VGG-Net

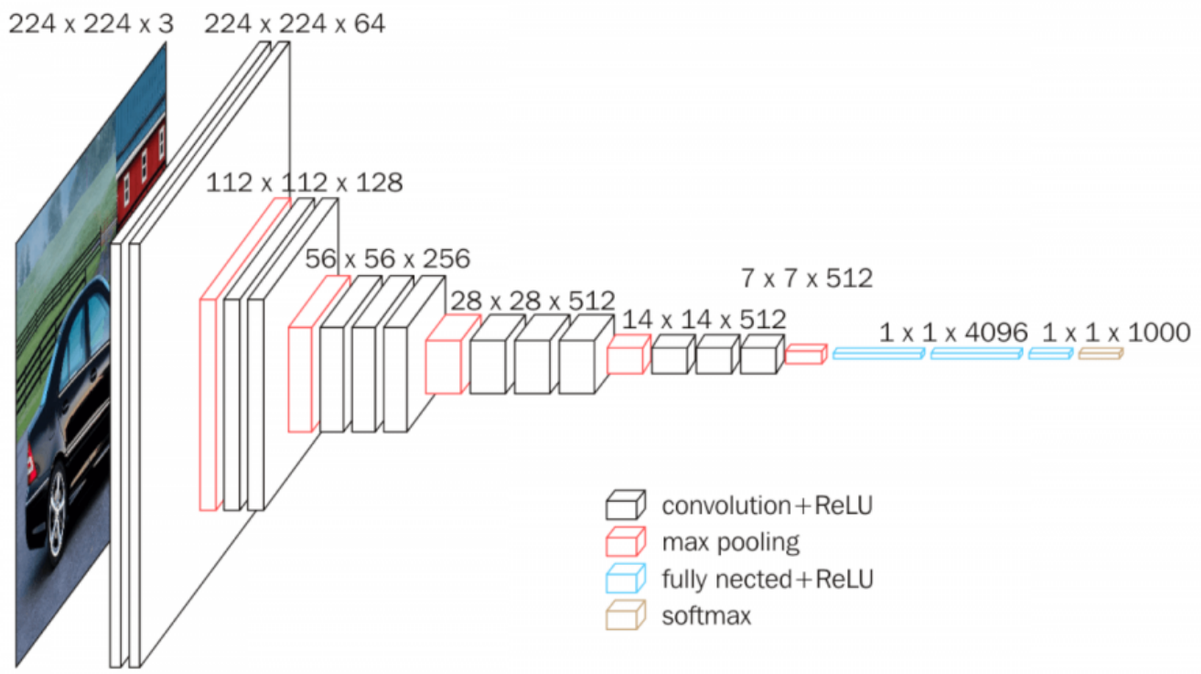
Introduction

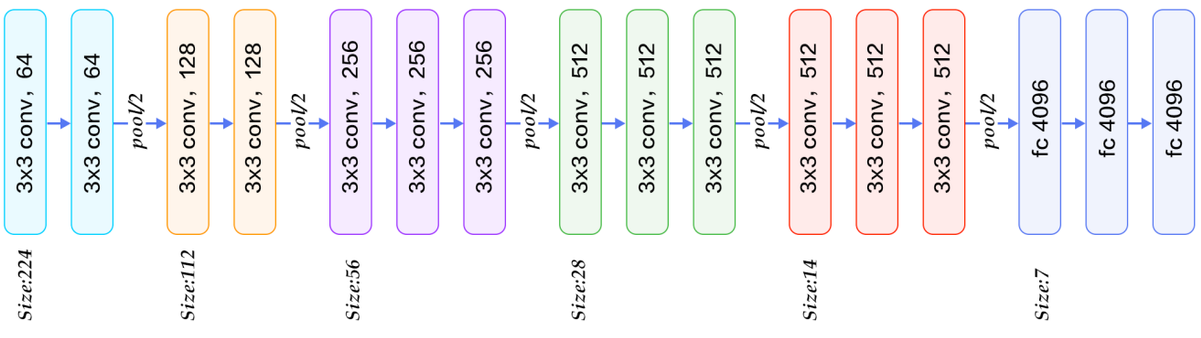
> The full name of VGG is the Visual Geometry Group, which belongs to the Department of Science and Engineering of Oxford University. It has released a series of convolutional network models beginning with VGG, which can be applied to face recognition and image classification, from VGG16 to VGG19. The 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 (http://ethereon.github.io/netscope/#/gist/dc5003de6943ea5a6b8b)](http://ethereon.github.io/netscope/#/gist/dc5003de6943ea5a6b8b)

The network structure

> The input of VGG is set to an RGB image of 224x244 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:

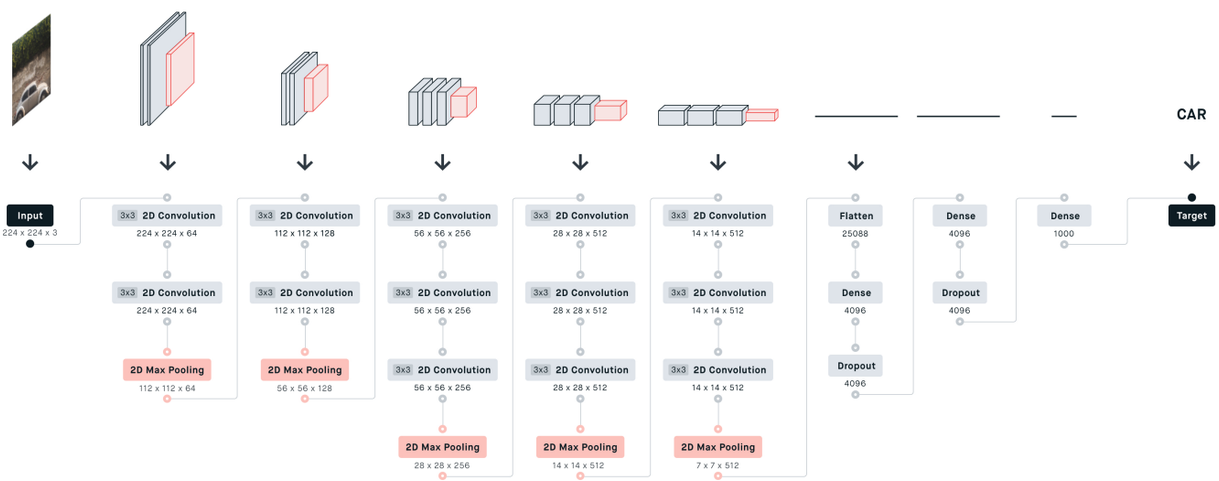




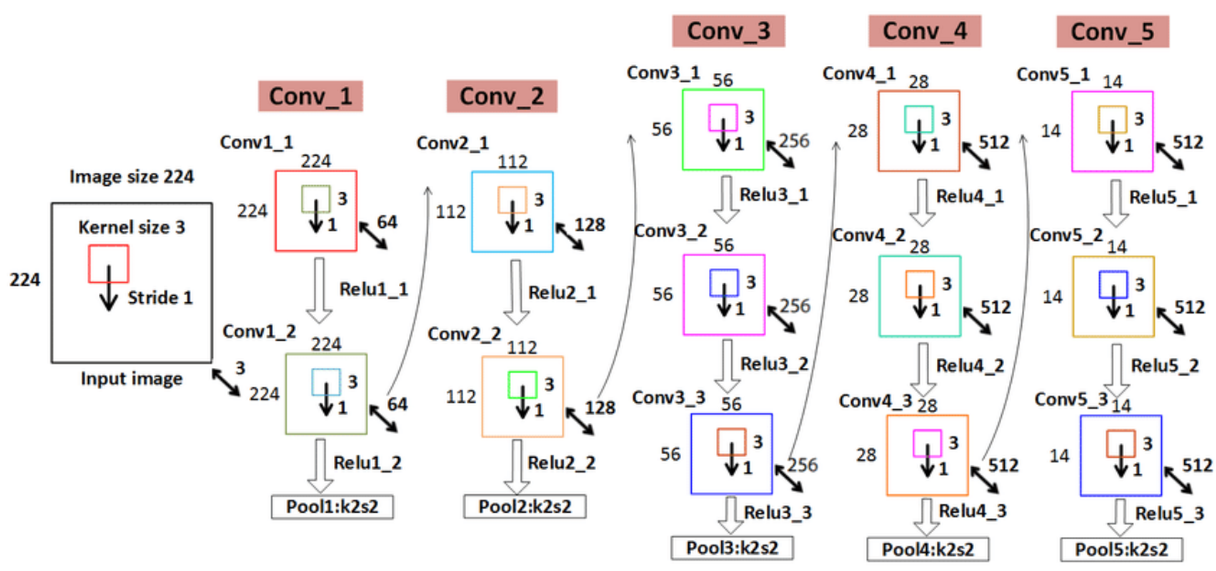
> VGG16 contains 16 layers and VGG19 contains 19 layers. A series of VGGs are exactly the same in the last three fully connected layers. The overall structure includes 5 sets of convolutional layers, followed by a MaxPool. The difference is that more and more cascaded convolutional layers are included 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 * 3 convolutions are equivalent to a 5 * 5 receptive field, and three are equivalent to 7 * 7. The advantages of using three 3 * 3 convolutions instead of one 7 * 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 * 3 require 3 (3 * 3 * C * C) = 27 * C * C, and 1 convolutional layer using 7 * 7 requires 7 * 7 * C * C = 49C * C. This can be seen as applying a kind of regularization to the 7 * 7 convolution, so that it is decomposed into three 3 * 3 convolutions.

> The 1 * 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 * 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 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **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

In [2]:
```
!pip install tflearn
```

```
Looking in indexes: https://pypi.org/simple, (https://pypi.org/simple,) https://us-pyth
on.pkg.dev/colab-wheels/public/simple/ (https://us-python.pkg.dev/colab-wheels/public/s
imple/)
Collecting tflearn
  Downloading tflearn-0.5.0.tar.gz (107 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 107.3/107.3 kB 11.9 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from t
flearn) (1.22.4)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from tfl
earn) (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
=f270efc507b05ff73c8125856b43368ccef0ca18caf050fe770d806a4f8b359f
  Stored in directory: /root/.cache/pip/wheels/55/fb/7b/e06204a0ceefa45443930b9a250cb5e
be31def0e4e8245a465
Successfully built tflearn
Installing collected packages: tflearn
Successfully installed tflearn-0.5.0
```

In [1]:
```
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
```

In [3]:
```
# 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/compa
t/v2_compat.py:107: disable_resource_variables (from tensorflow.python.ops.variable_sco
pe) is deprecated and will be removed in a future version.
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

Succesfully downloaded 17flowers.tgz 60270631 bytes.
File Extracted
Starting to parse images...
Parsing Done!
```

In [4]:
```
print(x_train.shape)
print(y_train.shape)
```

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

In [8]:

```
model = Sequential()
model.add(Conv2D(input_shape=(224,224,3),filters=64,kernel_size=(3,3),padding="same", ac
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_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_13 (Conv2D)          (None, 224, 224, 64)      1792

 conv2d_14 (Conv2D)          (None, 224, 224, 64)      36928

 max_pooling2d_5 (MaxPooling  (None, 112, 112, 64)     0
 2D)

 conv2d_15 (Conv2D)          (None, 112, 112, 128)     73856

 conv2d_16 (Conv2D)          (None, 112, 112, 128)     147584

 max_pooling2d_6 (MaxPooling  (None, 56, 56, 128)      0
 2D)

 conv2d_17 (Conv2D)          (None, 56, 56, 256)       295168

 conv2d_18 (Conv2D)          (None, 56, 56, 256)       590080

 conv2d_19 (Conv2D)          (None, 56, 56, 256)       590080

 max_pooling2d_7 (MaxPooling  (None, 28, 28, 256)      0
 2D)

 conv2d_20 (Conv2D)          (None, 28, 28, 512)       1180160

 conv2d_21 (Conv2D)          (None, 28, 28, 512)       2359808

 conv2d_22 (Conv2D)          (None, 28, 28, 512)       2359808

 max_pooling2d_8 (MaxPooling  (None, 14, 14, 512)      0
 2D)

 conv2d_23 (Conv2D)          (None, 14, 14, 512)       2359808

 conv2d_24 (Conv2D)          (None, 14, 14, 512)       2359808

 conv2d_25 (Conv2D)          (None, 14, 14, 512)       2359808

 max_pooling2d_9 (MaxPooling  (None, 7, 7, 512)        0
 2D)

 flatten_1 (Flatten)         (None, 25088)             0

 dense_3 (Dense)             (None, 4096)              102764544

 dense_4 (Dense)             (None, 4096)              16781312

 dense_5 (Dense)             (None, 17)                69649

=================================================================
Total params: 134,330,193
Trainable params: 134,330,193
Non-trainable params: 0
_________________________________________________________________
```

In [9]:
```
# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

In [10]:
```
# Train
model.fit(x_train, y_train, batch_size=64, epochs=5, verbose=1,validation_split=0.2, shu
```

```
Train on 1088 samples, validate on 272 samples
Epoch 1/5
1088/1088 [==============================] - ETA: 0s - loss: 2.8376 - acc: 0.0478

/usr/local/lib/python3.10/dist-packages/keras/engine/training_v1.py:2335: UserWarning:
`Model.state_updates` will be removed in a future version. This property should not be
used in TensorFlow 2.0, as `updates` are applied automatically.
  updates = self.state_updates

1088/1088 [==============================] - 44s 41ms/sample - loss: 2.8376 - acc: 0.04
78 - val_loss: 2.8353 - val_acc: 0.0331
Epoch 2/5
1088/1088 [==============================] - 15s 13ms/sample - loss: 2.8342 - acc: 0.05
42 - val_loss: 2.8366 - val_acc: 0.0331
Epoch 3/5
1088/1088 [==============================] - 15s 14ms/sample - loss: 2.8335 - acc: 0.06
53 - val_loss: 2.8365 - val_acc: 0.0331
Epoch 4/5
1088/1088 [==============================] - 15s 14ms/sample - loss: 2.8332 - acc: 0.06
53 - val_loss: 2.8387 - val_acc: 0.0331
Epoch 5/5
1088/1088 [==============================] - 15s 14ms/sample - loss: 2.8330 - acc: 0.06
53 - val_loss: 2.8391 - val_acc: 0.0331
```

Out[10]:
```
<keras.callbacks.History at 0x7fe540c4b4f0>
```

In [ ]:
```

```

VGG Pretrained

In [11]:
```
# download the data from g drive

import gdown
url = "https://drive.google.com/file/d/12jiQxJzYSYl3wnC8x5wHAhRzzJmmsCXP/view?usp=sharin
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
(https://drive.google.com/uc?/export=download&id=12jiQxJzYSYl3wnC8x5wHAhRzzJmmsCXP)
To: /content/catdog.zip
100%|██████████| 9.09M/9.09M [00:00<00:00, 118MB/s]
```

Out[11]:
```
'catdog.zip'
```

In [12]:
```
!unzip catdog.zip
```

```
Archive:  catdog.zip
   creating: train/
   creating: train/Cat/
  inflating: train/Cat/0.jpg
  inflating: train/Cat/1.jpg
  inflating: train/Cat/2.jpg
  inflating: train/Cat/cat.2405.jpg
  inflating: train/Cat/cat.2406.jpg
  inflating: train/Cat/cat.2436.jpg
  inflating: train/Cat/cat.2437.jpg
  inflating: train/Cat/cat.2438.jpg
  inflating: train/Cat/cat.2439.jpg
  inflating: train/Cat/cat.2440.jpg
  inflating: train/Cat/cat.2441.jpg
  inflating: train/Cat/cat.2442.jpg
  inflating: train/Cat/cat.2443.jpg
  inflating: train/Cat/cat.2444.jpg
  inflating: train/Cat/cat.2445.jpg
  inflating: train/Cat/cat.2446.jpg
```

In [14]:

```
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))
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')
```

```
Found 337 images belonging to 2 classes.
Found 59 images belonging to 2 classes.
Epoch 1/5
125/125 [==============================] - ETA: 0s - batch: 62.0000 - size: 15.2800 - l
oss: 2.4077 - acc: 0.9665
```

```
/usr/local/lib/python3.10/dist-packages/keras/engine/training_v1.py:2335: UserWarning:
`Model.state_updates` will be removed in a future version. This property should not be
used in TensorFlow 2.0, as `updates` are applied automatically.
  updates = self.state_updates

125/125 [==============================] - 19s 138ms/step - batch: 62.0000 - size: 15.2
800 - loss: 2.4086 - acc: 0.9665 - val_loss: 10.4672 - val_acc: 0.9297
Epoch 2/5
125/125 [==============================] - 17s 139ms/step - batch: 62.0000 - size: 15.2
800 - loss: 0.1754 - acc: 0.9958 - val_loss: 3.5723 - val_acc: 0.9311
Epoch 3/5
125/125 [==============================] - 17s 135ms/step - batch: 62.0000 - size: 15.4
000 - loss: 0.0877 - acc: 0.9984 - val_loss: 4.5018 - val_acc: 0.9473
Epoch 4/5
125/125 [==============================] - 17s 134ms/step - batch: 62.0000 - size: 15.2
800 - loss: 0.0918 - acc: 0.9969 - val_loss: 3.2619 - val_acc: 0.9824
Epoch 5/5
125/125 [==============================] - 16s 129ms/step - batch: 62.0000 - size: 15.2
800 - loss: 4.0183e-07 - acc: 1.0000 - val_loss: 2.7819 - val_acc: 0.9824
```

In [ ]:

In [ ]:

Inception ¶

Also known as GoogLeNet , it is a 22-layer network that won the 2014 ILSVRC Championship.

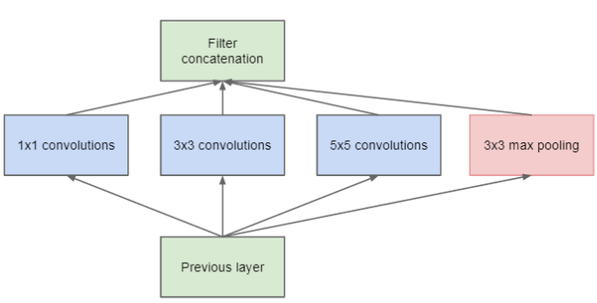
1. The original intention of the design is to expand the width and depth on its basis .
2. which is designed motives derived from improving the performance of the depth of the network generally can increase the size of the network and increase the size of the data set to increase, but at the same time cause the network parameters and easily fit through excessive , computing resources inefficient and The production of high-quality data sets is an expensive issue.
3. Its design philosophy is to change the full connection to a sparse architecture and try to change it to a sparse architecture inside the convolution.
4. The main idea is to design an inception module and increase the depth and width of the network by continuously copying these inception modules , but GooLeNet mainly extends these inception modules in depth.

There are four parallel channels in each inception module , and concat is performed at the end of the channel .

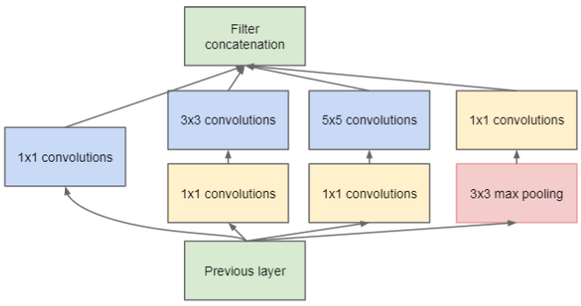
1x1 conv is mainly used to reduce the dimensions in the article to avoid calculation bottlenecks. It also adds additional softmax loss to some branches of the previous network layer to avoid the problem of gradient disappearance.

Four parallel channels:

- 1x1 conv: Borrowed from [Network in Network], the input feature map can be reduced in dimension and upgraded without too much loss of the input spatial information;
- 1x1conv followed by 3x3 conv: 3x3 conv increases the receptive field of the feature map, and changes the dimension through 1x1conv;
- 1x1 conv followed by 5x5 conv: 5x5 conv further increases the receptive field of the feature map, and changes the dimensions through 1x1 conv;
- 3x3 max pooling followed by 1x1 conv: The author believes that although the pooling layer will lose space information, it has been effectively applied in many fields, which proves its effectiveness, so a parallel channel is added, and it is changed by 1x1 conv Its output dimension.

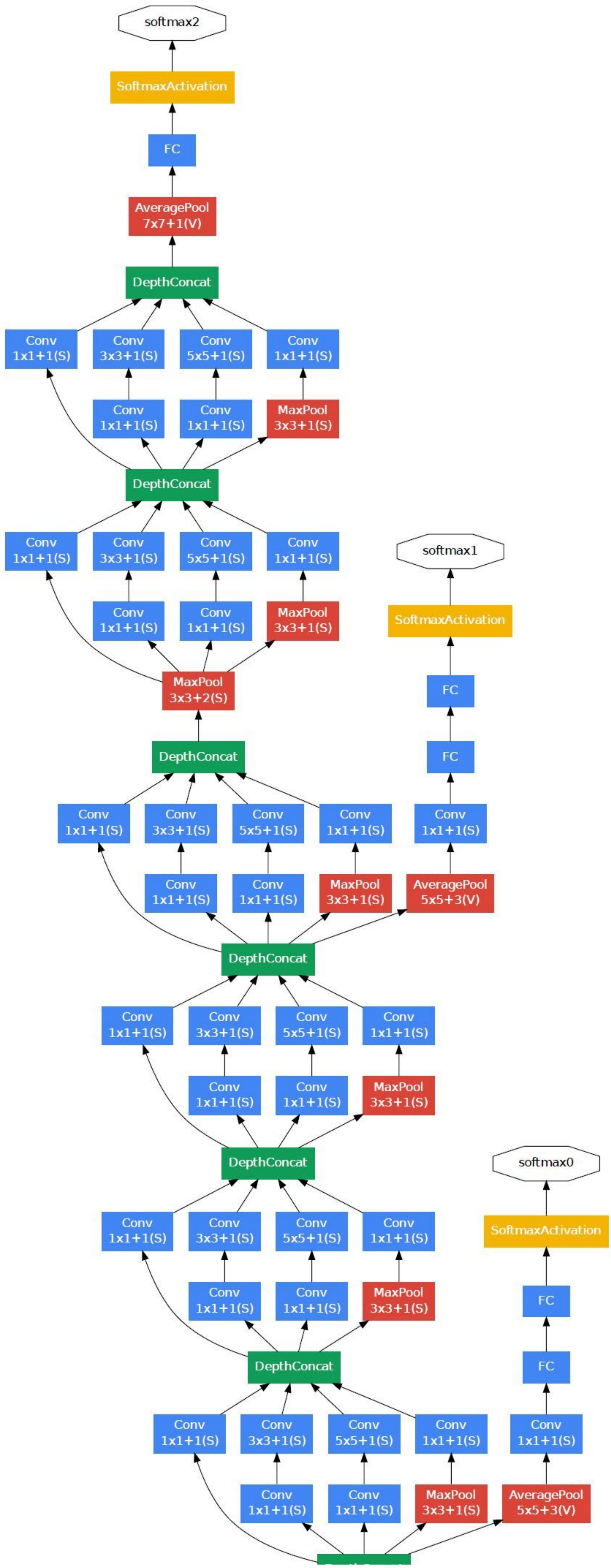


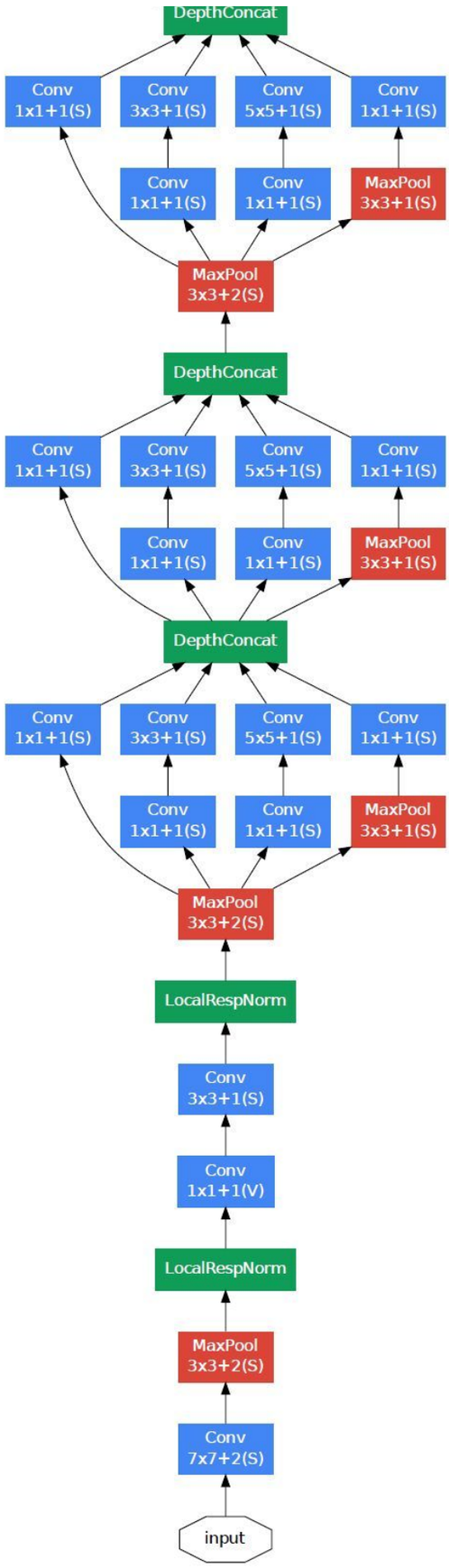
(a) Inception module, naïve version



(b) Inception module with dimension reductions

Complete network design : -


softmax2
SoftmaxActivation
FC
AveragePool 7x7+1(V)
DepthConcat
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
DepthConcat
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
softmax1
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
SoftmaxActivation
MaxPool 3x3+2(S)
FC
FC
DepthConcat
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
AveragePool 5x5+3(V)
DepthConcat
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
DepthConcat
softmax0
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
SoftmaxActivation
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
FC
DepthConcat
FC
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
AveragePool 5x5+3(V)



Two ways to improve network performance:

The most direct way to improve the performance of deep neural networks is to increase their size . This includes depth, the number of levels, and their width, the size of each level unit .

Another easy and safe way is to **increase the size of the training data**.

However, *both methods have two disadvantages* .

Larger models mean more parameters, which makes it easier for the network to overfit , especially when the number of label samples in the training data set is limited.

At the same time, because the production of high-quality training sets is tricky and expensive ,especially when some human experts do it , there is a large error rate . As shown below.

(a) Siberian husky

(b) Eskimo dog

Figure 1: Two distinct classes from the 1000 classes of the ILSVRC 2014 classification challenge.

Another shortcoming is that uniformly increasing the size of the network will increase the use of computing resources . For example, in a deep network, if two convolutions are chained, any unified improvement of their convolution kernels will cause demand for resources.

Power increase: If the increased capacity is inefficient, for example, if most of the weights end with 0 , then a lot of computing resources are wasted. But because the computing resources are always limited, an effective computational distribution always tends to increase the size of the model indiscriminately, and even the main objective goal is to improve the performance of the results.

The basic method to solve these two problems is to finally change the fully connected network to a sparse architecture, even inside the convolution.

The details of the GooLeNet network layer are shown in the following table:

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

Table 1: GoogLeNet incarnation of the Inception architecture

To sum up:

- 128 1x1 convolution kernels are used to reduce dimensions and modify linear activation units
- A fully connected layer of 1024 units and a modified linear activation unit;
- A dropout layer that drops neuron connections with a 70% probability;
- A linear layer with softmax loss as classification Predict 1000 categories, but removed during the inference phase

Training Methodology

> The momentum is set to 0.9 and the learning rate is set to decrease by 4% every 8 epochs.

> Seven models were trained . To make the problem more detailed, some models were trained on small crops, and some were trained on large crops .

> The factors that make the model train well include : the sampling of patches of various sizes in the image , the size of which is evenly distributed between 8% and 100%, and the aspect ratio between 3/4 and 4/3.

> **Illumination changes have an effect on avoiding overfitting.**

> **Later, random interpolation is used to resize the image.**

Inception-v2(2015)

This architecture is a landmark in the development of deep network models . The most prominent contribution is to propose a normalized Batch Normalization layer to unify the output range of the network. It is fixed in a relatively uniform range. If the BN layer is not added, the value range of the network input and output of each layer is greatly different, so the size of the learning rate will be different. The BN layer avoids this situation This accelerates the training of the network and gives the network regular terms to a certain extent , reducing the degree of overfitting of the network. In the subsequent development of network models, most models have more or less added BN layers to the model.

In this paper, the BN layer is standardized before being input to the activation function. At the same time, VGG uses 2 3x3 convs instead of 5x5 convs in the inception module to reduce the amount of parameters and speed up the calculation.

Algorithm advantages:

1. **Improved learning rate** : In the BN model, a higher learning rate is used to accelerate training convergence, but it will not cause other effects. Because if the scale of each layer is different, then the learning rate required by each layer is different. The scale of the same layer dimension often also needs different learning rates. Usually, the minimum learning is required to ensure the loss function to decrease, but The BN layer keeps the scale of each layer and dimension consistent, so you can directly use a higher learning rate for optimization.
2. **Remove the dropout layer** : The BN layer makes full use of the goals of the dropout layer. Remove the dropout layer from the BN-Inception model, but no overfitting will occur.
3. **Decrease the attenuation coefficient of L2 weight** : Although the L2 loss controls the overfitting of the Inception model, the loss of weight has been reduced by five times in the BN-Inception model.
4. **Accelerate the decay of the learning rate** : When training the Inception model, we let the learning rate decrease exponentially. Because our network is faster than Inception, we will increase the speed of reducing the learning rate by 6 times.
5. **Remove the local response layer** : Although this layer has a certain role, but after the BN layer is added, this layer is not necessary.
6. **Scramble training samples more thoroughly** : We scramble training samples, which can prevent the same samples from appearing in a mini-batch. This can improve the accuracy of the validation set by 1%, which is the advantage of the BN layer as a regular term. In our method, random selection is more effective when the model sees different samples each time.
7. **To reduce image distortion**: Because BN network training is faster and observes each training sample less often, we want the model to see a more realistic image instead of a distorted image.

Inception-v3-2015

This architecture focuses, how to use the convolution kernel two or more smaller size of the convolution kernel to replace, but also the introduction of **asymmetrical layers i.e. a convolution dimensional convolution** has also been proposed for pooling layer Some remedies that can cause loss of spatial information; there are ideas such as **label-smoothing , BN-ahxiliary** .

Experiments were performed on inputs with different resolutions . The results show that although low-resolution inputs require more time to train, the accuracy and high-resolution achieved are not much different.

The computational cost is reduced while improving the accuracy of the network.

General Design Principles

We will describe some design principles that have been proposed through extensive experiments with different architectural designs for convolutional networks. At this point, full use of the following principles can be guessed, and some additional experiments in the future will be necessary to estimate their accuracy and effectiveness.

1. **Prevent bottlenecks in characterization** . The so-called bottleneck of feature description is that a large proportion of features are compressed in the middle layer (such as using a pooling operation). This operation will cause the loss of feature space information and the loss of features. Although the operation of pooling in CNN is important, there are some methods that can be used to avoid this loss as much as possible (I note: later hole convolution operations).
2. **The higher the dimensionality of the feature, the faster the training converges** . That is, the independence of features has a great relationship with the speed of model convergence. The more independent features, the more thoroughly the input feature information is decomposed. It is easier to converge if the correlation is strong. Hebbin principle : fire together, wire together.
3. **Reduce the amount of calculation through dimensionality reduction** . In v1, the feature is first reduced by 1x1 convolutional dimensionality reduction. There is a certain correlation between different dimensions. Dimension reduction can be understood as a lossless or low-loss compression. Even if the dimensions are reduced, the correlation can still be used to restore its original information.
4. **Balance the depth and width of the network** . Only by increasing the depth and width of the network in the same proportion can the performance of the model be maximized.

Factorizing Convolutions with Large Filter Size

GooLeNet uses many dimensionality reduction methods, which has achieved certain results. Consider the example of a 1x1 convolutional layer used to reduce dimensions before a 3x3 convolutional layer. In the network, we expect the network to be highly correlated between the output neighboring elements at the activation function. Therefore, we can reduce their activation values before aggregation , which should generate similar local expression descriptions.

This paper explores experiments to decompose the network layer into different factors under different settings in order to improve the computational efficiency of the method . Because the Inception network is fully convolutional, each weight value corresponds to a product operation each time it is activated.

Therefore, any reduction in computational cost will result in a reduction in parameters. This means that we can use some suitable decomposition factors to reduce the parameters and thus speed up the training.

3.1 **Factorizing Convolutions with Large Filter Size**

With the same number of convolution kernels, larger convolution kernels (such as 5x5 or 7x7) are more expensive to calculate than 3x3 convolution kernels , which is about a multiple of 25/9 = 2.78. Of course, the 5x5 convolution kernel can obtain more correlations between the information and activation units in the previous network, but under the premise of huge consumption of computing resources, a physical reduction in the size of the convolution kernel still appears.

However, we still want to know whether a 5x5 convolutional layer can be replaced by a multi-layer convolutional layer with fewer parameters when the input and output sizes are consistent . If we scale the calculation map of 5x5 convolution, we can see that each output is like a small fully connected network sliding on the input window with a size of 5x5. Refer to Figure 1.

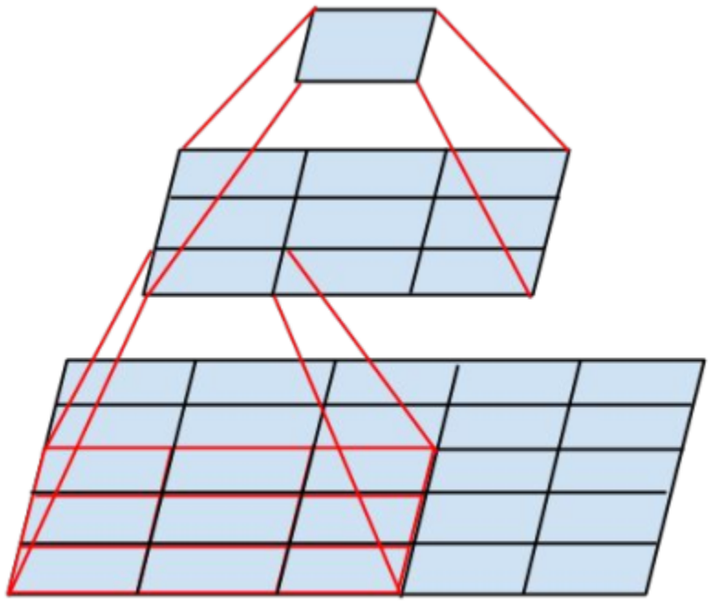


Figure 1. Mini-network replacing the 5×5 convolutions.

Therefore, we have developed a network that explores translation invariance and replaces one layer of convolution with two layers of convolution: the first layer is a 3x3 convolution layer and the second layer is a fully connected layer . Refer to Figure 1. We ended up replacing two 5x5 convolutional layers with two 3x3 convolutional layers. Refer to Figure 4 Figure 5. This operation can realize the weight sharing of neighboring layers. It is about (9 + 9) / 25 times reduction in computational consumption.

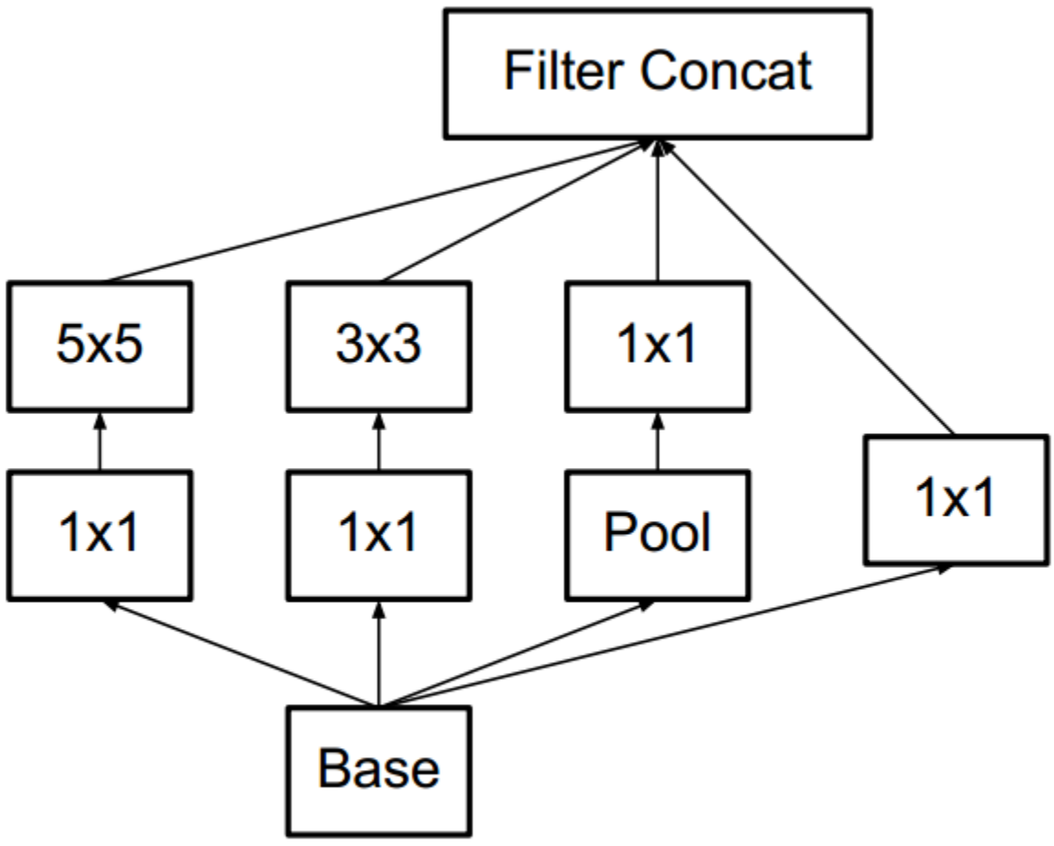


Figure 4. Original Inception module as described in [20].

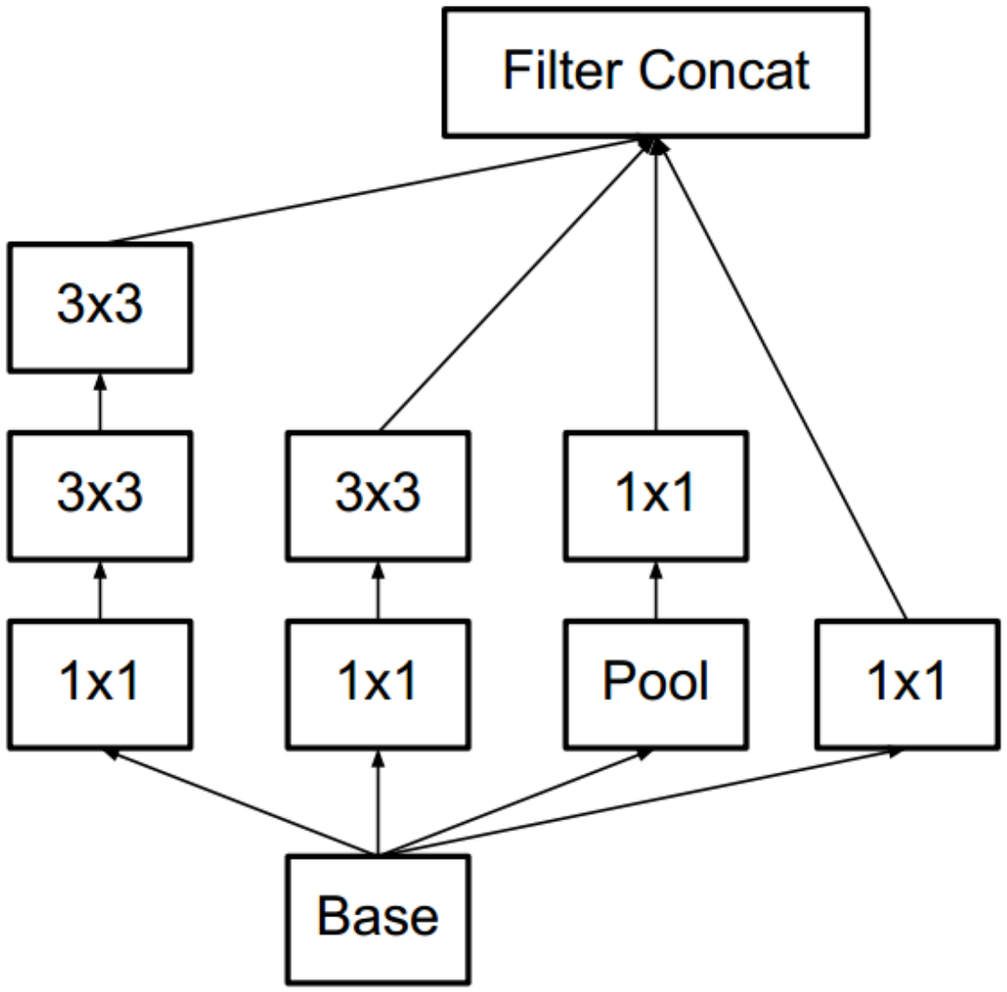


Figure 5. Inception modules where each 5×5 convolution is replaced by two 3×3 convolution, as suggested by principle 3 of Section 2.

Spatial Factorization into Asymmetric Convolutions

We are wondering if the convolution kernel can be made smaller, such as 2x2, but there is an asymmetric method that can be better than this method. That is to use nx1 size convolution. For example, using the [3x1 + 1x3] convolution layer. In this case, a single 3x3 convolution has the same receptive field. Refer to Figure 3. This asymmetric method can save [((3x3)-(3 + 3)) / (3x3) = 33%] computing resources, and replacing two 2x2 only saves [11%] Computing resources.

In theory, we can have a deeper discussion and use the convolution of [1xn + nx1] instead of the convolutional layer of nxn. Refer to Figure 6. But this situation is not very good in the previous layer, but it can perform better on a medium-sized feature map [mxm, m is between 12 and 20]. In this case, use [1x7 + 7x1] convolutional layer can get a very good result.

Utility of Auxiliary Classifiers

Inception-v1 introduced some auxiliary classifiers (referring to some branches of the previous layer adding the softmax layer to calculate the loss back propagation) to improve the aggregation problem in deep networks. The original motive is to pass the gradient back to the previous convolutional layer , so that they can effectively and improve the aggregation of features and avoid the problem of vanishing gradients.

Traditionally, pooling layers are used in convolutional networks to reduce the size of feature maps . In order to avoid bottlenecks in the expression of spatial information, the number of convolution kernels in the network can be expanded before using max pooling or average pooling.

For example, for a dxd network layer with K feature maps, to generate a network layer with 2K [d / 2 xd / 2] feature maps, we can use 2K convolution kernels with a step size of 1. Convolution and then add a pooling layer to get it, then this operation requires [2d 2 K 2]. But using pooling instead of convolution, the approximate operation is [2 * (d / 2) 2 xK 2], which reduces the operation by four times. However, this will cause a description bottleneck, because the feature map is reduced to [(d / 2) 2 xK], which will definitely cause the loss of spatial information on the network. Refer to Figure 9. However, we have adopted a different method to avoid this bottleneck, refer to Figure 10. That is, two parallel channels are used , one is a pooling layer (max or average), the step size is 2, and the other is a convolution layer , and then it is concatenated during output.

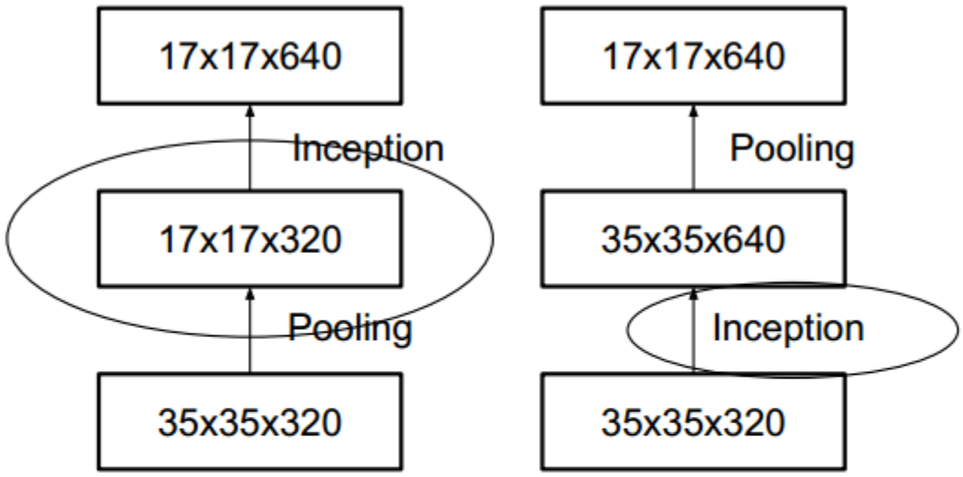


Figure 9. Two alternative ways of reducing the grid size. The solution on the left violates the principle 1 of not introducing an representational bottleneck from Section 2. The version on the right is 3 times more expensive computationally.

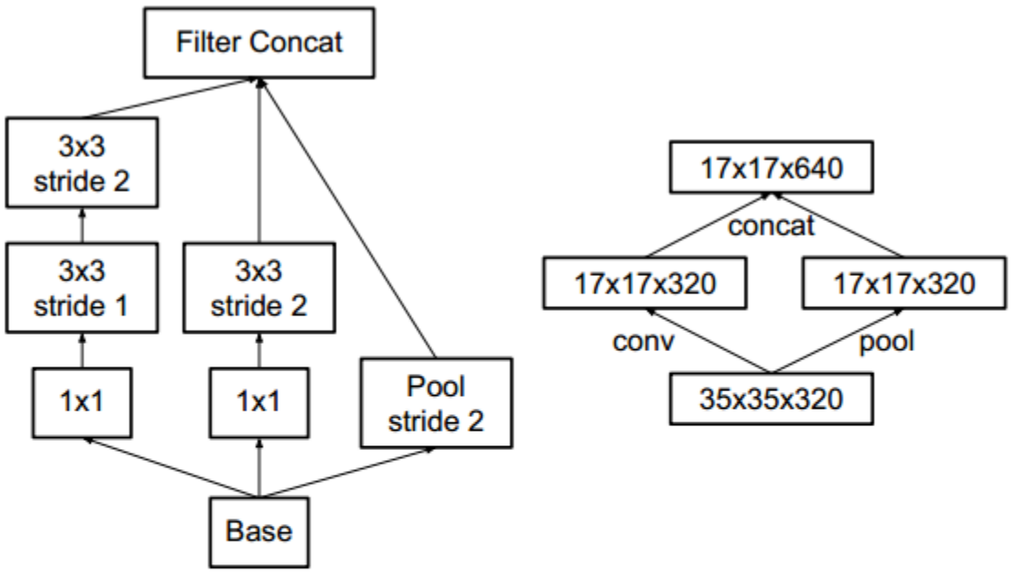


Figure 10. Inception module that reduces the grid-size while expands the filter banks. It is both cheap and avoids the representational bottleneck as is suggested by principle 1. The diagram on the right represents the same solution but from the perspective of grid sizes rather than the operations.

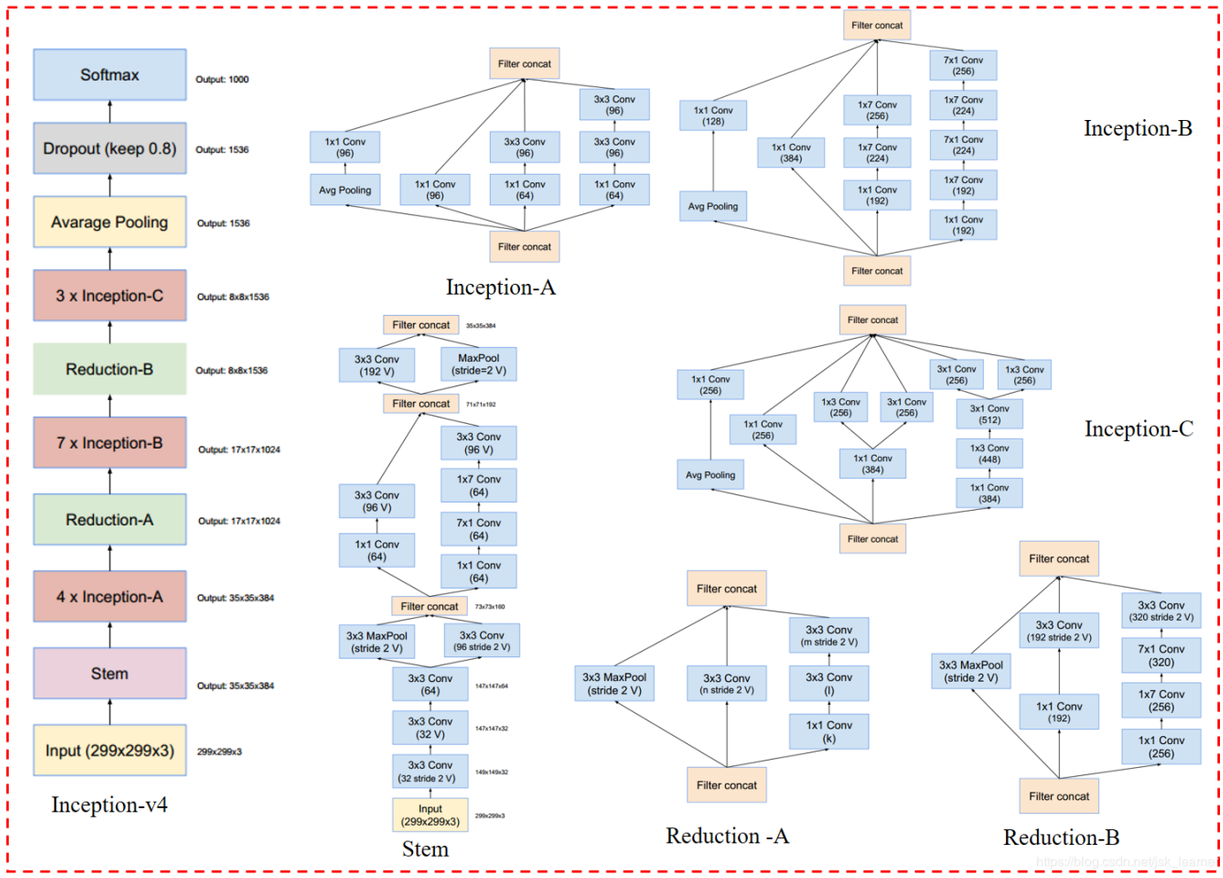
Inception-v4-2016

After ResNet appeared, ResNet residual structure was added.

It is based on Inception-v3 and added the skip connection structure in ResNet. Finally, under the structure of 3 residual and 1 inception-v4 , it reached the top-5 error 3.08% in CLS (ImageNet calssification) .

1-Introduction Residual conn works well when training very deep networks. Because the Inception network architecture can be very deep, it is reasonable to use residual conn instead of concat.

Compared with v3, Inception-v4 has more unified simplified structure and more inception modules.



The big picture of Inception-v4:

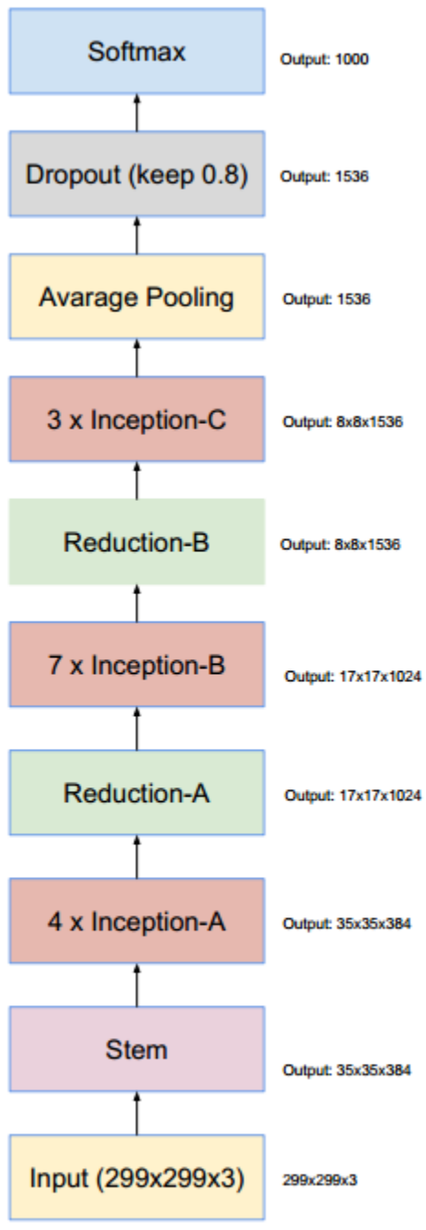


Figure 9. The overall schema of the Inception-v4 network. For the detailed modules, please refer to Figures 3, 4, 5, 6, 7 and 8 for the detailed structure of the various components.

Fig9 is an overall picture, and Fig3,4,5,6,7,8 are all local structures. For the specific structure of each module, see the end of the article.

Residual Inception Blocks

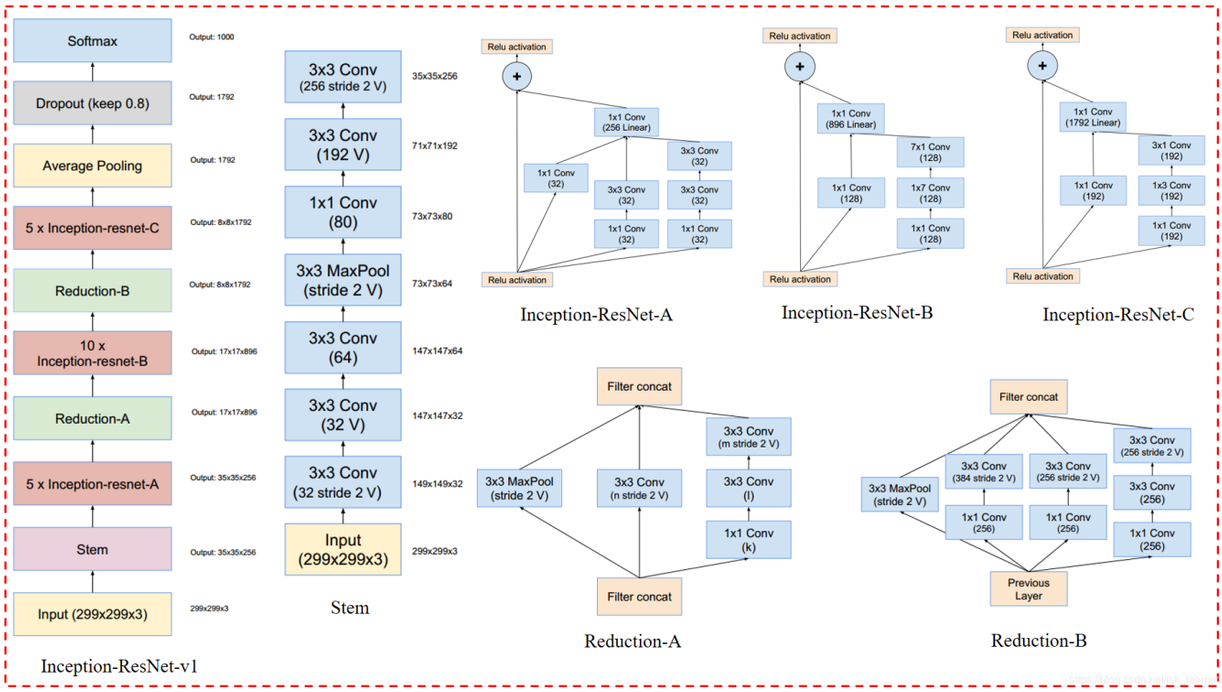
For the residual version in the Inception network, we use an Inception module that consumes less than the original Inception. The convolution kernel (followed by 1x1) of each Inception module is used to modify the dimension, which can compensate the reduction of the Inception dimension to some extent.

One is named **Inception-ResNet-v1**, which is consistent with the calculation cost of Inception-v3. One is named **Inception-ResNet-v2**, which is consistent with the calculation cost of Inception-v4.

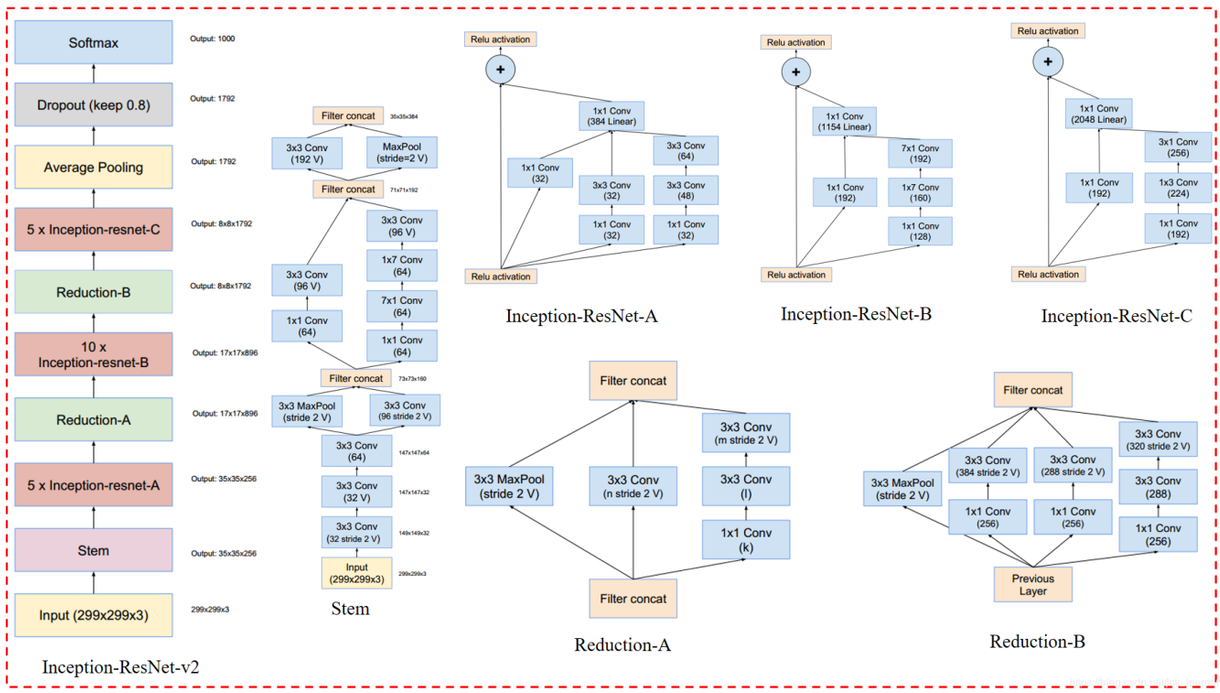
Figure 15 shows the structure of both. However, Inception-v4 is actually slower in practice, probably because it has more layers.

Another small technique is that we use the BN layer in the header of the traditional layer in the Inception-ResNet module, but not in the header of the summations. ** There is reason to believe that the BN layer is effective. But in order to add more Inception modules, we made a compromise between the two.

Inception-ResNet-v1



Inception-ResNet-v2



Scaling of the Residuals

This paper finds that when the number of convolution kernels exceeds 1,000 , the residual variants will start to show instability , and the network will die in the early stages of training, which means that the last layer before the average pooling layer is in the Very few iterations start with just a zero value . This situation

cannot be prevented by reducing the learning rate or by adding a BN layer . Hekaiming's ResNet article also mentions this phenomenon.

This article finds that scale can stabilize the training process before adding the residual module to the activation layer . This article sets the scale coefficient between 0.1 and 0.3.

In order to prevent the occurrence of unstable training of deep residual networks, He suggested in the article that it is divided into two stages of training. The first stage is called warm-up (preheating) , that is, training the model with a very low learning first. In the second stage, a higher learning rate is used. And this article finds that if the convolution sum is very high, even a learning rate of 0.00001 cannot solve this training instability problem, and the high learning rate will also destroy the effect. But this article considers scale residuals to be more reliable than warm-up.

Even if scal is not strictly necessary, it has no effect on the final accuracy, but it can stabilize the training process.

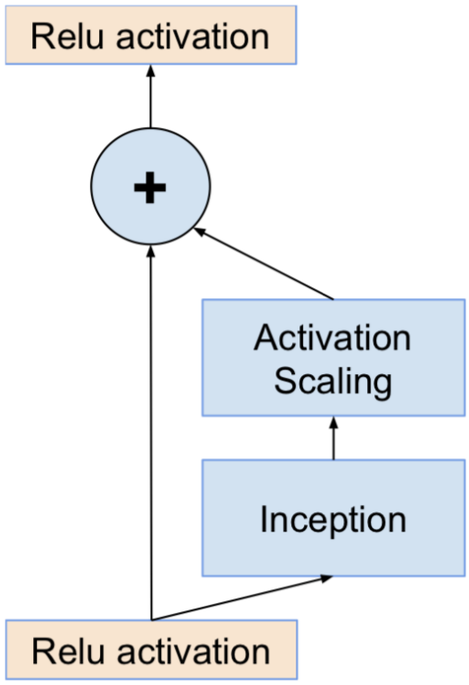


Figure 20. The general schema for scaling combined Inception-resnet moduels. We expect that the same idea is useful in the general resnet case, where instead of the Inception block an arbitrary subnetwork is used. The scaling block just scales the last linear activations by a suitable constant, typically around 0.1.

Conclusion

Inception-ResNet-v1 : a network architecture combining inception module and resnet module with similar calculation cost to Inception-v3;

Inception-ResNet-v2 : A more expensive but better performing network architecture.

Inception-v4 : A pure inception module, without residual connections, but with performance similar to Inception-ResNet-v2.

A big picture of the various module structures of Inception-v4 / Inception-ResNet-v1 / v2:

- Fig3-Stem: (Inception-v4 & Inception-ResNet-v2)

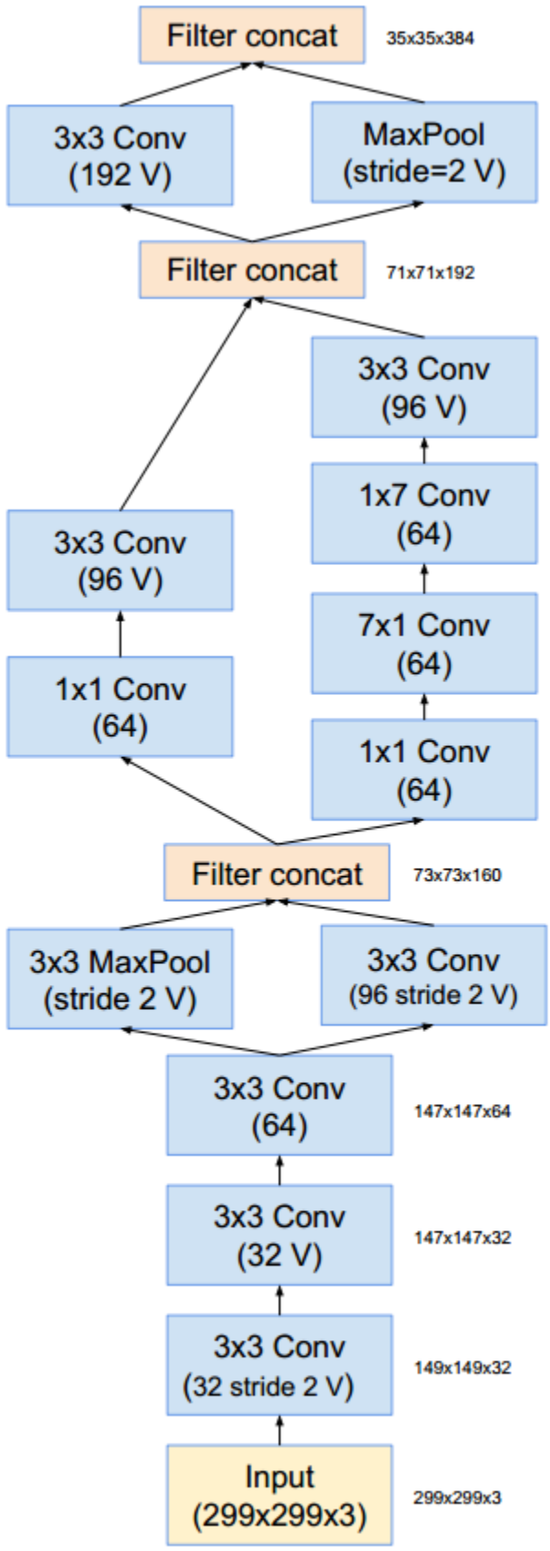


Figure 3. The schema for stem of the pure Inception-v4 and Inception-ResNet-v2 networks. This is the input part of those networks. Cf. Figures 9 and 15

- Fig4-Inception-A: (Inception-v4)

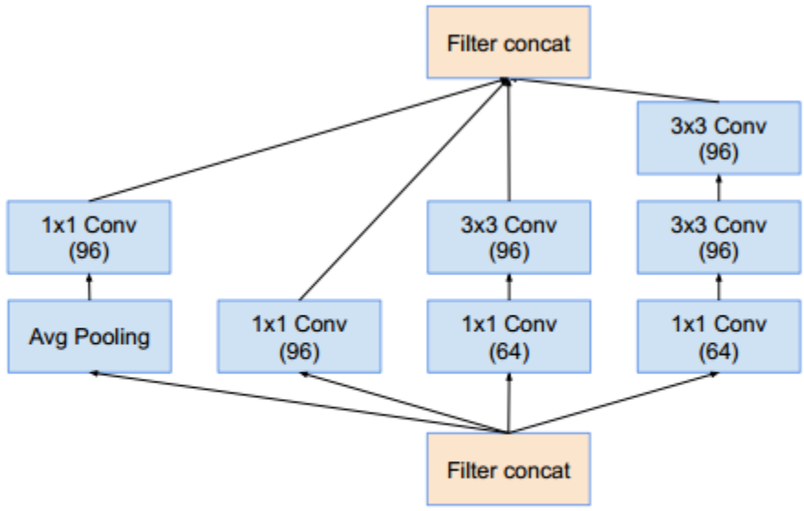


Figure 4. The schema for 35 × 35 grid modules of the pure Inception-v4 network. This is the Inception-A block of Figure 9.

- Fig5-Inception-B: (Inception-v4)

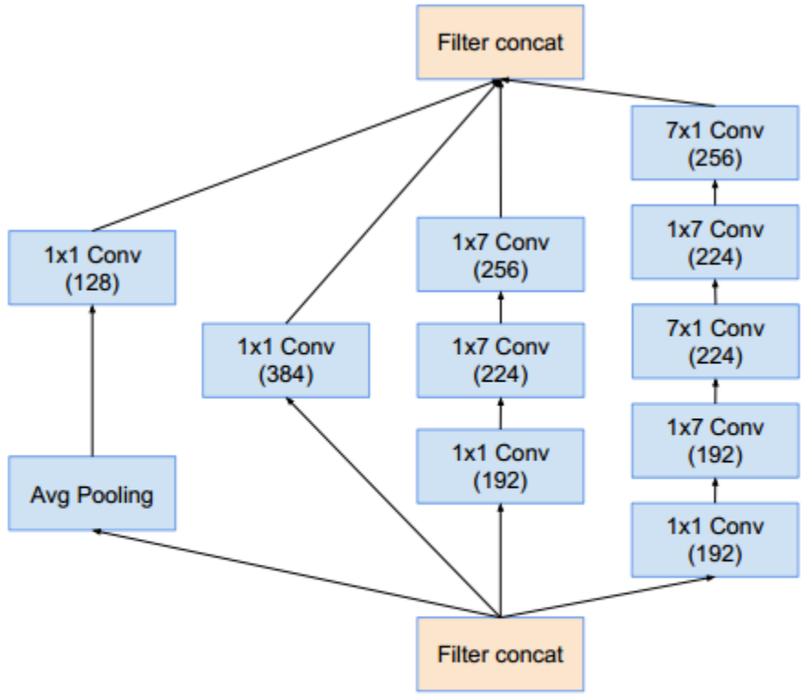


Figure 5. The schema for 17 × 17 grid modules of the pure Inception-v4 network. This is the Inception-B block of Figure 9.

- Fig6-Inception-C: (Inception-v4)

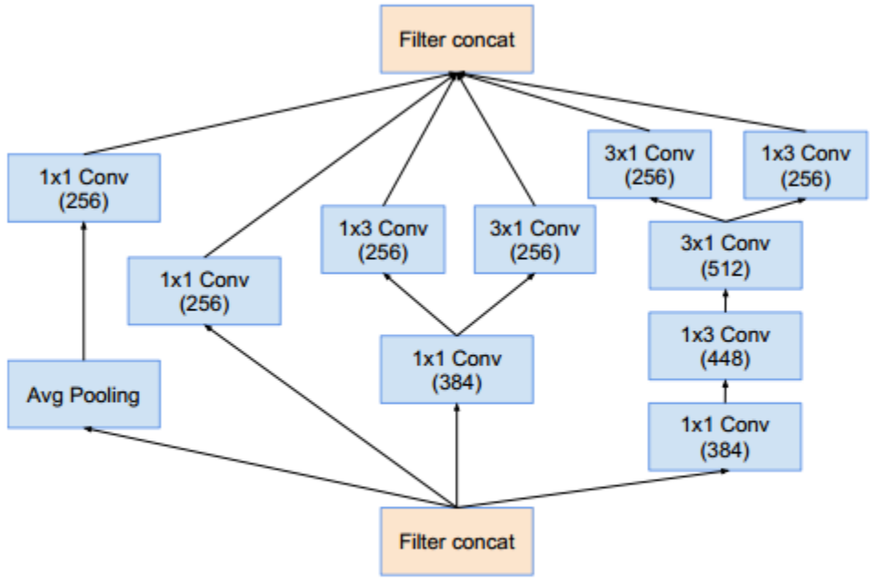


Figure 6. The schema for 8 × 8 grid modules of the pure Inception-v4 network. This is the Inception-C block of Figure 9.

- Fig7-Reduction-A: (Inception-v4 & Inception-ResNet-v1 & Inception-ResNet-v2)

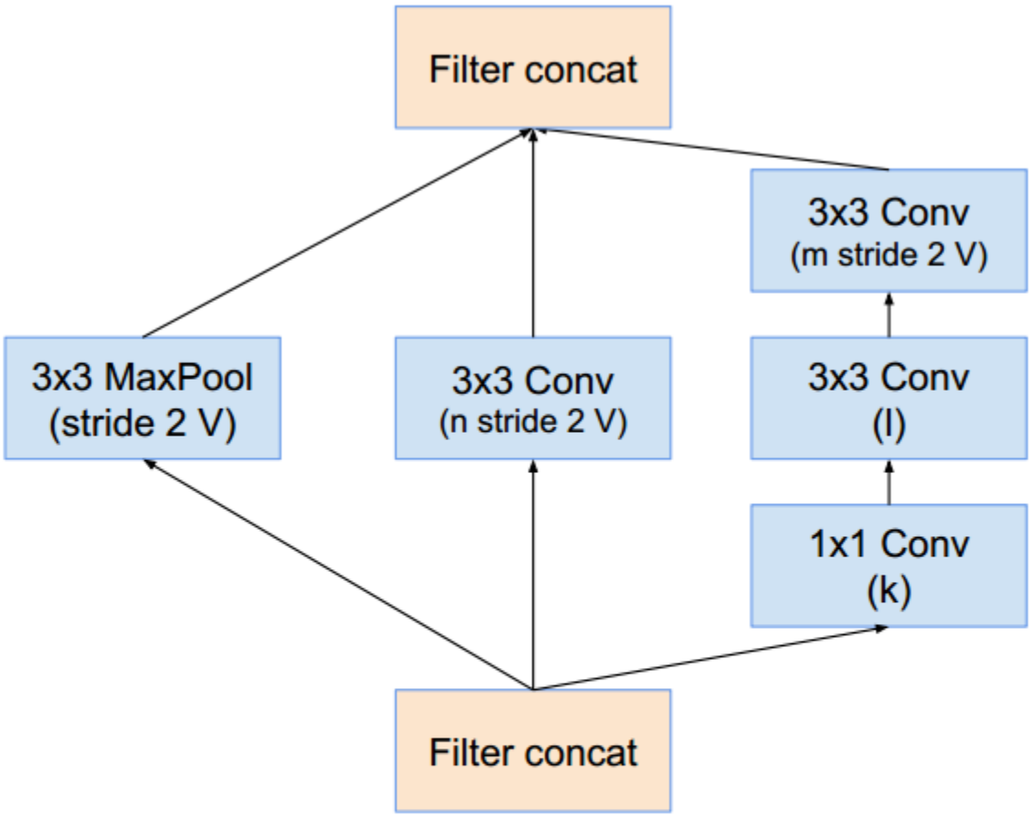


Figure 7. The schema for 35×35 to 17×17 reduction module. Different variants of this blocks (with various number of filters) are used in Figure 9, and 15 in each of the new Inception(-v4, -ResNet-v1, -ResNet-v2) variants presented in this paper. The k, l, m, n numbers represent filter bank sizes which can be looked up in Table 1.

- Fig8-Reduction-B: (Inception-v4)

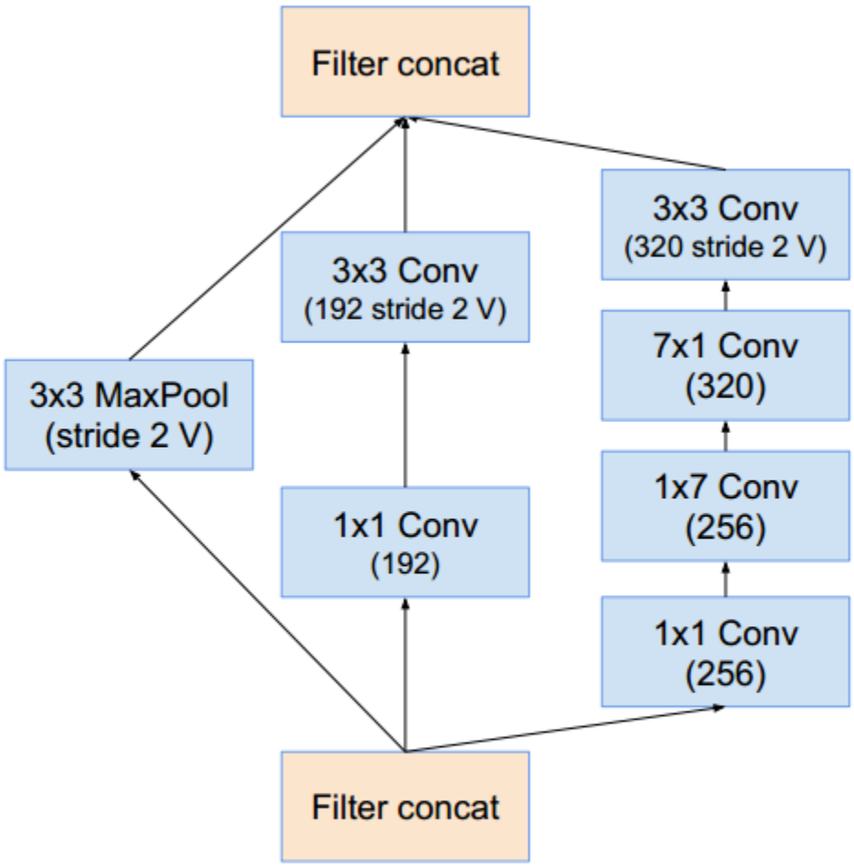


Figure 8. The schema for 17×17 to 8×8 grid-reduction module. This is the reduction module used by the pure Inception-v4 network in Figure 9.

- Fig10-Inception-ResNet-A: (Inception-ResNet-v1)

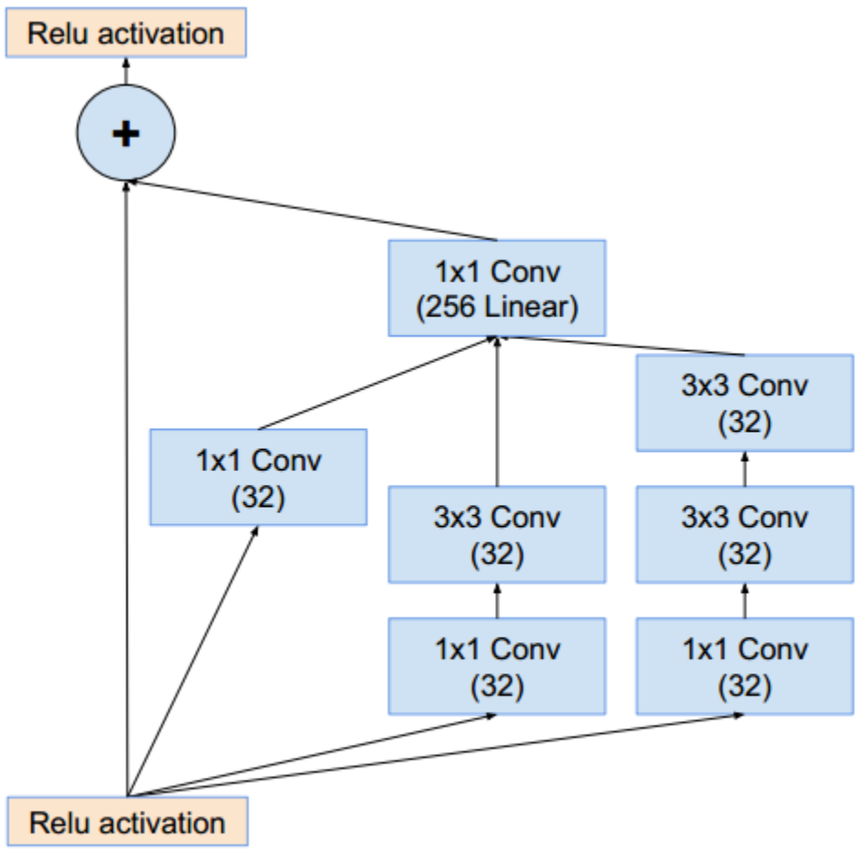


Figure 10. The schema for 35 × 35 grid (Inception-ResNet-A) module of Inception-ResNet-v1 network.

- Fig11-Inception-ResNet-B: (Inception-ResNet-v1)

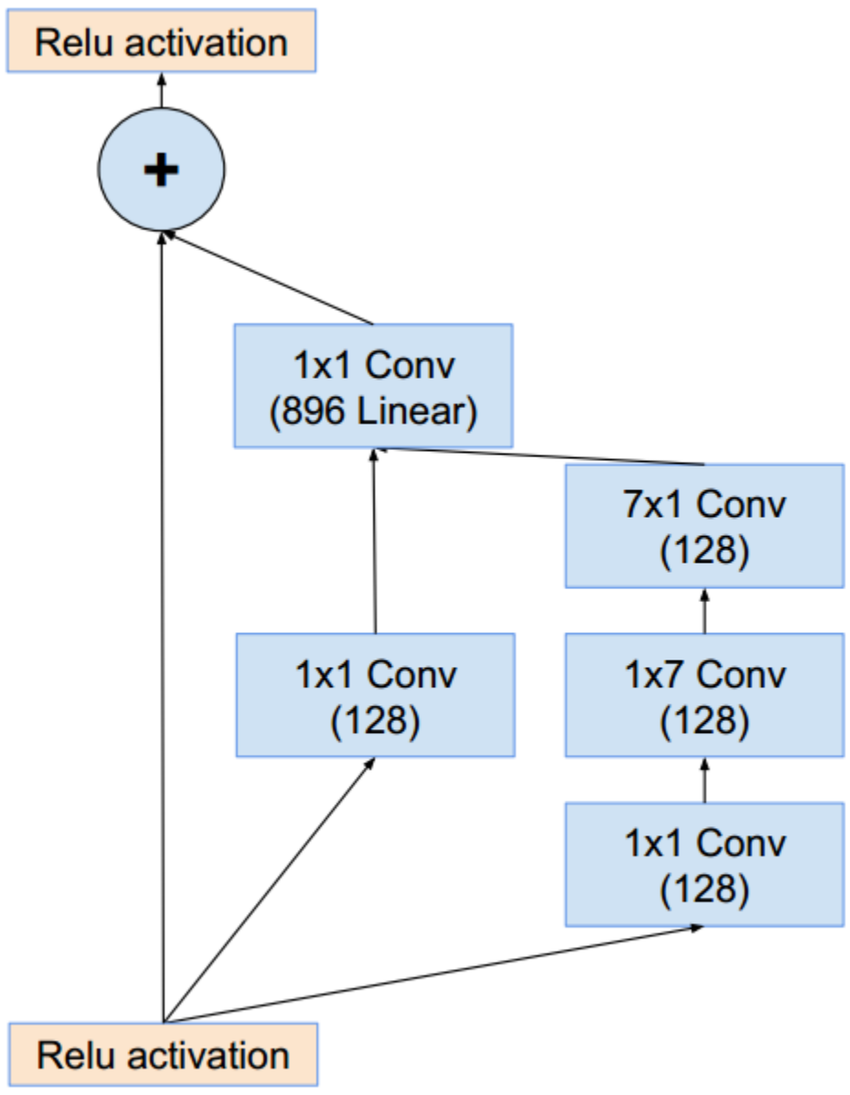


Figure 11. The schema for 17 × 17 grid (Inception-ResNet-B) module of Inception-ResNet-v1 network.

- Fig12-Reduction-B: (Inception-ResNet-v1)

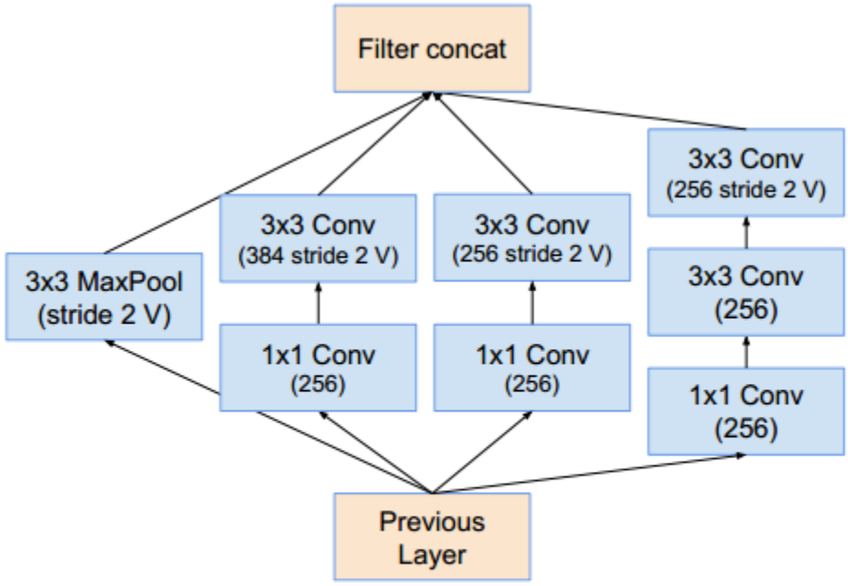


Figure 12. "Reduction-B" 17×17 to 8×8 grid-reduction module. This module used by the smaller Inception-ResNet-v1 network in Figure 15.

- Fig13-Inception-ResNet-C: (Inception-ResNet-v1)

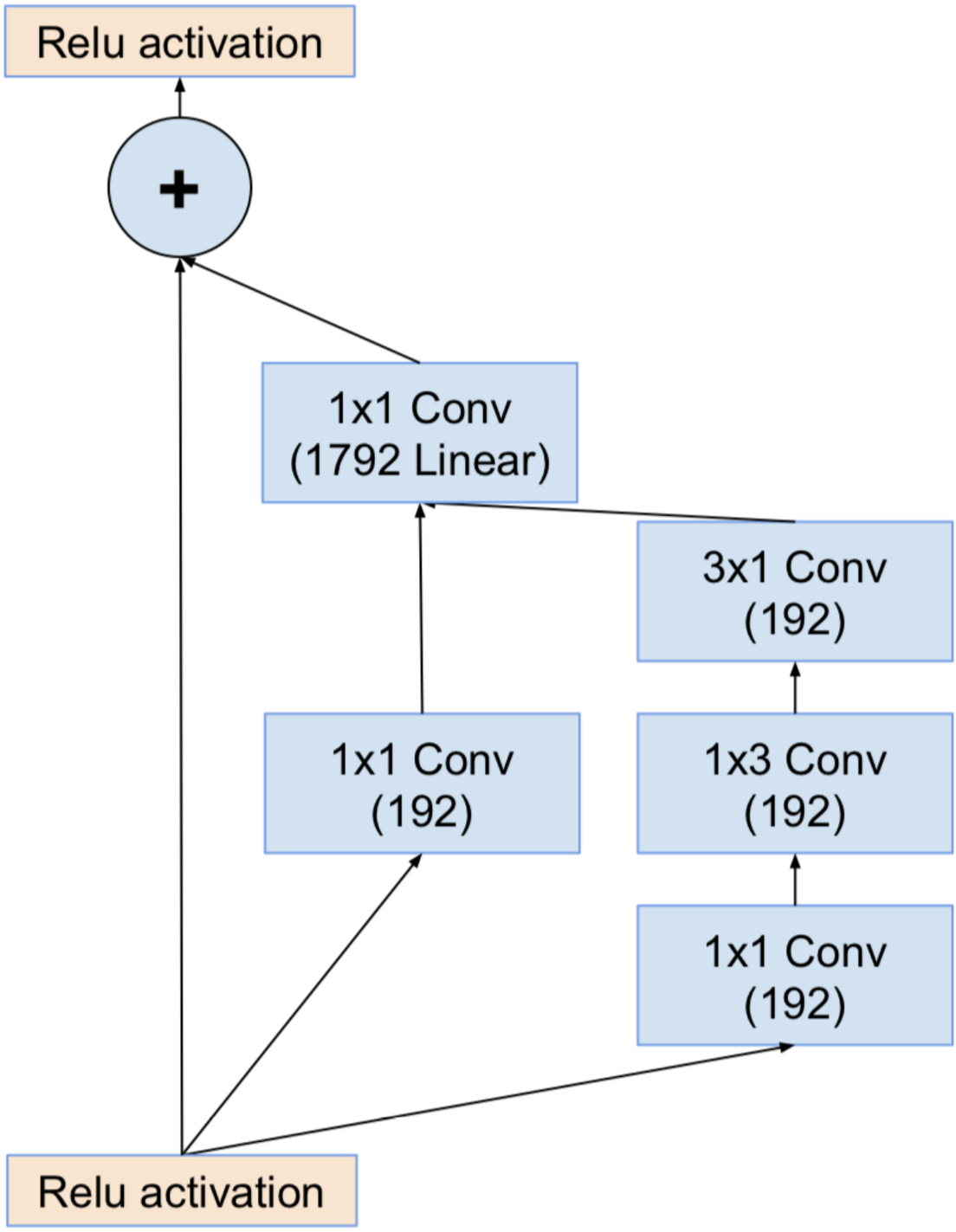
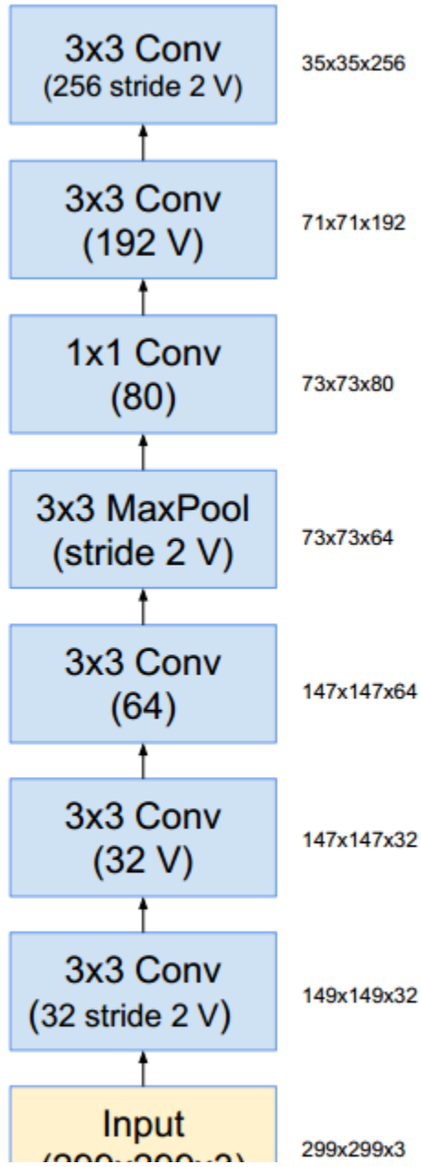


Figure 13. The schema for 8×8 grid (Inception-ResNet-C) module of Inception-ResNet-v1 network.

- Fig14-Stem: (Inception-ResNet-v1)



Code implementation

From Scratch

In [2]:
```
!pip install tflearn
```

```
Looking in indexes: https://pypi.org/simple, (https://pypi.org/simple,) https://us-pyth
on.pkg.dev/colab-wheels/public/simple/ (https://us-python.pkg.dev/colab-wheels/public/s
imple/)
Collecting tflearn
  Downloading tflearn-0.5.0.tar.gz (107 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 107.3/107.3 kB 6.7 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from t
flearn) (1.22.4)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from tfl
earn) (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
=f0daf99db3e5e956ce8947fe092b62231f5895e4149dfeaf4134e3d979b1729b
  Stored in directory: /root/.cache/pip/wheels/55/fb/7b/e06204a0ceefa45443930b9a250cb5e
be31def0e4e8245a465
Successfully built tflearn
Installing collected packages: tflearn
Successfully installed tflearn-0.5.0
```

In [1]:
```
from tensorflow import keras
from keras.models import Model
from keras.layers import Input, Conv2D, MaxPooling2D, concatenate, Flatten, Dense, Avera
from keras.optimizers import Adam
```

In [4]:
```
# 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/compa
t/v2_compat.py:107: disable_resource_variables (from tensorflow.python.ops.variable_sco
pe) is deprecated and will be removed in a future version.
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

Succesfully downloaded 17flowers.tgz 60270631 bytes.
File Extracted
Starting to parse images...
Parsing Done!
```

In [5]:
```
print(x_train.shape)
print(y_train.shape)
```

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

In [3]:
```
# Inception block
def inception_block(x, filters):
    tower_1 = Conv2D(filters[0], (1, 1), padding='same', activation='relu')(x)
    tower_1 = Conv2D(filters[1], (3, 3), padding='same', activation='relu')(tower_1)

    tower_2 = Conv2D(filters[2], (1, 1), padding='same', activation='relu')(x)
    tower_2 = Conv2D(filters[3], (5, 5), padding='same', activation='relu')(tower_2)

    tower_3 = MaxPooling2D((3, 3), strides=(1, 1), padding='same')(x)
    tower_3 = Conv2D(filters[4], (1, 1), padding='same', activation='relu')(tower_3)

    output = concatenate([tower_1, tower_2, tower_3], axis=3)
    return output

# Build the Inception model
def inception(input_shape, num_classes):
    inputs = Input(shape=input_shape)

    x = Conv2D(64, (3, 3), padding='same', activation='relu')(inputs)
    x = MaxPooling2D((2, 2))(x)

    x = inception_block(x, filters=[64, 96, 128, 16, 32])
    x = inception_block(x, filters=[128, 128, 192, 32, 96])
    x = MaxPooling2D((2, 2))(x)

    x = inception_block(x, filters=[192, 96, 208, 16, 48])
    x = inception_block(x, filters=[160, 112, 224, 24, 64])
    x = inception_block(x, filters=[128, 128, 256, 24, 64])
    x = inception_block(x, filters=[112, 144, 288, 32, 64])
    x = MaxPooling2D((2, 2))(x)

    x = inception_block(x, filters=[256, 160, 320, 32, 128])
    x = inception_block(x, filters=[256, 160, 320, 32, 128])
    x = inception_block(x, filters=[384, 192, 384, 48, 128])

    x = AveragePooling2D((4, 4))(x)
    x = Flatten()(x)
    outputs = Dense(num_classes, activation='softmax')(x)

    model = Model(inputs=inputs, outputs=outputs)
    return model
```

In [6]:
```
# Create the Inception model
model = inception(input_shape=(224, 224, 3), num_classes=17)

# Compile the model
model.compile(optimizer=Adam(lr=0.001), loss='categorical_crossentropy', metrics=['accur

# Print a summary of the model
model.summary()

# Train
model.fit(x_train, y_train, batch_size=64, epochs=5, verbose=1,validation_split=0.2, shu
```

```
Model: "model"
__________________________________________________________________________________
_______________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================
==============
 input_1 (InputLayer)           [(None, 224, 224, 3  0           []
                                )]

 conv2d (Conv2D)                (None, 224, 224, 64  1792        ['input_1[0][0]']
                                )

 max_pooling2d (MaxPooling2D)   (None, 112, 112, 64  0           ['conv2d[0][0]']
                                )

 conv2d_1 (Conv2D)              (None, 112, 112, 64  4160        ['max_pooling2d[0]
[0]']
                                )
```

Pretrained

In [7]:
```
# download the data from g drive

import gdown
url = "https://drive.google.com/file/d/12jiQxJzYSYl3wnC8x5wHAhRzzJmmsCXP/view?usp=sharin
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
(https://drive.google.com/uc?/export=download&id=12jiQxJzYSYl3wnC8x5wHAhRzzJmmsCXP)
To: /content/catdog.zip
100%|██████████| 9.09M/9.09M [00:01<00:00, 7.59MB/s]
```

Out[7]:
```
'catdog.zip'
```

In [8]:
```
!unzip catdog.zip
```

```
Archive:  catdog.zip
   creating: train/
   creating: train/Cat/
  inflating: train/Cat/0.jpg
  inflating: train/Cat/1.jpg
  inflating: train/Cat/2.jpg
  inflating: train/Cat/cat.2405.jpg
  inflating: train/Cat/cat.2406.jpg
  inflating: train/Cat/cat.2436.jpg
  inflating: train/Cat/cat.2437.jpg
  inflating: train/Cat/cat.2438.jpg
  inflating: train/Cat/cat.2439.jpg
  inflating: train/Cat/cat.2440.jpg
  inflating: train/Cat/cat.2441.jpg
  inflating: train/Cat/cat.2442.jpg
  inflating: train/Cat/cat.2443.jpg
  inflating: train/Cat/cat.2444.jpg
  inflating: train/Cat/cat.2445.jpg
  inflating: train/Cat/cat.2446.jpg
```

In [9]:

```
from tensorflow import keras
from tensorflow.keras.applications.inception_v3 import InceptionV3, 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 InceptionV3 model without the top layer
base_model = InceptionV3(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))
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')
```

```
WARNING:tensorflow:From /usr/local/lib/python3.10/dist-packages/keras/layers/normalizat
ion/batch_normalization.py:581: _colocate_with (from tensorflow.python.framework.ops) i
s deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/inception_v3/inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5 (https://storage.googleapis.com/tensorflow/keras-applications/inception_v3/inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5)
87910968/87910968 [==============================] - 3s 0us/step
Found 337 images belonging to 2 classes.
Found 59 images belonging to 2 classes.
Epoch 1/5
125/125 [==============================] - 20s 135ms/step - batch: 62.0000 - size: 15.2800 - loss: 2.2705 - acc: 0.8613 - val_loss: 0.4283 - val_acc: 0.9500
Epoch 2/5
125/125 [==============================] - 15s 118ms/step - batch: 62.0000 - size: 15.2800 - loss: 1.0321 - acc: 0.8927 - val_loss: 0.3787 - val_acc: 0.9662
Epoch 3/5
125/125 [==============================] - 14s 109ms/step - batch: 62.0000 - size: 15.4000 - loss: 0.1464 - acc: 0.9605 - val_loss: 0.1556 - val_acc: 0.9676
Epoch 4/5
125/125 [==============================] - 14s 116ms/step - batch: 62.0000 - size: 15.2800 - loss: 0.3796 - acc: 0.9314 - val_loss: 0.2708 - val_acc: 0.9676
Epoch 5/5
125/125 [==============================] - 14s 109ms/step - batch: 62.0000 - size: 15.4000 - loss: 0.2161 - acc: 0.9626 - val_loss: 0.2489 - val_acc: 0.9662
```

In [ ]: