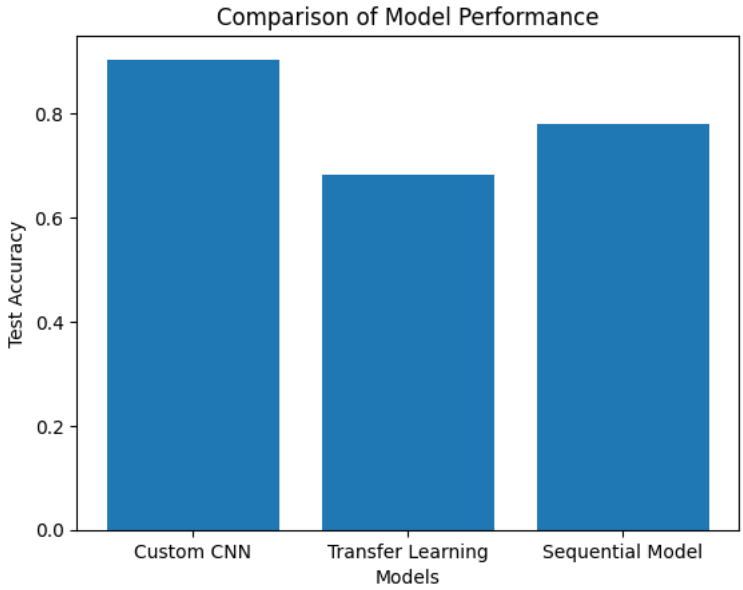
Pre-trained MobileNetV2 Model with Transfer Learning

```
base_model = MobileNetV2(weights='imagenet', include_top=False, input_shape=(image_height, image_width, num_channels))

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

transfer_model = Sequential([
base_model,
GlobalAveragePooling2D(),
Dense(num_classes, activation='softmax')
])

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

transfer_train_images = preprocess_input(train_images * 255)
transfer_test_images = preprocess_input(test_images * 255)

transfer_model.fit(transfer_train_images, train_labels, batch_size=32, epochs=10, validation_data=(transfer_test_images, test_lab

transfer_test_loss, transfer_test_acc = transfer_model.evaluate(transfer_test_images, test_labels)
print('Transfer Learning (MobileNetV2) Test accuracy:', transfer_test_acc)
```

```
WARNING:tensorflow:`input_shape` is undefined or non-square, or `rows` is not in [96, 128, 160, 192, 224]. Weights for input
Epoch 1/10
100/100 [==============================] - 12s 65ms/step - loss: 0.6600 - accuracy: 0.6751 - val_loss: 0.6397 - val_accuracy
Epoch 2/10
100/100 [==============================] - 4s 37ms/step - loss: 0.6294 - accuracy: 0.6955 - val_loss: 0.6234 - val_accuracy:
Epoch 3/10
100/100 [==============================] - 4s 44ms/step - loss: 0.6127 - accuracy: 0.6984 - val_loss: 0.6144 - val_accuracy:
Epoch 4/10
100/100 [==============================] - 7s 73ms/step - loss: 0.6018 - accuracy: 0.6993 - val_loss: 0.6087 - val_accuracy:
Epoch 5/10
100/100 [==============================] - 4s 43ms/step - loss: 0.5937 - accuracy: 0.6999 - val_loss: 0.6048 - val_accuracy:
Epoch 6/10
100/100 [==============================] - 5s 54ms/step - loss: 0.5875 - accuracy: 0.6993 - val_loss: 0.6025 - val_accuracy:
Epoch 7/10
100/100 [==============================] - 4s 37ms/step - loss: 0.5829 - accuracy: 0.7024 - val_loss: 0.6006 - val_accuracy:
Epoch 8/10
100/100 [==============================] - 4s 36ms/step - loss: 0.5790 - accuracy: 0.7028 - val_loss: 0.5994 - val_accuracy:
Epoch 9/10
100/100 [==============================] - 5s 54ms/step - loss: 0.5757 - accuracy: 0.7053 - val_loss: 0.5985 - val_accuracy:
Epoch 10/10
100/100 [==============================] - 4s 43ms/step - loss: 0.5729 - accuracy: 0.7043 - val_loss: 0.5980 - val_accuracy:
25/25 [==============================] - 1s 29ms/step - loss: 0.5980 - accuracy: 0.6826
Transfer Learning (MobileNetV2) Test accuracy: 0.682559609413147
```

Pre-trained MobileNetV2 Model with Transfer Learning received a good accuracy of 0.689. This tells us about data that it has visual patterns that are similar to the ones present in MobileNetV2 mode that was pretrained.

```
models = ['Custom CNN', 'Transfer Learning', 'Sequential Model']
accuracies = [cnn_test_acc, transfer_test_acc, seq_test_acc]

plt.bar(models, accuracies)
plt.xlabel('Models')
plt.ylabel('Test Accuracy')
plt.title('Comparison of Model Performance')
plt.show()
```



Analysys of the Performce of Various Approaches

If we compare the results from the sequential, the custom CNN and MobileNetV2 transfer learning models, we can see that we got better results with the custom CNN model. The CNN model received an accuracy of 91.47%, the transfer learning model received an accuracy of 68.12%, and the Sequential model received an accuracy of 79%. With the customer CNN model, it seems that the accuracy increases as the number of epochs increases, which shows that the model was learning and making improvements over time. We can see the same thing with the sequential model: the accuracy increases as the number of epochs increases, which also indicates to us that the sequential model was learning over time. With the transfer learning model, we can see that the accuracy remained about the same, indicating that it wasn't making any progress over time. The CNN model received the highest accuracy perhaps due to the fact that the model was designed and trained for the given classification task. The transfer learning model received the lowest accuracy perhaps because it was trained on a different dataset, and it was not designed to be able to be optimized for particular features of a dataset.