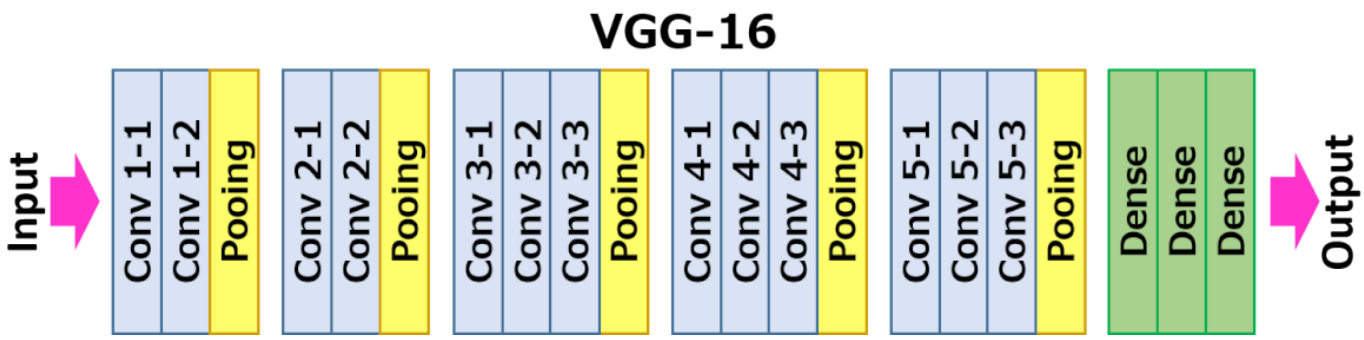


The pre-trained model we'll be working with to classify images of cats and dogs is called VGG16, which is the model that won the 2014 ImageNet competition.

In the ImageNet competition, multiple teams compete to build a model that best classifies images from the ImageNet library. The ImageNet library houses thousands of images belonging to 1000 different categories.

We'll import this VGG16 model and then fine-tune it using Keras. The fine-tuned model will not classify images as one of the 1000 categories for which it was trained on, but instead it will only work to classify images as either cats or dogs.

+ Code + Text

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

```
Mounted at /content/gdrive
```

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Activation, Dense, Flatten, BatchNormalization, Conv2D, M
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.metrics import categorical_crossentropy
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import KFold
import itertools
import os
import shutil
import random
import glob
import matplotlib.pyplot as plt
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
%matplotlib inline
```

```
train_path = 'gdrive/MyDrive/cat_dog/training_set'
test_path =  'gdrive/MyDrive/cat_dog/test_set'
train_batches = ImageDataGenerator(preprocessing_function=tf.keras.applications.vgg16.preproc
    .flow_from_directory(directory=train_path, target_size=(224,224), classes=['cats', 'dogs'
test_batches = ImageDataGenerator(preprocessing_function=tf.keras.applications.vgg16.preproce
    .flow_from_directory(directory=test_path, target_size=(224,224), classes=['cats', 'dogs']
```

```
Found 8005 images belonging to 2 classes.
Found 2023 images belonging to 2 classes.
```

```
imgs, labels = next(train_batches)
def plotImages(images_arr):
    fig, axes = plt.subplots(1, 10, figsize=(20,20))
    axes = axes.flatten()
    for img, ax in zip( images_arr, axes):
        ax.imshow(img)
        ax.axis('off')
    plt.tight_layout()
    plt.show()

plotImages(imgs)
print(labels)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0
```

```
[[1. 0.]
 [0. 1.]
 [1. 0.]
 [0. 1.]
 [0. 1.]
 [1. 0.]
 [1. 0.]
 [1. 0.]
 [1. 0.]
 [0. 1.]]
```

```
vgg16_model = tf.keras.applications.vgg16.VGG16()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16
553467904/553467096 [==============================] - 5s 0us/step
```

```
vgg16_model.summary()
```

```
Model: "vgg16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
fc1 (Dense)                  (None, 4096)              102764544
_________________________________________________________________
fc2 (Dense)                  (None, 4096)              16781312
_________________________________________________________________
predictions (Dense)          (None, 1000)              4097000
```

```
=================================================================
Total params: 138,357,544
Trainable params: 138,357,544
Non-trainable params: 0
_________________________________________________________________
```

```
type(vgg16_model)
```

```
tensorflow.python.keras.engine.functional.Functional
```

For now, we're going to go through a process to convert the Functional model to a Sequential model, so that it will be easier for us to work with given our current knowledge.

We first create a new model of type Sequential. We then iterate over each of the layers in vgg16_model, except for the last layer, and add each layer to the new Sequential model.

```
model = Sequential()
for layer in vgg16_model.layers[:-1]:
    model.add(layer)


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


model.add(Dense(units=2, activation='softmax'))
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
```

```
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
fc1 (Dense)                  (None, 4096)              102764544
_________________________________________________________________
fc2 (Dense)                  (None, 4096)              16781312
_________________________________________________________________
dense (Dense)                (None, 2)                 8194
=================================================================
Total params: 134,268,738
Trainable params: 8,194
Non-trainable params: 134,260,544
_________________________________________________________________
```

Train A Fine-Tuned Neural Network With TensorFlow's Keras API

```
model.compile(optimizer=Adam(learning_rate=0.0001), loss='categorical_crossentropy', metrics=
```

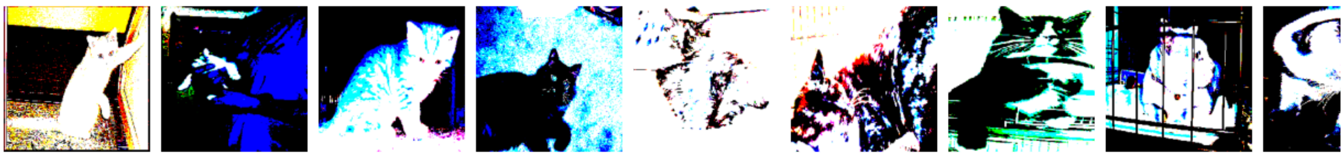
```
model.fit(x = train_batches,
          steps_per_epoch = len(train_batches),
          epochs = 5
         )
```

```
Epoch 1/5
801/801 [==============================] - 1581s 2s/step - loss: 0.2016 - accuracy: 0.92
Epoch 2/5
801/801 [==============================] - 49s 61ms/step - loss: 0.0459 - accuracy: 0.98
Epoch 3/5
801/801 [==============================] - 49s 61ms/step - loss: 0.0290 - accuracy: 0.98
Epoch 4/5
801/801 [==============================] - 49s 61ms/step - loss: 0.0235 - accuracy: 0.99
Epoch 5/5
801/801 [==============================] - 49s 61ms/step - loss: 0.0209 - accuracy: 0.99
<tensorflow.python.keras.callbacks.History at 0x7f387f3e4358>
```

```
test imgs, test labels = next(test batches)
```

```
plotImages(test_imgs)
print(test_labels)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [
```



```
[[1. 0.]
 [1. 0.]
 [1. 0.]
 [1. 0.]
 [1. 0.]
 [1. 0.]
 [1. 0.]
 [1. 0.]
 [1. 0.]
 [1. 0.]]
```

```
predictions = model.predict(x = test_batches, steps = len(test_batches), verbose = 0)
```

```
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
```

```
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
            horizontalalignment="center",
            color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

cm = confusion_matrix(y_true=test_batches.classes, y_pred=np.argmax(predictions, axis=-1))
cm_plot_labels = ['cat','dog']
plot_confusion_matrix(cm=cm, classes=cm_plot_labels, title='Confusion Matrix')
```

```
 Confusion matrix, without normalization
 [[ 987   24]
  [  10 1002]]
```