In this notebook I have shown how we can use a pre-trained model to train a binary classifier to detects cats and dogs. I have covered

1. How to train a binary classifier using transfer learning on a pre-trained VGG16 model.
2. How to evaluate the model on test set using different classification metrics.
3. How to visualize the images present in the training and test sets.

I hope you find this notebook helpful and some **UPVOTES** would be very much appreciated.

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

1. Import the Required Libraries

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

import os
os.environ["TF_CPP_MIN_LOG_LEVEL"]="2"
import warnings
warnings.filterwarnings("ignore")

from sklearn.metrics import confusion_matrix,classification_report

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense, GlobalAveragePooling2D, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.losses import BinaryCrossentropy
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import VGG16, VGG19
```

```
!pip install tensorflow==2.7.0
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: tensorflow==2.7.0 in /usr/local/lib/python3.8/dist-packages (2.7.0)
Requirement already satisfied: libclang>=9.0.1 in /usr/local/lib/python3.8/dist-packages (from tensorflow==2.7.0) (15.0.6.1)
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.8/dist-packages (from tensorflow==2.7.0) (1.14.1)
Requirement already satisfied: gast<0.5.0,>=0.2.1 in /usr/local/lib/python3.8/dist-packages (from tensorflow==2.7.0) (0.4.0)
Requirement already satisfied: flatbuffers<3.0,>=1.12 in /usr/local/lib/python3.8/dist-packages (from tensorflow==2.7.0) (2.0.7)
Requirement already satisfied: protobuf>=3.9.2 in /usr/local/lib/python3.8/dist-packages (from tensorflow==2.7.0) (3.19.6)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.8/dist-packages (from tensorflow==2.7.0) (2.2.0)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.8/dist-packages (from tensorflow==2.7.0) (1.15.0)
Requirement already satisfied: keras-preprocessing>=1.1.1 in /usr/local/lib/python3.8/dist-packages (from tensorflow==2.7.0) (1.1.2
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.8/dist-packages (from tensorflow==2.7.0) (0.2.0)
Requirement already satisfied: tensorboard~=2.6 in /usr/local/lib/python3.8/dist-packages (from tensorflow==2.7.0) (2.11.2)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.8/dist-packages (from tensorflow==2.7.0) (3.3.0)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.8/dist-packages (from tensorflow==2.7.0) (4.4.0)
Requirement already satisfied: absl-py>=0.4.0 in /usr/local/lib/python3.8/dist-packages (from tensorflow==2.7.0) (1.4.0)
Requirement already satisfied: h5py>=2.9.0 in /usr/local/lib/python3.8/dist-packages (from tensorflow==2.7.0) (3.1.0)
Requirement already satisfied: numpy>=1.14.5 in /usr/local/lib/python3.8/dist-packages (from tensorflow==2.7.0) (1.21.6)
Requirement already satisfied: wheel<1.0,>=0.32.0 in /usr/local/lib/python3.8/dist-packages (from tensorflow==2.7.0) (0.38.4)
Requirement already satisfied: keras<2.8,>=2.7.0rc0 in /usr/local/lib/python3.8/dist-packages (from tensorflow==2.7.0) (2.7.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.8/dist-packages (from tensorflow==2.7.0) (1.6.3)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.21.0 in /usr/local/lib/python3.8/dist-packages (from tensorflow==2.7
Requirement already satisfied: tensorflow-estimator<2.8,~=2.7.0rc0 in /usr/local/lib/python3.8/dist-packages (from tensorflow==2.7.
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.8/dist-packages (from tensorflow==2.7.0) (1.51.1)
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.8/dist-packages (from tensorboard~=2.6->tensorflow==2.7.0)
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.8/dist-packages (from tensorboard~=2.6->tensorflow==2.
Requirement already satisfied: tensorboard-data-server<0.7.0,>=0.6.0 in /usr/local/lib/python3.8/dist-packages (from tensorboard~=2
Requirement already satisfied: setuptools>=41.0.0 in /usr/local/lib/python3.8/dist-packages (from tensorboard~=2.6->tensorflow==2.7
Requirement already satisfied: google-auth<3,>=1.6.3 in /usr/local/lib/python3.8/dist-packages (from tensorboard~=2.6->tensorflow==
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.8/dist-packages (from tensorboard~=2.6->tensorflow==2.7.0)
Requirement already satisfied: tensorboard-plugin-wit>=1.6.0 in /usr/local/lib/python3.8/dist-packages (from tensorboard~=2.6->tens
Requirement already satisfied: google-auth-oauthlib<0.5,>=0.4.1 in /usr/local/lib/python3.8/dist-packages (from tensorboard~=2.6->t
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.8/dist-packages (from google-auth<3,>=1.6.3->tensorb
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.8/dist-packages (from google-auth<3,>=1.6.3->tensorboard~=2.
Requirement already satisfied: cachetools<6.0,>=2.0.0 in /usr/local/lib/python3.8/dist-packages (from google-auth<3,>=1.6.3->tensor
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python3.8/dist-packages (from google-auth-oauthlib<0.5,>=
Requirement already satisfied: importlib-metadata>=4.4 in /usr/local/lib/python3.8/dist-packages (from markdown>=2.6.8->tensorboard
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.8/dist-packages (from requests<3,>=2.21.0->tensorboa
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.8/dist-packages (from requests<3,>=2.21.0->tensorboard~
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.8/dist-packages (from requests<3,>=2.21.0->tensorboard~=2.6->
Requirement already satisfied: chardet<5,>=3.0.2 in /usr/local/lib/python3.8/dist-packages (from requests<3,>=2.21.0->tensorboard~=
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.8/dist-packages (from importlib-metadata>=4.4->markdown>=2.6.8->
Requirement already satisfied: pyasn1<0.5.0,>=0.4.6 in /usr/local/lib/python3.8/dist-packages (from pyasn1-modules>=0.2.1->google-a
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.8/dist-packages (from requests-oauthlib>=0.7.0->google-aut
```

2. Load the Image Training, Validation and Testing Datasets

i. Get the Image Datasets Paths

```
train_dataset_path = "/content/drive/MyDrive/data/train"
test_dataset_path = "/content/drive/MyDrive/data/test"
```

ii. Load Image Datasets and Apply Image Augmentations

Before feeding the image in the model, we need to resize the images present in the datasets to a fixed size. I have choosen the image width and height as 256x256px .

```
BATCH_SIZE = 32
IMG_WIDTH = 256
IMG_HEIGHT = 256
IMG_CHANNELS = 3
IMG_SHAPE = (IMG_WIDTH, IMG_HEIGHT, IMG_CHANNELS)

print(f"Batch Size: {BATCH_SIZE}")
print(f"Image Shape: {IMG_SHAPE}")
```

```
Batch Size: 32
Image Shape: (256, 256, 3)
```

Loading the training dataset and applying augmentations to it.

Also since there is no validation dataset available, we will keep 10% of the training dataset for validation purpose.

```
train_datagen = ImageDataGenerator(rescale=1.0/255,
                                   zoom_range = 0.2,
                                   width_shift_range = 0.2,
                                   height_shift_range = 0.2,
                                   vertical_flip = True,
                                   fill_mode = 'nearest',
                                   validation_split = 0.1)

train_generator = train_datagen.flow_from_directory(train_dataset_path,
                                                    target_size = (IMG_WIDTH, IMG_HEIGHT),
                                                    batch_size = BATCH_SIZE,
                                                    class_mode = 'binary',
                                                    shuffle = True,
                                                    subset = 'training',
                                                    seed = 2)
```

```
Found 144 images belonging to 2 classes.
```

```
validation_generator = train_datagen.flow_from_directory(train_dataset_path,
                                                         target_size = (IMG_WIDTH, IMG_HEIGHT),
                                                         batch_size = BATCH_SIZE,
                                                         class_mode = 'binary',
                                                         subset = 'validation',
                                                         shuffle = True,
                                                         seed = 2)
```

```
Found 16 images belonging to 2 classes.
```

Loading the test dataset.

```
test_datagen = ImageDataGenerator(rescale = 1.0/ 255)

test_generator = test_datagen.flow_from_directory(test_dataset_path,
                                                  target_size = (IMG_WIDTH, IMG_HEIGHT),
                                                  batch_size = BATCH_SIZE,
                                                  class_mode = 'binary',
                                                  shuffle = False)
```

```
Found 40 images belonging to 2 classes.
```

iii. Get the Label Mappings

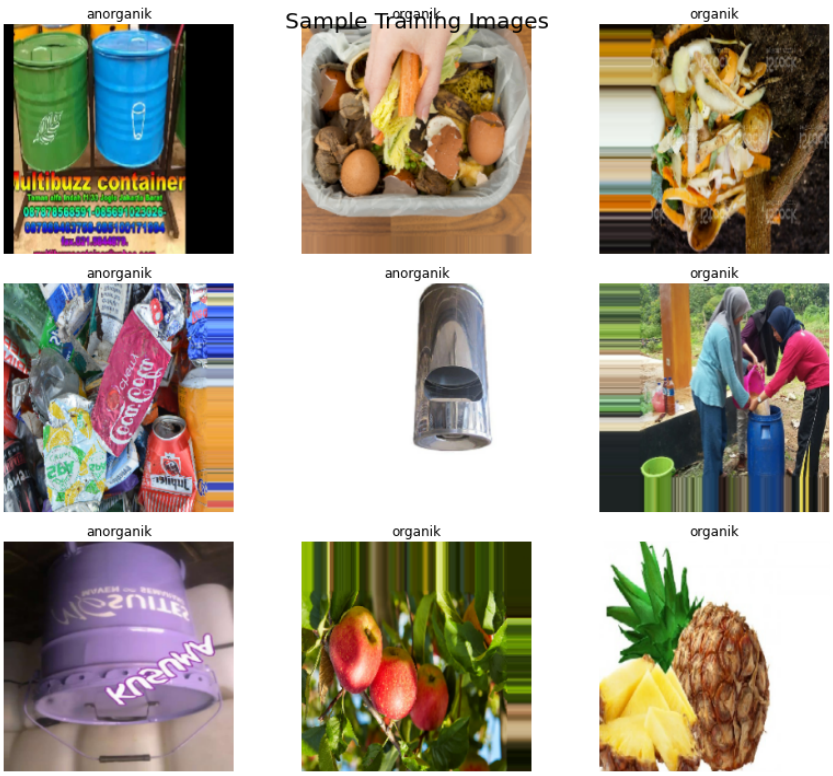
The labels dictionary is made in order to retrive the class names against the label indices used for training the model

```
labels = {value: key for key, value in train_generator.class_indices.items()}

print("Label Mappings for classes present in the training and validation datasets\n")
for key, value in labels.items():
    print(f"{key} : {value}")
```

```
Label Mappings for classes present in the training and validation datasets

0 : anorganik
1 : organik
```

3. Plotting Sample Training Images

```
fig, ax = plt.subplots(nrows = 3, ncols = 3, figsize = (12, 10))
idx = 0
plt.suptitle("Sample Training Images", fontsize = 20)
for i in range(3):
    for j in range(3):
        label = labels[train_generator[0][1][idx]]
        ax[i, j].set_title(f"{label}")
        ax[i, j].imshow(train_generator[0][0][idx][:, :, :])
        ax[i, j].axis("off")
        idx += 1

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



4. Training a Pretrained Model Using Transfer Learning

Since the training, validation and test set is available, let's first create a new model using the VGG16 pretrained model and train it on our dataset.

i. Creating the Model

```
base_model = VGG19(input_shape = IMG_SHAPE, include_top=False, weights='imagenet')
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg19/vgg19_weights_tf_dim_ordering_tf_kernels_n
80142336/80134624 [==============================] - 0s 0us/step
```

```
80150528/80134624 [==============================] - 0s 0us/step
```

```
base_model.summary()
```

```
Model: "vgg19"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 256, 256, 3)]     0

 block1_conv1 (Conv2D)       (None, 256, 256, 64)      1792

 block1_conv2 (Conv2D)       (None, 256, 256, 64)      36928

 block1_pool (MaxPooling2D)  (None, 128, 128, 64)      0

 block2_conv1 (Conv2D)       (None, 128, 128, 128)     73856

 block2_conv2 (Conv2D)       (None, 128, 128, 128)     147584

 block2_pool (MaxPooling2D)  (None, 64, 64, 128)       0

 block3_conv1 (Conv2D)       (None, 64, 64, 256)       295168

 block3_conv2 (Conv2D)       (None, 64, 64, 256)       590080

 block3_conv3 (Conv2D)       (None, 64, 64, 256)       590080

 block3_conv4 (Conv2D)       (None, 64, 64, 256)       590080

 block3_pool (MaxPooling2D)  (None, 32, 32, 256)       0

 block4_conv1 (Conv2D)       (None, 32, 32, 512)       1180160

 block4_conv2 (Conv2D)       (None, 32, 32, 512)       2359808

 block4_conv3 (Conv2D)       (None, 32, 32, 512)       2359808

 block4_conv4 (Conv2D)       (None, 32, 32, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 16, 16, 512)       0

 block5_conv1 (Conv2D)       (None, 16, 16, 512)       2359808

 block5_conv2 (Conv2D)       (None, 16, 16, 512)       2359808

 block5_conv3 (Conv2D)       (None, 16, 16, 512)       2359808

 block5_conv4 (Conv2D)       (None, 16, 16, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 8, 8, 512)         0

=================================================================
Total params: 20,024,384
Trainable params: 20,024,384
Non-trainable params: 0
_________________________________________________________________
```

Freeze all the top layers of the base model

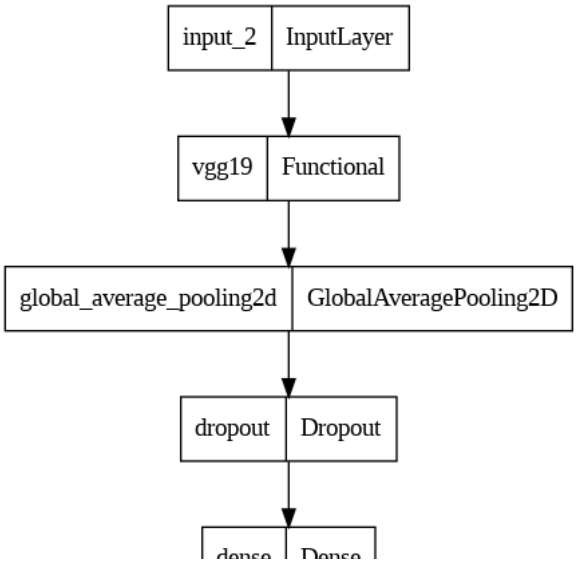
```
base_model.trainable = False
```

Since the base model contains only the convolution layers, let's add a classification layers at the bottom and the model will be trained only on the last layers (classification layer).

```
inputs = Input(shape = IMG_SHAPE)

x = base_model(inputs, training = False)
x = GlobalAveragePooling2D()(x)
x = Dropout(0.5)(x)
outputs = Dense(1, activation='sigmoid')(x)

model = Model(inputs = inputs, outputs = outputs)
```

This is how the model looks. We have added a Global Average Pooling layer after the VGG Model to flatten the output and then a Dropout and Dense Layer to predict the classes.

```
tf.keras.utils.plot_model(model)
```



ii. Defining Callbacks (Optional)

A callback is an object that can perform actions at various stages of training (e.g. at the start or end of an epoch, before or after a single batch, etc).

a. Reduce Learning Rate on Plateau

```
reduce_lr = ReduceLROnPlateau(monitor='val_loss',
                              factor=np.sqrt(0.1),
                               patience=5)
```

+ Code + Text

iii. Defining the Optimizer

```
base_learning_rate = 0.1
optimizer = Adam(learning_rate=base_learning_rate)
```

iv. Compile the Model

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

Let's first see the model's accuracy and loss before training the model

```
loss_0, accuracy_0 = model.evaluate(test_generator)

print(f"Initial Loss: {loss_0:.2f}")
print(f"Initial Accuracy: {accuracy_0: .2f}")
```

```
2/2 [==============================] - 11s 1s/step - loss: 0.7813 - accuracy: 0.5000
Initial Loss: 0.78
Initial Accuracy:  0.50
```

v. Training the Model

```
initial_epochs = 30

history = model.fit(
    train_generator,
    epochs=initial_epochs,
    validation_data=validation_generator,
    callbacks=[reduce_lr]
)
```

```
Epoch 2/30
5/5 [==============================] - 7s 1s/step - loss: 0.9075 - accuracy: 0.5972 - val_loss: 0.3992 - val_accuracy: 0.8750 - l
Epoch 3/30
```

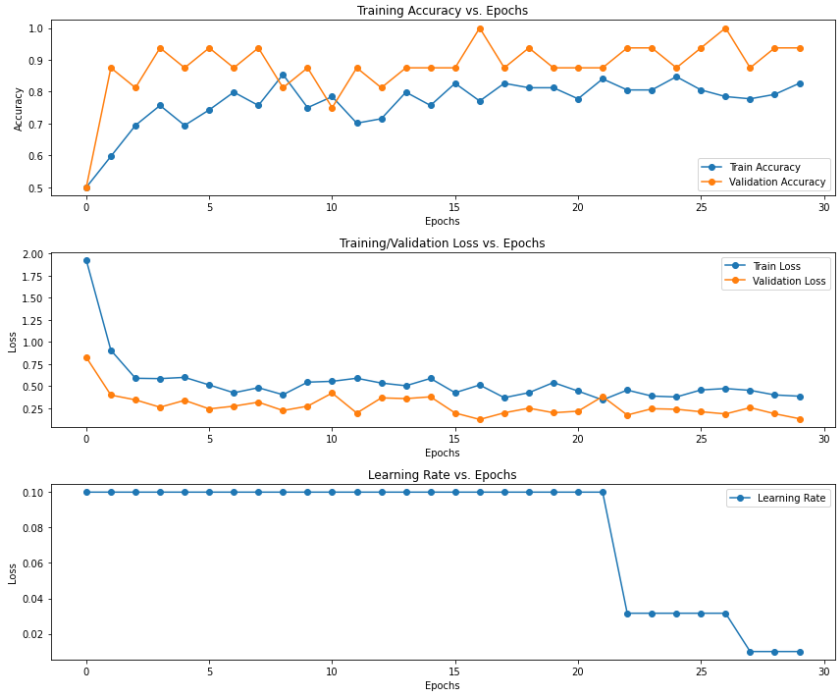
```
5/5 [==============================] - 6s 1s/step - loss: 0.5855 - accuracy: 0.7569 - val_loss: 0.2616 - val_accuracy: 0.9375 - l
Epoch 5/30
5/5 [==============================] - 6s 977ms/step - loss: 0.5997 - accuracy: 0.6944 - val_loss: 0.3395 - val_accuracy: 0.8750
Epoch 6/30
5/5 [==============================] - 7s 2s/step - loss: 0.5139 - accuracy: 0.7431 - val_loss: 0.2426 - val_accuracy: 0.9375 - l
Epoch 7/30
5/5 [==============================] - 6s 1s/step - loss: 0.4251 - accuracy: 0.7986 - val_loss: 0.2736 - val_accuracy: 0.8750 - l
Epoch 8/30
5/5 [==============================] - 6s 945ms/step - loss: 0.4826 - accuracy: 0.7569 - val_loss: 0.3187 - val_accuracy: 0.9375
Epoch 9/30
5/5 [==============================] - 5s 954ms/step - loss: 0.4033 - accuracy: 0.8542 - val_loss: 0.2242 - val_accuracy: 0.8125
Epoch 10/30
5/5 [==============================] - 7s 1s/step - loss: 0.5454 - accuracy: 0.7500 - val_loss: 0.2734 - val_accuracy: 0.8750 - l
Epoch 11/30
5/5 [==============================] - 5s 1s/step - loss: 0.5548 - accuracy: 0.7847 - val_loss: 0.4221 - val_accuracy: 0.7500 - l
Epoch 12/30
5/5 [==============================] - 7s 1s/step - loss: 0.5898 - accuracy: 0.7014 - val_loss: 0.1955 - val_accuracy: 0.8750 - l
Epoch 13/30
5/5 [==============================] - 6s 1s/step - loss: 0.5352 - accuracy: 0.7153 - val_loss: 0.3673 - val_accuracy: 0.8125 - l
Epoch 14/30
5/5 [==============================] - 7s 2s/step - loss: 0.5048 - accuracy: 0.7986 - val_loss: 0.3592 - val_accuracy: 0.8750 - l
Epoch 15/30
5/5 [==============================] - 6s 1s/step - loss: 0.5877 - accuracy: 0.7569 - val_loss: 0.3792 - val_accuracy: 0.8750 - l
Epoch 16/30
5/5 [==============================] - 6s 1s/step - loss: 0.4259 - accuracy: 0.8264 - val_loss: 0.1961 - val_accuracy: 0.8750 - l
Epoch 17/30
5/5 [==============================] - 7s 1s/step - loss: 0.5135 - accuracy: 0.7708 - val_loss: 0.1231 - val_accuracy: 1.0000 - l
Epoch 18/30
5/5 [==============================] - 6s 1s/step - loss: 0.3697 - accuracy: 0.8264 - val_loss: 0.1982 - val_accuracy: 0.8750 - l
Epoch 19/30
5/5 [==============================] - 7s 2s/step - loss: 0.4262 - accuracy: 0.8125 - val_loss: 0.2508 - val_accuracy: 0.9375 - l
Epoch 20/30
5/5 [==============================] - 6s 1s/step - loss: 0.5412 - accuracy: 0.8125 - val_loss: 0.1992 - val_accuracy: 0.8750 - l
Epoch 21/30
5/5 [==============================] - 6s 1s/step - loss: 0.4443 - accuracy: 0.7778 - val_loss: 0.2165 - val_accuracy: 0.8750 - l
Epoch 22/30
5/5 [==============================] - 7s 2s/step - loss: 0.3442 - accuracy: 0.8403 - val_loss: 0.3836 - val_accuracy: 0.8750 - l
Epoch 23/30
5/5 [==============================] - 5s 987ms/step - loss: 0.4560 - accuracy: 0.8056 - val_loss: 0.1731 - val_accuracy: 0.9375
Epoch 24/30
5/5 [==============================] - 7s 1s/step - loss: 0.3878 - accuracy: 0.8056 - val_loss: 0.2446 - val_accuracy: 0.9375 - l
Epoch 25/30
5/5 [==============================] - 5s 1s/step - loss: 0.3778 - accuracy: 0.8472 - val_loss: 0.2387 - val_accuracy: 0.8750 - l
Epoch 26/30
5/5 [==============================] - 7s 1s/step - loss: 0.4574 - accuracy: 0.8056 - val_loss: 0.2104 - val_accuracy: 0.9375 - l
Epoch 27/30
5/5 [==============================] - 6s 1s/step - loss: 0.4730 - accuracy: 0.7847 - val_loss: 0.1857 - val_accuracy: 1.0000 - l
Epoch 28/30
5/5 [==============================] - 6s 1s/step - loss: 0.4528 - accuracy: 0.7778 - val_loss: 0.2570 - val_accuracy: 0.8750 - l
Epoch 29/30
5/5 [==============================] - 5s 1s/step - loss: 0.4005 - accuracy: 0.7917 - val_loss: 0.1875 - val_accuracy: 0.9375 - l
Epoch 30/30
```

5. Plotting the Model Metrics

i. Plotting the training and validation accuracy, loss and learning rate

```
train_accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']

train_loss = history.history['loss']
val_loss = history.history['val_loss']

learning_rate = history.history['lr']


fig, ax = plt.subplots(nrows=3, ncols=1, figsize=(12, 10))

ax[0].set_title('Training Accuracy vs. Epochs')
ax[0].plot(train_accuracy, 'o-', label='Train Accuracy')
ax[0].plot(val_accuracy, 'o-', label='Validation Accuracy')
ax[0].set_xlabel('Epochs')
ax[0].set_ylabel('Accuracy')
ax[0].legend(loc='best')

ax[1].set_title('Training/Validation Loss vs. Epochs')
ax[1].plot(train_loss, 'o-', label='Train Loss')
ax[1].plot(val_loss, 'o-', label='Validation Loss')
ax[1].set_xlabel('Epochs')
ax[1].set_ylabel('Loss')
ax[1].legend(loc='best')

ax[2].set_title('Learning Rate vs. Epochs')
ax[2].plot(learning_rate, 'o-', label='Learning Rate')
ax[2].set_xlabel('Epochs')
ax[2].set_ylabel('Loss')
ax[2].legend(loc='best')
```

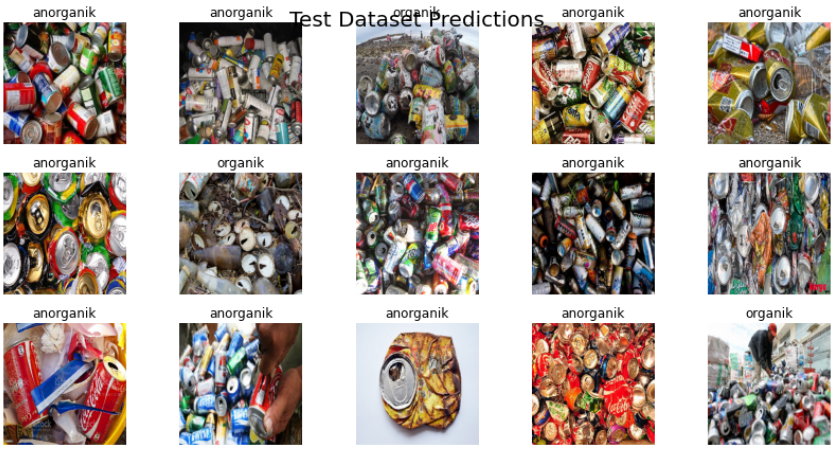
```
plt.tight_layout()
plt.show()
```



6. Testing the Model on Test dataset

```
predictions = model.predict(test_generator)
```

```
fig, ax = plt.subplots(nrows=5, ncols=5, figsize=(12, 10))
idx = 0
plt.suptitle('Test Dataset Predictions', fontsize=20)

for i in range(5):
    for j in range(5):
        predicted_label = labels[np.round(predictions.T[0][idx])]
        ax[i, j].set_title(f"{predicted_label}")
        ax[i, j].imshow(test_generator[0][0][idx])
        ax[i, j].axis("off")
        idx += 1

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



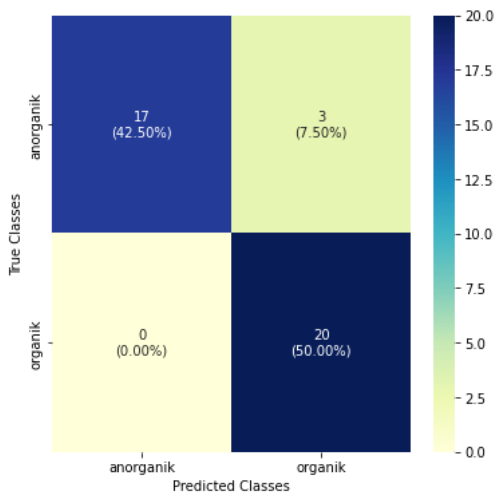

```
test_loss, test_accuracy = model.evaluate(test_generator, batch_size=BATCH_SIZE)

print(f"Test Loss:     {test_loss}")
print(f"Test Accuracy: {test_accuracy}")
```

```
2/2 [==============================] - 2s 85ms/step - loss: 0.1365 - accuracy: 0.9250
Test Loss:     0.1365206390619278
Test Accuracy: 0.925000011920929
```

6. Plotting the Classification Metrics

i. Confusion Matrix

```
y_pred = np.round(predictions)
y_pred = y_pred.T[0]          # to make the shape of y_true and y_pred same
y_true = test_generator.classes


cf_mtx = confusion_matrix(y_true, y_pred)

group_counts = ["{0:0.0f}".format(value) for value in cf_mtx.flatten()]
group_percentages = ["{0:.2%}".format(value) for value in cf_mtx.flatten()/np.sum(cf_mtx)]
box_labels = [f"{v1}\n({v2})" for v1, v2 in zip(group_counts, group_percentages)]
box_labels = np.asarray(box_labels).reshape(2, 2)

plt.figure(figsize = (6, 6))
sns.heatmap(cf_mtx, xticklabels=labels.values(), yticklabels=labels.values(),
            cmap="YlGnBu", fmt="", annot=box_labels)

plt.xlabel('Predicted Classes')
plt.ylabel('True Classes')
plt.show()
```




```
print(classification_report(y_true, y_pred, target_names=labels.values()))
```

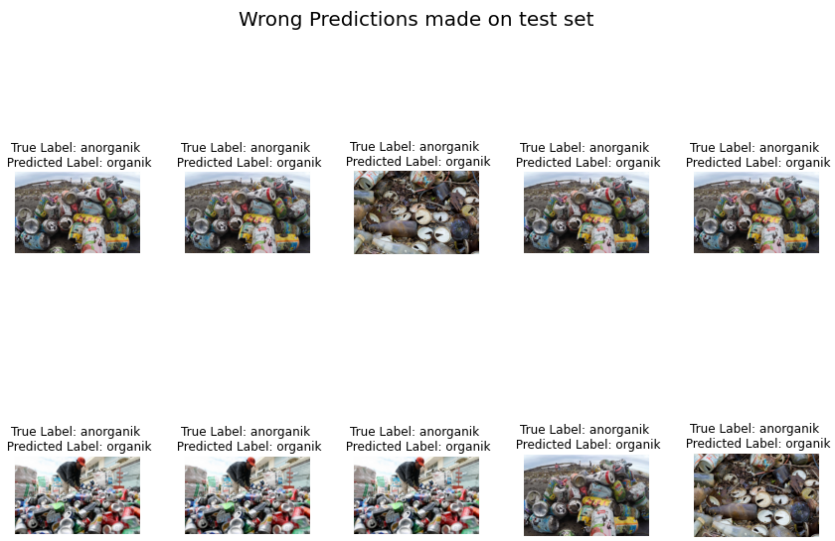
```
              precision    recall  f1-score   support

   anorganik       1.00      0.85      0.92        20
     organik       0.87      1.00      0.93        20

    accuracy                           0.93        40
```

```
       macro avg       0.93      0.93      0.92        40
    weighted avg       0.93      0.93      0.92        40
```

9. Wrong Predictions

Let's see where the model has given wrong predictions and what were the actual predictions on those images.

```
errors = (y_true - y_pred != 0)
y_true_errors = y_true[errors]
y_pred_errors = y_pred[errors]


test_images = test_generator.filenames
test_img = np.asarray(test_images)[errors]


import cv2

fig, ax = plt.subplots(nrows=2, ncols=5, figsize=(12, 10))
idx = 0

for i in range(2):
    for j in range(5):
        idx = np.random.randint(0, len(test_img))
        true_index = y_true_errors[idx]
        true_label = labels[true_index]
        predicted_index = y_pred_errors[idx]
        predicted_label = labels[predicted_index]
        ax[i, j].set_title(f"True Label: {true_label} \n Predicted Label: {predicted_label}")
        img_path = os.path.join(test_dataset_path, test_img[idx])
        img = cv2.imread(img_path)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        ax[i, j].imshow(img)
        ax[i, j].axis("off")

plt.tight_layout()
plt.suptitle('Wrong Predictions made on test set', fontsize=20)
plt.show()
```



I hope you found this notebook helpful. Suggestions are welcome.

UPVOTE if you like my work.

3s completed at 12:23 PM

In this notebook I have shown how we can use a pre-trained model to train a binary classifier to detects cats and dogs. I have covered

1. How to train a binary classifier using transfer learning on a pre-trained VGG16 model.
2. How to evaluate the model on test set using different classification metrics.
3. How to visualize the images present in the training and test sets.

I hope you find this notebook helpful and some **UPVOTES** would be very much appreciated.

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

1. Import the Required Libraries

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

import os
os.environ["TF_CPP_MIN_LOG_LEVEL"]="2"
import warnings
warnings.filterwarnings("ignore")

from sklearn.metrics import confusion_matrix,classification_report

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense, GlobalAveragePooling2D, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.losses import BinaryCrossentropy
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import VGG16, ResNet50
```

```
!pip install tensorflow==2.7.0
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: tensorflow==2.7.0 in /usr/local/lib/python3.8/dist-packages (2.7.0)
Requirement already satisfied: flatbuffers<3.0,>=1.12 in /usr/local/lib/python3.8/dist-packages (from tensorflow==2.7.0) (2.0.7)
Requirement already satisfied: protobuf>=3.9.2 in /usr/local/lib/python3.8/dist-packages (from tensorflow==2.7.0) (3.19.6)
Requirement already satisfied: tensorboard~=2.6 in /usr/local/lib/python3.8/dist-packages (from tensorflow==2.7.0) (2.11.2)
Requirement already satisfied: gast<0.5.0,>=0.2.1 in /usr/local/lib/python3.8/dist-packages (from tensorflow==2.7.0) (0.4.0)
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.8/dist-packages (from tensorflow==2.7.0) (0.2.0)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.8/dist-packages (from tensorflow==2.7.0) (1.15.0)
Requirement already satisfied: tensorflow-estimator<2.8,~=2.7.0rc0 in /usr/local/lib/python3.8/dist-packages (from tensorflow==2.7.
Requirement already satisfied: h5py>=2.9.0 in /usr/local/lib/python3.8/dist-packages (from tensorflow==2.7.0) (3.1.0)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.8/dist-packages (from tensorflow==2.7.0) (3.3.0)
Requirement already satisfied: keras<2.8,>=2.7.0rc0 in /usr/local/lib/python3.8/dist-packages (from tensorflow==2.7.0) (2.7.0)
Requirement already satisfied: numpy>=1.14.5 in /usr/local/lib/python3.8/dist-packages (from tensorflow==2.7.0) (1.21.6)
Requirement already satisfied: keras-preprocessing>=1.1.1 in /usr/local/lib/python3.8/dist-packages (from tensorflow==2.7.0) (1.1.2
Requirement already satisfied: absl-py>=0.4.0 in /usr/local/lib/python3.8/dist-packages (from tensorflow==2.7.0) (1.4.0)
Requirement already satisfied: libclang>=9.0.1 in /usr/local/lib/python3.8/dist-packages (from tensorflow==2.7.0) (15.0.6.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.21.0 in /usr/local/lib/python3.8/dist-packages (from tensorflow==2.7
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.8/dist-packages (from tensorflow==2.7.0) (1.6.3)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.8/dist-packages (from tensorflow==2.7.0) (4.4.0)
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.8/dist-packages (from tensorflow==2.7.0) (1.14.1)
Requirement already satisfied: wheel<1.0,>=0.32.0 in /usr/local/lib/python3.8/dist-packages (from tensorflow==2.7.0) (0.38.4)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.8/dist-packages (from tensorflow==2.7.0) (2.2.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.8/dist-packages (from tensorflow==2.7.0) (1.51.1)
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.8/dist-packages (from tensorboard~=2.6->tensorflow==2.
Requirement already satisfied: google-auth-oauthlib<0.5,>=0.4.1 in /usr/local/lib/python3.8/dist-packages (from tensorboard~=2.6->t
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.8/dist-packages (from tensorboard~=2.6->tensorflow==2.7.0)
Requirement already satisfied: google-auth<3,>=1.6.3 in /usr/local/lib/python3.8/dist-packages (from tensorboard~=2.6->tensorflow==
Requirement already satisfied: setuptools>=41.0.0 in /usr/local/lib/python3.8/dist-packages (from tensorboard~=2.6->tensorflow==2.7
Requirement already satisfied: tensorboard-data-server<0.7.0,>=0.6.0 in /usr/local/lib/python3.8/dist-packages (from tensorboard~=2
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.8/dist-packages (from tensorboard~=2.6->tensorflow==2.7.0)
Requirement already satisfied: tensorboard-plugin-wit>=1.6.0 in /usr/local/lib/python3.8/dist-packages (from tensorboard~=2.6->tens
Requirement already satisfied: cachetools<6.0,>=2.0.0 in /usr/local/lib/python3.8/dist-packages (from google-auth<3,>=1.6.3->tensor
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.8/dist-packages (from google-auth<3,>=1.6.3->tensorb
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.8/dist-packages (from google-auth<3,>=1.6.3->tensorboard~=2.
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python3.8/dist-packages (from google-auth-oauthlib<0.5,>=
Requirement already satisfied: importlib-metadata>=4.4 in /usr/local/lib/python3.8/dist-packages (from markdown>=2.6.8->tensorboard
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.8/dist-packages (from requests<3,>=2.21.0->tensorboa
Requirement already satisfied: chardet<5,>=3.0.2 in /usr/local/lib/python3.8/dist-packages (from requests<3,>=2.21.0->tensorboard~=
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.8/dist-packages (from requests<3,>=2.21.0->tensorboard~
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.8/dist-packages (from requests<3,>=2.21.0->tensorboard~=2.6->
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.8/dist-packages (from importlib-metadata>=4.4->markdown>=2.6.8->
Requirement already satisfied: pyasn1<0.5.0,>=0.4.6 in /usr/local/lib/python3.8/dist-packages (from pyasn1-modules>=0.2.1->google-a
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.8/dist-packages (from requests-oauthlib>=0.7.0->google-aut
```

2. Load the Image Training, Validation and Testing Datasets

i. Get the Image Datasets Paths

```
train_dataset_path = "/content/drive/MyDrive/data/train"
test_dataset_path = "/content/drive/MyDrive/data/test"
```

ii. Load Image Datasets and Apply Image Augmentations

Before feeding the image in the model, we need to resize the images present in the datasets to a fixed size. I have choosen the image width and height as 256x256px .

```
BATCH_SIZE = 32
IMG_WIDTH = 256
IMG_HEIGHT = 256
IMG_CHANNELS = 3
IMG_SHAPE = (IMG_WIDTH, IMG_HEIGHT, IMG_CHANNELS)

print(f"Batch Size: {BATCH_SIZE}")
print(f"Image Shape: {IMG_SHAPE}")
```

```
Batch Size: 32
Image Shape: (256, 256, 3)
```

Loading the training dataset and applying augmentations to it.

Also since there is no validation dataset available, we will keep 10% of the training dataset for validation purpose.

```
train_datagen = ImageDataGenerator(rescale=1.0/255,
                                   zoom_range = 0.2,
                                   width_shift_range = 0.2,
                                   height_shift_range = 0.2,
                                   vertical_flip = True,
                                   fill_mode = 'nearest',
                                   validation_split = 0.1)

train_generator = train_datagen.flow_from_directory(train_dataset_path,
                                                    target_size = (IMG_WIDTH, IMG_HEIGHT),
                                                    batch_size = BATCH_SIZE,
                                                    class_mode = 'binary',
                                                    shuffle = True,
                                                    subset = 'training',
                                                    seed = 2)
```

```
Found 144 images belonging to 2 classes.
```

```
validation_generator = train_datagen.flow_from_directory(train_dataset_path,
                                                         target_size = (IMG_WIDTH, IMG_HEIGHT),
                                                         batch_size = BATCH_SIZE,
                                                         class_mode = 'binary',
                                                         subset = 'validation',
                                                         shuffle = True,
                                                         seed = 2)
```

```
Found 16 images belonging to 2 classes.
```

Loading the test dataset.

```
test_datagen = ImageDataGenerator(rescale = 1.0/ 255)

test_generator = test_datagen.flow_from_directory(test_dataset_path,
                                                  target_size = (IMG_WIDTH, IMG_HEIGHT),
                                                  batch_size = BATCH_SIZE,
                                                  class_mode = 'binary',
                                                  shuffle = False)
```

```
Found 40 images belonging to 2 classes.
```

iii. Get the Label Mappings

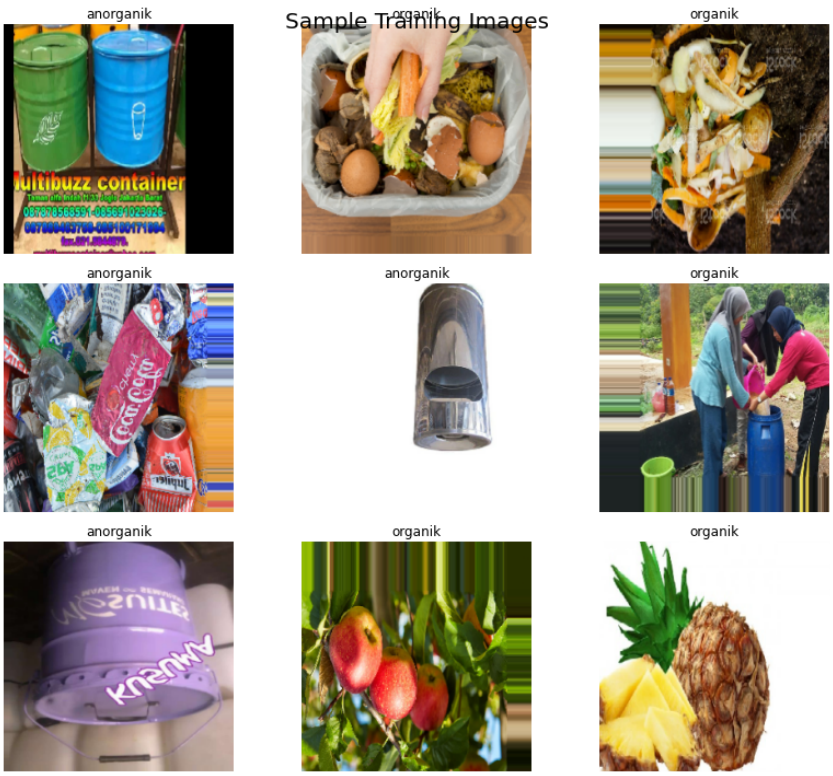
The labels dictionary is made in order to retrive the class names against the label indices used for training the model

```
labels = {value: key for key, value in train_generator.class_indices.items()}

print("Label Mappings for classes present in the training and validation datasets\n")
for key, value in labels.items():
    print(f"{key} : {value}")
```

```
Label Mappings for classes present in the training and validation datasets

0 : anorganik
1 : organik
```

3. Plotting Sample Training Images

```
fig, ax = plt.subplots(nrows = 3, ncols = 3, figsize = (12, 10))
idx = 0
plt.suptitle("Sample Training Images", fontsize = 20)
for i in range(3):
    for j in range(3):
        label = labels[train_generator[0][1][idx]]
        ax[i, j].set_title(f"{label}")
        ax[i, j].imshow(train_generator[0][0][idx][:, :, :])
        ax[i, j].axis("off")
        idx += 1

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



4. Training a Pretrained Model Using Transfer Learning

Since the training, validation and test set is available, let's first create a new model using the VGG16 pretrained model and train it on our dataset.

i. Creating the Model

```
base_model = VGG16(input_shape = IMG_SHAPE, include_top=False, weights='imagenet')
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_n
58892288/58889256 [==============================] - 0s 0us/step
```

```
58900480/58889256 [==============================] - 0s 0us/step
```

```
base_model.summary()
```

```
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 256, 256, 3)]     0

 block1_conv1 (Conv2D)       (None, 256, 256, 64)      1792

 block1_conv2 (Conv2D)       (None, 256, 256, 64)      36928

 block1_pool (MaxPooling2D)  (None, 128, 128, 64)      0

 block2_conv1 (Conv2D)       (None, 128, 128, 128)     73856

 block2_conv2 (Conv2D)       (None, 128, 128, 128)     147584

 block2_pool (MaxPooling2D)  (None, 64, 64, 128)       0

 block3_conv1 (Conv2D)       (None, 64, 64, 256)       295168

 block3_conv2 (Conv2D)       (None, 64, 64, 256)       590080

 block3_conv3 (Conv2D)       (None, 64, 64, 256)       590080

 block3_pool (MaxPooling2D)  (None, 32, 32, 256)       0

 block4_conv1 (Conv2D)       (None, 32, 32, 512)       1180160

 block4_conv2 (Conv2D)       (None, 32, 32, 512)       2359808

 block4_conv3 (Conv2D)       (None, 32, 32, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 16, 16, 512)       0

 block5_conv1 (Conv2D)       (None, 16, 16, 512)       2359808

 block5_conv2 (Conv2D)       (None, 16, 16, 512)       2359808

 block5_conv3 (Conv2D)       (None, 16, 16, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 8, 8, 512)         0

=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
_________________________________________________________________
```

Freeze all the top layers of the base model

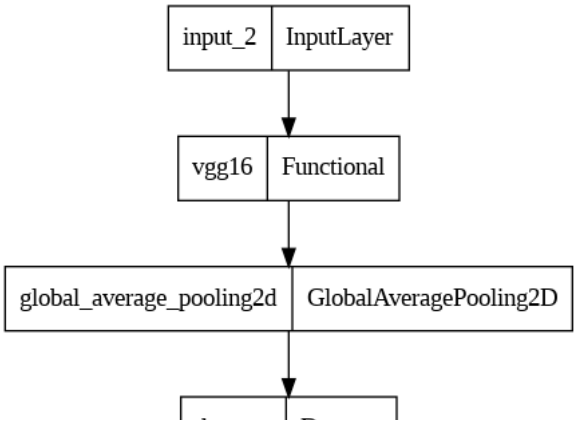
```
base_model.trainable = False
```

Since the base model contains only the convolution layers, let's add a classification layers at the bottom and the model will be trained only on the last layers (classification layer).

```
inputs = Input(shape = IMG_SHAPE)

x = base_model(inputs, training = False)
x = GlobalAveragePooling2D()(x)
x = Dropout(0.5)(x)
outputs = Dense(1, activation='sigmoid')(x)

model = Model(inputs = inputs, outputs = outputs)
```

This is how the model looks. We have added a Global Average Pooling layer after the VGG Model to flatten the output and then a Dropout and Dense Layer to predict the classes.

```
tf.keras.utils.plot_model(model)
```



ii. Defining Callbacks (Optional)

A callback is an object that can perform actions at various stages of training (e.g. at the start or end of an epoch, before or after a single batch, etc).

a. Reduce Learning Rate on Plateau

```
reduce_lr = ReduceLROnPlateau(monitor='val_loss',
                              factor=np.sqrt(0.1),
                               patience=5)
```

iii. Defining the Optimizer

```
base_learning_rate = 0.1
optimizer = Adam(learning_rate=base_learning_rate)
```

iv. Compile the Model

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

Let's first see the model's accuracy and loss before training the model

```
loss_0, accuracy_0 = model.evaluate(test_generator)

print(f"Initial Loss: {loss_0:.2f}")
print(f"Initial Accuracy: {accuracy_0: .2f}")
```

```
2/2 [==============================] - 12s 1s/step - loss: 0.8943 - accuracy: 0.5000
Initial Loss: 0.89
Initial Accuracy:  0.50
```

v. Training the Model

```
initial_epochs = 30

history = model.fit(
    train_generator,
    epochs=initial_epochs,
    validation_data=validation_generator,
    callbacks=[reduce_lr]
)
```

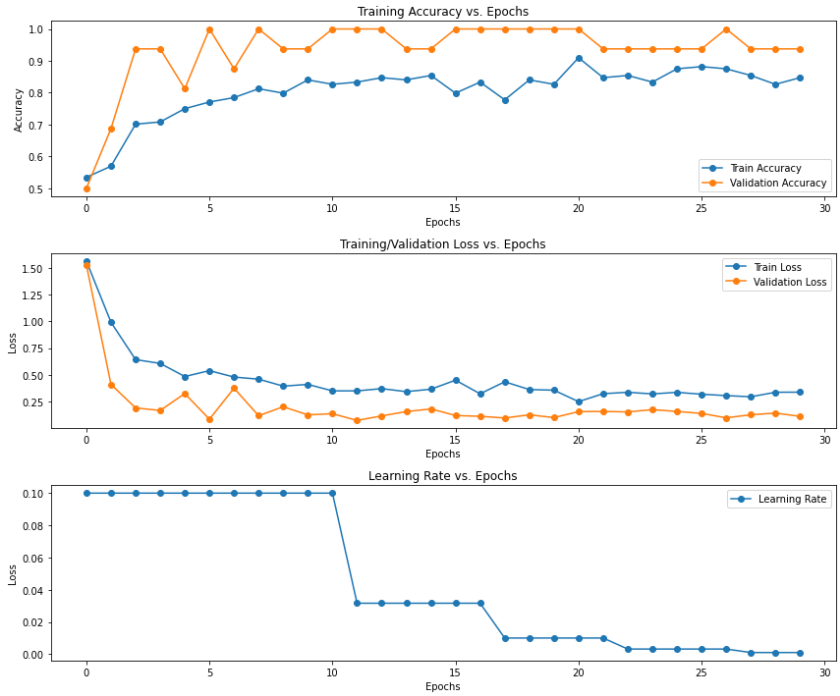
```
Epoch 2/30
5/5 [==============================] - 7s 1s/step - loss: 0.9917 - accuracy: 0.5694 - val_loss: 0.4123 - val_accuracy: 0.6875 - l
Epoch 3/30
5/5 [==============================] - 5s 973ms/step - loss: 0.6437 - accuracy: 0.7014 - val_loss: 0.1929 - val_accuracy: 0.9375
Epoch 4/30
5/5 [==============================] - 7s 1s/step - loss: 0.6078 - accuracy: 0.7083 - val_loss: 0.1693 - val_accuracy: 0.9375 - l
Epoch 5/30
5/5 [==============================] - 5s 920ms/step - loss: 0.4859 - accuracy: 0.7500 - val_loss: 0.3268 - val_accuracy: 0.8125
Epoch 6/30
```

```
5/5 [==============================] - 5s 921ms/step - loss: 0.4807 - accuracy: 0.7847 - val_loss: 0.3768 - val_accuracy: 0.8750
Epoch 8/30
5/5 [==============================] - 5s 1s/step - loss: 0.4611 - accuracy: 0.8125 - val_loss: 0.1202 - val_accuracy: 1.0000 - l
Epoch 9/30
5/5 [==============================] - 7s 1s/step - loss: 0.3962 - accuracy: 0.7986 - val_loss: 0.2041 - val_accuracy: 0.9375 - l
Epoch 10/30
5/5 [==============================] - 5s 1s/step - loss: 0.4122 - accuracy: 0.8403 - val_loss: 0.1281 - val_accuracy: 0.9375 - l
Epoch 11/30
5/5 [==============================] - 6s 1s/step - loss: 0.3519 - accuracy: 0.8264 - val_loss: 0.1391 - val_accuracy: 1.0000 - l
Epoch 12/30
5/5 [==============================] - 6s 1s/step - loss: 0.3520 - accuracy: 0.8333 - val_loss: 0.0761 - val_accuracy: 1.0000 - l
Epoch 13/30
5/5 [==============================] - 5s 1s/step - loss: 0.3729 - accuracy: 0.8472 - val_loss: 0.1185 - val_accuracy: 1.0000 - l
Epoch 14/30
5/5 [==============================] - 7s 2s/step - loss: 0.3448 - accuracy: 0.8403 - val_loss: 0.1586 - val_accuracy: 0.9375 - l
Epoch 15/30
5/5 [==============================] - 5s 997ms/step - loss: 0.3674 - accuracy: 0.8542 - val_loss: 0.1837 - val_accuracy: 0.9375
Epoch 16/30
5/5 [==============================] - 5s 1s/step - loss: 0.4517 - accuracy: 0.7986 - val_loss: 0.1222 - val_accuracy: 1.0000 - l
Epoch 17/30
5/5 [==============================] - 5s 1s/step - loss: 0.3247 - accuracy: 0.8333 - val_loss: 0.1147 - val_accuracy: 1.0000 - l
Epoch 18/30
5/5 [==============================] - 6s 1s/step - loss: 0.4366 - accuracy: 0.7778 - val_loss: 0.0989 - val_accuracy: 1.0000 - l
Epoch 19/30
5/5 [==============================] - 6s 1s/step - loss: 0.3643 - accuracy: 0.8403 - val_loss: 0.1287 - val_accuracy: 1.0000 - l
Epoch 20/30
5/5 [==============================] - 5s 1s/step - loss: 0.3582 - accuracy: 0.8264 - val_loss: 0.1027 - val_accuracy: 1.0000 - l
Epoch 21/30
5/5 [==============================] - 7s 1s/step - loss: 0.2519 - accuracy: 0.9097 - val_loss: 0.1584 - val_accuracy: 1.0000 - l
Epoch 22/30
5/5 [==============================] - 5s 1s/step - loss: 0.3256 - accuracy: 0.8472 - val_loss: 0.1600 - val_accuracy: 0.9375 - l
Epoch 23/30
5/5 [==============================] - 5s 934ms/step - loss: 0.3388 - accuracy: 0.8542 - val_loss: 0.1557 - val_accuracy: 0.9375
Epoch 24/30
5/5 [==============================] - 5s 985ms/step - loss: 0.3235 - accuracy: 0.8333 - val_loss: 0.1767 - val_accuracy: 0.9375
Epoch 25/30
5/5 [==============================] - 6s 1s/step - loss: 0.3382 - accuracy: 0.8750 - val_loss: 0.1601 - val_accuracy: 0.9375 - l
Epoch 26/30
5/5 [==============================] - 6s 1s/step - loss: 0.3207 - accuracy: 0.8819 - val_loss: 0.1414 - val_accuracy: 0.9375 - l
Epoch 27/30
5/5 [==============================] - 5s 1s/step - loss: 0.3078 - accuracy: 0.8750 - val_loss: 0.1006 - val_accuracy: 1.0000 - l
Epoch 28/30
5/5 [==============================] - 7s 1s/step - loss: 0.2968 - accuracy: 0.8542 - val_loss: 0.1291 - val_accuracy: 0.9375 - l
Epoch 29/30
5/5 [==============================] - 5s 1s/step - loss: 0.3390 - accuracy: 0.8264 - val_loss: 0.1446 - val_accuracy: 0.9375 - l
Epoch 30/30
```

5. Plotting the Model Metrics

i. Plotting the training and validation accuracy, loss and learning rate

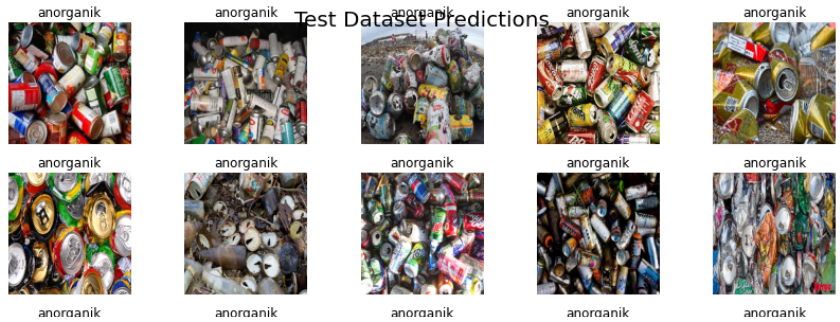
```
train_accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']

train_loss = history.history['loss']
val_loss = history.history['val_loss']

learning_rate = history.history['lr']


fig, ax = plt.subplots(nrows=3, ncols=1, figsize=(12, 10))

ax[0].set_title('Training Accuracy vs. Epochs')
ax[0].plot(train_accuracy, 'o-', label='Train Accuracy')
ax[0].plot(val_accuracy, 'o-', label='Validation Accuracy')
ax[0].set_xlabel('Epochs')
ax[0].set_ylabel('Accuracy')
ax[0].legend(loc='best')

ax[1].set_title('Training/Validation Loss vs. Epochs')
ax[1].plot(train_loss, 'o-', label='Train Loss')
ax[1].plot(val_loss, 'o-', label='Validation Loss')
ax[1].set_xlabel('Epochs')
ax[1].set_ylabel('Loss')
ax[1].legend(loc='best')

ax[2].set_title('Learning Rate vs. Epochs')
ax[2].plot(learning_rate, 'o-', label='Learning Rate')
ax[2].set_xlabel('Epochs')
ax[2].set_ylabel('Loss')
ax[2].legend(loc='best')

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



6. Testing the Model on Test dataset

```
predictions = model.predict(test_generator)
```

```
fig, ax = plt.subplots(nrows=5, ncols=5, figsize=(12, 10))
idx = 0
plt.suptitle('Test Dataset Predictions', fontsize=20)

for i in range(5):
    for j in range(5):
        predicted_label = labels[np.round(predictions.T[0][idx])]
        ax[i, j].set_title(f"{predicted_label}")
        ax[i, j].imshow(test_generator[0][0][idx])
        ax[i, j].axis("off")
        idx += 1

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



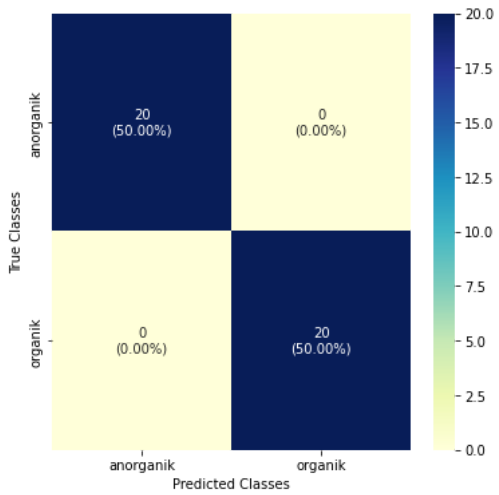

```
test_loss, test_accuracy = model.evaluate(test_generator, batch_size=BATCH_SIZE)

print(f"Test Loss:     {test_loss}")
print(f"Test Accuracy: {test_accuracy}")
```

```
2/2 [==============================] - 1s 96ms/step - loss: 0.0887 - accuracy: 1.0000
Test Loss:     0.08871717751026154
Test Accuracy: 1.0
```

6. Plotting the Classification Metrics

i. Confusion Matrix

```
y_pred = np.round(predictions)
y_pred = y_pred.T[0]          # to make the shape of y_true and y_pred same
y_true = test_generator.classes


cf_mtx = confusion_matrix(y_true, y_pred)

group_counts = ["{0:0.0f}".format(value) for value in cf_mtx.flatten()]
group_percentages = ["{0:.2%}".format(value) for value in cf_mtx.flatten()/np.sum(cf_mtx)]
box_labels = [f"{v1}\n({v2})" for v1, v2 in zip(group_counts, group_percentages)]
box_labels = np.asarray(box_labels).reshape(2, 2)

plt.figure(figsize = (6, 6))
sns.heatmap(cf_mtx, xticklabels=labels.values(), yticklabels=labels.values(),
           cmap="YlGnBu", fmt="", annot=box_labels)

plt.xlabel('Predicted Classes')
plt.ylabel('True Classes')
plt.show()
```




```
print(classification_report(y_true, y_pred, target_names=labels.values()))
```

```
              precision    recall  f1-score   support

   anorganik       1.00      1.00      1.00        20
     organik       1.00      1.00      1.00        20

    accuracy                           1.00        40
   macro avg       1.00      1.00      1.00        40
weighted avg       1.00      1.00      1.00        40
```

9. Wrong Predictions

Let's see where the model has given wrong predictions and what were the actual predictions on those images.

```
errors = (y_true - y_pred != 0)
y_true_errors = y_true[errors]
y_pred_errors = y_pred[errors]
```

```
test_images = test_generator.filenames
test_img = np.asarray(test_images)[errors]
```

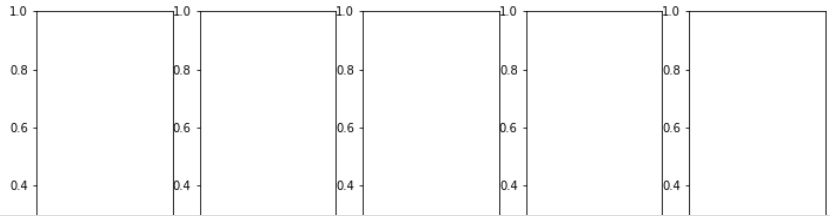
```
import cv2

fig, ax = plt.subplots(nrows=2, ncols=5, figsize=(12, 10))
idx = 0

for i in range(2):
    for j in range(5):
        idx = np.random.randint(0, len(test_img))
        true_index = y_true_errors[idx]
        true_label = labels[true_index]
        predicted_index = y_pred_errors[idx]
        predicted_label = labels[predicted_index]
        ax[i, j].set_title(f"True Label: {true_label} \n Predicted Label: {predicted_label}")
        img_path = os.path.join(test_dataset_path, test_img[idx])
        img = cv2.imread(img_path)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        ax[i, j].imshow(img)
        ax[i, j].axis("off")

plt.tight_layout()
plt.suptitle('Wrong Predictions made on test set', fontsize=20)
plt.show()
```

I hope you found this notebook helpful. Suggestions are welcome.

UPVOTE if you like my work.

```
----> 8         idx = np.random.randint(0, len(test_img))
      9         true_index = y_true_errors[idx]
     10         true_label = labels[true_index]

mtrand.pyx in numpy.random.mtrand.RandomState.randint()

_bounded_integers.pyx in numpy.random._bounded_integers._rand_int64()

ValueError: high <= 0
```

SEARCH STACK OVERFLOW



1s completed at 12:10 PM