

# VGG-Net

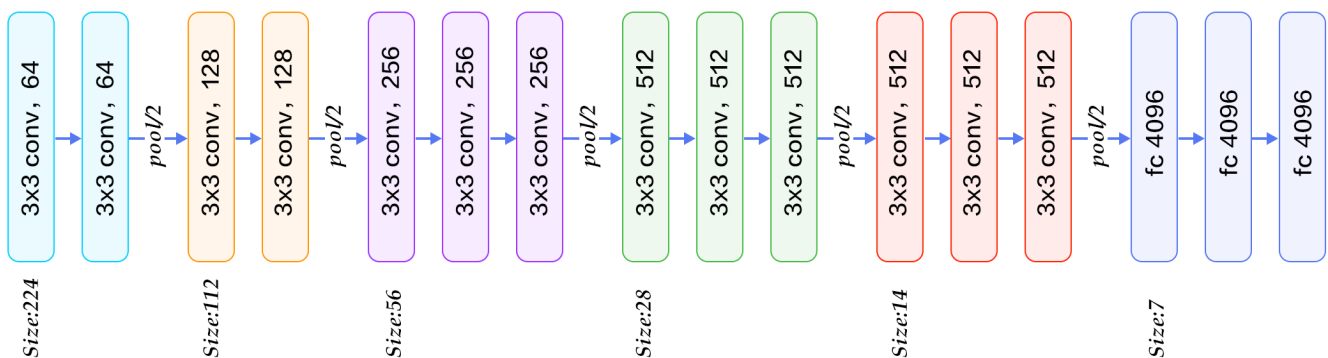
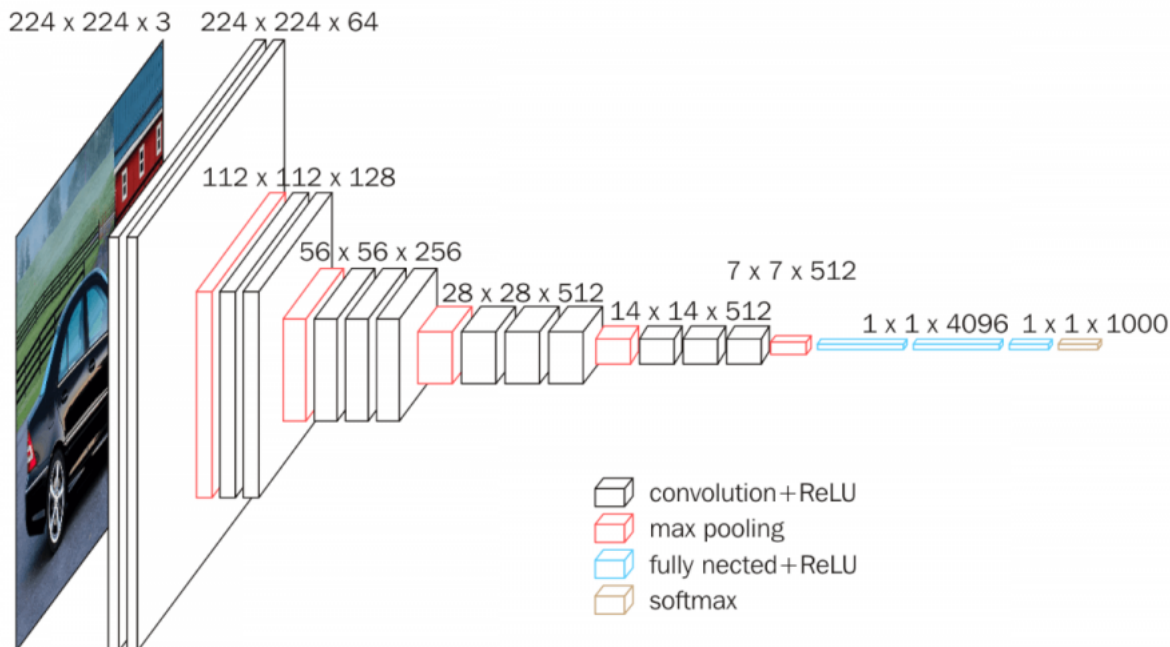
## Introduction

The full name of VGG is the Visual Geometry Group, which belongs to the Department of Science and Engineering of Oxford University. It has released a series of convolutional network models beginning with VGG, which can be applied to face recognition and image classification, from VGG16 to VGG19. The original purpose of VGG's research on the depth of convolutional networks is to understand how the depth of convolutional networks affects the accuracy and accuracy of large-scale image classification and recognition. -Deep-16 CNN), in order to deepen the number of network layers and to avoid too many parameters, a small 3x3 convolution kernel is used in all layers.

## Network Structure of VGG19

### The network structure

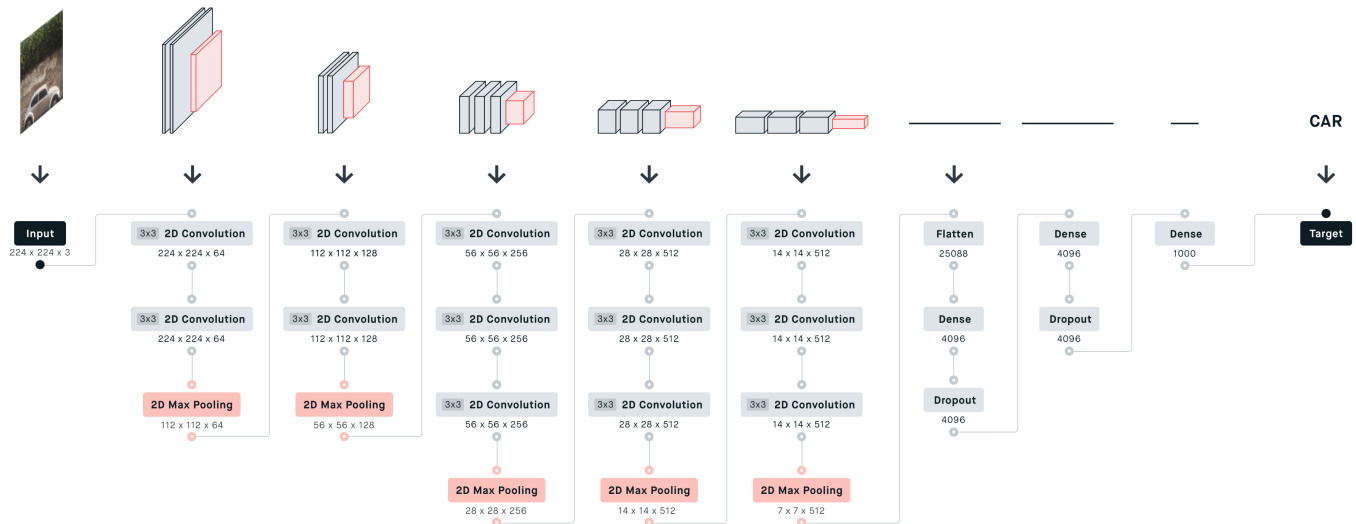
The input of VGG is set to an RGB image of 224x224 size. The average RGB value is calculated for all images on the training set image, and then the image is input as an input to the VGG convolution network. A 3x3 or 1x1 filter is used, and the convolution step is fixed. . There are 3 VGG fully connected layers, which can vary from VGG11 to VGG19 according to the total number of convolutional layers + fully connected layers. The minimum VGG11 has 8 convolutional layers and 3 fully connected layers. The maximum VGG19 has 16 convolutional layers. +3 fully connected layers. In addition, the VGG network is not followed by a pooling layer behind each convolutional layer, or a total of 5 pooling layers distributed under different convolutional layers. The following figure is VGG Structure diagram:



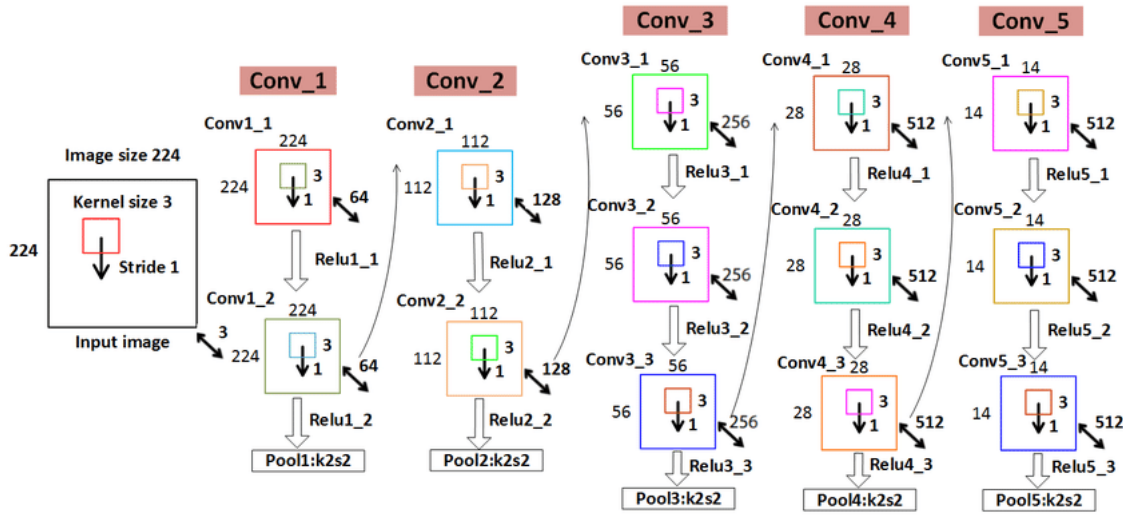
VGG16 contains 16 layers and VGG19 contains 19 layers. A series of VGGS are exactly the same in the last three fully connected layers. The overall structure includes 5 sets of convolutional layers, followed by a MaxPool. The difference is that more and more cascaded convolutional layers are included in the five sets of convolutional layers .

	Layer	Feature Map	Size	Kernel Size	Stride	Activation
Input	Image	1	224 x 224 x 3	-	-	-
1	2 X Convolution	64	224 x 224 x 64	3x3	1	relu
	Max Pooling	64	112 x 112 x 64	3x3	2	relu
3	2 X Convolution	128	112 x 112 x 128	3x3	1	relu
	Max Pooling	128	56 x 56 x 128	3x3	2	relu
5	2 X Convolution	256	56 x 56 x 256	3x3	1	relu
	Max Pooling	256	28 x 28 x 256	3x3	2	relu
7	3 X Convolution	512	28 x 28 x 512	3x3	1	relu
	Max Pooling	512	14 x 14 x 512	3x3	2	relu
10	3 X Convolution	512	14 x 14 x 512	3x3	1	relu
	Max Pooling	512	7 x 7 x 512	3x3	2	relu
13	FC	-	25088	-	-	relu
14	FC	-	4096	-	-	relu
15	FC	-	4096	-	-	relu
Output	FC	-	1000	-	-	Softmax

Each convolutional layer in AlexNet contains only one convolution, and the size of the convolution kernel is  $7 * 7$ . In VGGNet, each convolution layer contains 2 to 4 convolution operations. The size of the convolution kernel is  $3 * 3$ , the convolution step size is 1, the pooling kernel is  $2 * 2$ , and the step size is 2. The most obvious improvement of VGGNet is to reduce the size of the convolution kernel and increase the number of convolution layers.



Using multiple convolution layers with smaller convolution kernels instead of a larger convolution layer with convolution kernels can reduce parameters on the one hand, and the author believes that it is equivalent to more non-linear mapping, which increases the Fit expression ability.



Two consecutive  $3 \times 3$  convolutions are equivalent to a  $5 \times 5$  receptive field, and three are equivalent to  $7 \times 7$ . The advantages of using three  $3 \times 3$  convolutions instead of one  $7 \times 7$  convolution are twofold: one, including three ReLU layers instead of one, makes the decision function more discriminative; and two, reducing parameters. For example, the input and output are all  $C$  channels. 3 convolutional layers using  $3 \times 3$  require  $3(3 \times 3 \times C \times C) = 27 \times C \times C$ , and 1 convolutional layer using  $7 \times 7$  requires  $7 \times 7 \times C \times C = 49 \times C \times C$ . This can be seen as applying a kind of regularization to the  $7 \times 7$  convolution, so that it is decomposed into three  $3 \times 3$  convolutions.

The  $1 \times 1$  convolution layer is mainly to increase the non-linearity of the decision function without affecting the receptive field of the convolution layer. Although the  $1 \times 1$  convolution operation is linear, ReLU adds non-linearity.

### Network Configuration

Table 1 shows all network configurations. These networks follow the same design principles, but differ in depth.

**Table 1: ConvNet configurations** (shown in columns). The depth of the configurations increases from the left (A) to the right (E), as more layers are added (the added layers are shown in bold). The convolutional layer parameters are denoted as “conv<receptive field size>-<number of channels>”. The ReLU activation function is not shown for brevity.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input ( $224 \times 224$ RGB image)					
conv3-64	conv3-64 <b>LRN</b>	conv3-64 <b>conv3-64</b>	conv3-64	conv3-64	conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 <b>conv3-128</b>	conv3-128	conv3-128	conv3-128
maxpool					
conv3-256	conv3-256	conv3-256	conv3-256	conv3-256	conv3-256
conv3-256	conv3-256	conv3-256	conv3-256 <b>conv1-256</b>	conv3-256 <b>conv3-256</b>	conv3-256
maxpool					
conv3-512	conv3-512	conv3-512	conv3-512	conv3-512	conv3-512
conv3-512	conv3-512	conv3-512	conv3-512 <b>conv1-512</b>	conv3-512 <b>conv3-512</b>	conv3-512
maxpool					
conv3-512	conv3-512	conv3-512	conv3-512	conv3-512	conv3-512
conv3-512	conv3-512	conv3-512	conv3-512 <b>conv1-512</b>	conv3-512 <b>conv3-512</b>	conv3-512
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

This picture is definitely used when introducing VGG16. This picture contains a lot of information. My interpretation here may be limited. If you have any supplements, please leave a message.

- **Number 1** : This is a comparison chart of 6 networks. From A to E, the network is getting deeper. Several layers have been added to verify the effect.
- **Number 2** : Each column explains the structure of each network in detail.
- **Number 3**: This is a correct way to do experiments, that is, use the simplest method to solve the problem , and then gradually optimize for the problems that occur.

**Network A:** First mention a shallow network, this network can easily converge on ImageNet. And then?

**Network A-LRN:** Add something that someone else (AlexNet) has experimented to say is effective (LRN), but it seems useless. And then?

**Network B:** Then try adding 2 layers? Seems to be effective. And then?

**Network C:** Add two more layers of  $1 * 1$  convolution, and it will definitely converge. The effect seems to be better. A little excited. And then?

**Network D:** Change the  $1 * 1$  convolution kernel to  $3 * 3$ . Try it. The effect has improved again. Seems to be the best (2014).

## Training

**The optimization method** is a stochastic gradient descent SGD + momentum (0.9) with momentum. The batch size is 256.

**Regularization** : L2 regularization is used, and the weight decay is  $5e-4$ . Dropout is after the first two fully connected layers,  $p = 0.5$ .

Although it is deeper and has more parameters than the AlexNet network, we speculate that VGGNet can converge in less cycles for two reasons: one, the greater depth and smaller convolutions bring implicit regularization ; Second, some layers of pre-training.

**Parameter initialization** : For a shallow A network, parameters are randomly initialized, the weight  $w$  is sampled from  $N(0, 0.01)$ , and the bias is initialized to 0. Then, for deeper networks, first the first four convolutional layers and three fully connected layers are initialized with the parameters of the A network. However, it was later discovered that it is also possible to directly initialize it without using pre-trained parameters.

In order to obtain a  $224 * 224$  input image, each rescaled image is randomly cropped in each SGD iteration. In order to enhance the data set, the cropped image is also randomly flipped horizontally and RGB color shifted.

## Summary of VGGNet improvement points

1. A smaller  $3 * 3$  convolution kernel and a deeper network are used . The stack of two  $3 * 3$  convolution kernels is relative to the field of view of a  $5 * 5$  convolution kernel, and the stack of three  $3 * 3$  convolution kernels is equivalent to the field of view of a  $7 * 7$  convolution kernel. In this way, there can be fewer parameters (3 stacked  $3 * 3$  structures have only  $7 * 7$  structural parameters  $(3 * 3 * 3) / (7 * 7) = 55\%$ ); on the other hand, they have more The non-linear transformation increases the ability of CNN to learn features.
2. In the convolutional structure of VGGNet, a  $1 * 1$  convolution kernel is introduced. Without affecting the input and output dimensions, non-linear transformation is introduced to increase the expressive power of the network and reduce the amount of calculation.
3. During training, first train a simple (low-level) VGGNet A-level network, and then use the weights of the A network to initialize the complex models that follow to speed up the convergence of training .

## Some basic questions

**Q1: Why can 3 3x3 convolutions replace 7x7 convolutions?**

**Answer 1**

3 3x3 convolutions, using 3 non-linear activation functions, increasing non-linear expression capabilities, making the segmentation plane more separable Reduce the number of parameters. For the convolution kernel of C channels, 7x7 contains parameters , and the number of 3 3x3 parameters is greatly reduced.

**Q2: The role of 1x1 convolution kernel**

**Answer 2**

Increase the nonlinearity of the model without affecting the receptive field 1x1 winding machine is equivalent to linear transformation, and the non-linear activation function plays a non-linear role

**Q3: The effect of network depth on results (in the same year, Google also independently released the network GoogleNet with a depth of 22 layers)**

**Answer 3**

VGG and GoogleNet models are deep Small convolution VGG only uses 3x3, while GoogleNet uses 1x1, 3x3, 5x5, the model is more complicated (the model began to use a large convolution kernel to reduce the calculation of the subsequent machine layer)

## Code Implementation

### From Scratch

```
!pip install tflearn

Collecting tflearn
  Downloading tflearn-0.5.0.tar.gz (107 kB)
    107.3/107.3 kB 2.6 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from tflearn) (1.22.4)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from tflearn) (1.16.0)
Requirement already satisfied: Pillow in /usr/local/lib/python3.10/dist-packages (from tflearn) (8.4.0)
Building wheels for collected packages: tflearn
  Building wheel for tflearn (setup.py) ... done
  Created wheel for tflearn: filename=tflearn-0.5.0-py3-none-any.whl size=127283 sha256=3ea44dcd49641671aa70465d30be732878df8a6be7b
  Stored in directory: /root/.cache/pip/wheels/55/fb/7b/e06204a0ceefa45443930b9a250cb5ebe31def0e4e8245a465
Successfully built tflearn
Installing collected packages: tflearn
Successfully installed tflearn-0.5.0
```

```
from tensorflow import keras
import keras,os
from keras.models import Sequential
from keras.layers import Dense, Conv2D, MaxPool2D , Flatten
from keras.preprocessing.image import ImageDataGenerator
import numpy as np

# Get Data
import tflearn.datasets.oxflower17 as oxflower17
from keras.utils import to_categorical

x, y = oxflower17.load_data()

x_train = x.astype('float32') / 255.0
y_train = to_categorical(y, num_classes=17)

WARNING:tensorflow:From /usr/local/lib/python3.10/dist-packages/tensorflow/python/compat/v2_compat.py:107: disable_resource_variables
Instructions for updating:
non-resource variables are not supported in the long term
Downloading Oxford 17 category Flower Dataset, Please wait...
100.0% 60276736 / 60270631
Successfully downloaded 17flowers.tgz 60270631 bytes.
File Extracted
Starting to parse images...
Parsing Done!
```

```
print(x_train.shape)
print(y_train.shape)

(1360, 224, 224, 3)
(1360, 17)

model = Sequential()
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)))
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)))
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)))
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)))
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)))

model.add(Flatten())
model.add(Dense(units=4096,activation="relu"))
model.add(Dense(units=4096,activation="relu"))
model.add(Dense(units=17, activation="softmax"))
model.summary()

Model: "sequential"
Layer (type)                Output Shape                Param #
=====
conv2d (Conv2D)              (None, 224, 224, 64)       1792
```

conv2d_1 (Conv2D)	(None, 224, 224, 64)	36928
max_pooling2d (MaxPooling2D)	(None, 112, 112, 64)	0
conv2d_2 (Conv2D)	(None, 112, 112, 128)	73856
conv2d_3 (Conv2D)	(None, 112, 112, 128)	147584
max_pooling2d_1 (MaxPooling2D)	(None, 56, 56, 128)	0
conv2d_4 (Conv2D)	(None, 56, 56, 256)	295168
conv2d_5 (Conv2D)	(None, 56, 56, 256)	590080
conv2d_6 (Conv2D)	(None, 56, 56, 256)	590080
max_pooling2d_2 (MaxPooling2D)	(None, 28, 28, 256)	0
conv2d_7 (Conv2D)	(None, 28, 28, 512)	1180160
conv2d_8 (Conv2D)	(None, 28, 28, 512)	2359808
conv2d_9 (Conv2D)	(None, 28, 28, 512)	2359808
max_pooling2d_3 (MaxPooling2D)	(None, 14, 14, 512)	0
conv2d_10 (Conv2D)	(None, 14, 14, 512)	2359808
conv2d_11 (Conv2D)	(None, 14, 14, 512)	2359808
conv2d_12 (Conv2D)	(None, 14, 14, 512)	2359808
max_pooling2d_4 (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
dense (Dense)	(None, 4096)	102764544
dense_1 (Dense)	(None, 4096)	16781312
dense_2 (Dense)	(None, 17)	69649
=====		
Total params: 134,330,193		
Trainable params: 134,330,193		
Non-trainable params: 0		

```
# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Train
model.fit(x_train, y_train, batch_size=64, epochs=5, verbose=1, validation_split=0.2, shuffle=True)

Train on 1088 samples, validate on 272 samples
Epoch 1/5
1088/1088 [=====] - ETA: 0s - loss: 2.8362 - acc: 0.0524/usr/local/lib/python3.10/dist-packages/keras/engi
updates = self.state_updates
1088/1088 [=====] - 44s 40ms/sample - loss: 2.8362 - acc: 0.0524 - val_loss: 2.8360 - val_acc: 0.0551
Epoch 2/5
1088/1088 [=====] - 14s 13ms/sample - loss: 2.8341 - acc: 0.0607 - val_loss: 2.8381 - val_acc: 0.0404
Epoch 3/5
1088/1088 [=====] - 14s 13ms/sample - loss: 2.8330 - acc: 0.0588 - val_loss: 2.8397 - val_acc: 0.0331
Epoch 4/5
1088/1088 [=====] - 14s 13ms/sample - loss: 2.8334 - acc: 0.0653 - val_loss: 2.8400 - val_acc: 0.0331
Epoch 5/5
1088/1088 [=====] - 14s 13ms/sample - loss: 2.8332 - acc: 0.0653 - val_loss: 2.8404 - val_acc: 0.0331
<keras.callbacks.History at 0x7ab33c10fc10>
```

▼ VGG Pretrained

```
# download the data from g drive

import gdown
```

```
url = "https://drive.google.com/file/d/12jiQxJzYSYl3wnC8x5wHAhRzzJmmsCXP/view?usp=sharing"
file_id = url.split("/")[-2]
print(file_id)
prefix = 'https://drive.google.com/uc?export=download&id='
gdown.download(prefix+file_id, "catdog.zip")

12jiQxJzYSYl3wnC8x5wHAhRzzJmmsCXP
Downloading...
From: https://drive.google.com/uc?export=download&id=12jiQxJzYSYl3wnC8x5wHAhRzzJmmsCXP
To: /content/catdog.zip
100%|██████████| 9.09M/9.09M [00:00<00:00, 34.1MB/s]
'catdog.zip'
```

```
!unzip catdog.zip
```

```

  inflating: validation/Cat/cat.2409.jpg
  inflating: validation/Cat/cat.2410.jpg
  inflating: validation/Cat/cat.2411.jpg
  inflating: validation/Cat/cat.2412.jpg
  inflating: validation/Cat/cat.2413.jpg
  inflating: validation/Cat/cat.2414.jpg
  inflating: validation/Cat/cat.2415.jpg
  inflating: validation/Cat/cat.2416.jpg
  inflating: validation/Cat/cat.2417.jpg
  inflating: validation/Cat/cat.2418.jpg
  inflating: validation/Cat/cat.2419.jpg
  inflating: validation/Cat/cat.2420.jpg
  inflating: validation/Cat/cat.2421.jpg
  inflating: validation/Cat/cat.2422.jpg
  inflating: validation/Cat/cat.2423.jpg
  inflating: validation/Cat/cat.2424.jpg
  inflating: validation/Cat/cat.2425.jpg
  inflating: validation/Cat/cat.2426.jpg
  inflating: validation/Cat/cat.2427.jpg
  inflating: validation/Cat/cat.2428.jpg
  inflating: validation/Cat/cat.2429.jpg
  inflating: validation/Cat/cat.2430.jpg
  inflating: validation/Cat/cat.2431.jpg
  inflating: validation/Cat/cat.2432.jpg
  inflating: validation/Cat/cat.2433.jpg
  inflating: validation/Cat/cat.2434.jpg
  inflating: validation/Cat/cat.2435.jpg
  creating: validation/Dog/
  inflating: validation/Dog/dog.2402.jpg
  inflating: validation/Dog/dog.2403.jpg
  inflating: validation/Dog/dog.2404.jpg
  inflating: validation/Dog/dog.2405.jpg
  inflating: validation/Dog/dog.2406.jpg
  inflating: validation/Dog/dog.2407.jpg
  inflating: validation/Dog/dog.2408.jpg
  inflating: validation/Dog/dog.2409.jpg
  inflating: validation/Dog/dog.2410.jpg
  inflating: validation/Dog/dog.2411.jpg
  inflating: validation/Dog/dog.2412.jpg
  inflating: validation/Dog/dog.2413.jpg
  inflating: validation/Dog/dog.2414.jpg
  inflating: validation/Dog/dog.2415.jpg
  inflating: validation/Dog/dog.2416.jpg
  inflating: validation/Dog/dog.2417.jpg
  inflating: validation/Dog/dog.2418.jpg
  inflating: validation/Dog/dog.2419.jpg
  inflating: validation/Dog/dog.2420.jpg
  inflating: validation/Dog/dog.2421.jpg
  inflating: validation/Dog/dog.2422.jpg
  inflating: validation/Dog/dog.2423.jpg
  inflating: validation/Dog/dog.2424.jpg
  inflating: validation/Dog/dog.2425.jpg
  inflating: validation/Dog/dog.2426.jpg
  inflating: validation/Dog/dog.2427.jpg
  inflating: validation/Dog/dog.2428.jpg
  inflating: validation/Dog/dog.2429.jpg
  inflating: validation/Dog/dog.2430.jpg
  inflating: validation/Dog/dog.2431.jpg
```

```
from tensorflow import keras
from keras.applications.vgg16 import VGG16, preprocess_input
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
```

```
# Set the path to your training and validation data
train_data_dir = '/content/train'
validation_data_dir = '/content/validation'
```

```
# Set the number of training and validation samples
num_train_samples = 2000
num_validation_samples = 800
```

```

# Set the number of epochs and batch size
epochs = 5
batch_size = 16

# Load the VGG16 model without the top layer
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

# Freeze the base model layers
for layer in base_model.layers:
    layer.trainable = False

# Create a new model
model = Sequential()

# Add the base model as a layer
model.add(base_model)

# Add custom layers on top of the base model
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5)) # reduce the overfitting
model.add(Dense(1, activation='sigmoid'))

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Preprocess the training and validation data
train_datagen = ImageDataGenerator(preprocessing_function=preprocess_input)
validation_datagen = ImageDataGenerator(preprocessing_function=preprocess_input)


train_generator = train_datagen.flow_from_directory(
    train_data_dir,
    target_size=(224, 224),
    batch_size=batch_size,
    class_mode='binary')

validation_generator = validation_datagen.flow_from_directory(
    validation_data_dir,
    target_size=(224, 224),
    batch_size=batch_size,
    class_mode='binary')

# Train the model
model.fit(
    train_generator,
    steps_per_epoch=num_train_samples // batch_size,
    epochs=epochs,
    validation_data=validation_generator,
    validation_steps=num_validation_samples // batch_size)

# Save the trained model
model.save('dog_cat_classifier.h5')

```

 Downloading data from [https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16\\_weights\\_tf\\_dim\\_ordering\\_tf\\_kernels\\_n58889256/58889256](https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_n58889256/58889256) [=====] - 0s 0us/step  
 Found 337 images belonging to 2 classes.  
 Found 59 images belonging to 2 classes.  
 Epoch 1/5  
 125/125 [=====] - 18s 137ms/step - batch: 62.0000 - size: 15.2800 - loss: 2.9239 - acc: 0.9529 - val\_loss:  
 Epoch 2/5  
 125/125 [=====] - 16s 125ms/step - batch: 62.0000 - size: 15.4000 - loss: 0.0673 - acc: 0.9964 - val\_loss:  
 Epoch 3/5  
 125/125 [=====] - 14s 113ms/step - batch: 62.0000 - size: 15.2800 - loss: 0.0965 - acc: 0.9974 - val\_loss:  
 Epoch 4/5  
 125/125 [=====] - 14s 113ms/step - batch: 62.0000 - size: 15.4000 - loss: 0.0184 - acc: 0.9990 - val\_loss:  
 Epoch 5/5  
 125/125 [=====] - 14s 113ms/step - batch: 62.0000 - size: 15.1600 - loss: 1.7005e-13 - acc: 1.0000 - val\_l