Resnet

> Indented block

Introduction

ResNet is a network structure proposed by the He Kaiming, Sun Jian and others of Microsoft Research Asia in 2015, and won the first place in the ILSVRC-2015 classification task. At the same time, it won the first place in ImageNet detection, ImageNet localization, COCO detection, and COCO segmentation tasks. It was a sensation at the time.

ResNet, also known as residual neural network, refers to the idea of adding residual learning to the traditional convolutional neural network, which solves the problem of gradient dispersion and accuracy degradation (training set) in deep networks, so that the network can get more and more The deeper, both the accuracy and the speed are controlled.

> Deep Residual Learning for Image Recognition Original link : ResNet Paper

The problem caused by increasing depth

- The first problem brought by increasing depth is the problem of gradient explosion / dissipation . This is because as the number of layers increases, the gradient of backpropagation in the network will become unstable with continuous multiplication, and become particularly large or special. small. Among them , the problem of gradient dissipation often occurs .
- In order to overcome gradient dissipation, many solutions have been devised, such as using BatchNorm, replacing the activation function with ReLu, using Xaiver initialization, etc. It can be said that gradient dissipation has been well solved
- Another problem of increasing depth is the problem of network degradation, that is, as the depth increases, the performance of the network will become worse and worse, which is directly reflected in the decrease in accuracy on the training set. The residual network article solves this problem. And after this problem is solved, the depth of the network has increased by several orders of magnitude.

Degradation of deep network

> With network depth increasing, accuracy gets saturated (which might be unsurprising) and then degrades rapidly. Unexpectedly, such degradation is not caused by overfitting, and adding more layers to a favored deep model leads to higher training error.

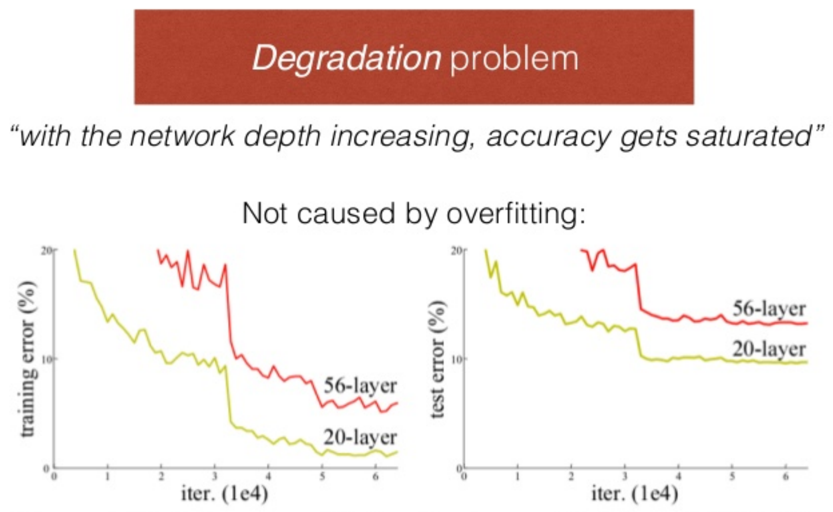


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error.

- The above figure is the error rate of the training set classified by the network on the CIFAR10-data set with the increase of the network depth . It can be seen that if we directly stack the convolutional layers, as the number of layers increases, the error rate increases significantly. The trend is that the deepest 56-layer network has the worst accuracy . We verified it on the VGG network. For the CIFAR-10 dataset, it took 5 minutes on the 18-layer VGG network to get the full network training. The 80% accuracy rate was achieved, and the 34-layer VGG model took 8 minutes to get the 72% accuracy rate. The problem of network degradation does exist.
- The decrease in the training set error rate indicates that the problem of degradation is not caused by overfitting. The specific reason is that it is left for further study. The author's other paper "Identity Mappings in Deep Residual Networks" proved the occurrence of degradation. It is because the optimization performance is not good, which indicates that the deeper the network, the more difficult the reverse gradient is to conduct.

Deep Residual Networks

From 10 to 100 layers

We can imagine that *when we simply stack the network directly to a particularly long length, the internal characteristics of the network have reached the best situation in one of the layers. At this time, the remaining layers should not make any changes to the characteristics and learn automatically. The form of identity mapping*. That is to say, for a particularly deep deep network, the solution space of the shallow form of the network should be a subset of the solution space of the deep network, in other words, a network deeper than the shallow network will not have at least Worse effect, but this is not true because of network degradation.

Then, we settle for the second best. In the case of network degradation, if we do not add depth, we can improve the accuracy. Can we at least make the deep network achieve the same performance as the shallow network, that is, let the layers behind the deep network achieve at least The role of identity mapping . Based on this idea, the author proposes a residual module to help the network achieve identity mapping.

`To understand ResNet, we must first understand what kind of problems will occur when the network becomes deeper.`

The first problem brought by increasing the network depth is the disappearance and explosion of the gradient.

This problem was successfully solved after Szegedy proposed the **BN (Batch Normalization)** structure. The BN layer can normalize the output of each layer. The size can still be kept stable after the reverse layer transfer, and it will not be too small or too large.

Is it easy to converge after adding BN and then increasing the depth?

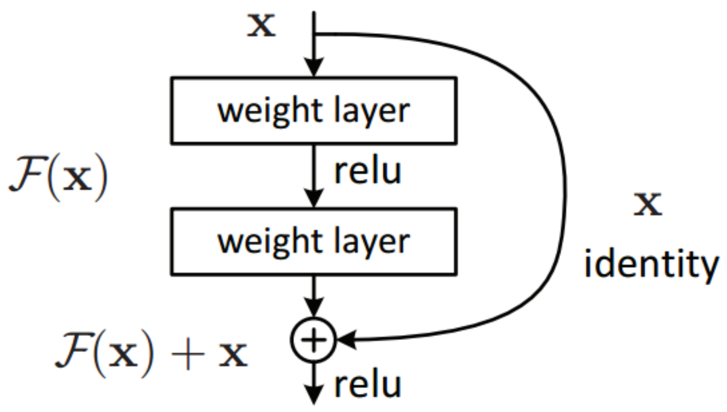
The answer is still **negative**. The author mentioned the second problem-**the degradation problem**: when the level reaches a certain level, the accuracy will saturate and then decline rapidly. This decline is not caused by the disappearance of the gradient. It is not caused by overfit, but because the network is so complicated that it is difficult to achieve the ideal error rate by unconstrained stocking training alone.

The degradation problem is not a problem of the network structure itself, but is caused by the current insufficient training methods. The currently widely used training methods, whether it is SGD, AdaGrad, or RMSProp, cannot reach the theoretically optimal convergence result after the network depth becomes larger.

We can also prove that as long as there is an ideal training method, deeper networks will definitely perform better than shallow networks.

The proof process is also very simple: `Suppose that several layers are added behind a network A to form a new network B. If the added level is just an identity mapping of the output of A, that is, the output of A is after the level of B becomes the output of B, there is no change, so the error rates of network A and network B are equal, which proves that the deepened network will not be worse than the network before deepening.`

He Kaiming proposed a residual structure to implement the above identity mapping (Below Figure): In addition to the normal convolution layer output, the entire module has a branch directly connecting the input to the output. The output and the output of the convolution do The final output is obtained by arithmetic addition. The formula is H (x) = F (x) + x, x is the input, F (x) is the output of the convolution branch, and H (x) is the output of the entire structure. It can be shown that if all parameters in the F (x) branch are 0, H (x) is an identity mapping. The residual structure artificially creates an identity map, which can make the entire structure converge in the direction of the identity map, ensuring that the final error rate will not become worse because the depth becomes larger. If a network can achieve the desired result by simply setting the parameter values by hand, then this structure can easily converge to the result through training. This is a rule that is unsuccessful when designing complex networks. Recall that in order to restore the original distribution after BN processing, the formula y = rx + delta is used. When r is manually set to standard deviation and delta is the mean, y is the distribution before BN processing. This is the use of this Rules.


x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

What does residual learning mean?

The idea of residual learning is the above picture, which can be understood as a block, defined as follows:

$y = F(x, \{W_i\}) + x$

The residual learning block contains two branches or two mappings:

1. Identity mapping refers to the curved curve on the right side of the figure above. As its name implies, identity mapping refers to its own mapping, which is x itself;
2. *F*(x) Residual mapping refers to another branch, that is, part. This part is called residual mapping (y -x) .

What role does the residual module play in back propagation?

- The residual module will significantly reduce the parameter value in the module, so that the parameters in the network have a more sensitive response ability to the loss of reverse conduction, although the fundamental It does not solve the problem that the loss of backhaul is too small, but it reduces the parameters. Relatively speaking, it increases the effect of backhaul loss and also generates a certain regularization effect.
- Secondly, because there are branches of the identity mapping in the forward process, the gradient conduction in the back-propagation process also has more simple paths , and the gradient can be transmitted to the previous module after only one relu.
- The so-called backpropagation is that the network outputs a value, and then compares it with the real value to an error loss. At the same time, the loss is changed to change the parameter. The returned loss depends on the original loss and gradient. Since the purpose is to change the parameter, The problem is that if the intensity of changing the parameter is too small, the value of the parameter can be reduced, so that the loss of the intensity of changing the parameter is relatively greater.
- Therefore, the most important role of the residual module is to change the way of forward and backward information transmission, thereby greatly promoting the optimization of the network.
- Using the four criteria proposed by Inceptionv3, we will use them again to improve the residual module. Using criterion 3, the dimensionality reduction before spatial aggregation will not cause information loss, so the same method is also used here, adding 1 * 1 convolution The kernel is used to increase the non-linearity and reduce the depth of the output to reduce the computational cost. You get the form of a residual module that becomes a bottleneck. The figure above shows the basic form on the left and the bottleneck form on the right.
- To sum up, the shortcut module will help the features in the network perform identity mapping in the forward process, and help conduct gradients in the reverse process, so that deeper models can be successfully trained.

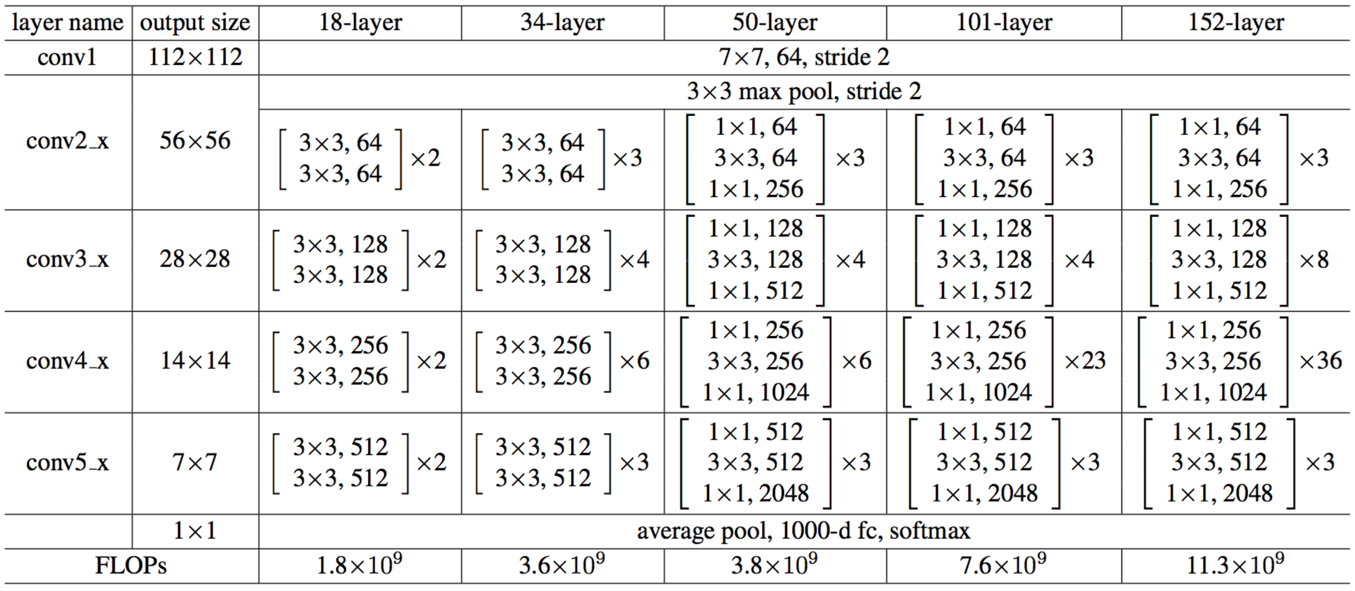
Why can the residual learning solve the problem of "the accuracy of the network deepening declines"?

For a neural network model, if the model is optimal, then training can easily optimize the residual mapping to 0, and only identity mapping is left at this time. No matter how you increase the depth, the network will always be in an optimal state in theory. Because it is equivalent to all the subsequent added networks to carry information transmission along the identity mapping (self), it can be understood that the number of layers behind the optimal network is discarded (without the ability to extract features), and it does not actually play a role. . In this way, the performance of the network will not decrease with increasing depth.

The author used two types of data, **ImageNet** and **CIFAR**, to prove the effectiveness of ResNet:

The first is ImageNet. The authors compared the training effect of ResNet structure and traditional structure with the same number of layers. The left side of Figure is a VGG-19 network with a traditional structure (each followed by BN), the middle is a 34-layer network with a traditional structure (each followed by BN), and the right side is 34 layers ResNet (the solid line indicates a direct connection, and the dashed line indicates a dimensional change using 1x1 convolution to match the number of features of the input and output). Figure 3 shows the results after training these types of networks.

The data on the left shows that the 34-layer network (red line) with the traditional structure has a higher error rate than the VGG-19 (blue-green line). Because the BN structure is added to each layer Therefore, the high error is not caused by the gradient disappearing after the level is increased, but by the degradation problem; the ResNet structure on the right side of Figure 3 shows that the 34-layer network (red line) has a higher error rate than the 18-layer network (blue-green line). Low, this is because the ResNet structure has overcome the degradation problem. In addition, the final error rate of the ResNet 18-layer network on the right is similar to the error rate of the traditional 18-layer network on the left. This is because the 18-layer network is simpler and can converge to a more ideal result even without the ResNet structure.



layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times23$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

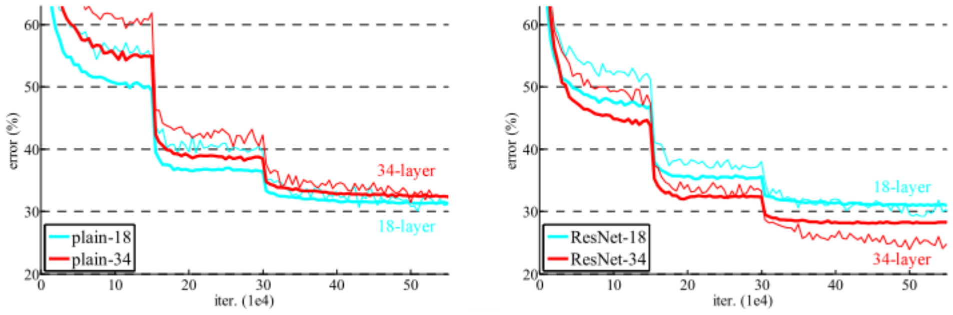
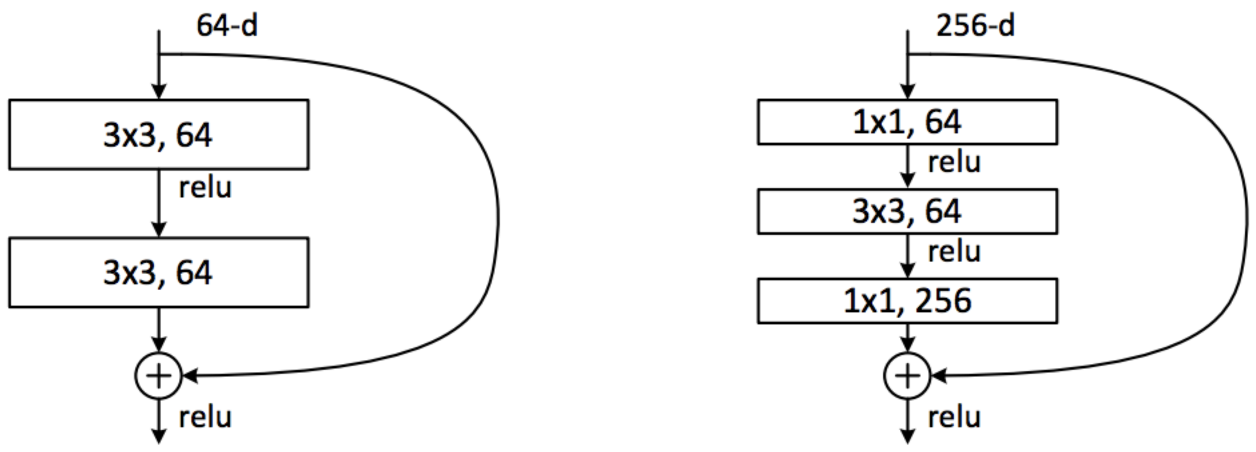


Figure 4. Training on **ImageNet**. Thin curves denote training error, and bold curves denote validation error of the center crops. Left: plain networks of 18 and 34 layers. Right: ResNets of 18 and 34 layers. In this plot, the residual networks have no extra parameter compared to their plain counterparts.

The ResNet structure like the left side of Fig. 4 is only used for shallow ResNet networks. If there are many network layers, the dimensions near the output end of the network will be very large. Still using the structure on the left side of Fig. 4 will cause a huge amount of calculation. For deeper networks, we all use the bottleneck structure on the right side of Figure 4, first using a 1x1 convolution for dimensionality reduction, then 3x3 convolution, and finally using 1x1 dimensionality to restore the original dimension.

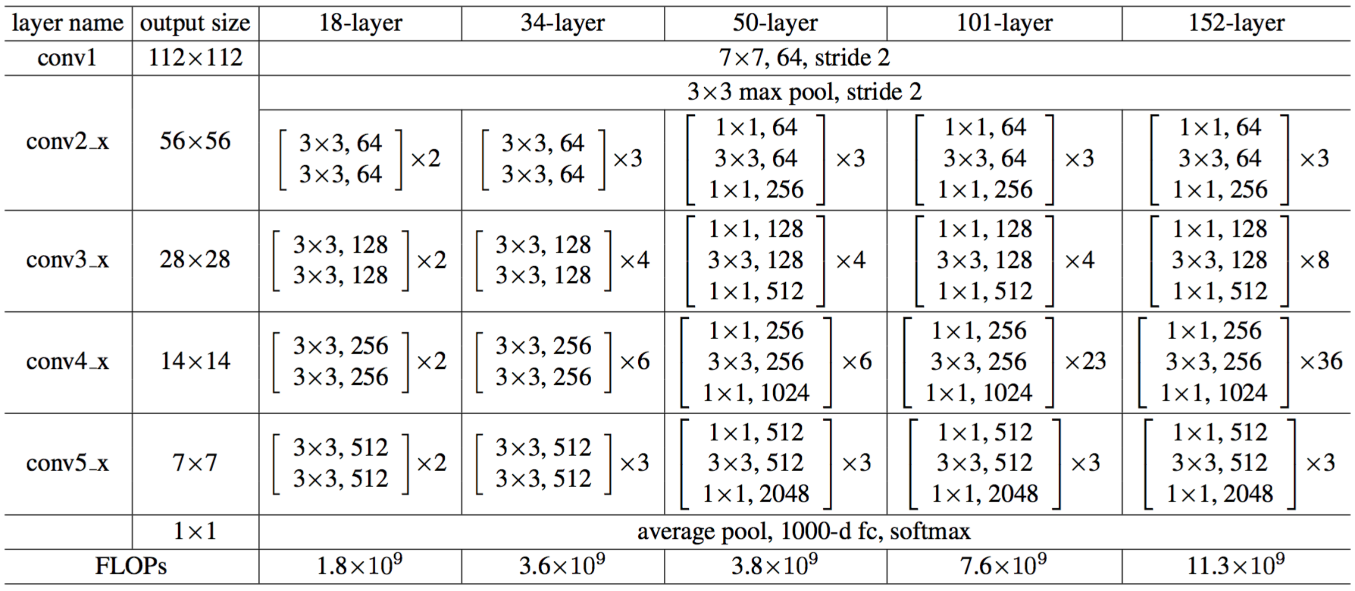
In practice, considering the cost of the calculation, the residual block is calculated and optimized, that is, the two 3x3 convolution layers are replaced with 1x1 + 3x3 + 1x1 , as shown below. The middle 3x3 convolutional layer in the new structure first reduces the calculation under one dimensionality-reduced 1x1 convolutional layer , and then restores it under another 1x1 convolutional layer , both maintaining accuracy and reducing the amount of calculation .



This is equivalent to reducing the amount of parameters for the same number of layers , so it can be extended to deeper models. So the author proposed ResNet with 50, 101 , and 152 layers , and not only did not have degradation problems, the error rate was greatly reduced, and the computational complexity was also kept at a very low level .

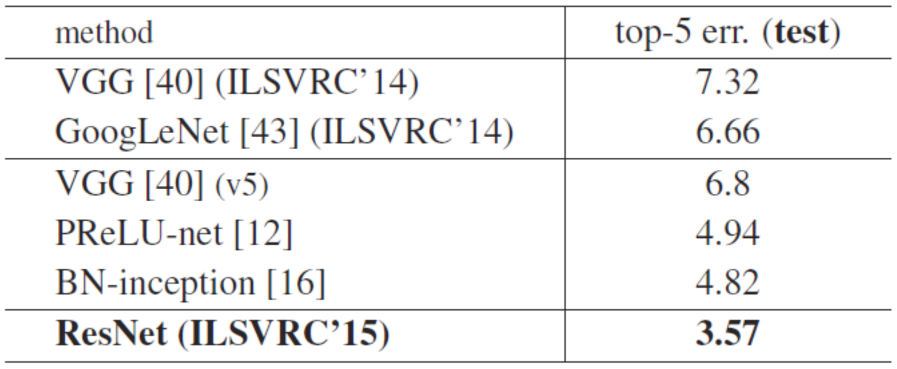
At this time, the error rate of ResNet has already dropped other networks a few streets, but it does not seem to be satisfied. Therefore, a more abnormal 1202 layer network has been built. For such a deep network, optimization is still not difficult, but it appears The problem of overfitting is quite normal. The author also said that the 1202 layer model will be further improved in the future.

Diffrent Variants : -



layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times23$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Below is the transcript of resnet, winning the championship at ImageNet2015



method	top-5 err. (**test**)
VGG [40] (ILSVRC'14)	7.32
GoogLeNet [43] (ILSVRC'14)	6.66
VGG [40] (v5)	6.8
PReLU-net [12]	4.94
BN-inception [16]	4.82
ResNet (ILSVRC'15)	**3.57**

Code Implementation

From Scratch

```
!pip install tflearn
```

```
    Collecting tflearn
      Downloading tflearn-0.5.0.tar.gz (107 kB)
         ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 107.3/107.3 kB 1.4 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=ea6740443604531aa09085a45f28dc739325f7f18b9
      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
from keras.models import Model
from keras.layers import Conv2D, BatchNormalization, Activation, Add, AveragePooling2D, Flatten, Dense
from keras.optimizers import Adam
```

```
# 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_variabl
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!
```

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

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

```
# Residual block
def residual_block(x, filters, downsample=False):
    strides = (2, 2) if downsample else (1, 1)

    # First convolutional layer of the block
    y = Conv2D(filters, kernel_size=(3, 3), strides=strides, padding='same')(x)
    y = BatchNormalization()(y)
    y = Activation('relu')(y)

    # Second convolutional layer of the block
    y = Conv2D(filters, kernel_size=(3, 3), strides=(1, 1), padding='same')(y)
    y = BatchNormalization()(y)

    # Skip connection if downsample or number of filters change
    if downsample:
        x = Conv2D(filters, kernel_size=(1, 1), strides=(2, 2), padding='same')(x)

    # Add skip connection
    y = Add()([x, y])
    y = Activation('relu')(y)
    return y

# Build the ResNet model
def resnet(input_shape, num_classes):
    inputs = keras.Input(shape=input_shape)

    # Initial convolutional layer
    x = Conv2D(16, kernel_size=(3, 3), strides=(1, 1), padding='same')(inputs)
    x = BatchNormalization()(x)
    x = Activation('relu')(x)

    # Residual blocks
    x = residual_block(x, filters=16)
    x = residual_block(x, filters=16)
    x = residual_block(x, filters=32, downsample=True)
    x = residual_block(x, filters=32)
    x = residual_block(x, filters=64, downsample=True)
    x = residual_block(x, filters=64)

    # Average pooling and output layer
    x = AveragePooling2D(pool_size=(8, 8))(x)
    x = Flatten()(x)
    outputs = Dense(num_classes, activation='softmax')(x)

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

```
# Create the ResNet model
model = resnet(input_shape=(224, 224, 3), num_classes=17)

# Compile the model
model.compile(optimizer=Adam(lr=0.001), 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)
```

```
WARNING:tensorflow:From /usr/local/lib/python3.10/dist-packages/keras/layers/normalization/batch_normalization.py:581: _colocate_wi
Instructions for updating:
Colocations handled automatically by placer.
/usr/local/lib/python3.10/dist-packages/keras/optimizers/legacy/adam.py:117: UserWarning: The `lr` argument is deprecated, use `lea
  super().__init__(name, **kwargs)
Train on 1088 samples, validate on 272 samples
Epoch 1/5
1088/1088 [==============================] - ETA: 0s - loss: 2.7651 - acc: 0.1820/usr/local/lib/python3.10/dist-packages/keras/engi
  updates = self.state_updates
1088/1088 [==============================] - 23s 21ms/sample - loss: 2.7651 - acc: 0.1820 - val_loss: 2.8806 - val_acc: 0.0772
Epoch 2/5
1088/1088 [==============================] - 9s 9ms/sample - loss: 1.6809 - acc: 0.4614 - val_loss: 3.0135 - val_acc: 0.0441
Epoch 3/5
1088/1088 [==============================] - 10s 9ms/sample - loss: 1.2754 - acc: 0.5836 - val_loss: 3.1346 - val_acc: 0.0441
Epoch 4/5
1088/1088 [==============================] - 9s 9ms/sample - loss: 0.9813 - acc: 0.6728 - val_loss: 3.6132 - val_acc: 0.0441
Epoch 5/5
1088/1088 [==============================] - 9s 9ms/sample - loss: 0.8721 - acc: 0.6958 - val_loss: 3.7739 - val_acc: 0.0441
<keras.callbacks.History at 0x7ff16ba62680>
```

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, 32.2MB/s]
'catdog.zip'
```

```
!unzip catdog.zip
```

```
  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
```

```python
from tensorflow import keras

from keras.datasets import cifar10
from tensorflow.keras.applications.resnet50 import ResNet50, 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 = ResNet50(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')
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kerne
94765736/94765736 [==============================] - 0s 0us/step
Found 337 images belonging to 2 classes.
Found 59 images belonging to 2 classes.
Epoch 1/5
125/125 [==============================] - 20s 136ms/step - batch: 62.0000 - size: 15.2800 - loss: 1.8437 - acc: 0.9487 - val_loss:
```