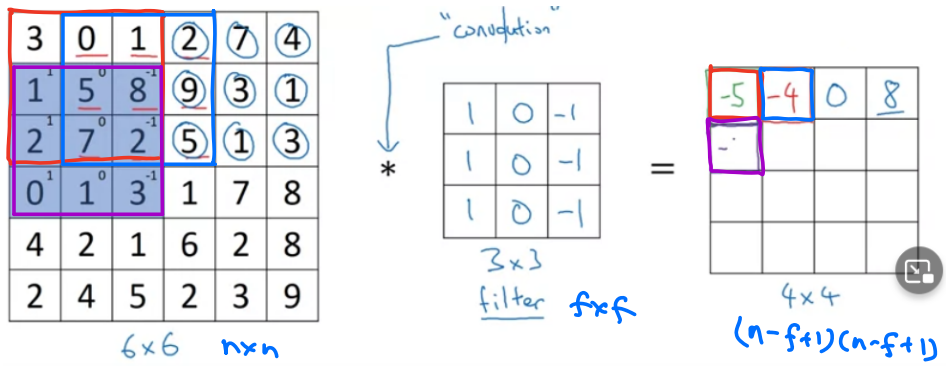
CNN week 1

Computer vision

- Image classification
- Object detection
- Neural Style Transfer

Edge detection Example

Vertical edge detection



Attention: just element-wise multiple! (对应元素相乘!)
not the multiplication of matrix!

Implement: Python: conv_forward
tensorflow: tf.nn.conv2d
keras: Conv2D

Different filter:

1	0	-1
2	0	-2
1	0	-1

Sobel filter

3	0	-3
10	0	-10
3	0	-3

Schoor filter

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

treat all the number as parameter!
And using backprop get these values

Padding

Downsides of above method

- Picture becomes smaller and smaller after each convolution. (shrinking output)
- The edge information can be used a few time (throw away info from edge)

Solution:

Padding

0	0	0	0	0	0	0	
0	3	0	1	2	7	4	0
0	1	5	8	9	3	1	0
0	2	7	2	5	1	3	0
0	0	1	3	1	7	8	0
0	4	2	1	6	2	8	0
0	2	4	5	2	3	9	0
0	0	0	0	0	0	0	0

6x6

Question: how to choose the number of paddle?

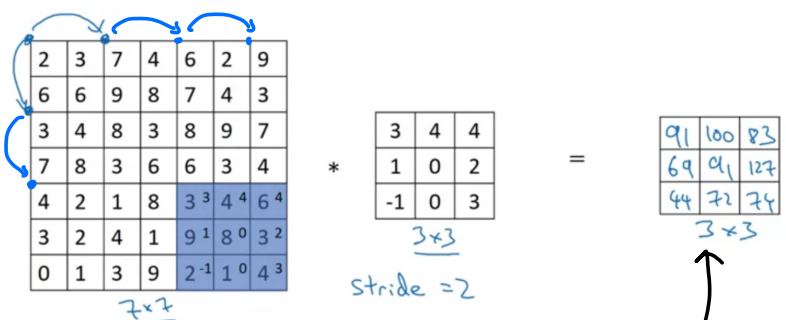
A: Valid convolution: $n \times n * f \times f \rightarrow (n-f+1) \times (n-f+1)$

Same convolution: Pad so that output size is the same as the input size.

$$(n+2p-f+1) \times (n+2p-f+1) = n$$

$$\Rightarrow p = f-1/2 \quad (f \text{ is usually odd})$$

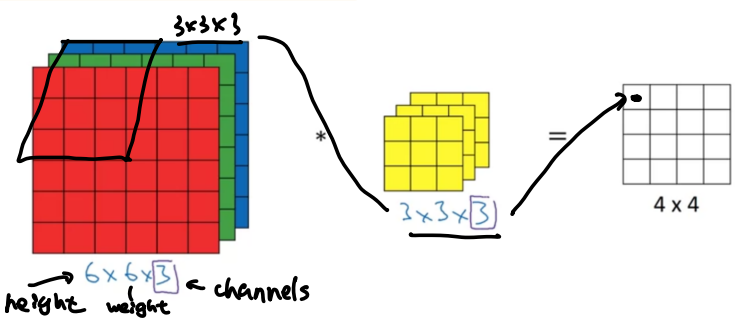
Strided Convolutions



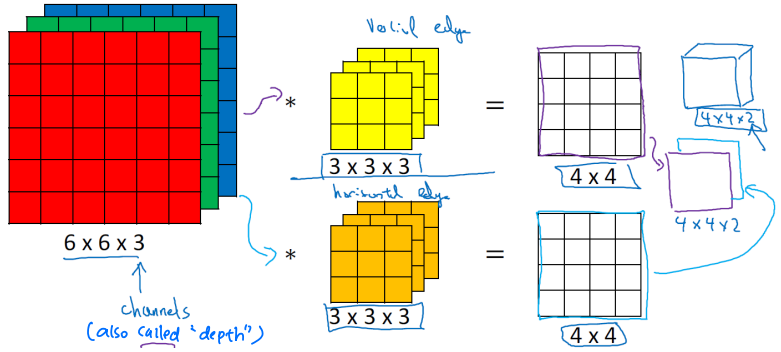
The output is $\left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor$

e.g. In above example: $\frac{7+2\times0-3}{2} + 1 = 3$

Convolutions Over Volume



Multiple filters

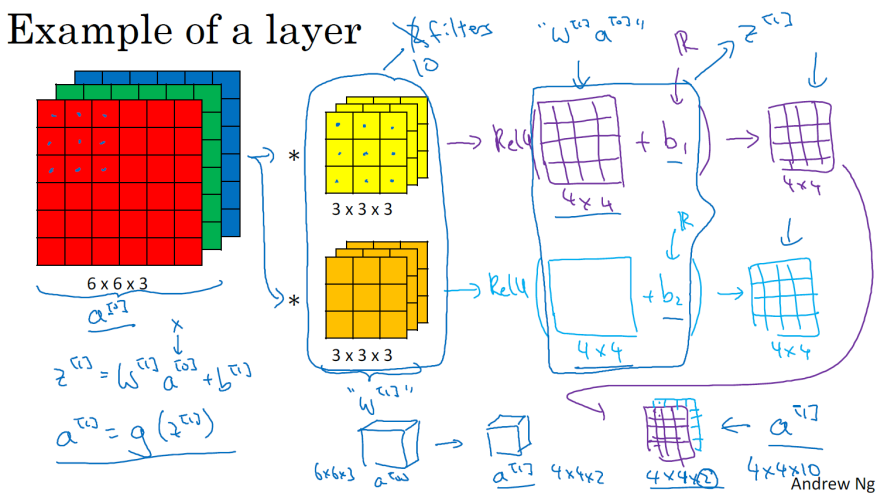


Summary: $n \times n \times n_c \times f \times f \times n_c = (n-f+1) \times (n-f+1)$

e.g. $6 \times 6 \times 3 \times 4 \times 4 \times 2 \quad 4 \times 4$

One layer of a convolutional Network

The comparision between neural network and CNN



Question: If you have 10 filters that are 3x3x3 in one layer of a neural network, how many parameters does that layer have?

Answer: $3 \times 3 \times 3 + 1 = 28$ parameters $\times 10 = 280$

(+1 for b)

Summary of notation:

$f^{[l]}$ = filter size

$p^{[l]}$ = padding

$s^{[l]}$ = stride

$n_c^{[l]}$ = number of filters

Input: $n_H^{[l-1]} \times n_W^{[l-1]} \times n_c^{[l-1]}$

Output: $n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$

where $n_H^{[l]} = \left\lfloor \frac{n_H^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right\rfloor$

$$n_W^{[l]} = \left\lfloor \frac{n_W^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right\rfloor$$

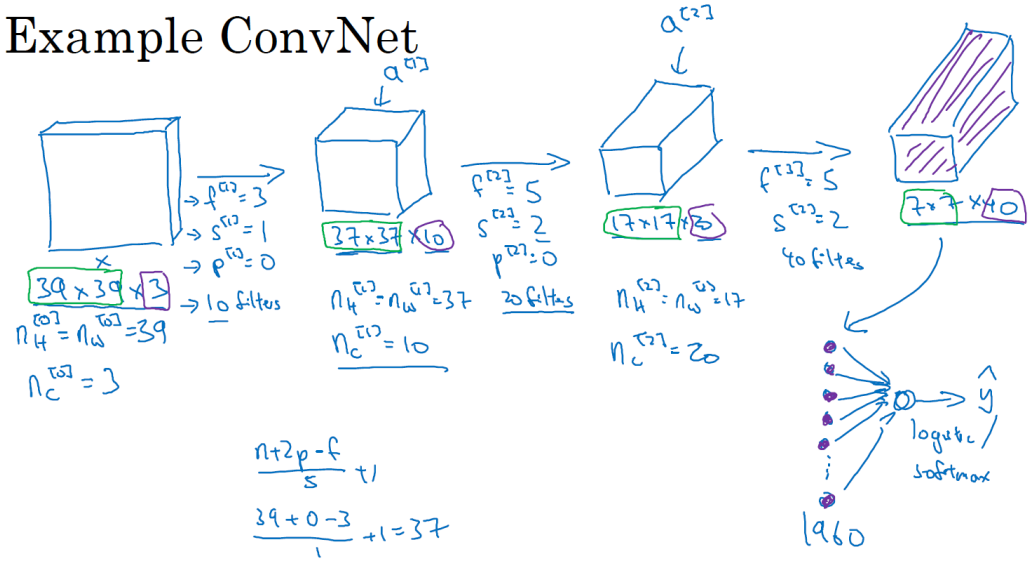
Each filter is: $f^{[l]} \times f^{[l]} \times n_C^{[l-1]}$

Activation: $a^{[l]} \rightarrow n_H^{[l]} \times n_W^{[l]} \times n_C^{[l]}$

Weights: $f^{[l]} \times f^{[l]} \times n_C^{[l-1]} \times n_C^{[l]}$

bias: $n_C^{[l]} - (1, 1, 1, n_C^{[l]})$

A simple convolution network example

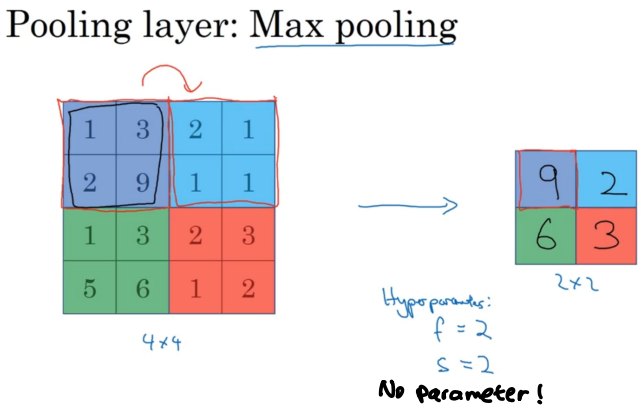


Note that { the size of image ↓
the number of channels ↑

Type of layer in a convolution network:

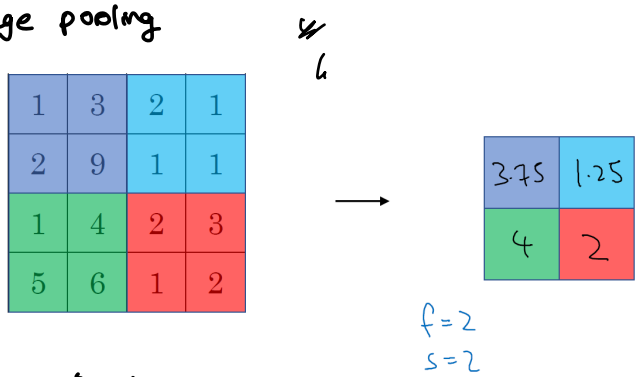
- Convolution
- Pooling
- Fully connected

Pooling layers



Note: the calculation of maximum is independent on each channel.

- Average pooling



Hyperparameters:

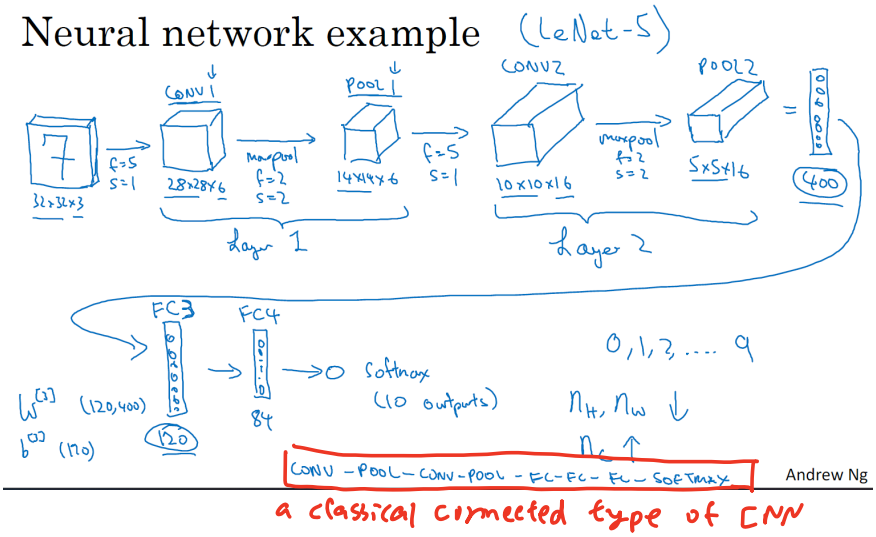
f: filter size

s: stride

Max or average pooling

~~not use padding in common~~

CNN Example



a classical connected type of CNN

Why Convolutions ?

- Parameter Sharing : A feature detector (such as vertical edge detector) that's useful in one part of the image is probably useful in another part of the image

- Sparsity of connection: In each layer, each output value depends only on a small number of inputs

Some question :

1. the benefits of padding.

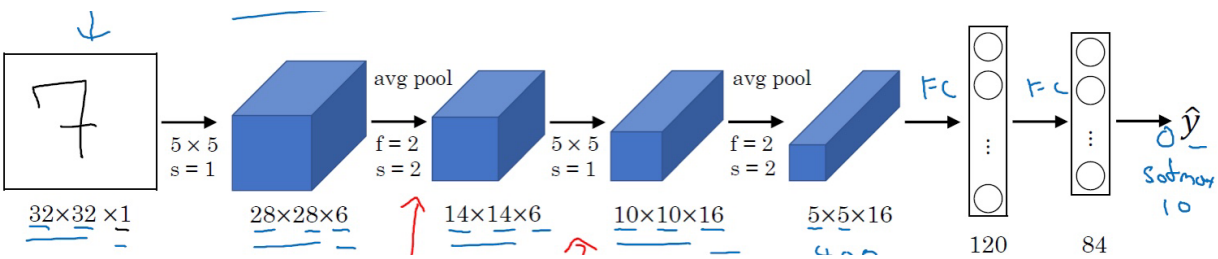
CNN Week 2 Case Study

Outline:

- Classical network:
  - LeNet-5
  - AlexNet
  - VGG
- ResNet
- Inception
- MobileNet

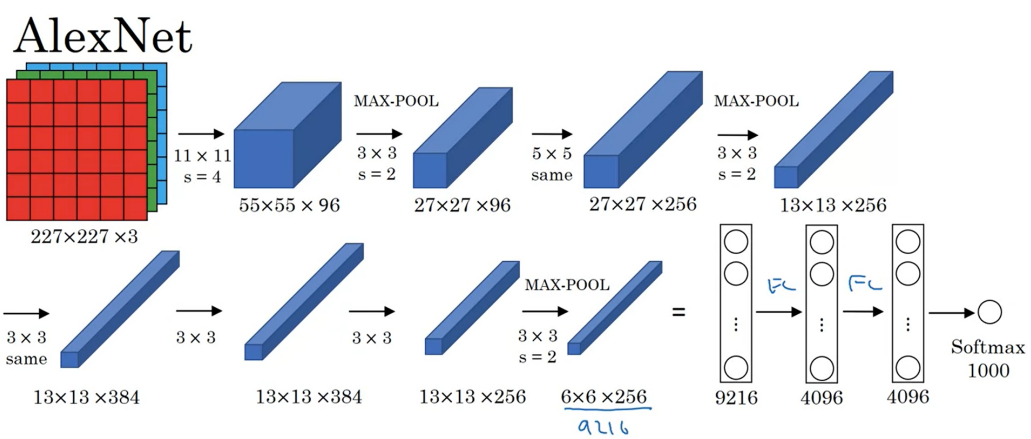
Classical Network

LeNet-5



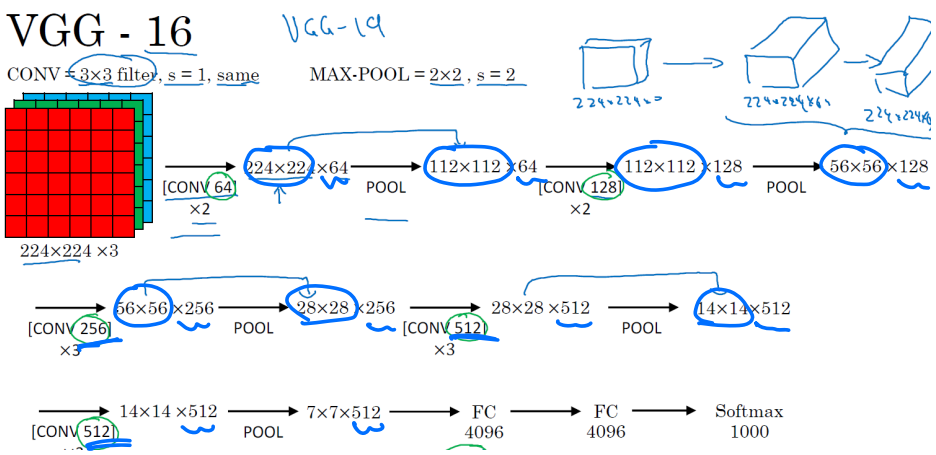
Classical architecture: conv → pool → conv → pool → fc → fc → output

AlexNet



- Similar to LeNet, but much bigger
- ReLU

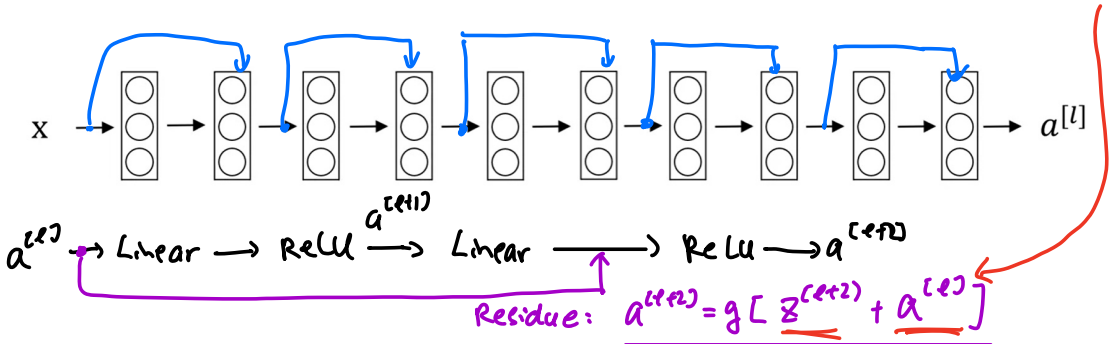
VGG-16

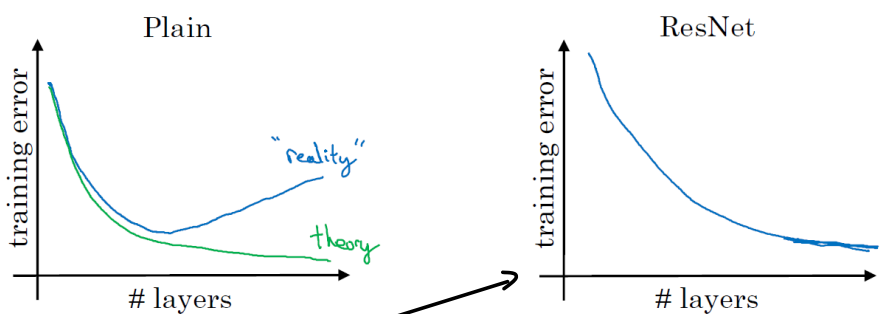


- Uniformity
- drawback: too many parameters

ResNets (Residual Networks)

Question:
① If different dimension after mapping ?





Why ResNets Work? △

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

$= g(w^{[l+2]} a^{[l+1]} + b^{[l+2]} + W_s a^{[l]}) = g(a^{[l]})$

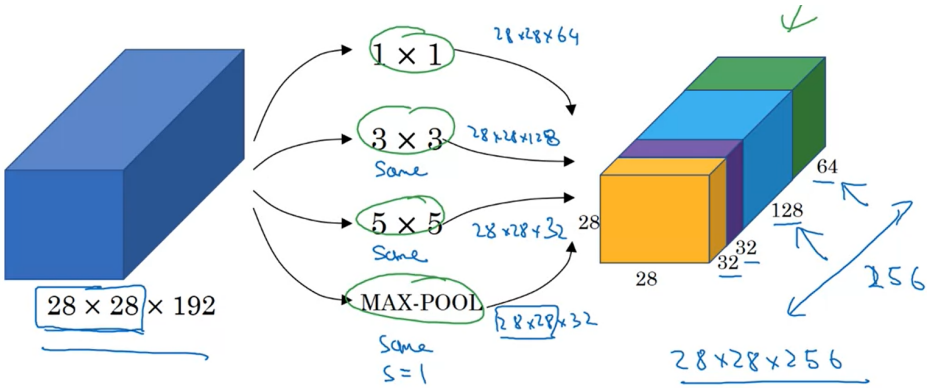
If $(w^{[l+2]}, b^{[l+2]} = 0)$ $= a^{[l]}$

One by One convolution (network in network)

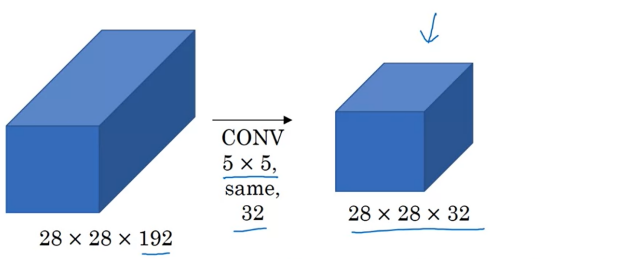
- Shrink the number of channels

Inception Network

Intuition: Use what sizes of conv layer or pooling layer? Do them together

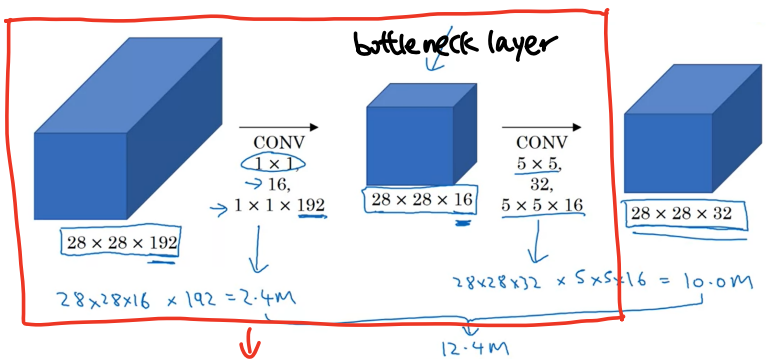


The computational cost



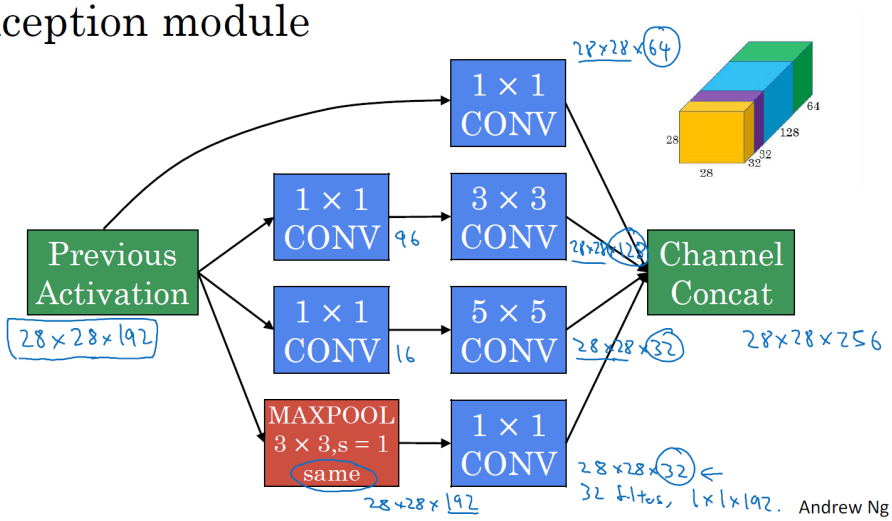
32 filters. filter one 5×5×192.

28×28×32 × 5×5×192 = 120M.

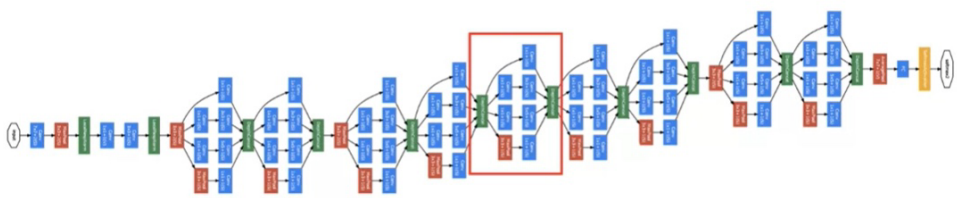


Does it influence the performance?

Inception module



Inception Network



Mobile Net

Motivation:

- Low computational cost at deployment
- Useful for mobile and embeded vision application
- Key idea: Normal vs depthwise-separable convolutions

Normal Convolution.

Computation cost = # filter params × # filter positions × # of filters

E.g.

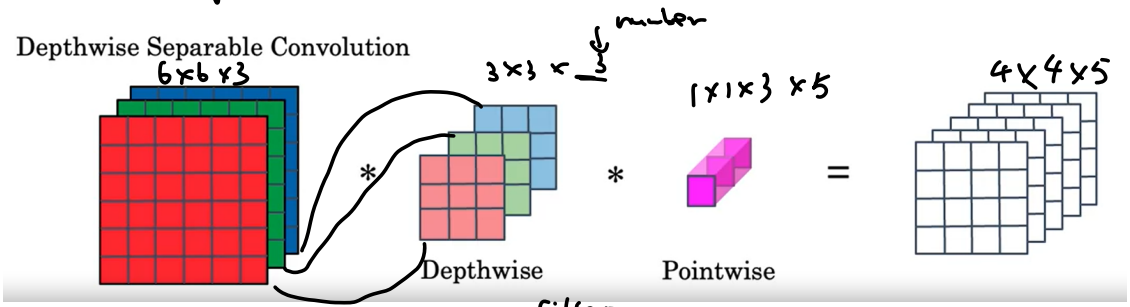
$6 \times 6 \times 3 \quad * \quad 3 \times 3 \times 3 \quad (n_c' \text{ filter}) \quad = \quad 4 \times 4 \times n_c'$

stride = 1
no padding

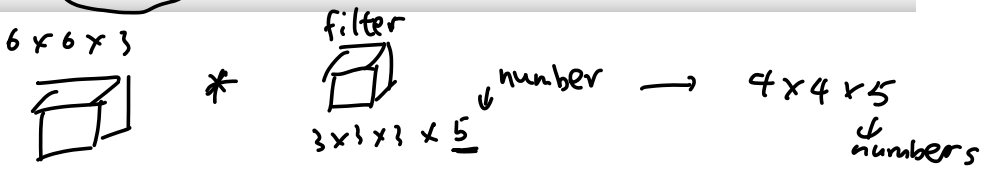
Assume $n_c' = 5$

Computational cost = $3 \times 3 \times 3 \times 4 \times 4 \times 5 = 2160$

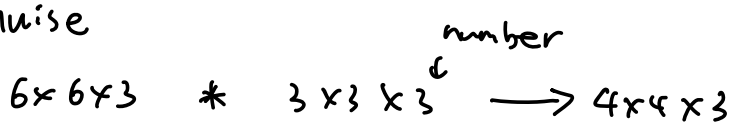
Depthwise Separable Convolution



- Normal



Depthwise



Depthwise Computation cost = $3\times3\times3\times4\times4 = 432$

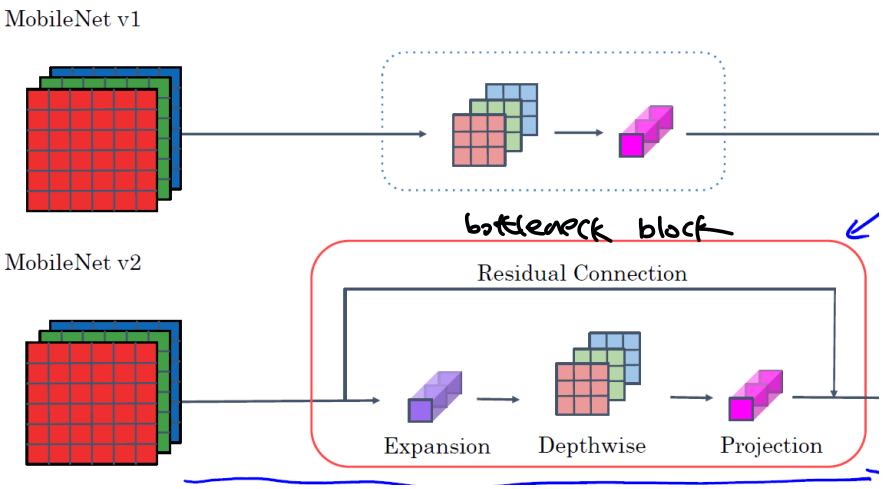
Pointwise computation cost = $1\times1\times3\times4\times4\times5 = 240$

Total : 672

Normal : 2160

} in general case : $\frac{1}{n'} + \frac{1}{f^2}$

MobileNet Architecture



- Computationally efficient!

Efficient Net

- Scale up or down the neural network based on the resources of a device

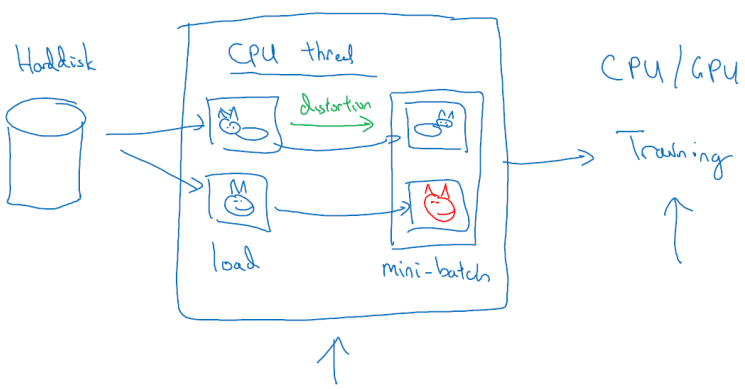
Practical Advice for Using ConvNets

Transfer Learning

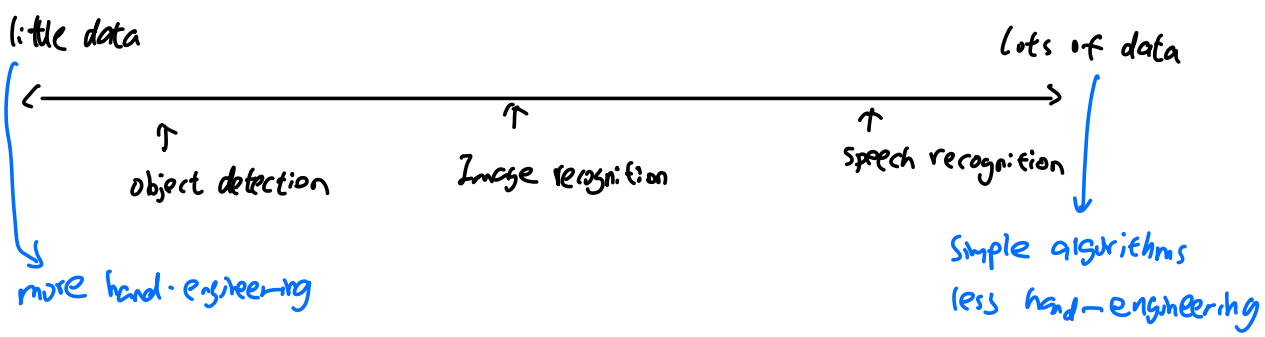
从网上下载之后 initialize 必须 hyperparameter. for the initialization of your own network.

下来的 + weight

- Data Augmentation
  - (CV) Mirror △
    Random Cropping △
    Rotation
    ⋮
  - Color shifting
    (method: PCA color augmentation)



- State of CV



Two sources of knowledge

- Labeled data
- Hand engineered features/network architecture

△ Tips for doing well on benchmarks / winning competitions

Ensembling — keep each network available anytime, cost a lot of memory

- Train several networks independently and average their outputs

Multi-crop at test time

- Run classifier on multiple versions of test image and average results

- Use architectures of networks published in the literature
- Use open source implementations if possible
- Use pretrained models and fine-tune on your dataset

Supplement

① Transfer learning

Scenario 1 – Size of the Data set is small while the Data similarity is very high – In this case, since the data similarity is very high, we do not need to retrain the model. All we need to do is to customize and modify the output layers according to our problem statement. We use the pretrained model as a feature extractor. Suppose we decide to use models trained on Imagenet to identify if the new set of images have cats or dogs. Here the images we need to identify would be similar to imagenet, however we just need two categories as my output – cats or dogs. In this case all we do is just modify the dense layers and the final softmax layer to output 2 categories instead of a 1000.

Scenario 2 – Size of the data is small as well as data similarity is very low – In this case we can freeze the initial (let's say k) layers of the pretrained model and train just the remaining(n-k) layers again. The top layers would then be customized to the new data set. Since the new data set has low similarity it is significant to retrain and customize the higher layers according to the new dataset. The small size of the data set is compensated by the fact that the initial layers are kept pretrained(which have been trained on a large dataset previously) and the weights for those layers are frozen.

Scenario 3 – Size of the data set is large however the Data similarity is very low – In this case, since we have a large dataset, our neural network training would be effective. However, since the data we have is very different as compared to the data used for training our pretrained models. The predictions made using pretrained models would not be effective. Hence, its best to train the neural network from scratch according to your data.

Scenario 4 – Size of the data is large as well as there is high data similarity – This is the ideal situation. In this case the pretrained model should be most effective. The best way to use the model is to retain the architecture of the model and the initial weights of the model. Then we can retrain this

② transfer learning 实现过程

① 划分训练集和验证集

image_dataset_from_directory( )

② Use prefetch( ) to prevent memory bottlenecks when reading from disk

③ data augment

```
data_augmentation = tf.keras.Sequential(
                    [layers.RandomFlip('horizontal_and_vertical'),
                     layers.RandomRotation(0.2),
                     ...
                    ])
```

or

```
def data_augmenter():
    '''
    Create a Sequential model composed of 2 layers
    Returns:
        tf.keras.Sequential
    '''
    ### START CODE HERE
    data_augmentation = tf.keras.Sequential()
    data_augmentation.add(RandomFlip('horizontal'))
    data_augmentation.add(RandomRotation(0.2))
    ### END CODE HERE

    return data_augmentation
```

④ Use a model for transfer learning

⑤ Layer freezing

1. Delete the top layer (the classification layer)
   - Set `include_top` in `base_model` as False
2. Add a new classifier layer
   - Train only one layer by freezing the rest of the network
   - As mentioned before, a single neuron is enough to solve a binary classification problem.
3. Freeze the base model and train the newly-created classifier layer
   - Set `base model.trainable=False` to avoid changing the weights and train *only* the new layer
   - Set training in `base_model` to False to avoid keeping track of statistics in the batch norm layer

去掉 base model 的分类器，冻结 base model，输入自己的训练集（处理过的），然后再加上 global avg pooling，Dropout，predict_layer，output

新 model = tf.keras.Model(input, outputs)

⑥ backprop 訓練新的 model.

```
base_learning_rate = 0.001
model2.compile(optimizer=tf.keras.optimizers.Adam(lr=base_learning_rate),
               loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
               metrics=['accuracy'])
```

```
initial_epochs = 5
history = model2.fit(train_dataset, validation_data=validation_dataset, epochs=initial_epochs)
```

⑦ 畫圖看 training and validation accuracy

⑧ Fine-tuning model

You could try fine-tuning the model by re-running the optimizer in the last layers to improve accuracy. When you use a smaller learning rate, you take smaller steps to adapt it a little more closely to the new data. In transfer learning, the way you achieve this is by unfreezing the layers at the end of the network, and then re-training your model on the final layers with a very low learning rate. Adapting your learning rate to go over these layers in smaller steps can yield more fine details - and higher accuracy.

The intuition for what's happening: when the network is in its earlier stages, it trains on low-level features, like edges. In the later layers, more complex, high-level features like wispy hair or pointy ears begin to emerge. For transfer learning, the low-level features can be kept the same, as they have common features for most images. When you add new data, you generally want the high-level features to adapt to it, which is rather like letting the network learn to detect features more related to your data, such as soft fur or big teeth.

To achieve this, just unfreeze the final layers and re-run the optimizer with a smaller learning rate, while keeping all the other layers frozen.

Where the final layers actually begin is a bit arbitrary, so feel free to play around with this number a bit. The important takeaway is that the later layers are the part of your network that contain the fine details (pointy ears, hairy tails) that are more specific to your problem.

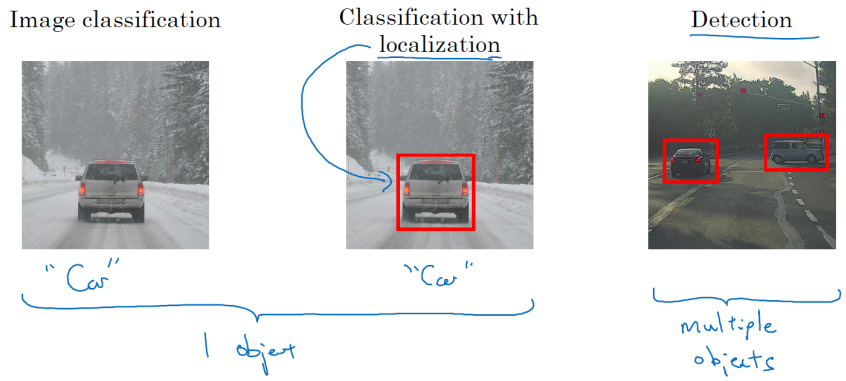
First, unfreeze the base model by setting `base_model.trainable=True`, set a layer to fine-tune from, then re-freeze all the layers before it. Run it again for another few epochs, and see if your accuracy improved!

CNN Week 3 Detection Algorithms

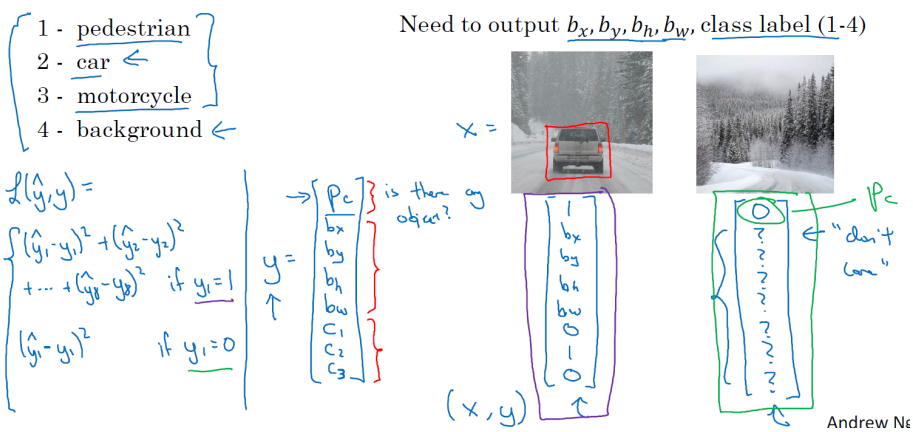
Object Localization

What objects are in this image and where in the image are those objects located?

Some definitions

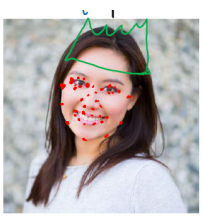


Defining the target label y



Landmark detection

The label of landmark should be same among the all training set.

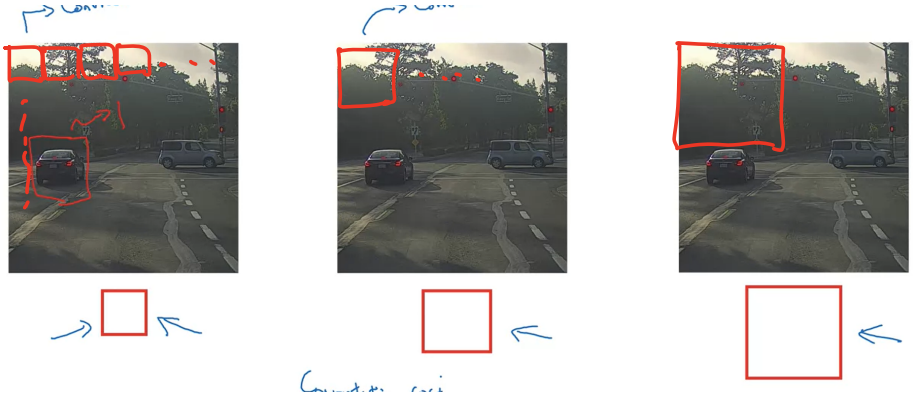


$l_{1x}, l_{1y}, l_{2x}, l_{2y}, l_{3x}, l_{3y}, l_{4x}, l_{4y}, \dots, l_{64}, l_{64y}$ → X, y

→ Conv Net → face ?, l_{1x}, l_{1y}, l_{2x}, l_{2y}, ⋮, l_{64x}, l_{64y}

Object detection

Sliding windows detection



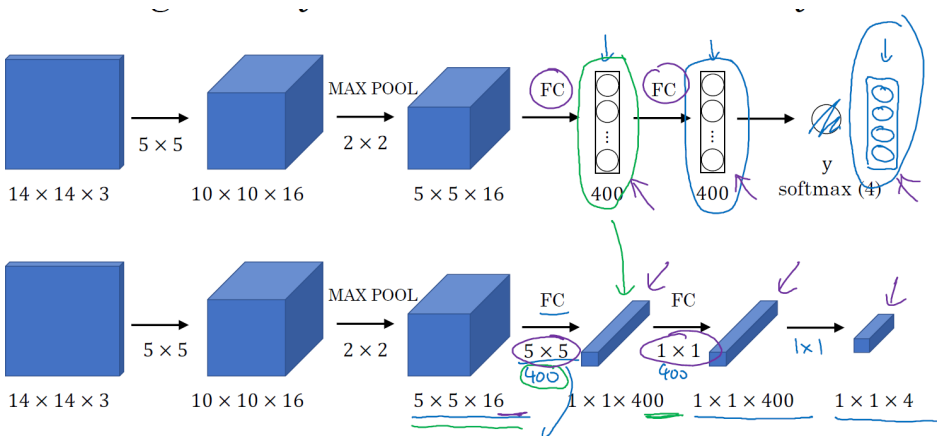
Training set:

x	y
	1
	1
	1
	0
	0

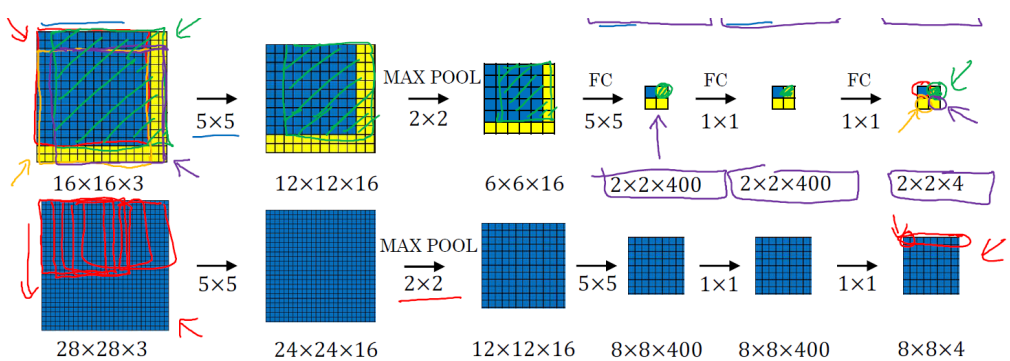
Drawback: high computational cost, unacceptable for CNNs.

Convolutional implementation of sliding windows

Turning FC layer into convolutional layers

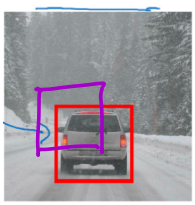


Convolution implementation of sliding windows



Implement all the sliding windows at the same time.

Problem: 无法精确输出边界框. (the purple frame is more common)

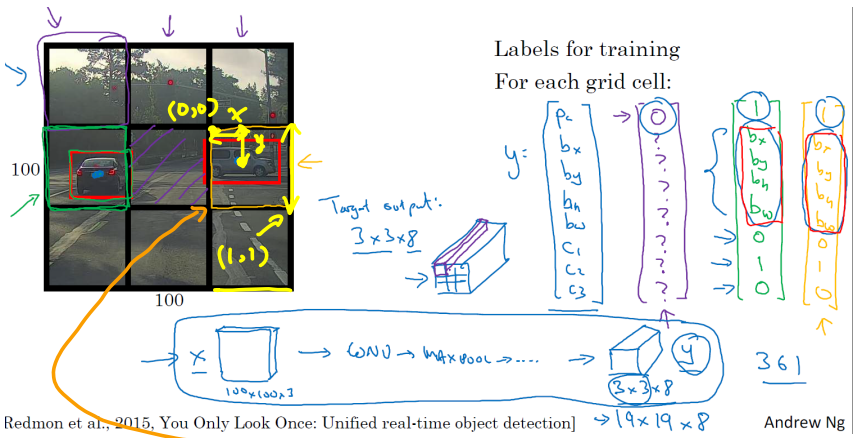


or the perfect frame is not a square.

how to solve this problem?

Building box Predictions

YOLO Algorithm



Advantage:

① 不会受限于滑动窗口的步长

② very fast

Question:

how to decide b_x, b_y, b_h, b_w

$$\begin{bmatrix} 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix} \rightarrow \begin{bmatrix} 1 \\ 0.4 \\ 0.3 \\ 0.9 \\ 0.5 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

b_x, b_y less than 1

b_h, b_w ← fraction of red line

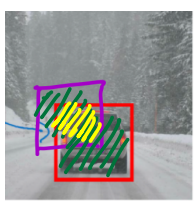
could be larger than 1

Intersection over Union

Evaluate the detection algorithm

Add another components in detection algorithm

$$IoU = \frac{\text{size of Yellow}}{\text{size of green}}$$



"correct" if $IoU \geq 0.5$

↑ just a convention

Non-max suppression (解决一个物体出现多个框问题)

-Just output the max probability one.

- Step:

  Each output prediction is $\begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \end{bmatrix}$

  Discard all boxes with $p_c \leq 0.6$

  While there are any remaining boxes:

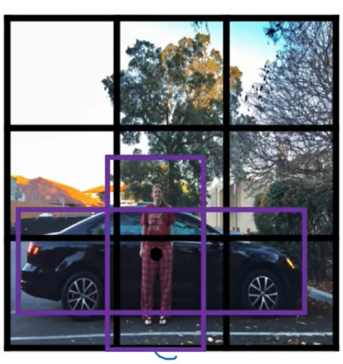
  - Pick the box with the largest p_c. Output that as a prediction
  - Discard any remaining box with IoU ≥ 0.5 with the box output in the previous step.

Anchor box (解决两个features中心重叠和特殊物体问题)

Previously: Each object in training image is assigned to grid cell that contains that object's midpoint

With two anchor boxes: Each object in training image is assigned to grid cell that contains object's midpoint and anchor box for the grid cell with highest IoU.

Example:



Anchor box 1: Anchor box 2:

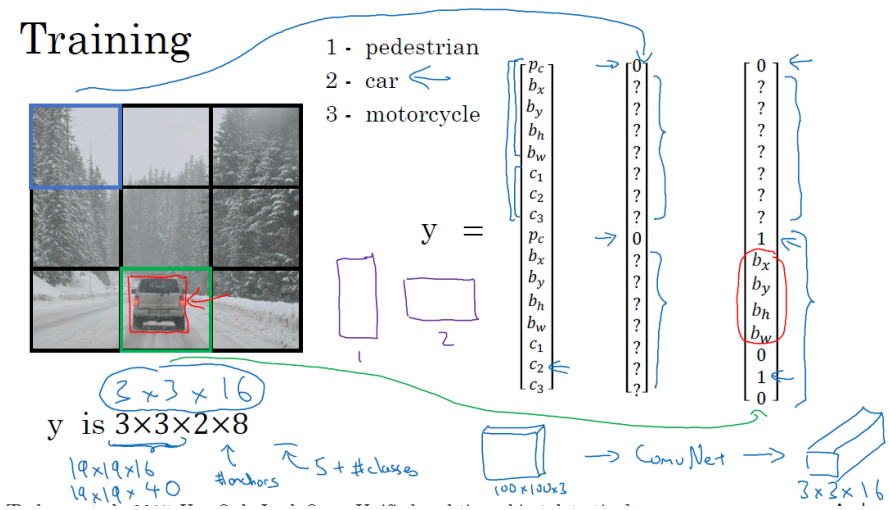
$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \\ p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 1 \\ 0 \\ 0 \\ 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 1 \\ ? \\ ? \\ ? \\ ? \\ ? \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ ? \\ ? \end{bmatrix}$$

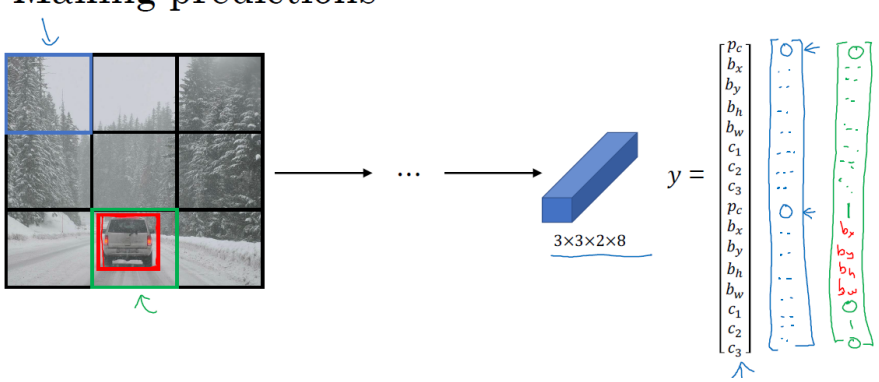
only car?

why anchor box ? - 处理两个对象出现在同一格子的问题
- 更能专注特定形状 (too tall/wide ...)

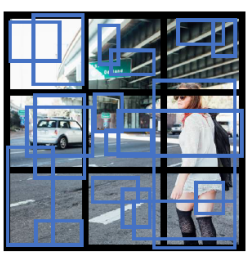
how to choose anchor box ? — kmeans

YOLO Algorithm (update) ☆☆





Outputting the non-max supressed outputs



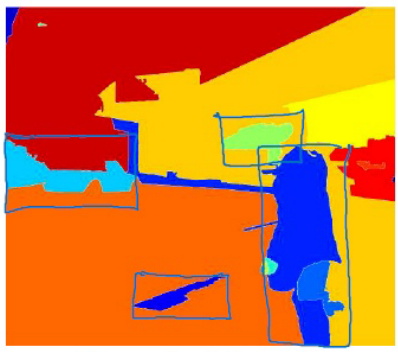
- For each grid call, get 2 predicted bounding boxes.
- Get rid of low probability predictions.
- For each class (pedestrian, car, motorcycle) use non-max suppression to generate final predictions.

Region proposals (R-CNN)

R-CNN: Propose regions. Classify proposed regions one at a time. Output label + bounding box.

Faster R-CNN: Propose regions. Use convolution implementation of sliding windows to classify all the proposed regions

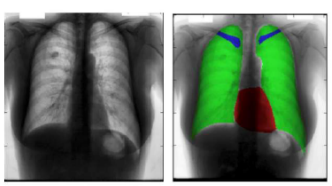
Faster R-CNN: Use convolutional network to propose regions



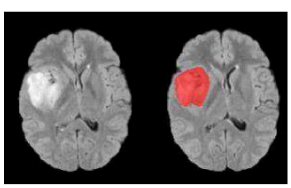
Segmentation algorithm

Semantic Segmentation with U-net (语义分割) ☆☆

Motivation for U-Net

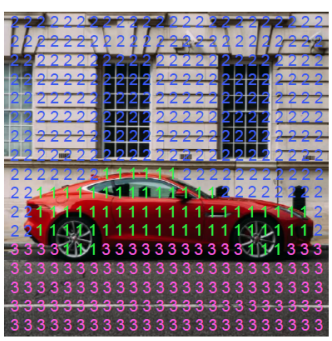


Chest X-Ray



Brain MRI

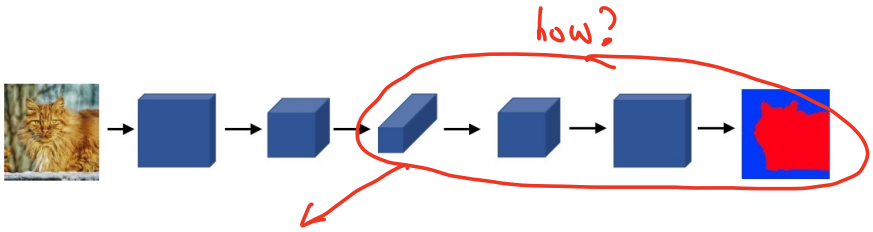
Per-pixel class labels



1. Car
2. Building
3. Road

Segmentation Map

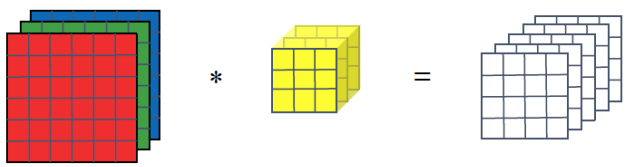
Deep Learning for Semantic Segmentation



Transpose Convolutions

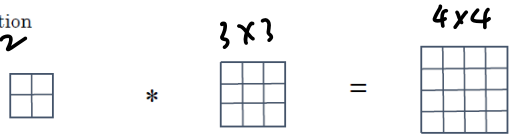
Transpose Convolution

Normal Convolution

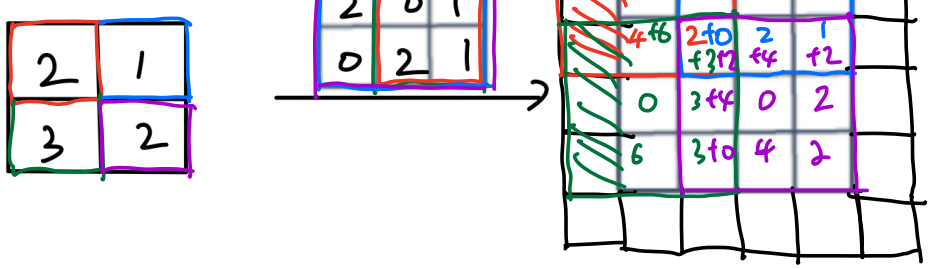


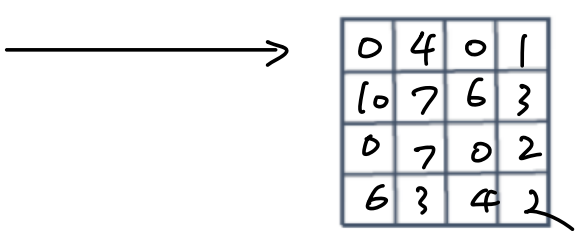
Transpose Convolution

2x2 * 3x3 = 4x4



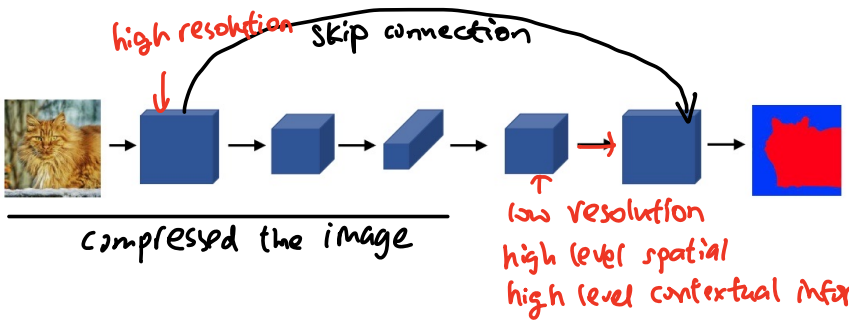
In detail.





U-Net Architecture Intuition

Deep Learning for Semantic Segmentation



U-Net Architecture

$$\left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor$$

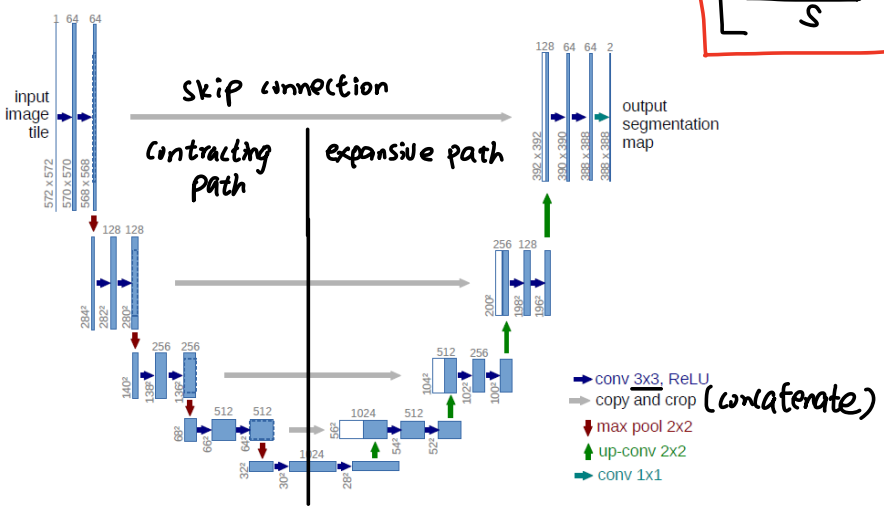


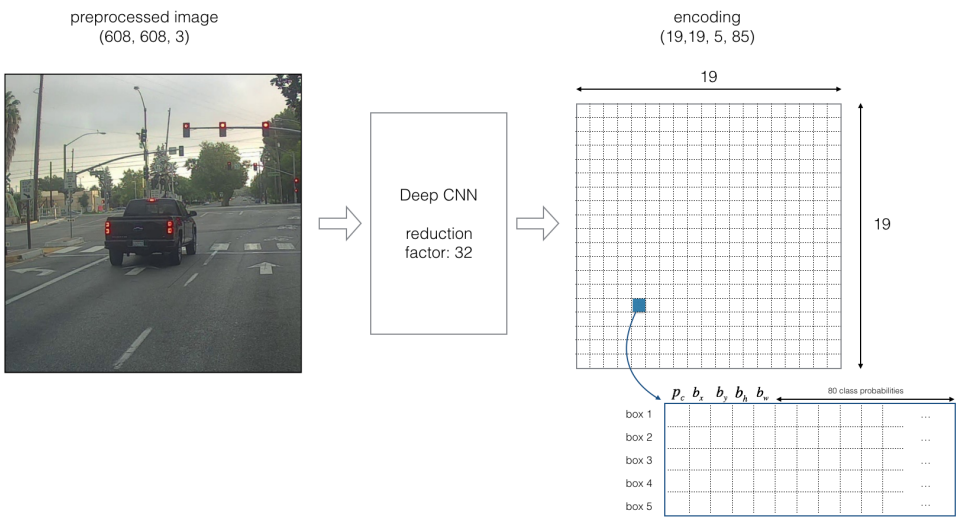
Fig. 1. U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations.

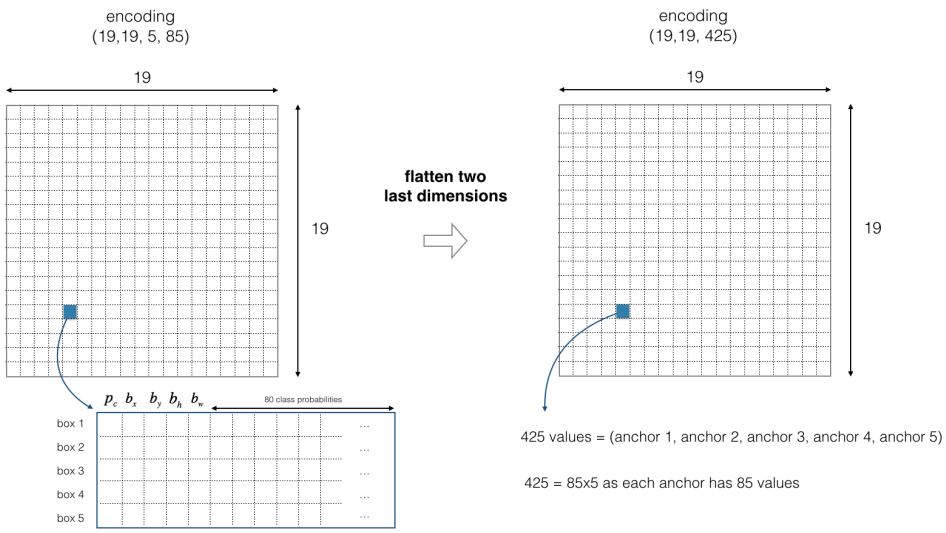
Implementation:

① conv1 = conv2D(filters = 64,
kernel_size = 3,
padding = 'same', (valid ?)
activation = 'relu',
kernel_initialization = 'he_normal')

Summary for YOLO ☆

- Input image (608, 608, 3)
- The input image goes through a CNN, resulting in a (19, 19, 5, 85) dimensional output.
- After flattening the last two dimensions, the output is a volume of shape (19, 19, 425)
  - Each cell in a 19 x 19 grid over the input image gives 425 numbers.
  - 425 = 5 x 85 because each cell contains predictions for 5 boxes, corresponding to 5 anchor boxes.
  - 85 = 80 + 5 where 5 is because $(P_c, b_x, b_y, b_h, b_w)$ has 5 numbers, and 80 is the number of classes we'd like to detect
- You then select only few boxes based on:
  - Score-thresholding: throw away boxes that have detected a class with a score less than the threshold
  - Non-max suppression: Compute the Intersection over Union and avoid selecting overlapping boxes.
- This gives you YOLO's final output.





425 values = (anchor 1, anchor 2, anchor 3, anchor 4, anchor 5)

425 = 85x5 as each anchor has 85 values

CNN Week 4 Face Recognition

One-Shot learning

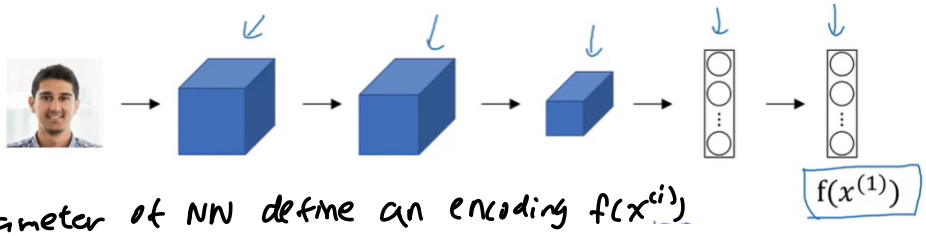
- Learning from one example to recognize the person again
- Learning a `similarity' function

  d(img1, img2) = degree of different between image

  If $d(img1, img2) \leq \tau$
  $> \tau$

- Question: How to learn `similarity' function ?

Siamese Network



- Parameter of NN define an encoding $f(x^{(i)})$
- Learn parameters so that:

  If $x^{(i)}, x^{(j)}$ are same person, $\|f(x^i) - f(x^j)\|$ is small

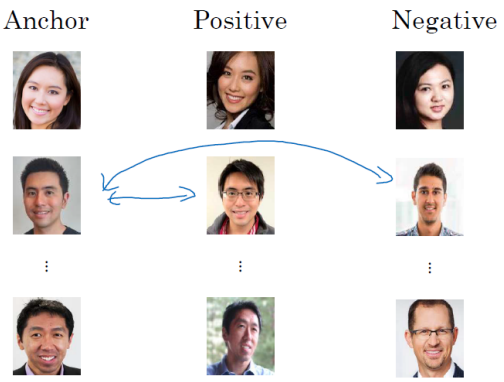
  different $\|f(x^i) - f(x^j)\|$ is large

Triplet loss

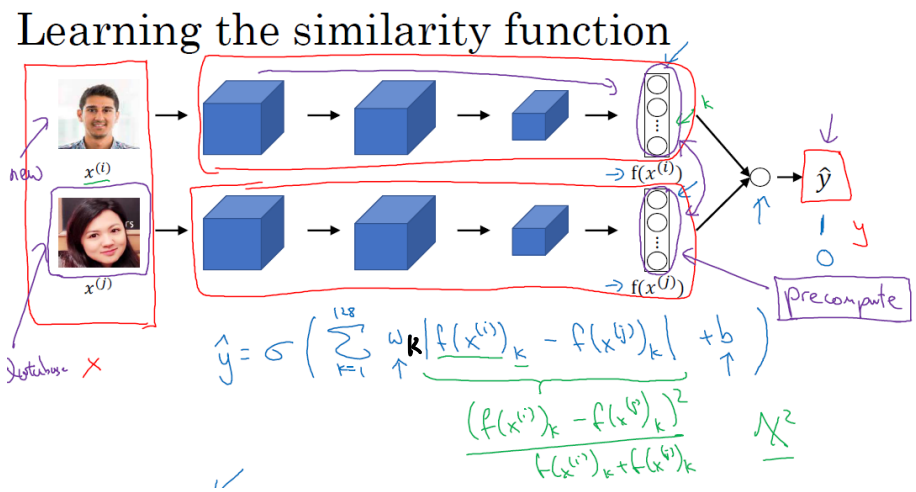
Given 3 image A, P, N:

$$l(A, P, N) = \max\left(\|f(A) - f(P)\|^2 - \|f(A) - f(N)\|^2 + \alpha, 0\right)$$

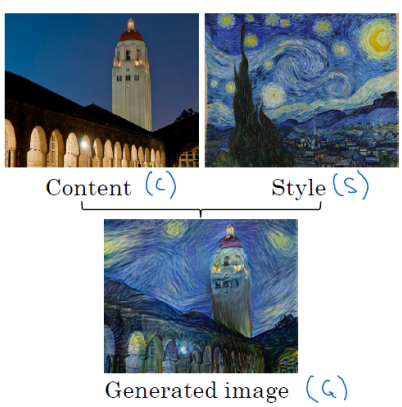
$$J = \sum_{i=1}^{m} l(A^{(i)}, P^{(i)}, N^{(i)})$$

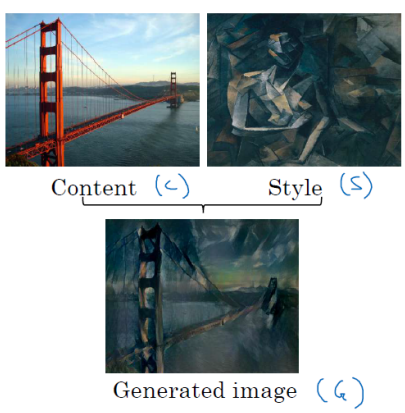


Face verification and binary classification



Neural Style Transfer





Cost Function

$$J(G) = \alpha J_{content}(C, G) + \beta J_{style}(S, G)$$

1. Initiate G randomly

   G: 100x100x3

2. Use gradient descent to minimize J(G)

Content Cost Function

Content cost function

$$J(G) = \alpha J_{content}(C, G) + \beta J_{style}(S, G)$$

- Say you use hidden layer l to compute content cost.
- Use pre-trained ConvNet. (E.g., VGG network)
- Let $a^{[l](C)}$ and $a^{[l](G)}$ be the activation of layer l on the images
- If $a^{[l](C)}$ and $a^{[l](G)}$ are similar, both images have similar content

$$J_{content}(C, G) = \frac{1}{2} \| a^{[l](C)} - a^{[l](G)} \|^2$$

- Style cost function