fln1mybyr

December 11, 2023

0.0.1 Creating Dataset With Three Classes

THe dataset for this assignment consists of three classes, mainly Leaf, Car and Book. We have created this dataset from scratch ie. capturing the images through our phone camera. We do have images with noise, which has been introduced on purpose!

We have 100 images for each class, split to train and test and valid at 80:10:10 ratio.

We have used ImageDataGenerator to load images onto the memory udring the training process from the respective train and test folder mounted on the Gdrive.

[2]:
```
from google.colab import drive
# Mount Google Drive
drive.mount('/content/drive')
# Navigate to the folder where your zip file is located

%cd '/content/drive/MyDrive/'
# Unzip the folder
#!unzip "data_small.zip" -d "/content/"
!pip install keras-tuner --upgrade
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
/content/drive/MyDrive
Collecting keras-tuner
  Downloading keras_tuner-1.4.6-py3-none-any.whl (128 kB)
                           128.9/128.9
kB 4.6 MB/s eta 0:00:00
Requirement already satisfied: keras in /usr/local/lib/python3.10/dist-
packages (from keras-tuner) (2.14.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-
packages (from keras-tuner) (23.2)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-
packages (from keras-tuner) (2.31.0)
Collecting kt-legacy (from keras-tuner)
  Downloading kt_legacy-1.0.5-py3-none-any.whl (9.6 kB)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from requests->keras-tuner) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-
```

```
packages (from requests->keras-tuner) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.10/dist-packages (from requests->keras-tuner) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.10/dist-packages (from requests->keras-tuner)
(2023.11.17)
Installing collected packages: kt-legacy, keras-tuner
Successfully installed keras-tuner-1.4.6 kt-legacy-1.0.5
```

[23]:
```
#!rm -r '/content/data_small'
```

[3]:
```
#load our necessary packages
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import ResNet50
from tensorflow.keras import layers, models, optimizers
from keras_tuner import RandomSearch
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
%matplotlib inline
from tensorflow.keras.preprocessing.image import img_to_array, load_img
import random
import os
#define the datapath
data_path = "/content/drive/MyDrive/data_small"
```

Train - Validate and Test set

[5]:
```
#define our directories for the data
book_train_dir = os.path.join('/content/drive/MyDrive/data_small/train/book')
car_train_dir = os.path.join('/content/drive/MyDrive/data_small/train/car')
leaf_train_dir = os.path.join('/content/drive/MyDrive/data_small/train/leaf')

book_test_dir = os.path.join('/content/drive/MyDrive/data_small/test/book')
car_test_dir = os.path.join('/content/drive/MyDrive/data_small/test/car')
leaf_test_dir = os.path.join('/content/drive/MyDrive/data_small/test/leaf')

book_valid_dir = os.path.join('/content/drive/MyDrive/data_small/valid/book')
car_valid_dir = os.path.join('/content/drive/MyDrive/data_small/valid/car')
leaf_valid_dir = os.path.join('/content/drive/MyDrive/data_small/valid/leaf')

print('Total training book images:', len(os.listdir(book_train_dir)))
print('Total training car images:', len(os.listdir(car_train_dir)))
print('Total training leaf images:', len(os.listdir(leaf_train_dir)))
print("-----------------------------------------------------")
print('Total test book images:', len(os.listdir(book_test_dir)))
```

```
print('Total test car images:', len(os.listdir(car_test_dir)))
print('Total test leaf images:', len(os.listdir(leaf_test_dir)))
print("-------------------------------------------------------")
print('Total valid book images:', len(os.listdir(book_test_dir)))
print('Total valid car images:', len(os.listdir(car_test_dir)))
print('Total valid leaf images:', len(os.listdir(leaf_test_dir)))
```

```
Total training book images: 80
Total training car images: 80
Total training leaf images: 80
-------------------------------------------------------
Total test book images: 10
Total test car images: 10
Total test leaf images: 10
-------------------------------------------------------
Total valid book images: 10
Total valid car images: 10
Total valid leaf images: 10
```

[6]:
```
book_files = os.listdir(book_train_dir)
print(book_files[:10])

car_files = os.listdir(car_train_dir)
print(car_files[:10])

leaf_files = os.listdir(leaf_train_dir)
print(leaf_files[:10])
```

```
['IMG_20231209_161645.jpg', 'IMG_20231209_162017.jpg',
'IMG_20231209_161825.jpg', 'IMG_20231209_161612.jpg', 'IMG_20231209_161830.jpg',
'IMG_20231209_161600.jpg', 'IMG_20231209_161703.jpg', 'IMG_20231209_162004.jpg',
'IMG_20231209_161658.jpg', 'IMG_20231209_161837.jpg']
['1000024683 - Copy.jpeg', 'car.PNG', 'IMG_20231109_153852.jpg',
'IMG_20231207_132436.jpg', 'IMG_20231208_112952.jpg', '20230416_224404.jpg',
'IMG_20230804_181909.jpg', '20230806_175923.jpg', 'IMG_20231109_153859.jpg',
'20230806_175920.jpg']
['IMG_20231209_160706.jpg', 'IMG_20231209_160549.jpg',
'IMG_20231209_160631.jpg', 'IMG_20231209_155928.jpg', 'IMG_20231209_155909.jpg',
'IMG_20231209_160104.jpg', 'IMG_20231209_160709.jpg', 'IMG_20231209_155813.jpg',
'IMG_20231209_160028.jpg', 'IMG_20231209_155833.jpg']
```

Display 5 images of each class:

[7]:
```
pic_index = 5
next_book = [os.path.join(book_train_dir, fname)
                for fname in book_files[pic_index:pic_index+5]]
next_car = [os.path.join(car_train_dir, fname)
                for fname in car_files[pic_index:pic_index+5]]
```

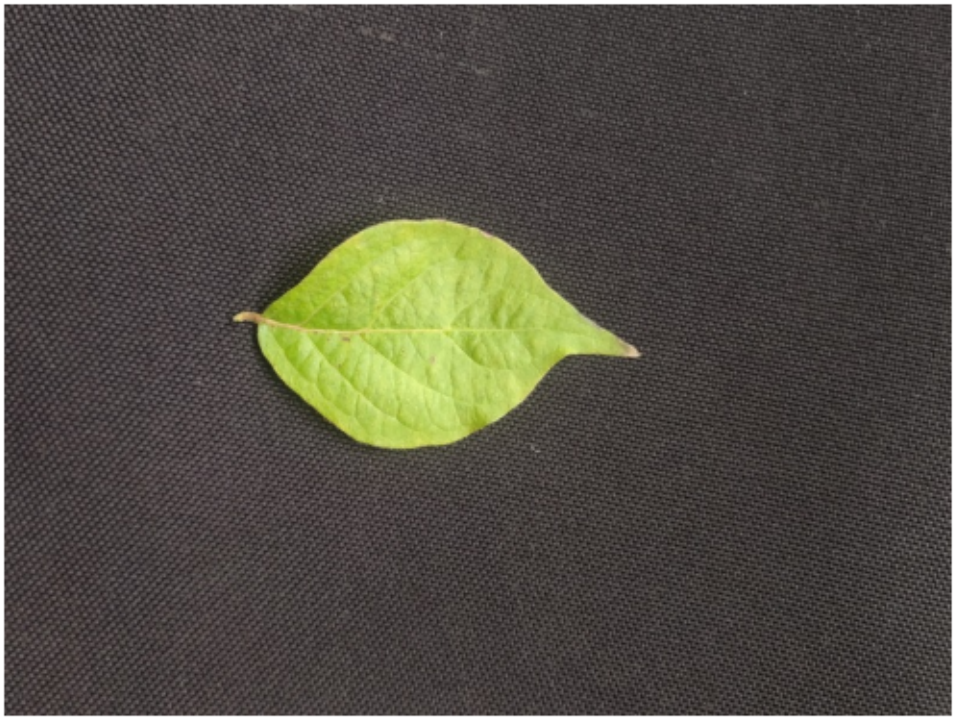
```
next_leaf = [os.path.join(leaf_train_dir, fname)
                for fname in leaf_files[pic_index:pic_index+5]]

for i, img_path in enumerate(next_book+next_car+next_leaf):
    img = mpimg.imread(img_path)
    plt.imshow(img)
    plt.axis('Off')
    plt.show()
```


URBAN
EDGE


STOP


ALDI


TOWARDS AIRPORT



Building the input Pipeline with Image Augmentation: Explaination:

rescale=1./255: Scales the pixel values to the range [0, 1].

rotation_range=40: Randomly rotates images in the range of [-40 degrees, 40 degrees].

width_shift_range=0.2 and height_shift_range=0.2: Randomly shifts the width and height of images by up to 20%.

shear_range=0.2: Applies shear transformations.

zoom_range=0.2: Randomly zooms into images.

horizontal_flip=True: Randomly flips images horizontally.

fill_mode='nearest': Fills in newly created pixels after a rotation or a width/height shift using the nearest available pixel.

[8]:
```
# Image size and batch size
img_size = (64, 64)
batch_size = 32
```

[9]:
```
TRAINING_DIR = "/content/drive/MyDrive/data_small/train/"
VALIDATION_DIR = "/content/drive/MyDrive/data_small/valid/"
TEST_DIR = "/content/drive/MyDrive/data_small/test/"

training_datagen = ImageDataGenerator(
    rescale = 1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest')

validation_datagen = ImageDataGenerator(rescale = 1./255)
test_datagen = ImageDataGenerator(rescale = 1./255)

train_generator = training_datagen.flow_from_directory(
    TRAINING_DIR,
    target_size=(150,150),
    class_mode='categorical')

validation_generator = validation_datagen.flow_from_directory(
    VALIDATION_DIR,
    target_size=(150,150),
    class_mode='categorical')

test_generator = validation_generator = test_datagen.flow_from_directory(
    TEST_DIR,
```

```
    target_size=(150,150),
    class_mode='categorical')
```

```
Found 240 images belonging to 3 classes.
Found 30 images belonging to 3 classes.
Found 30 images belonging to 3 classes.
```

Training A a DNN from scratch:

[10]:
```
model = tf.keras.models.Sequential([
    # This is the first convolution
    tf.keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(150, 150,␣
 ↪3)),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(16, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(8, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(4, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),

    # Flatten the results to feed into a DNN
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dropout(0.5),
    # 512 neuron hidden layer
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(3, activation='softmax')  # for our three classes
])
```

[11]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 148, 148, 32)      896

 max_pooling2d (MaxPooling2  (None, 74, 74, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 72, 72, 16)        4624

 max_pooling2d_1 (MaxPoolin  (None, 36, 36, 16)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 34, 34, 8)         1160

 max_pooling2d_2 (MaxPoolin  (None, 17, 17, 8)         0
```

```
 g2D)

 conv2d_3 (Conv2D)           (None, 15, 15, 4)         292

 max_pooling2d_3 (MaxPoolin  (None, 7, 7, 4)           0
 g2D)

 flatten (Flatten)           (None, 196)               0

 dropout (Dropout)           (None, 196)               0

 dense (Dense)               (None, 512)               100864

 dense_1 (Dense)             (None, 3)                 1539

=================================================================
Total params: 109375 (427.25 KB)
Trainable params: 109375 (427.25 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[12]:
```
model.compile(loss = 'categorical_crossentropy', optimizer='rmsprop',␣
↪metrics=['accuracy'])
```

[13]:
```
history = model.fit(train_generator, epochs=15, validation_data =␣
↪validation_generator, verbose = 1)
```

```
Epoch 1/15
8/8 [==============================] - 96s 12s/step - loss: 1.1100 - accuracy:
0.3708 - val_loss: 1.0724 - val_accuracy: 0.4000
Epoch 2/15
8/8 [==============================] - 9s 1s/step - loss: 1.0496 - accuracy:
0.4083 - val_loss: 1.0037 - val_accuracy: 0.6667
Epoch 3/15
8/8 [==============================] - 9s 1s/step - loss: 0.9939 - accuracy:
0.5167 - val_loss: 0.9417 - val_accuracy: 0.5000
Epoch 4/15
8/8 [==============================] - 10s 1s/step - loss: 0.9976 - accuracy:
0.5000 - val_loss: 0.8573 - val_accuracy: 0.6667
Epoch 5/15
8/8 [==============================] - 10s 1s/step - loss: 0.9094 - accuracy:
0.5750 - val_loss: 0.8031 - val_accuracy: 0.6667
Epoch 6/15
8/8 [==============================] - 9s 1s/step - loss: 0.8330 - accuracy:
0.6458 - val_loss: 0.9495 - val_accuracy: 0.6000
Epoch 7/15
8/8 [==============================] - 10s 1s/step - loss: 0.9012 - accuracy:
0.5750 - val_loss: 0.9073 - val_accuracy: 0.6667
```

```
Epoch 8/15
8/8 [==============================] - 10s 1s/step - loss: 0.8595 - accuracy:
0.6708 - val_loss: 0.8866 - val_accuracy: 0.6667
Epoch 9/15
8/8 [==============================] - 9s 1s/step - loss: 0.7880 - accuracy:
0.6708 - val_loss: 0.6193 - val_accuracy: 0.7333
Epoch 10/15
8/8 [==============================] - 10s 1s/step - loss: 0.7127 - accuracy:
0.6958 - val_loss: 0.5621 - val_accuracy: 0.8000
Epoch 11/15
8/8 [==============================] - 10s 1s/step - loss: 0.7447 - accuracy:
0.6958 - val_loss: 0.5295 - val_accuracy: 0.8333
Epoch 12/15
8/8 [==============================] - 9s 1s/step - loss: 0.6674 - accuracy:
0.7250 - val_loss: 0.4250 - val_accuracy: 0.9000
Epoch 13/15
8/8 [==============================] - 9s 1s/step - loss: 0.6186 - accuracy:
0.7625 - val_loss: 0.5556 - val_accuracy: 0.8000
Epoch 14/15
8/8 [==============================] - 10s 1s/step - loss: 0.6522 - accuracy:
0.7125 - val_loss: 0.5208 - val_accuracy: 0.8333
Epoch 15/15
8/8 [==============================] - 12s 2s/step - loss: 0.6494 - accuracy:
0.7500 - val_loss: 0.4682 - val_accuracy: 0.8333
```

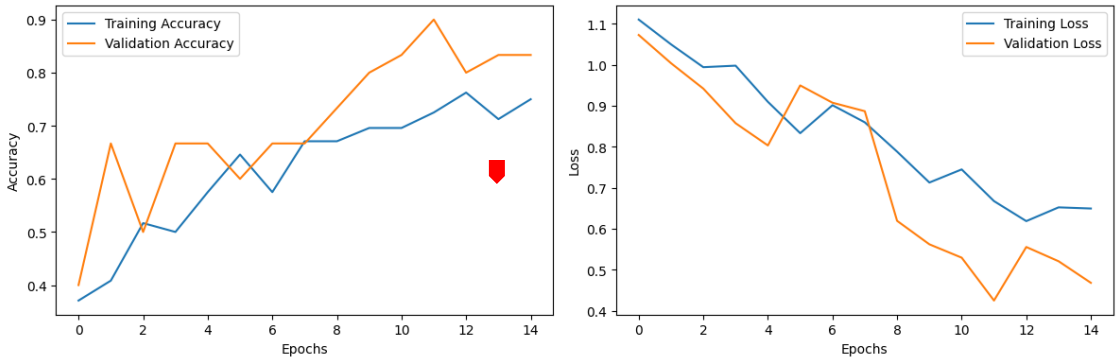
Report Accuracies:

[14]:
```python
# Evaluate the model on the test set
test_generator = test_datagen.flow_from_directory(
    '/content/drive/MyDrive/data_small/test/',
    target_size=(150,150),
    batch_size=32,
    class_mode='categorical'
)
accuracy = model.evaluate(test_generator)[1]
print(f"Test Accuracy: {accuracy}")
```

```
Found 30 images belonging to 3 classes.
1/1 [==============================] - 2s 2s/step - loss: 0.4682 - accuracy:
0.8333
Test Accuracy: 0.8333333134651184
```

[15]:
```python
#plot our train and test accuracy:
train_accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']
train_loss = history.history['loss']
val_loss = history.history['val_loss']
```

```
# Plot the accuracy over epochs
plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
plt.plot(train_accuracy, label='Training Accuracy')
plt.plot(val_accuracy, label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

# Plot the loss over epochs
plt.subplot(1, 2, 2)
plt.plot(train_loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.show()
```



Our predicted classes:

[16]:
```
predictions = model.predict(test_generator)
predicted_classes = tf.argmax(predictions, axis=1)
predicted_classes
```

```
1/1 [==============================] - 2s 2s/step
```

[16]:
```
<tf.Tensor: shape=(30,), dtype=int64, numpy=
array([2, 1, 2, 1, 2, 2, 0, 2, 1, 0, 0, 2, 2, 2, 1, 1, 2, 1, 0, 0, 2, 0,
       0, 0, 1, 1, 0, 1, 2, 0])>
```

True Labels:

[17]:
```
true_classes = test_generator.classes
true_classes
```

[17]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2], dtype=int32)
```

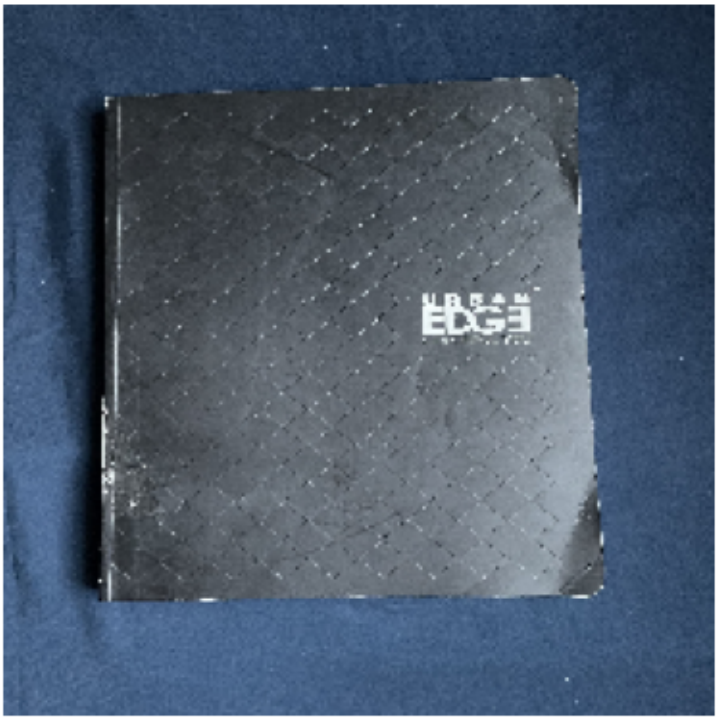
[18]:
```
#extracting misclassified images to display:
misclassified_indices = [i for i in range(0,len(predicted_classes)) if
 ↪true_classes[i] != predicted_classes[i]]
misclassified_images = [test_generator.filepaths[i] for i in
 ↪misclassified_indices]
```

Displaying few misclassifed images

[19]:
```
indices = [random.choice(misclassified_indices) for i in range(8)]
for i in indices:
    image_path = misclassified_images[misclassified_indices.index(i)]
    img = tf.keras.preprocessing.image.load_img(image_path, target_size=(224,
 ↪224))
    img_array = tf.keras.preprocessing.image.img_to_array(img)
    true_label = true_classes[i]
    predicted_label = np.argmax(predictions[i])
    plt.imshow(img_array / 255.0)  # Rescale to [0, 1]
    plt.title(f"True Label: {true_label}, Predicted Label: {predicted_label}")
    plt.axis('off')
    plt.show()
    plt.show()
```

True Label: 1, Predicted Label: 2

True Label: 0, Predicted Label: 1



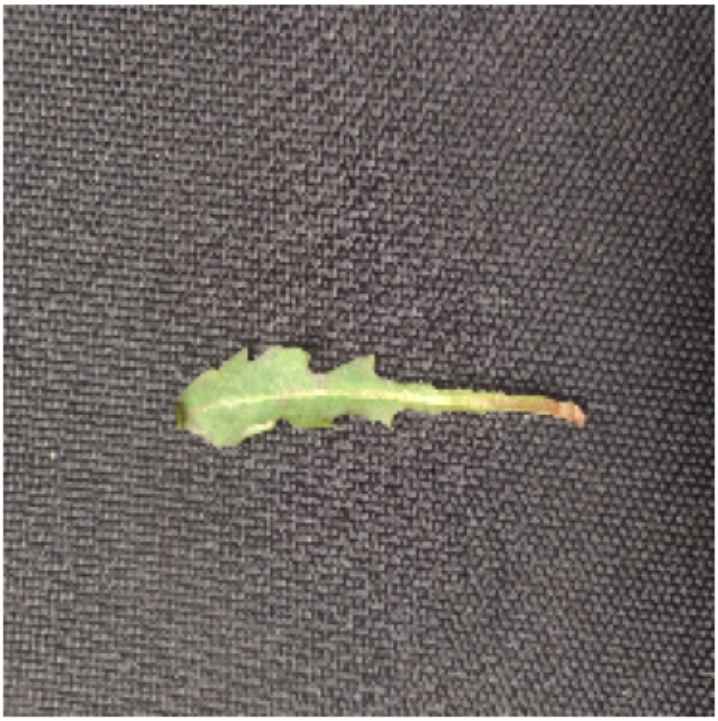
True Label: 2, Predicted Label: 1

True Label: 2, Predicted Label: 0

True Label: 1, Predicted Label: 2

True Label: 1, Predicted Label: 0

True Label: 2, Predicted Label: 1



True Label: 1, Predicted Label: 2

Training a Resnet50 model with imagenet weights:

[20]:
```
def build_model(hp):
    base_model = ResNet50(weights='imagenet', include_top=False,␣
 ↪input_shape=(32, 32, 3))
    for layer in base_model.layers:
        layer.trainable = False
    n_hidden = hp.Int("n_hidden", min_value=0, max_value=8, default=2)
    n_neurons = hp.Int("n_neurons", min_value=16, max_value=256)
    learning_rate = hp.Float("learning_rate", min_value=1e-4,␣
 ↪max_value=1e-2,sampling="log")
    optimizer = hp.Choice("optimizer", values=["sgd", "adam"])
    if optimizer == "sgd":
      optimizer = tf.keras.optimizers.SGD(learning_rate=learning_rate)
    else:
      optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)
    model = models.Sequential([base_model,
                               layers.GlobalAveragePooling2D(),
                               layers.Dense(hp.Int('dense_units',␣
 ↪min_value=128, max_value=512, step=32), activation='relu'),
                               layers.Dropout(0.5),
                               layers.Dense(3, activation='softmax')]) # 3␣
 ↪classes: book, car, leaf
    model.compile(
        optimizer=optimizer,
        loss='categorical_crossentropy',
        metrics=['accuracy'])
    return model
```

Fine tuning the model with RandomSearch:

[21]:
```
tuner = RandomSearch(
    build_model,
    objective='val_accuracy',
    overwrite=True,
    max_trials=5,  # Adjust based on your resources and time constraints
    directory='/content/tuning',  # Change this to a specific directory
    project_name='my_3_classifier')
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
94765736/94765736 [==============================] - 0s 0us/step
```

[22]:
```
# Perform the hyperparameter search
tuner.search(train_generator, epochs=10, validation_data=validation_generator)
```

```
Trial 5 Complete [00h 04m 13s]
val_accuracy: 0.5666666626930237

Best val_accuracy So Far: 0.7333333492279053
Total elapsed time: 00h 24m 02s
```

Get the top three modes with highest val accuracy:

[23]:
```
top3_models = tuner.get_best_models(num_models=3)
best_model = top3_models[0]
best_model
```

[23]:
```
<keras.src.engine.sequential.Sequential at 0x78ce0e9a2860>
```

[24]:
```
# Get the best hyperparameters
best_hps = tuner.get_best_hyperparameters(num_trials=1)[0]
print(f"Best Hyperparameters: {best_hps}")
```

```
Best Hyperparameters:
<keras_tuner.src.engine.hyperparameters.hyperparameters.HyperParameters object
at 0x78ce0f872260>
```

[25]:
```
top3_params = tuner.get_best_hyperparameters(num_trials=3)
top3_params[0].values # best hyperparameter values
```

[25]:
```
{'n_hidden': 3,
 'n_neurons': 155,
 'learning_rate': 0.0008183184659748614,
 'optimizer': 'sgd',
 'dense_units': 160}
```

Fit the model with the best parameters:

[36]:
```
best_model = tuner.hypermodel.build(best_hps)
history = best_model.fit(train_generator, epochs=10,␣
 ↪validation_data=validation_generator)

# Evaluate the best model
test_generator = test_datagen.flow_from_directory(
    "/content/drive/MyDrive/data_small/test/",
    target_size=img_size,
    batch_size=batch_size,
    class_mode='categorical'
)

test_loss, test_acc = best_model.evaluate(test_generator)
print(f'Test accuracy: {test_acc}')
```

```
Epoch 1/10
```

```
8/8 [==============================] - 45s 3s/step - loss: 1.1636 - accuracy:
0.3542 - val_loss: 1.0709 - val_accuracy: 0.6000
Epoch 2/10
8/8 [==============================] - 22s 3s/step - loss: 1.1910 - accuracy:
0.3417 - val_loss: 1.0695 - val_accuracy: 0.6000
Epoch 3/10
8/8 [==============================] - 22s 3s/step - loss: 1.2579 - accuracy:
0.3542 - val_loss: 1.0684 - val_accuracy: 0.6000
Epoch 4/10
8/8 [==============================] - 21s 3s/step - loss: 1.1947 - accuracy:
0.3208 - val_loss: 1.0676 - val_accuracy: 0.6000
Epoch 5/10
8/8 [==============================] - 22s 3s/step - loss: 1.1539 - accuracy:
0.3333 - val_loss: 1.0664 - val_accuracy: 0.6000
Epoch 6/10
8/8 [==============================] - 20s 2s/step - loss: 1.1013 - accuracy:
0.4292 - val_loss: 1.0661 - val_accuracy: 0.5667
Epoch 7/10
8/8 [==============================] - 22s 3s/step - loss: 1.1168 - accuracy:
0.3750 - val_loss: 1.0651 - val_accuracy: 0.5333
Epoch 8/10
8/8 [==============================] - 21s 3s/step - loss: 1.1764 - accuracy:
0.3375 - val_loss: 1.0632 - val_accuracy: 0.6000
Epoch 9/10
8/8 [==============================] - 22s 3s/step - loss: 1.2144 - accuracy:
0.3042 - val_loss: 1.0628 - val_accuracy: 0.6000
Epoch 10/10
8/8 [==============================] - 21s 3s/step - loss: 1.1435 - accuracy:
0.3375 - val_loss: 1.0625 - val_accuracy: 0.5667
Found 30 images belonging to 3 classes.
1/1 [==============================] - 3s 3s/step - loss: 1.1041 - accuracy:
0.3333
Test accuracy: 0.3333333432674408
```

[34]:
```
predictions = best_model.predict(test_generator)
predicted_classes = tf.argmax(predictions, axis=1)
predicted_classes
```

```
1/1 [==============================] - 3s 3s/step
```

[34]:
```
<tf.Tensor: shape=(30,), dtype=int64, numpy=
array([2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2])>
```

[35]:
```
true_classes = test_generator.classes
misclassified_indices = [i for i in range(len(true_classes)) if␣
 ↪predicted_classes[i] != true_classes[i]]
```

```
misclassified_images = [test_generator.filepaths[i] for i in␣
 ↪misclassified_indices]
true_classes
```

[35]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2], dtype=int32)
```

Display misclassified images:

[29]:
```
indices = [random.choice(misclassified_indices) for i in range(5)]
for i in indices:
    image_path = misclassified_images[misclassified_indices.index(i)]
    img = tf.keras.preprocessing.image.load_img(image_path, target_size=(224,␣
 ↪224))
    img_array = tf.keras.preprocessing.image.img_to_array(img)
    true_label = true_classes[i]
    predicted_label = np.argmax(predictions[i])
    plt.imshow(img_array / 255.0)  # Rescale to [0, 1]
    plt.title(f"True Label: {true_label}, Predicted Label: {predicted_label}")
    plt.axis('off')
    plt.show()
    plt.show()
```

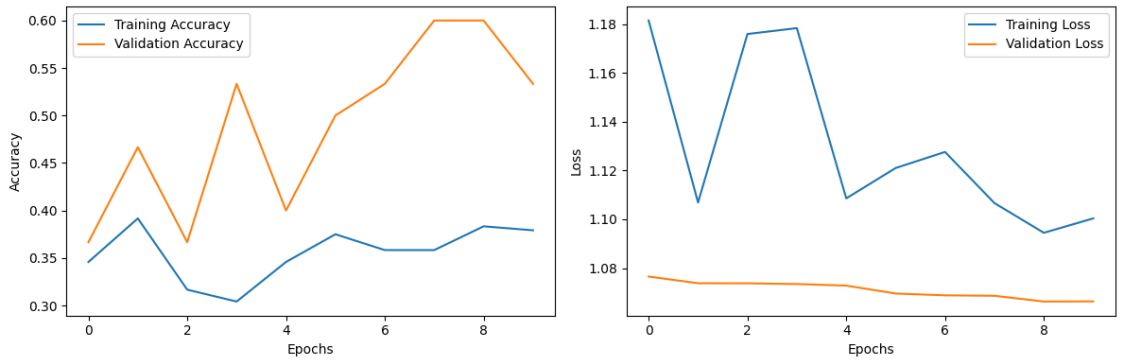
True Label: 2, Predicted Label: 0

True Label: 2, Predicted Label: 0


True Label: 2, Predicted Label: 0

[30]:
```
#plot our train and test accuracy:
train_accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']
train_loss = history.history['loss']
val_loss = history.history['val_loss']

# Plot the accuracy over epochs
plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
plt.plot(train_accuracy, label='Training Accuracy')
plt.plot(val_accuracy, label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

# Plot the loss over epochs
plt.subplot(1, 2, 2)
plt.plot(train_loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
```

```
plt.legend()

plt.tight_layout()
plt.show()
```



It does look like we havent reached accuracies that are not good enough. From the observations above, one can conclude that, we definitely need more data, data that is of good quality.

Perhaps the pre-trained model that i chose was not good enough for these category, but I certainly think its the lack of good quality data that i causing below average accuracies.

[ ]: