Case Study of Deep ConvNet

Montag, 20. Juli 2020 16:38

Classic ConvNet

LeNet

AlexNet

VGG - 16

Conv = 3*3 filter, s = 1, same padding Max-pool = 2*2, s=2

$224 \times 224 \times 3 \xrightarrow[\times 2]{[\text{CONV } 64]} 224 \times 224 \times 64 \xrightarrow{\text{Pool}} 112 \times 112 \times 64$

$\xrightarrow[\times 2]{[\text{CONV } 128]} 112 \times 112 \times 128 \xrightarrow{\text{Pool}} 56 \times 56 \times 128 \xrightarrow[\times 3]{[\text{CONV } 256]} 56 \times 56 \times 256$

$\xrightarrow{\text{Pool}} 28 \times 28 \times 256 \xrightarrow[\times 3]{[\text{CONV } 512]} 28 \times 28 \times 512$

$\xrightarrow{\text{Pool}} 14 \times 14 \times 512 \xrightarrow[\times 3]{[\text{CONV } 512]} 14 \times 14 \times 512$

$\xrightarrow{\text{Pool}} 7 \times 7 \times 512 \longrightarrow \underset{4096}{\text{FC}} \longrightarrow \text{FC} \longrightarrow \underset{1000}{\text{softmax}}$

~ 138 M parameters

$\longrightarrow$

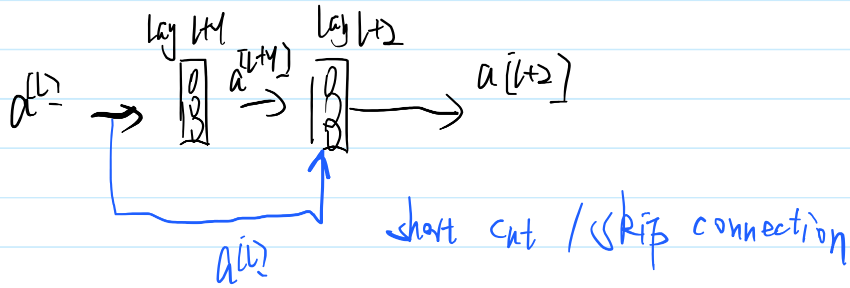
$n_H, n_W \downarrow$

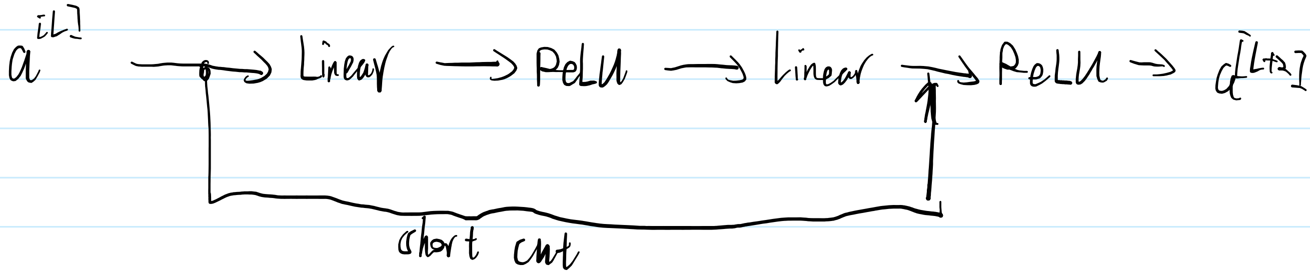
$n_C \uparrow$

Residual networks -- ResNets

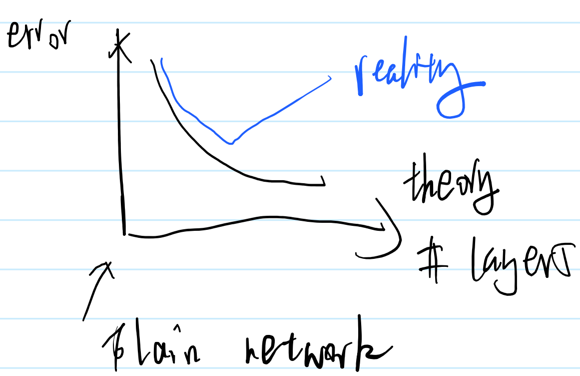
residual block creates short cut to build very deep network, allows you to take the activation from one layer and suddenly feed it to another layer even much deeper in the neural network.

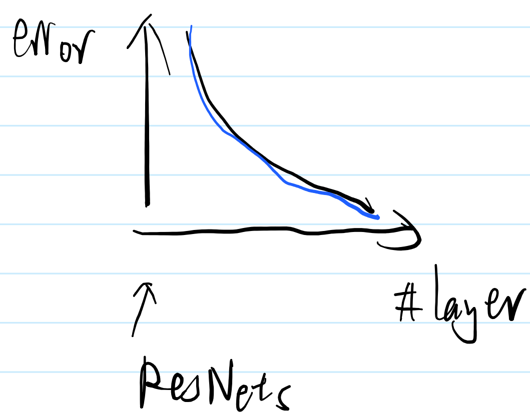
deep neural networks are difficult to train because of vanishing and exploding gradient types of problems.





$$a^{[l+2]} = g(z^{[l+2]} + a^{[l]})$$





1*1 Convolution / Network in network

$$28 \times 28 \times 192 \xrightarrow[\text{CONV } 1\times1]{\text{ReLU}} 28 \times 28 \times 32$$

→ n_H, n_W stay the same

→ decrease the # channels

32

$1 \times 1 \times 192$

32 filters

32 filters

Inception Network

do many things in one layer

28x28x192

1x1. 28x28x64

3x3 same 28x28x128

5x5 same 28x28x32

max pool. padding 28x28x32

28x28x 256.

computation cost

28x28x32 x 5x5x192

= 120M parameters

Using 1*1 convolution can decrese the computational cost rapidly

bottleneck layer

28x28x192 $\xrightarrow{}$ 28x28x16 $\xrightarrow{}$ 28x28x32

CONV 1x1 #16 1x1 x 192

CONV 5x5 #32 5x5x16

28x28x16 x1x1x192 = 2.4M

28x28x32 x 5x5x16. = 10 M

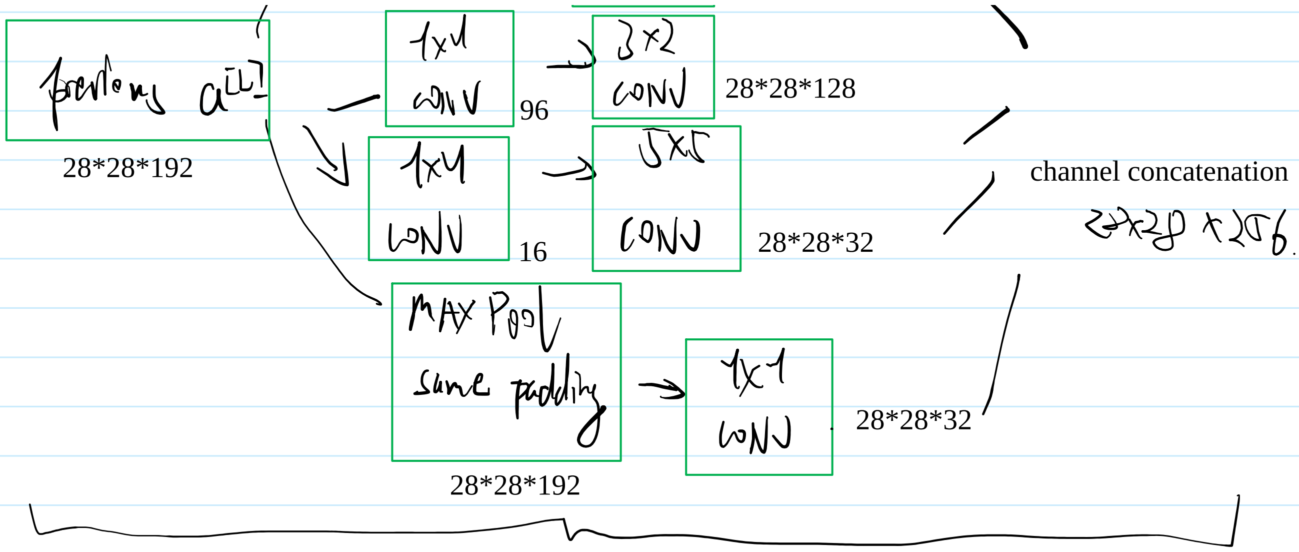
12.4 M

Previous a[l]

1x1 CONV 28*28*64

1x1 CONV

3x3 CONV 28*28*128



One Inception module

GoogleNet

Transfer Learning

https://towardsdatascience.com/transfer-learning-from-pre-trained-models-f2393f124751

① x → [] → [] → [] → [] → ~~softmax O~~ → ~~$\hat{Y}$~~ → code and weight from others' work

don't train again

freeze

O → $\hat{Y}$ → train for personal usage

② x → [] → [] → [] → [] → [] → [] → O → $\hat{Y}$

$\underbrace{\text{[] → [] → []}}_{\text{freeze}}$ $\underbrace{\text{[] → [] → [] → O → } \hat{Y}}_{\text{train}}$

③ x → [] → [] → [] → - — → O → $\hat{Y}$

download pre-trained weight to initionize weight and train

Take away from programming assignment:

- keras introduction

  a. Keras re-uses and overwrites the same variable at each step, which means there will be no Z1, A1, ...Zl, Al. (Except for X_input)
  b. X = ZeroPadding2D((3, 3))(X_input): this single line is equivalent to:
     i. ZP = ZeroPadding2D((3, 3))   # ZP is an object that can be called as a function
     ii. X = ZP(X_input)
  c. A outline in Keras to build a ML model:

```
def model(input)    # input = (height, width, channels)

    # Define the input placeholder as a tensor with shape input_shape.
    X_input = Input(input)

    # Zero-Padding: pads the border of X_input with zeroes
    X = ZeroPadding2D((3, 3))(X_input)

    # CONV -> BN -> RELU Block applied to X
    X = Conv2D(32, (7, 7), strides = (1, 1), name = 'conv0')(X)
    X = BatchNormalization(axis = 3, name = 'bn0')(X)
    X = Activation('relu')(X)

    # MAXPOOL
    X = MaxPooling2D((2, 2), name='max_pool')(X)

    # FLATTEN X (means convert it to a vector) + FULLYCONNECTED
    X = Flatten()(X)
    X = Dense(1, activation='sigmoid', name='fc')(X)

    # Create model. This creates the Keras model instance
    model = Model(inputs = X_input, outputs = X, name='HappyModel')

    return model
```

  a. There are other functions to build a model such as: AveragePooling2D(), GlobalMaxPooling2D(), Dropout(), etc.
  b. After build a model:
     - Create the model by calling the function
     - Compile the model by calling model.compile(optimizer = "...", loss = "...", metrics = ["accuracy"])
     - Train the model on train data by calling model.fit(x = ..., y = ..., epochs = ..., batch_size = ...)
     - Test the model on test data by calling model.evaluate(x = ..., y = ...)
  c. Optimiers: such as 'adam' and the loss such as 'binary_crossentropy' for binary classification problem
  d. model.summary(): prints the details of your layers in a table with the sizes of its inputs/outputs
  e. plot_model(): plots your graph in a nice layout. You can even save it as ".png" using SVG() if you'd like to share it on social media ;). It is saved in "File" then "Open..." in the upper bar of the notebook. -> plot_model(happyModel, to_file='HappyModel.png'; SVG(model_to_dot(happyModel).create(prog='dot', format='svg'))

- ResNet:
  a. a deeper network can represent very complex functions but hard to train because of vanishing gradients: very deep networks often have a gradient signal that goes to zero quickly, thus making gradient descent prohibitively slow.
  b. vanishing / exploding gradients: during gradient descent, as you backprop from the final layer back to the first layer, you are multiplying by the weight matrix on each step, and thus the gradient can decrease exponentially quickly to zero (or grow exponentially quickly and "explode" to take very large values).
  c. In ResNets, a "shortcut" or a "skip connection" allows the model to skip layers, which also allows one of the block to learn an identity function very easily.
  d. Two kinds of ResNet block:
     i. identity block
     ii. convolutional block (Conv2D) used to resize the x to the final addition with main path; For example, to reduce the X dimensions's height and width by a factor of 2, you can use a 1x1 convolution with a stride of 2.