<https://medium.com/pythoneers/vgg-16-architecture-implementation-and-practical-use-e0fef1d14557>

Vgg 16 Architecture, Implementation and Practical Use

[](https://abhayparashar31.medium.com/?source=post_page-----e0fef1d14557--------------------------------)

[Abhay Parashar](https://abhayparashar31.medium.com/?source=post_page-----e0fef1d14557--------------------------------)

Follow

[Oct 8, 2020](https://medium.com/pythoneers/vgg-16-architecture-implementation-and-practical-use-e0fef1d14557?source=post_page-----e0fef1d14557--------------------------------) · 6 min read

Step by Step Process to create an Image Classifier Using Vgg16

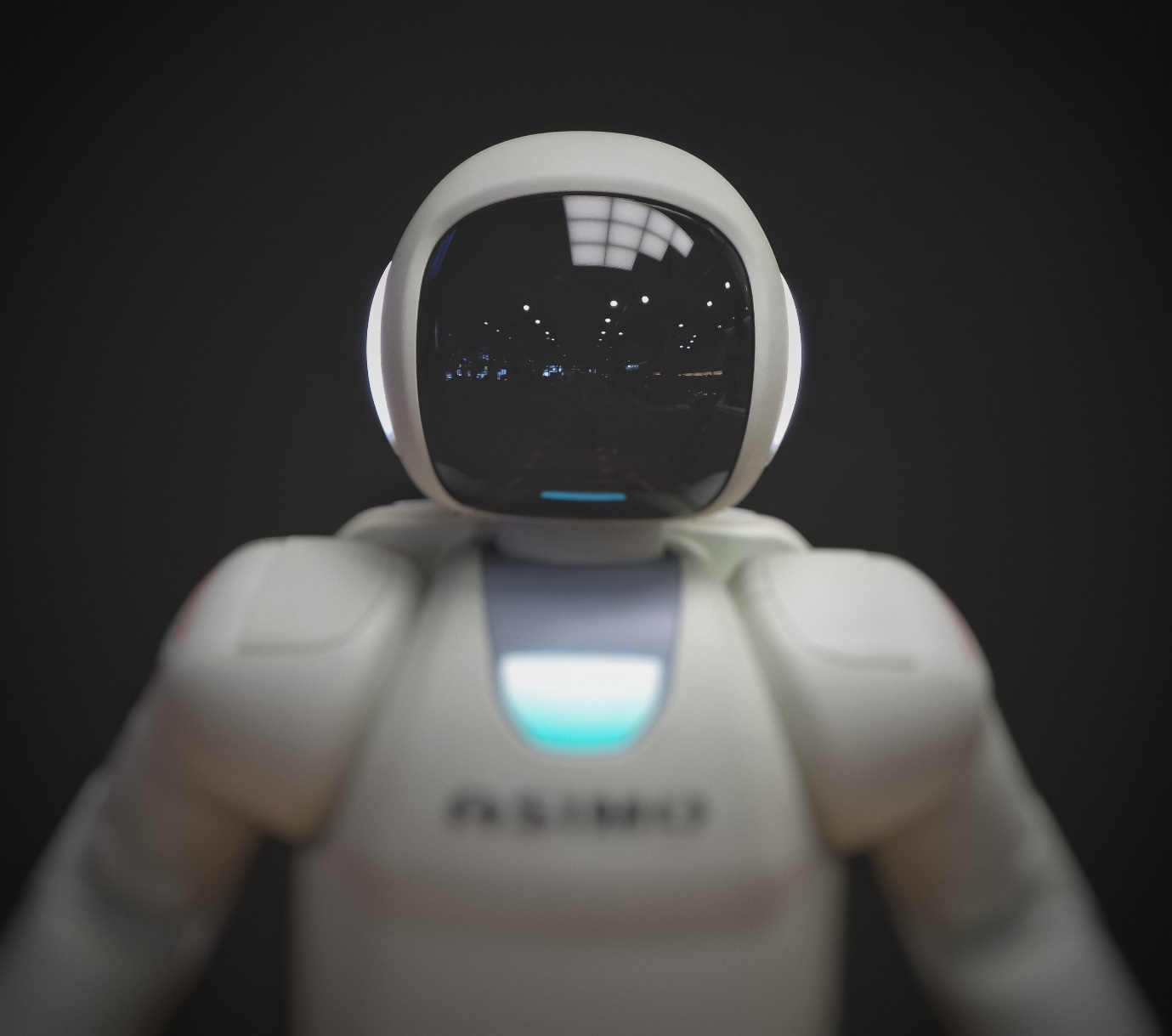


Photo by [Photos Hobby](https://unsplash.com/@photoshobby?utm_source=medium&utm_medium=referral) on [Unsplash](https://unsplash.com/?utm_source=medium&utm_medium=referral)

Hello there, I am Abhay

The era of Convolution Neural Network is at its peak in the 20th century and it is said that it is going to rise more by up to 10% in upcoming years. The reason behind that increase is data. In 2020 according to some resources, it is that that there is almost 90% of data that we used is unlabelled data, and from that 90%, almost 40–50% of data is in the form of images. whether you upload an image on your social media or you upload a post on twitter the image data is everywhere. we need some intelligent algorithms using which we can find the relative meaning from that data.

CNN is mainly used for image classification, segmentation, and also for other co-related fields. a CNN can predict the objects inside an image by just looking at it like we humans do.

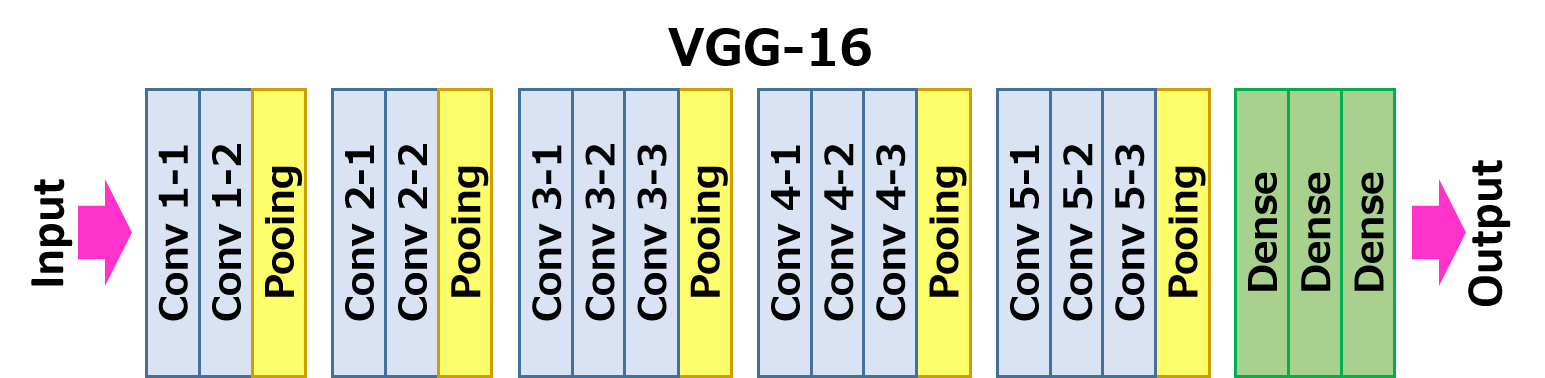
There are so many things to handle when we are implementing a CNN-like number of layers, size of the filter, padding type, and more. The solution from all of this is to use a pre-trained model for our image classification project. There are many pre-trained models out there like resents, inception, Vgg, and others.

One the easiest to understand and simple to build model is Vgg 16. it is also one of the commonly used models nowadays. using Vgg 16 we can solve almost every classification problem. if you are a beginner then Vgg 16 is best for you to start and build.

Key Features of Vgg 16

* It is also called the OxfordNet model, named after the Visual Geometry Group from Oxford.
* Number 16 refers that it has a total of **16 layers** that has some **weights**.
* It Only has Conv and pooling layers in it.
* always use a 3 x 3 Kernel for convolution.
* 2x2 size of the max pool.
* has a total of about 138 million parameters.
* Trained on ImageNet data
* It has an **accuracy** of 92.7%.
* it has one more version of it Vgg 19, a total of 19 layers with weights.

The architecture of Vgg 16



VGG-16 → [Source](http://www.renom.jp/notebooks/image_processing/neural-style-transfer/notebook.html)

**The Kernel size is 3x3 and the pool size is 2x2 for all the layers.**The input to the Vgg 16 model is 224x224x3 pixels images. then we have two convolution layers with each 224x224x64 size, then we have a pooling layer which reduces the height and width of the image to 112x112x64.

Then we have two conv128 layers with each 112x112x128 size after that we have a pooling layer which again reduces the height and width of the image to 56x56x128.

Then we have three conv256 layers with each 56x56x256 size, after that again a pooling layer reduces the image size to 28x28x256.

Then we have three conv512 layers with each 28x28x512 size, after that again a pooling layer reduces the image size to 14x14x512 =.

Then again we have three conv512 layers with each 14x14x521 layers, after that, we have a pooling layer with 7x7x521 and then we have two dense or fully-connected layers with each of 4090 nodes. and at last, we have a final dense or output layer with 1000 nodes of the size which classify between 1000 classes of image net.

Implementation of Vgg 16 Using Keras

First, we need to import necessary libraries for Keras to implement a vgg 16 model.

import keras,os  
from keras.models import Sequential  
from keras.layers import Dense, Conv2D, MaxPool2D , Flatten

Now we need to define a model Keras

model = Sequential()

Now just remember the architecture in mind and start adding the layers into the network. our **Kernel\_size is (3,3) and the pool\_size is (2,2)**always.

Sobel,canny,

***The first part has two conv64 with a pooling layer***

model.add(Conv2D(input\_shape=(224,224,3),filters=64,kernel\_size=(3,3),padding="same", activation="relu"))

model.add(Conv2D(filters=64,kernel\_size=(3,3),padding="same", activation="relu"))

model.add(MaxPool2D(pool\_size=(2,2),strides=(2,2)))

***The second part has two conv128 with a pooling layer***

model.add(Conv2D(filters=128, kernel\_size=(3,3), padding="same", activation="relu"))

model.add(Conv2D(filters=128, kernel\_size=(3,3), padding="same", activation="relu"))

model.add(MaxPool2D(pool\_size=(2,2),strides=(2,2)))

***The Third part has three conv256 with a pooling layer***

model.add(Conv2D(filters=256, kernel\_size=(3,3), padding="same", activation="relu"))

model.add(Conv2D(filters=256, kernel\_size=(3,3), padding="same", activation="relu"))

model.add(Conv2D(filters=256, kernel\_size=(3,3), padding="same", activation="relu"))

model.add(MaxPool2D(pool\_size=(2,2),strides=(2,2)))

***The fourth part has four conv512 with a pooling layer***

model.add(Conv2D(filters=512, kernel\_size=(3,3), padding="same", activation="relu"))

model.add(Conv2D(filters=512, kernel\_size=(3,3), padding="same", activation="relu"))

model.add(Conv2D(filters=512, kernel\_size=(3,3), padding="same", activation="relu"))

model.add(MaxPool2D(pool\_size=(2,2),strides=(2,2)))

***The fifth part has four conv512 with a pooling layer***

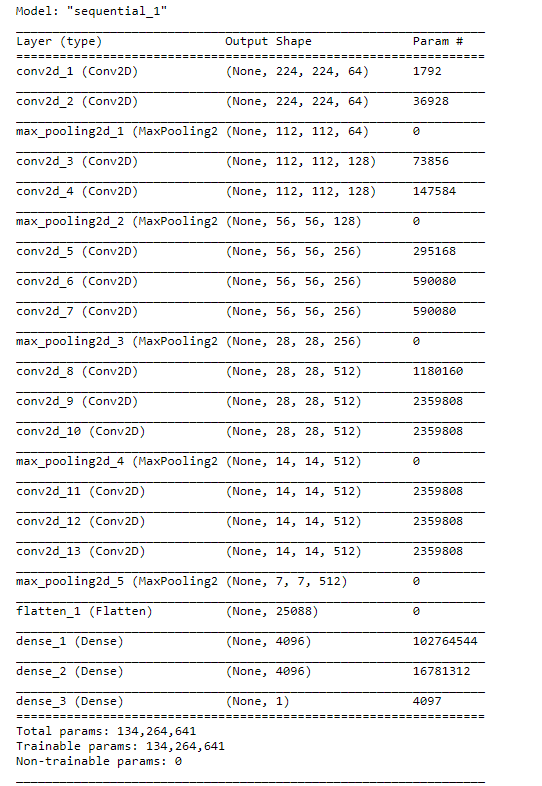
model.add(Conv2D(filters=512, kernel\_size=(3,3), padding="same", activation="relu"))model.add(Conv2D(filters=512, kernel\_size=(3,3), padding="same", activation="relu"))model.add(Conv2D(filters=512, kernel\_size=(3,3), padding="same", activation="relu"))model.add(MaxPool2D(pool\_size=(2,2),strides=(2,2)))

***At last, We have Two Dense layers with 4096 nodes and an output layer***

model.add(Flatten())  
model.add(Dense(units=4096,activation="relu"))  
model.add(Dense(units=4096,activation="relu"))

model.add(Dense(units=1, activation="sigmoid"))   
##units are 1 because we are using binary activation

***Summary of model***



Practical Usecase

-Build an image classifier that can classify between cats and dogs-

Classifying between dog vs cat is the most common problem which you see whenever you trying to learn anything in deep learning. It is a **binary classification**problem because we are trying to classify cats and dogs.  
For this purpose, we are going to use the pre-trained model of Vgg 16 with image net weights provided by Keras.  
**Dataset:** [Dog vs Cat](https://www.kaggle.com/c/dogs-vs-cats)

Transfer Learning

Let’s start by importing all the necessary libraries.

from keras.layers import Input, Lambda, Dense, Flatten  
from keras.models import Model  
from keras.applications.vgg16 import VGG16  
from keras.applications.vgg16 import preprocess\_input  
from keras.preprocessing import image  
from keras.preprocessing.image import ImageDataGenerator  
from keras.models import Sequential  
import numpy as np  
from keras.preprocessing.image import ImageDataGenerator

Because we are using a pre-trained model we need to define the size of the image. Vgg 16 model takes a 224x224x3 size of the image.

IMAGE\_SIZE = [224, 224]

Let’s load the Vgg 16 model

vgg = VGG16(input\_shape=IMAGE\_SIZE + [3], weights='imagenet', include\_top=False)  
vgg.summary()

Here we are taking the weights straight from the image net.

To use a pre-trained model just makes the middle layers non trainable and remove the last layer.

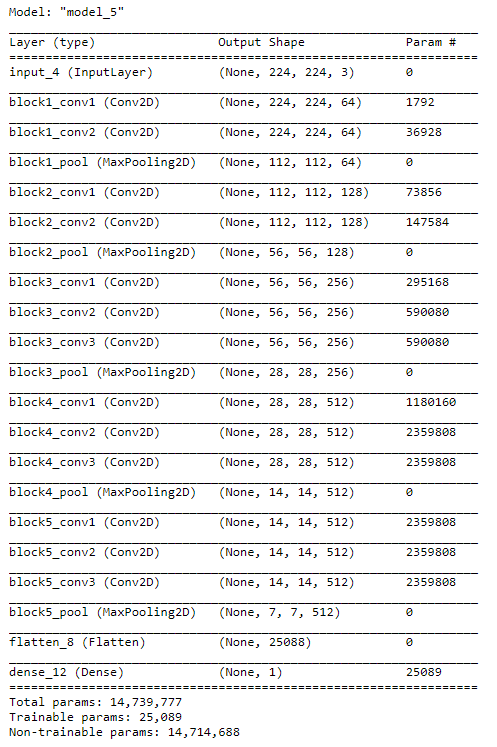
We are also going to do the same let’s makes the middle layers freeze.

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

We have already removed the output layer by include\_top = False.

Let’s add our own output layer with only one node.

x = Flatten()(vgg.output)  
prediction = Dense(1, activation='sigmoid')(x)  
model = Model(inputs=vgg.input, outputs=prediction)  
model.summary()



“Image By Author”

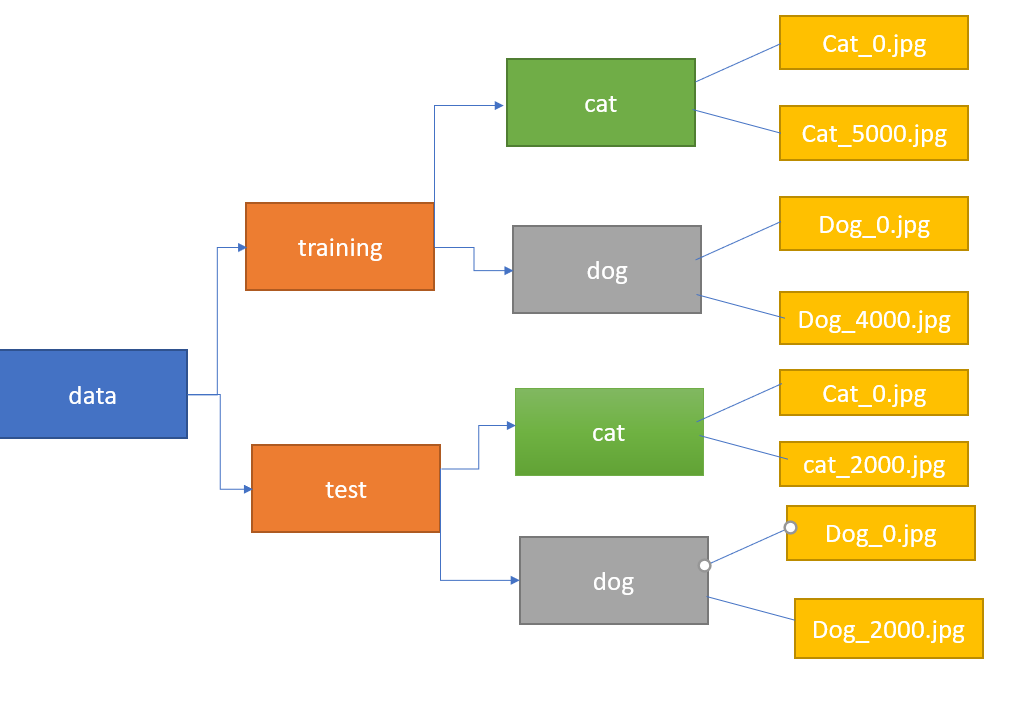
MachineLearningmastery.com

Now we need to compile our model so that we can train it. our problem is a binary classification problem so we are going to use binary\_crossentropy and optimizer we are going to use as Adam and for metrics, we are going to use accuracy.

model.compile(  
 loss='binary\_crossentropy',  
 optimizer='adam',  
 metrics=['accuracy']  
)

Let’s generate training and validation data using the data generator

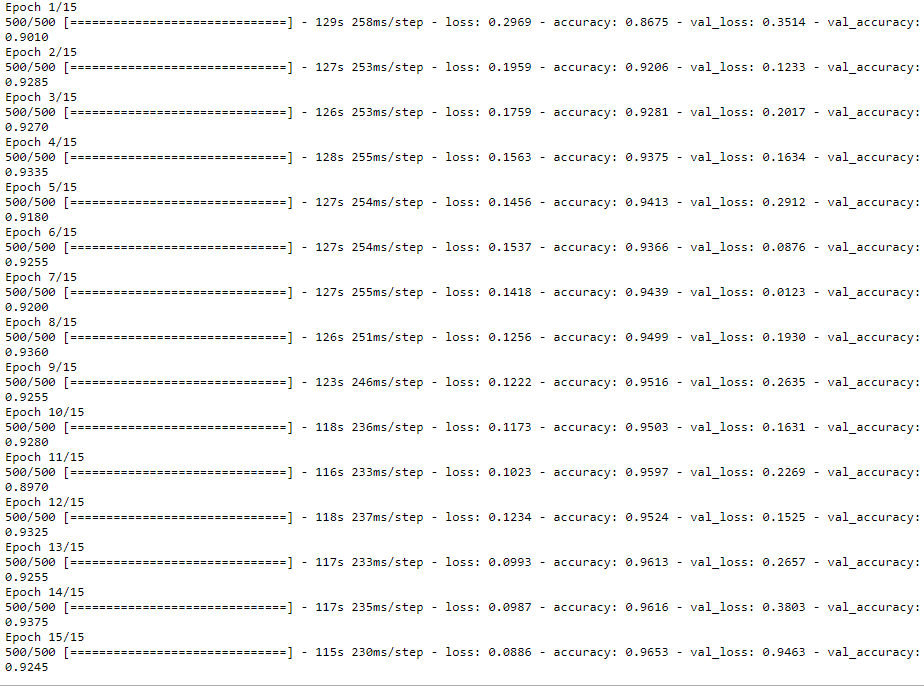
train\_path = 'data/training\_set'  
valid\_path = 'data/test\_set'  
  
train\_datagen = ImageDataGenerator(rescale = 1./255,  
 shear\_range = 0.2,  
 zoom\_range = 0.2,  
 horizontal\_flip = True)test\_datagen = ImageDataGenerator(rescale = 1./255)training\_set = train\_datagen.flow\_from\_directory(train\_path,  
 target\_size = (224, 224),  
 batch\_size = 16,  
 class\_mode = 'binary')test\_set = test\_datagen.flow\_from\_directory(valid\_path,  
 target\_size = (224, 224),  
 batch\_size = 16,  
 class\_mode = 'binary')



“Folder Structure In which we are going to pass images to data generator — Image By Author”

Let’s train our model

r = model.fit\_generator(  
 training\_set,   
 validation\_data=test\_set,  
 epochs=15,  
 steps\_per\_epoch=len(training\_set),  
 validation\_steps=len(test\_set)  
)



“Training — Image By Author”

At final we get an accuracy of 92%.

Let’s test our model on new images.

from IPython.display import Image  
from keras.preprocessing import image  
import tensorflowimg\_path = "data\single\_prediction\cat\_or\_dog\_4.jpg"##new image path  
test\_image = image.load\_img(img\_path, target\_size = [224,224])  
test\_image = image.img\_to\_array(test\_image)  
test\_image = np.expand\_dims(test\_image, axis = 0)result = model.predict(test\_image)  
if result==0:  
 print("CAT")  
else:  
 print("DOG")

Conclusion

We can say that Vgg 16 is easy to understand and implement. It only contains a convolution and pooling layer. The architecture of Vgg 16 looks similar to the architecture of stack. the main aim of this model is to create a deep neural network by using a stacked representation of Conv and pooling layers.