Abhay Parashar

Oct 8, 2020 . 6 min read ★ . Listen

Vgg 16 Architecture, Implementation and Practical Use

Step by Step Process to create an Image Classifier Using Vgg16

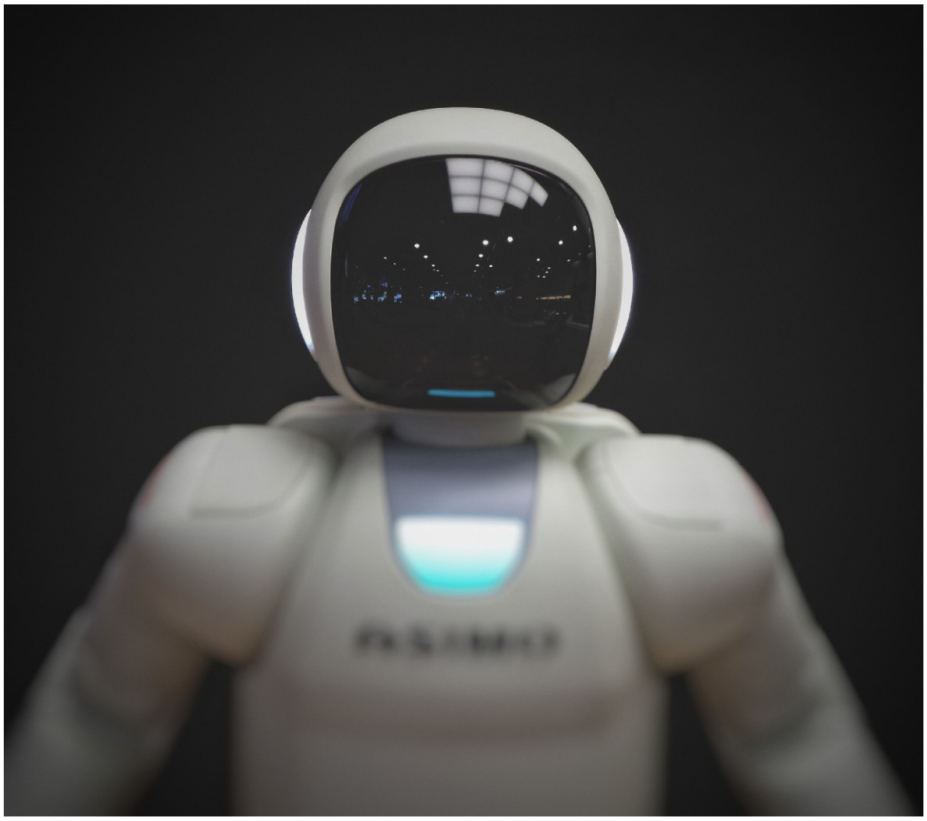


Photo by Photos Hobby on Unsplash

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.

The hardest part with CNN is building a CNN. They look great on paper and also when they are implemented, but when you try to implement a CNN it sucks always sucks. 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.

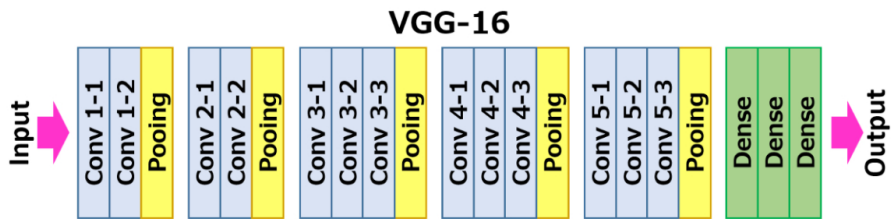
One the easiest to understand and simple to build model is Vgg 16. it is also one of the commonly used models nowadays. using Vgg 16 we can solve almost every classification problem. if you are a beginner then Vgg 16 is best for you to start and build.

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

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.

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

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 224, 224, 64)      1792
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 224, 224, 64)      36928
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 112, 112, 64)      0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 112, 112, 128)     73856
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 112, 112, 128)     147584
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 56, 56, 128)       0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 56, 56, 256)       295168
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 56, 56, 256)       590080
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 56, 56, 256)       590080
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 28, 28, 256)       0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 28, 28, 512)       1180160
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 28, 28, 512)       2359808
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 28, 28, 512)       2359808
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 14, 14, 512)       0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 14, 14, 512)       2359808
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 14, 14, 512)       2359808
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 14, 14, 512)       2359808
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 7, 7, 512)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 25088)             0
_________________________________________________________________
dense_1 (Dense)              (None, 4096)              102764544
_________________________________________________________________
dense_2 (Dense)              (None, 4096)              16781312
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 4097
=================================================================
Total params: 134,264,641
Trainable params: 134,264,641
Non-trainable params: 0
_________________________________________________________________
```

. . .

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

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()
```

```
Model: "model_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_4 (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_8 (Flatten)          (None, 25088)             0
_________________________________________________________________
dense_12 (Dense)             (None, 1)                 25089
=================================================================
Total params: 14,739,777
Trainable params: 25,089
Non-trainable params: 14,714,688
```

“Image By Author”

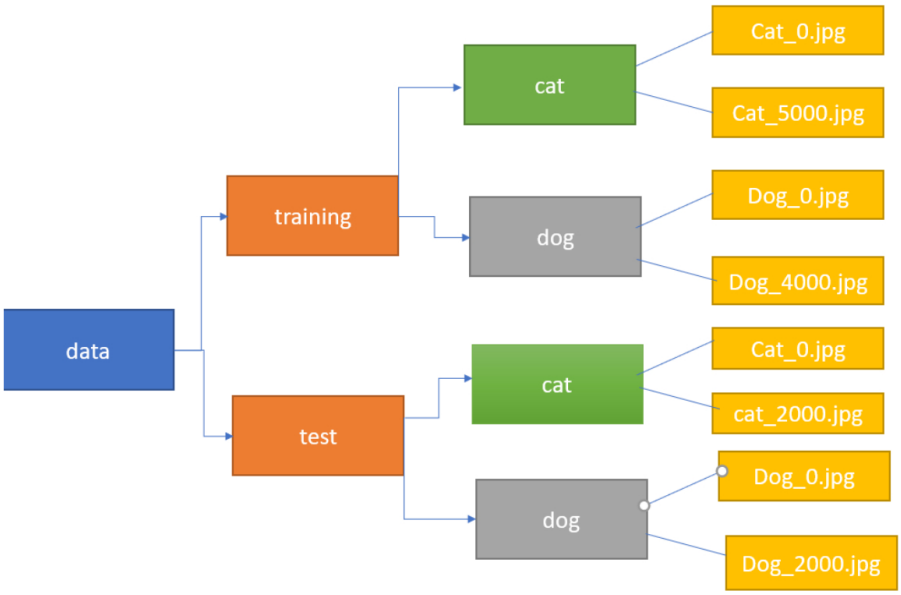
Now we need to compile our model so that we can train it. our problem is a binary classification problem so we are going to use binary_crossentropy and optimizer we are going to use as Adam and for metrics, we are going to use accuracy.

```
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)
)
```

```
Epoch 1/15
500/500 [==============================] - 129s 258ms/step - loss: 0.2969 - accuracy: 0.8675 - val_loss: 0.3514 - val_accuracy:
0.9010
Epoch 2/15
500/500 [==============================] - 127s 253ms/step - loss: 0.1959 - accuracy: 0.9206 - val_loss: 0.1233 - val_accuracy:
0.9285
Epoch 3/15
500/500 [==============================] - 126s 253ms/step - loss: 0.1759 - accuracy: 0.9281 - val_loss: 0.2017 - val_accuracy:
0.9270
Epoch 4/15
500/500 [==============================] - 128s 255ms/step - loss: 0.1563 - accuracy: 0.9375 - val_loss: 0.1634 - val_accuracy:
0.9335
Epoch 5/15
500/500 [==============================] - 127s 254ms/step - loss: 0.1456 - accuracy: 0.9413 - val_loss: 0.2912 - val_accuracy:
0.9180
Epoch 6/15
```

```
500/500 [==============================] - 127s 254ms/step - loss: 0.1537 - accuracy: 0.9366 - val_loss: 0.0876 - val_accuracy:
0.9255
Epoch 7/15
500/500 [==============================] - 127s 255ms/step - loss: 0.1418 - accuracy: 0.9439 - val_loss: 0.0123 - val_accuracy:
0.9200
Epoch 8/15
500/500 [==============================] - 126s 251ms/step - loss: 0.1256 - accuracy: 0.9499 - val_loss: 0.1930 - val_accuracy:
0.9360
Epoch 9/15
500/500 [==============================] - 123s 246ms/step - loss: 0.1222 - accuracy: 0.9516 - val_loss: 0.2635 - val_accuracy:
0.9255
Epoch 10/15
500/500 [==============================] - 118s 236ms/step - loss: 0.1173 - accuracy: 0.9503 - val_loss: 0.1631 - val_accuracy:
0.9280
Epoch 11/15
500/500 [==============================] - 116s 233ms/step - loss: 0.1023 - accuracy: 0.9597 - val_loss: 0.2269 - val_accuracy:
0.8970
Epoch 12/15
500/500 [==============================] - 118s 237ms/step - loss: 0.1234 - accuracy: 0.9524 - val_loss: 0.1525 - val_accuracy:
0.9325
Epoch 13/15
500/500 [==============================] - 117s 233ms/step - loss: 0.0993 - accuracy: 0.9613 - val_loss: 0.2657 - val_accuracy:
0.9255
Epoch 14/15
500/500 [==============================] - 117s 235ms/step - loss: 0.0987 - accuracy: 0.9616 - val_loss: 0.3803 - val_accuracy:
0.9375
Epoch 15/15
500/500 [==============================] - 115s 230ms/step - loss: 0.0886 - accuracy: 0.9653 - val_loss: 0.9463 - val_accuracy:
0.9245
```

“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 tensorflow

img_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.

FUTURE READING

Keras documentation: Keras Applications

Keras Applications are deep learning models that are made available alongside pre-trained weights. These models can be...

keras.io

Keras documentation: VGG16 and VGG19

Instantiates the VGG16 model. Reference By default, it loads weights pre-trained on ImageNet. Check 'weights' for other...

keras.io

Thanks For Reading😀

28

Sign up for The Weekly Picks

By The Pythoneers

A Newsletter That Contains Top Stories From The Week, Coming On Your Door Every Saturday Take a look.

Get this newsletter

Emails will be sent to iamsanju0707@gmail.com.
Not you?

More from The Pythoneers

Follow

We Share Innovative Stories Related to Python Programming, Machine learning, Data Science, Computer Vision, Automation, Web Scraping, Software Development, and more related to AI.

Abhay Parashar · Oct 8, 2020 ★

Develop a Single Linked List Using Python

Perform and Understand Insertion and Deletion Operations — A linked list is a part of data structures. It is one of the most important data structures. In simple words, we can say that a linked list is a collection...

Python 5 min read

Share your ideas with millions of readers. Write on Medium

Abhay Parashar · Oct 7, 2020 ★

Cuda Installation on Windows(2021)-Step by Step Process

Train Your Deep learning Models Using GPU The world is changing day by day, if you say 10 years ago that after 10 years there will be...

Artificial Intelligence 5 min read

Abhay Parashar · Oct 6, 2020 ★

Python Script That Sends Birthday Wishes to Your Friends

Let Python become the savior of yours — Hi there, I am Abhay Recently on Friday, there is the birthday of my old school friend and...

Programming 5 min read

Abhay Parashar · Oct 3, 2020 ★

6 Python Intermediate Projects that Everyone Should Build

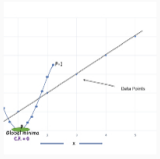
The best Python projects that you can find on the internet today — We are in 2020. we all know that it industry is growing day by day. if you s...

Programming 7 min read

Abhay Parashar · Oct 3, 2020 ★

Linear Regression: Zero to Hero

Complete intuition of The Most Important Algorithm In ML & DL — Hello there, I m Abhay In this blog, we are going to discuss the most important algorithm in machine learning and deep learning Linear...



Machine Learning 6 min read

Read more from The Pythoneers