In this notebook I have shown how we can use a pre-trained model to train a binary classifier to detect 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.

```python
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

```python
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os
import warnings
import tensorflow as tf
import tensorflow.keras.applications
import tensorflow.keras.preprocessing.image
import tensorflow.keras.callbacks
import tensorflow.keras.layers
import tensorflow.keras.models
import tensorflow
import sklearn.metrics
import seaborn as sns
import matplotlib.pyplot as plt
import numpy
import keras
```

How to evaluate the model on test set using different classification metrics.

```python
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount('/content/drive', force_remount=True).
```
2. Load the Image Training, Validation and Testing Datasets

i. Get the Image Datasets Paths

```python
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 chosen the image width and height as 256x256px.

```python
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.

```python
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.
```

```python
validation_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 = 'validation',
                                                          seed = 2)
```

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

Loading the test dataset.

```python
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 retrieve 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("{}: {}\n".format(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("{}\n".format(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_notop.h5 (88142336/88134624) - 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

```python
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).

```python
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.

```python
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

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

iii. Defining the Optimizer

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

iv. Compile the Model

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

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

```python
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

```python
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 - 1
Epoch 3/30
5. Plotting the Model Metrics

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

```python
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']
```

```python
fig, ax = plt.subplots(nrows=3, ncols=1, figsize=(12, 18))
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')
```
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()
```python
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

```python
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 = "{v1}
(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=True)
plt.xlabel('Predicted Classes')
plt.ylabel('True Classes')
plt.show()

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

<table>
<thead>
<tr>
<th></th>
<th>precision</th>
<th>recall</th>
<th>f1-score</th>
<th>support</th>
</tr>
</thead>
<tbody>
<tr>
<td>anorganik</td>
<td>1.00</td>
<td>0.85</td>
<td>0.92</td>
<td>20</td>
</tr>
<tr>
<td>organik</td>
<td>0.87</td>
<td>1.00</td>
<td>0.93</td>
<td>20</td>
</tr>
<tr>
<td>accuracy</td>
<td>0.93</td>
<td></td>
<td></td>
<td>40</td>
</tr>
</tbody>
</table>
```
<table>
<thead>
<tr>
<th></th>
<th>0.93</th>
<th>0.93</th>
<th>0.92</th>
<th>40</th>
</tr>
</thead>
<tbody>
<tr>
<td>weighted avg</td>
<td>0.93</td>
<td>0.93</td>
<td>0.92</td>
<td>40</td>
</tr>
</tbody>
</table>

### 9. Wrong Predictions

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

```python
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} 
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()
```

Wrong Predictions made on test set

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

**UPVOTE** if you like my work.
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
from plotly import 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.keras import Input, Dense, GlobalAveragePooling2D, Dropout
from tensorflow.keras.layers import Input, Dense, GlobalAveragePooling2D, Dropout
from tensorflow.keras.models import Model
from tensorflow.keras.applications import VGG16, ResNet50
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.models import Sequential
from tensorflow.keras.preprocessing import sequence, image

!pip install tensorflow==2.7.0


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<5.2.0;>3.9.2 in /usr/local/lib/python3.8/dist-packages (from tensorflow==2.7.0) (3.19.6)
Requirement already satisfied: tensorboard<2.0;>2.6.2 in /usr/local/lib/python3.8/dist-packages (from tensorflow==2.7.0) (2.11.2)
Requirement already satisfied: gast<0.10;>0.8.5 in /usr/local/lib/python3.8/dist-packages (from tensorflow==2.7.0) (0.0.4)
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: tensorflow==2.7.0 in /usr/local/lib/python3.8/dist-packages (2.7.0)

How to visualize the images present in the training and test sets.

How to evaluate the model on test set using different classification metrics.
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 chosen 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("Batch Size: {}".format(BATCH_SIZE))
print("Image Shape: {}".format(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',
                                                            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 retrieve 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

```python
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

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

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).

```python
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

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

iii. Defining the Optimizer

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

iv. Compile the Model

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

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

```python
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

```python
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 [==============================] - 7s 1s/step - loss: 0.9917 - accuracy: 0.5694 - val_loss: 0.4123 - val_accuracy: 0.6875 - l
Epoch 4/30
5/5 [==============================] - 5s 920ms/step - loss: 0.8878 - accuracy: 0.7883 - val_loss: 0.1929 - val_accuracy: 0.9375
Epoch 5/30
5/5 [==============================] - 5s 920ms/step - loss: 0.8878 - accuracy: 0.7883 - val_loss: 0.1929 - val_accuracy: 0.9375 - l
Epoch 6/30
5/5 [==============================] - 5s 920ms/step - loss: 0.8878 - accuracy: 0.7883 - val_loss: 0.1929 - val_accuracy: 0.9375 - l
5. Plotting the Model Metrics

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

```python
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']
```

```python
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('Learning Rate')
ax[2].legend(loc='best')

plt.tight_layout()
plt.show()
```
6. Testing the Model on Test dataset

```python
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("Test Loss: \{test_loss\}")
print("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

6.1. 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 = ['{v1}
(\{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()))

<table>
<thead>
<tr>
<th></th>
<th>precision</th>
<th>recall</th>
<th>f1-score</th>
<th>support</th>
</tr>
</thead>
<tbody>
<tr>
<td>anorganik</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>20</td>
</tr>
<tr>
<td>organik</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>20</td>
</tr>
<tr>
<td>accuracy</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>40</td>
</tr>
<tr>
<td>macro avg</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>40</td>
</tr>
<tr>
<td>weighted avg</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>40</td>
</tr>
</tbody>
</table>
9. Wrong Predictions

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

```python
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}
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
```

1s completed at 12:10 PM