CONVOLUTIONAL NEURAL NETWORKS

By: Mohamed Aziz Tousli

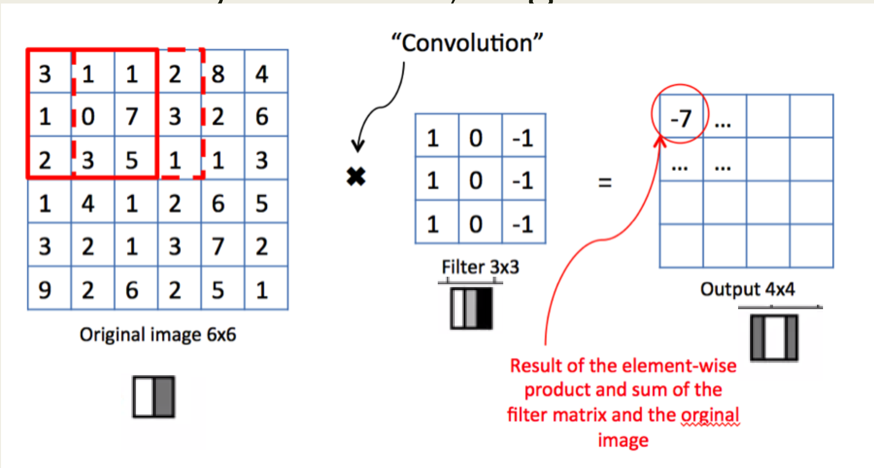
Convolution

- Computer vision problems: Image classification, Object detection, Neural style transfer, Edge detection…

Vertical edge detection: $\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}, \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}, \begin{bmatrix} 3 & 0 & -3 \\ 10 & 0 & -10 \\ 3 & 0 & -3 \end{bmatrix}$

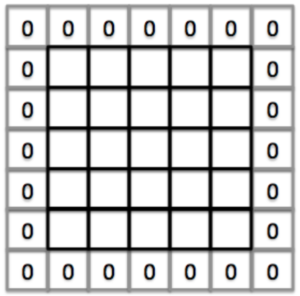
Horizontal edge detection: $\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$

- Detect edges in general: $\begin{bmatrix} w_1 & w_2 & w_3 \\ w_4 & w_5 & w_6 \\ w_7 & w_8 & w_9 \end{bmatrix}$ → Parameters to learn

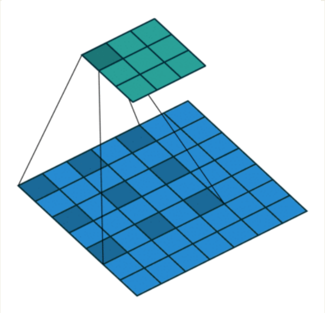


→ Shrinking output + Throwing away information from edge → <u>Solution</u>: **Padding**

- **Valid convolution**: No padding (p=0)
- **Same convolution**: Padding so that output size is same as input size $p = \frac{f-1}{2}$



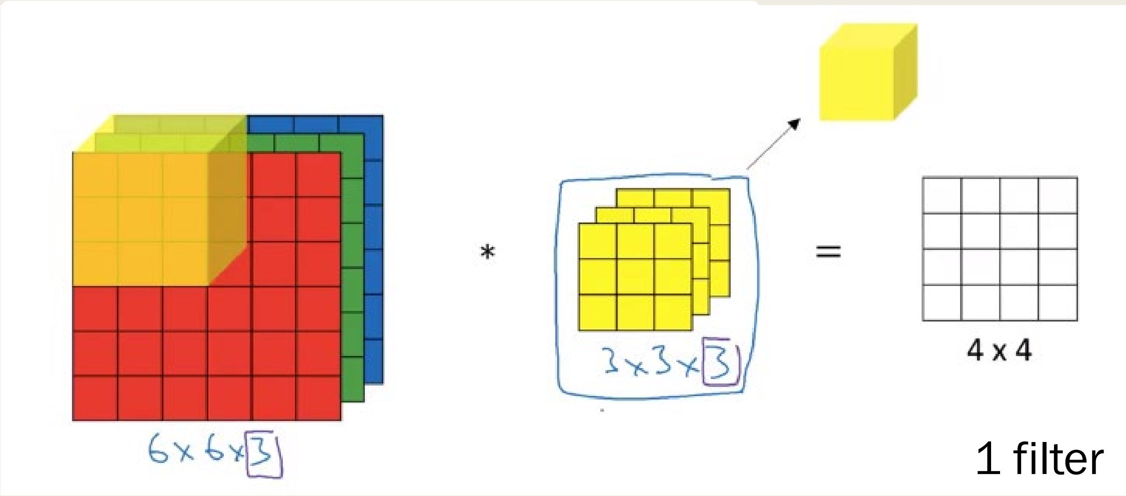
- **Strided** convolution: Convolution with <u>pace</u>
- $(\boldsymbol{n}, \boldsymbol{n}) * (\boldsymbol{f}, \boldsymbol{f}) \rightarrow \left(\left\lfloor \frac{\boldsymbol{n}+2\boldsymbol{p}-\boldsymbol{f}}{\boldsymbol{s}} + \mathbf{1} \right\rfloor, \left\lfloor \frac{\boldsymbol{n}+2\boldsymbol{p}-\boldsymbol{f}}{\boldsymbol{s}} + \mathbf{1} \right\rfloor\right)$



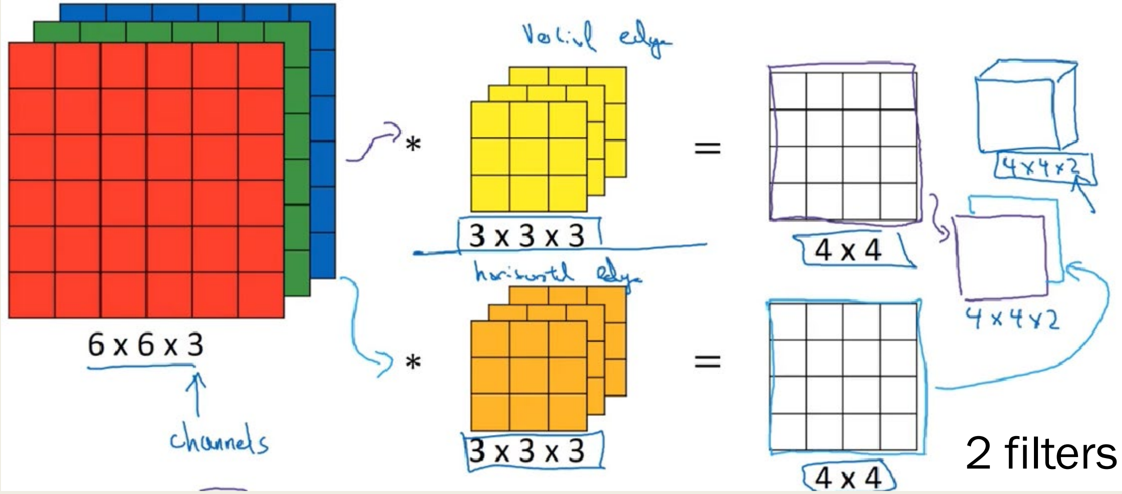
PS: <u>Cross-correlation</u> : Convolution with rotating the filter

PS: Convolution can be done in 1D and 3D the same way as in 2D

Convolution Layer



1 filter



2 filters

$(\boldsymbol{n},\boldsymbol{n},\boldsymbol{n_c}) * (\boldsymbol{f},\boldsymbol{f},\boldsymbol{n_c}) \to (\boldsymbol{n}-\boldsymbol{f}+\mathbf{1},\boldsymbol{n}-\boldsymbol{f}+\mathbf{1},\boldsymbol{n_c'}); n_c = number\ of\ \boldsymbol{channels}; = number\ of\ \boldsymbol{filters}$

- $\boldsymbol{a}^{[\mathbf{1}]} = \boldsymbol{g}\left(\boldsymbol{w}^{[\mathbf{1}]}.\boldsymbol{a}^{[\mathbf{0}]} + \boldsymbol{b}^{[\mathbf{1}]}\right); w = filters, a^{[0]} = input, b = bias$
- Filter size: $f^{[l]}$, Padding: $p^{[l]}$, Stride: $s^{[l]}$, Number of filters: $n_c^{[l]}$
- **Filter:** $(f^{[l]}, f^{[l]}, n_c^{[l-1]})$, **Activation:** $\left(n_H^{[l]}, n_W^{[l]}, n_c^{[l]}\right)$, **Weights:** $(f^{[l]}, f^{[l]}, n_c^{[l-1]}, n_c^{[l]})$, **Bias:** $(1,1,1,n_c^{[l]})$
- **Input:** $\left(n_H^{[l-1]}, n_W^{[l-1]}, n_c^{[l-1]}\right)$, **Output:** $\left(n_H^{[l]}, n_W^{[l]}, n_c^{[l]}\right)$, $n_{H/W}^{[l]} = [\frac{n_{H/W}^{[l-]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1]$

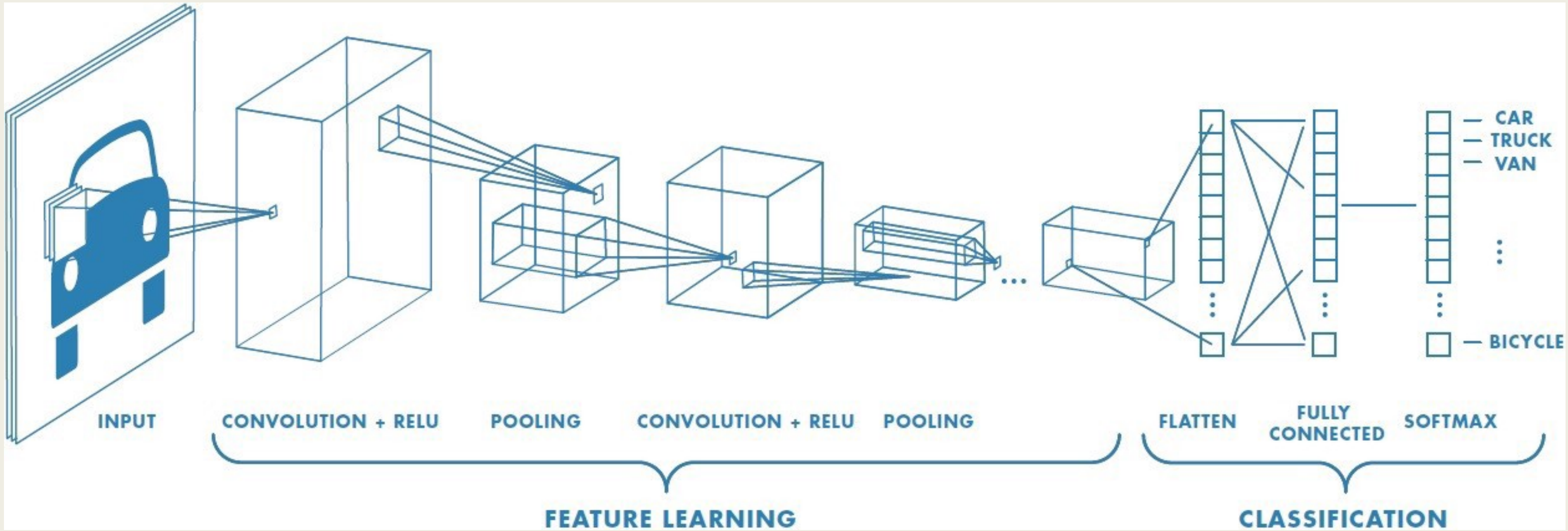
Convolutional Network

- Types of a layer in a CN: Convolution (Conv), Pooling (Pool), Fully connected (FC)
- **Max pooling:** (Hyper parameters: f (filter size), s (stride))
- Input size: $(n_H, n_W, n_c) \rightarrow \left(\left[\frac{n_H - f}{s} + 1\right], \left[\frac{n_H - f}{s} + 1\right], n_c\right)$

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

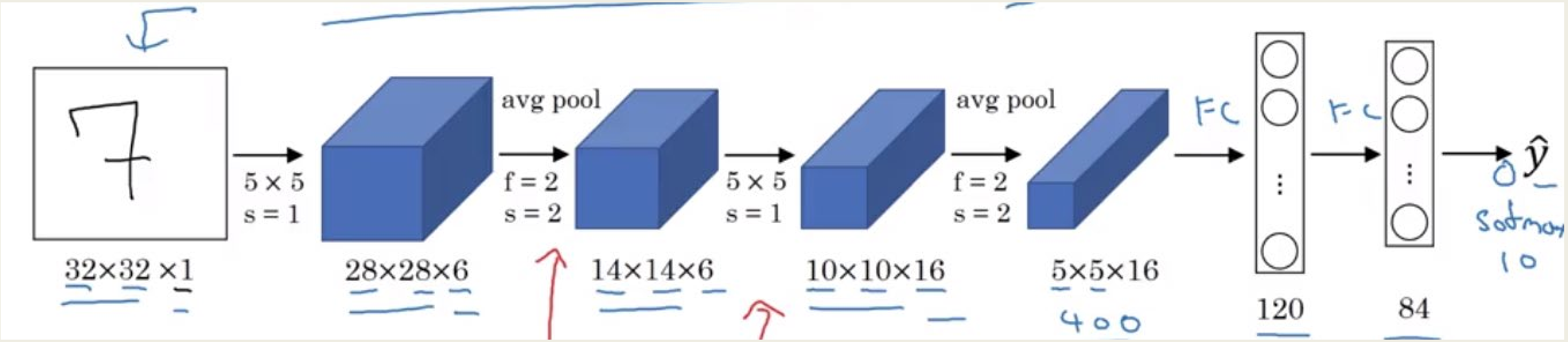
2	4
7	9

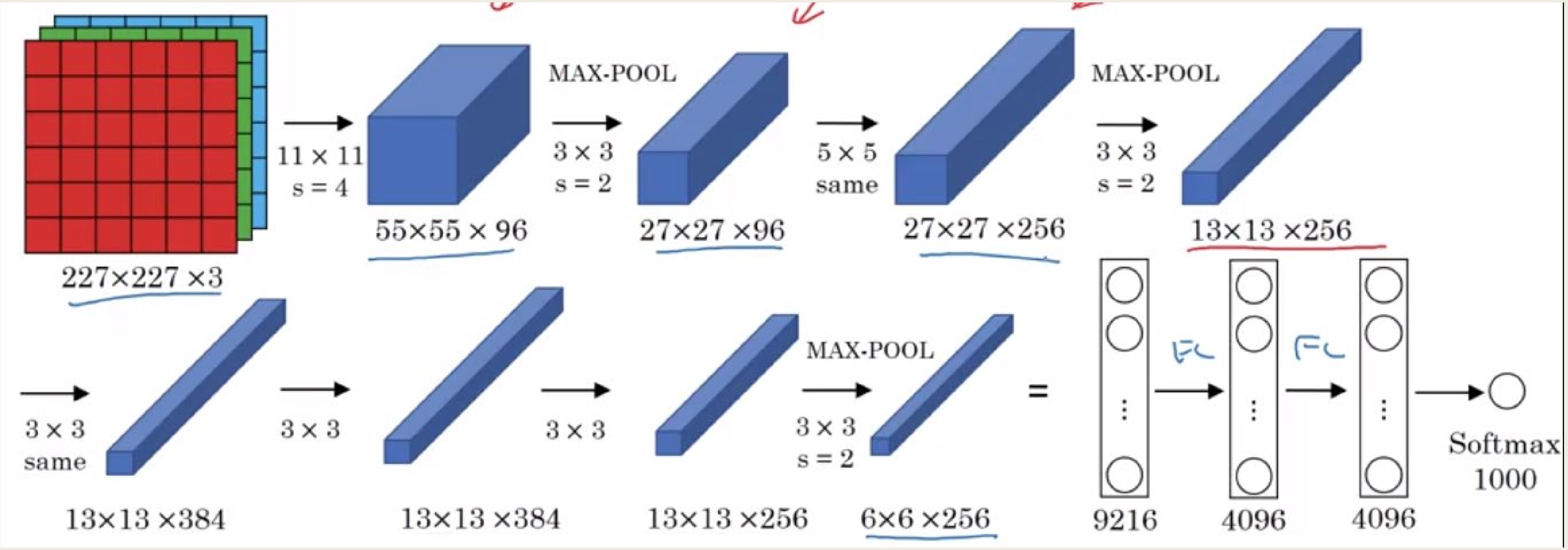
PS: There is also **average** pooling + There is no padding in pooling + There are no parameters to learn



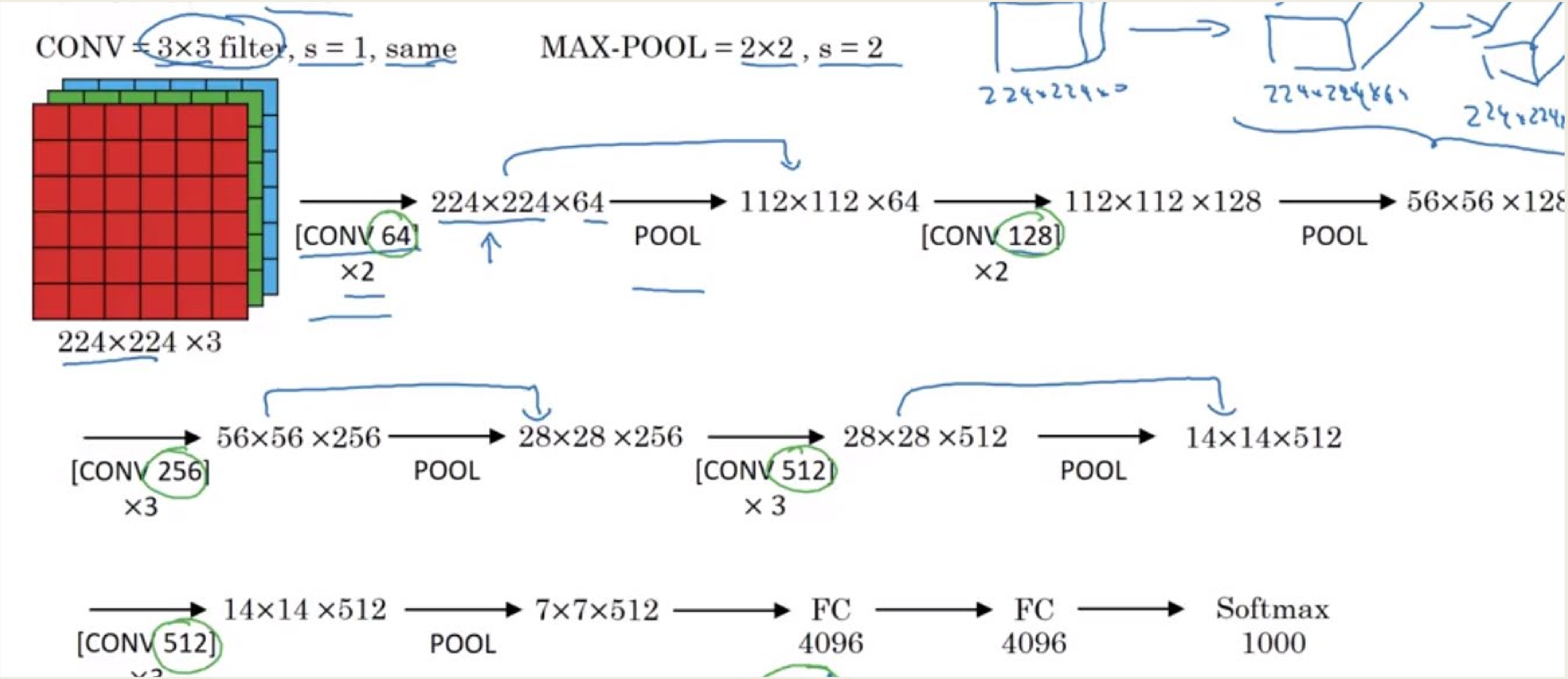
- Why convolutions?
  - **Parameter sharing:** A feature detector is probably useful in many parts of the image
  - **Sparsity of connections:** In each layer, each output value depends only on a small number of inputs

Examples: LeNet5 – AlexNet

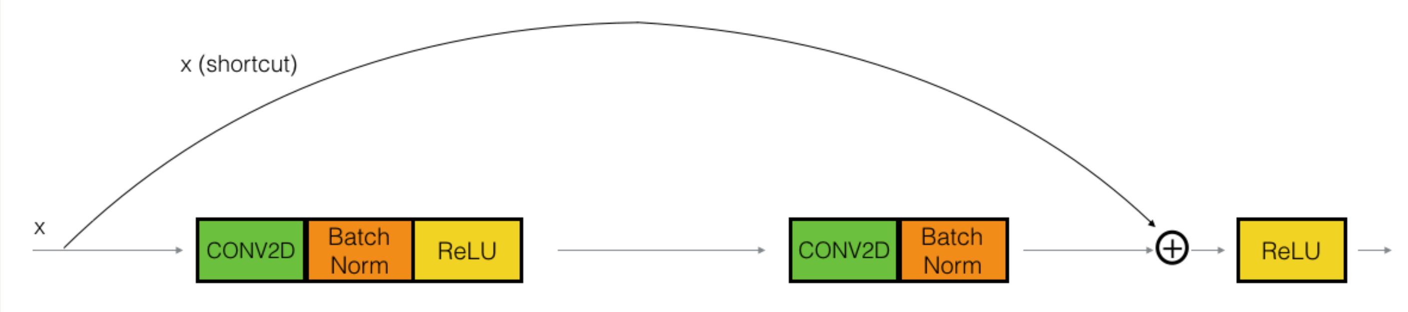




Examples: VGG16

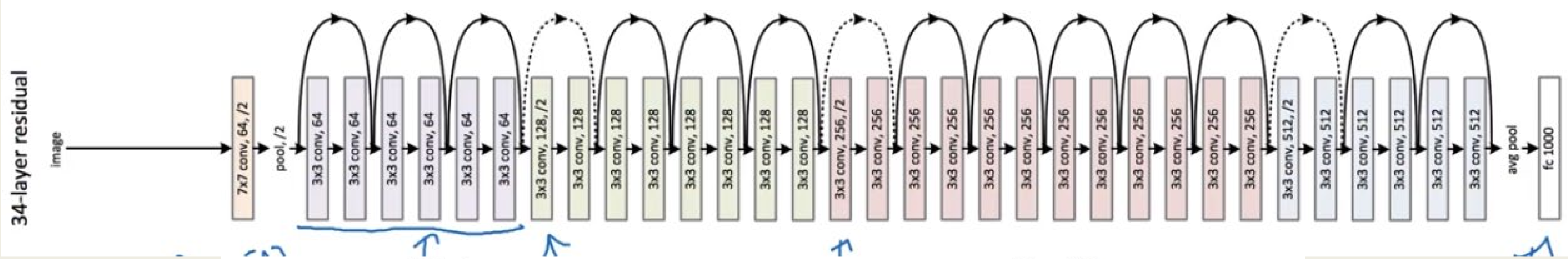


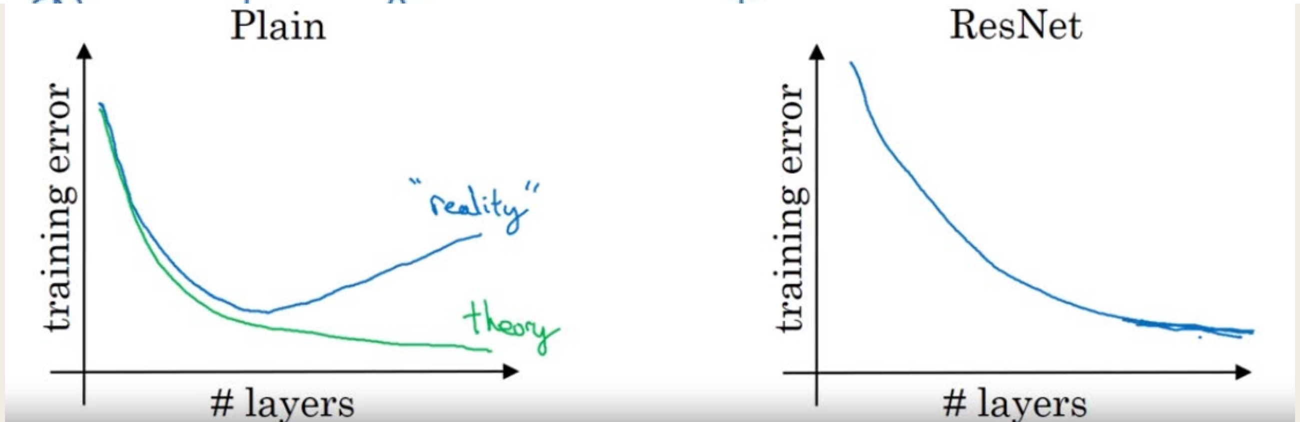
Residual Network



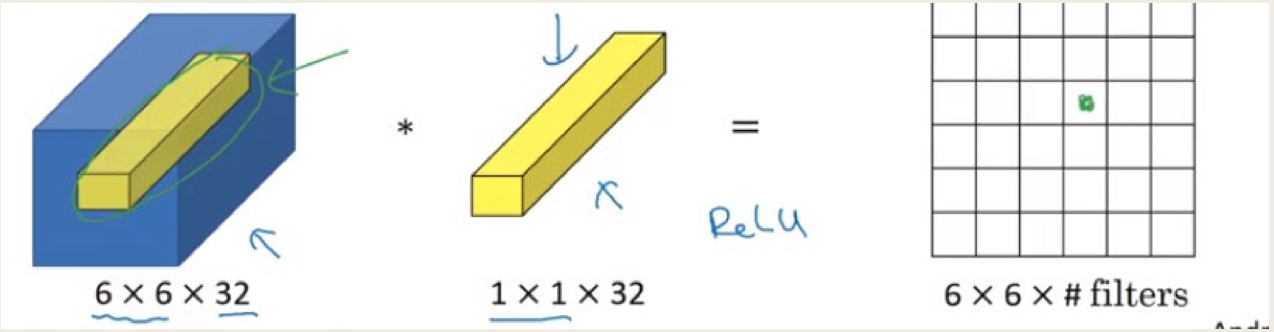
- **Shortcut = Skip connection:** $a^{[l+2]} = g\left(z^{[l+2]} + a^{[l]}\right)$
- **Residual network** vs **Plain network:**
  - → Identity function is easier for Residual block to learn (If $W^{[l+2]} = 0\ and\ b^{[l+2]} = 0$)

Ps: If sizes don't much → $a^{[l+2]} = g\left(z^{[l+2]} + \boldsymbol{W_s}, a^{[l]}\right)$; **Exp:** $a^{[l+2]} \to 256, a^{[l]} \to 128, \boldsymbol{W_s} \to (256{,}128)$





1x1 Convolution

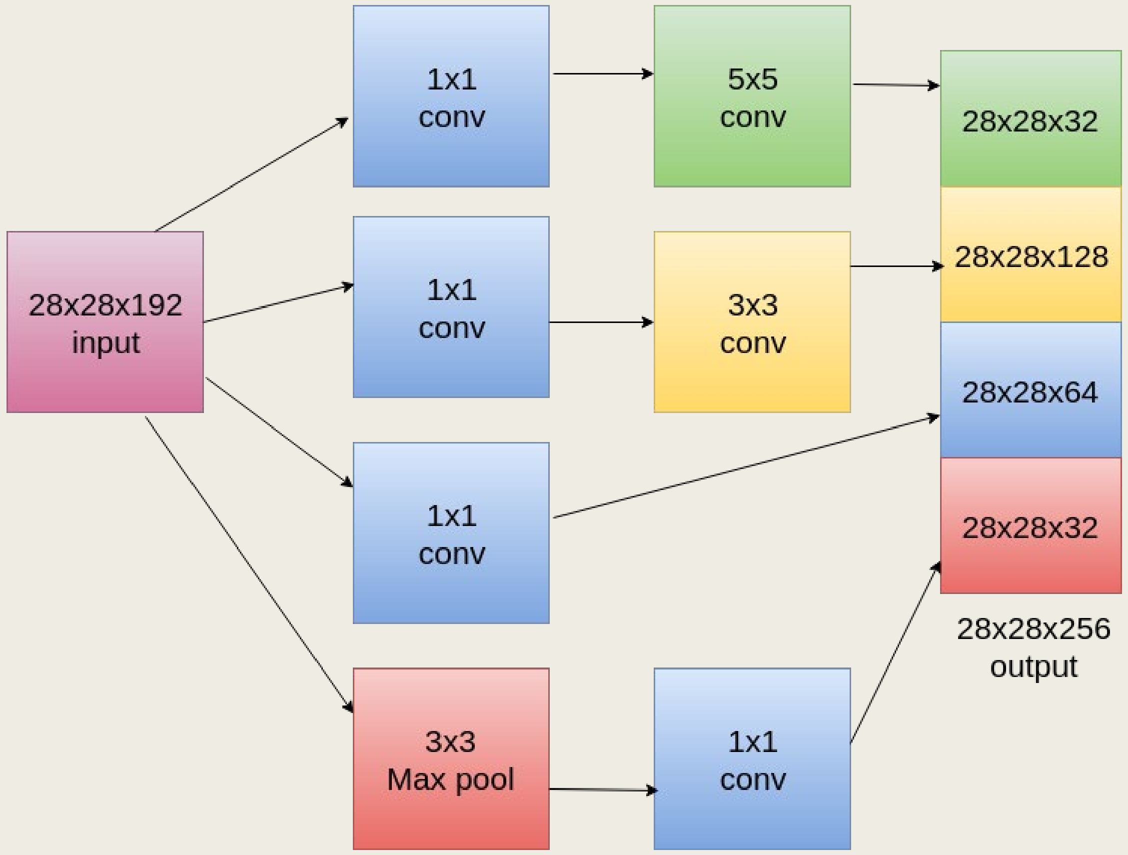


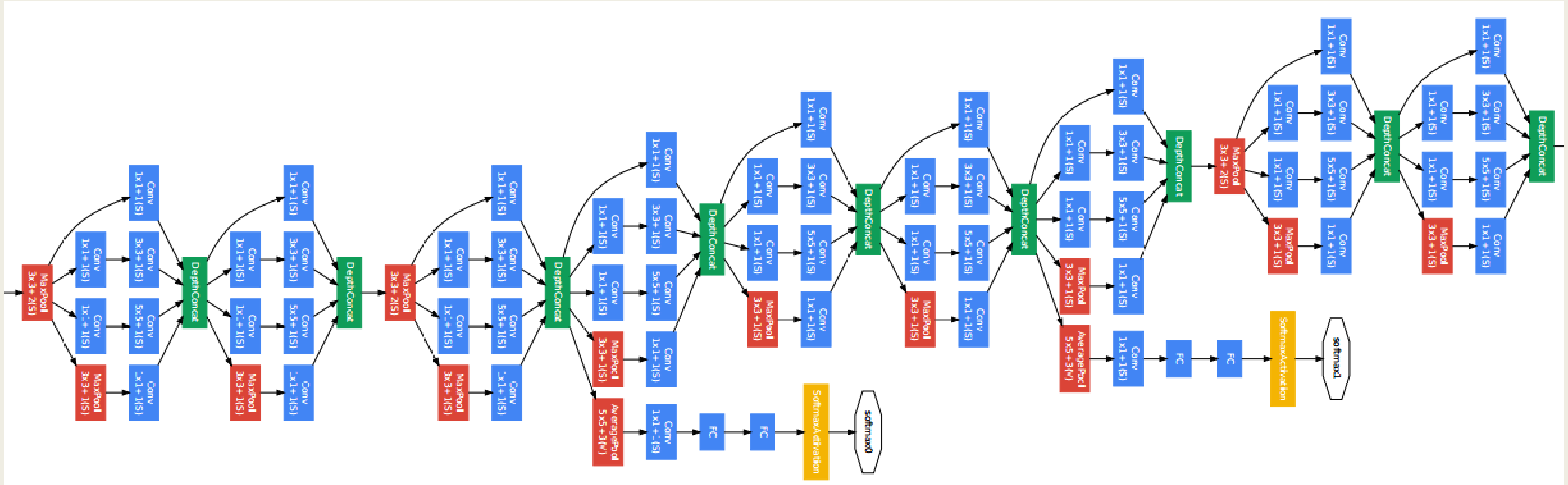
- <u>Goal</u>: Change size from $(n, n, value)$ to $(n, n, \#filters)$ thanks to $(1,1, value)$ #filters

→ Reduce computational cost (1x1 layer is called « bottleneck » layer

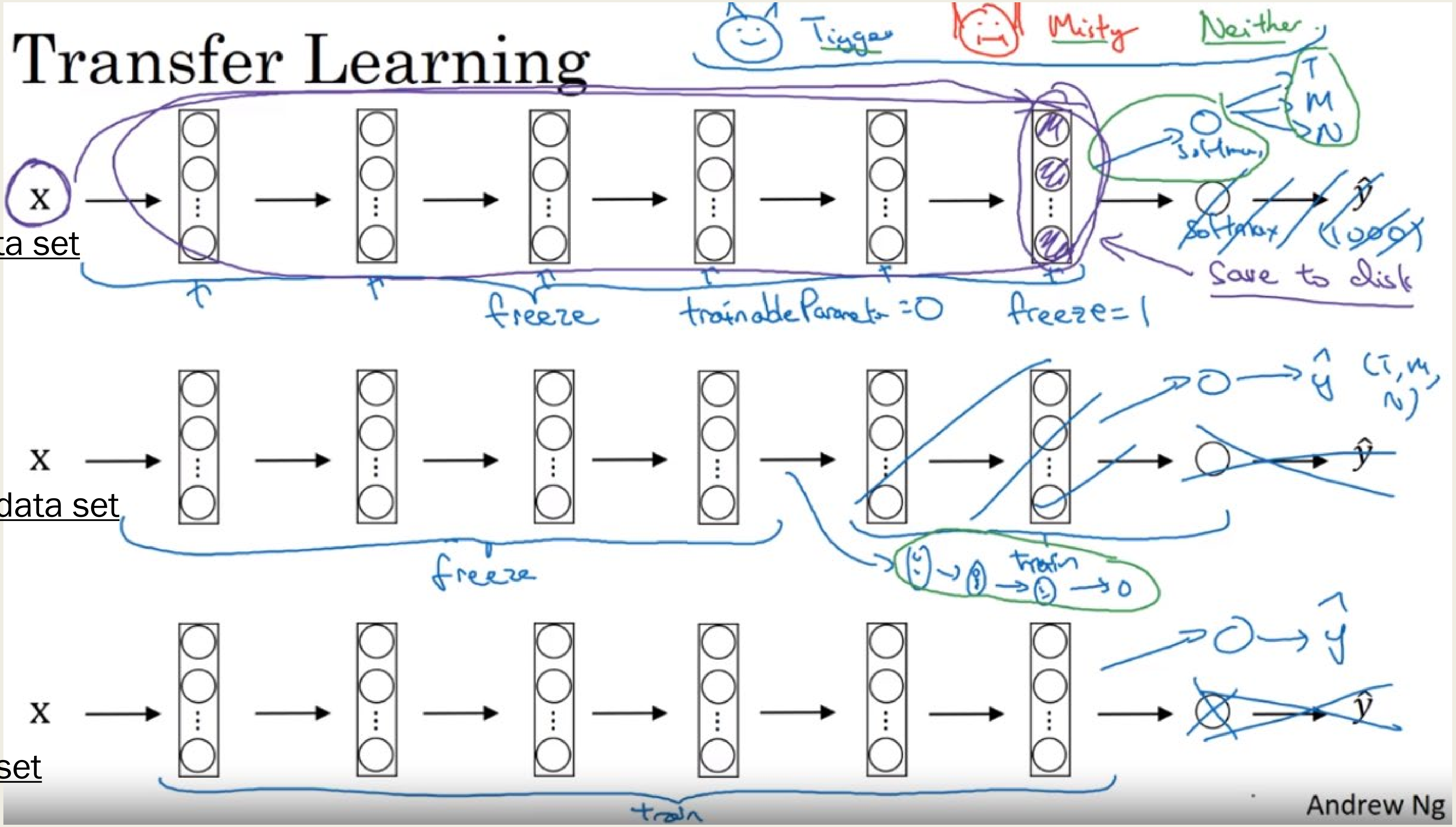
- <u>Tip</u>: Use open source code
  - Use architectures of networks published in the literature
  - Use open source implementations if possible
  - Use pretrained models and fine-tune on your dataset

Inception Network





Transfer Learning



Small data set

Medium data set

Big data set

Data Augmentation

- <u>Techniques</u>: Mirroring, Random cropping, Rotation, Shearing, Local warping, Color shifting…
- Implementing distortions during training → Parallel job using threads:
  - CPU thread to distort picture
  - CPU thread to train data
- <u>Two sources of knowledge</u>:
  - Labeled data
  - Hand engineered features/network architecture/other components
- **Little** data (More hand-engineering) vs **Much** data (Less hand-engineering ⇔ Simpler algorithms)
- **Little** data: <u>Object detection</u> → <u>Image recognition</u> → <u>Speech recognition</u>: **Much** data
- <u>Tips for doing well on benchmarks</u>:
  - **Ensembling**: Train several networks independently and average their outputs
  - **Multi-crop at test time**: Run classifier on multiple versions of test images and average results

10-crop

Keras {1}

- High level framework that provides additional abstractions (Higher than TensorFlow)
- In Keras, instead of creating a new variable on each step of forward propagation (X, Z1, A1, Z2, A2…) we just reassign X to a new value

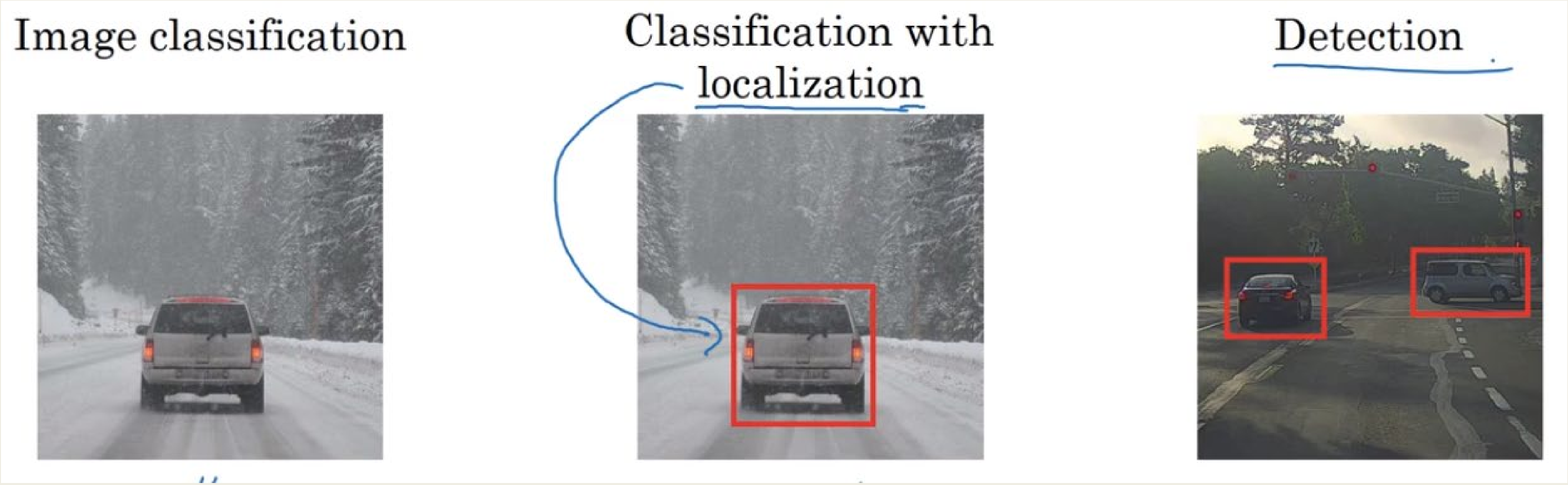
```
def model(input_shape):
    # Define the input placeholder as a tensor with shape input_shape. Think of this as your input image!
    X_input = Input(input_shape)

    # 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 your Keras model instance, you'll use this instance to train/test the model.
    model = Model(inputs = X_input, outputs = X, name='HappyModel')

    return model
```

Keras {2}

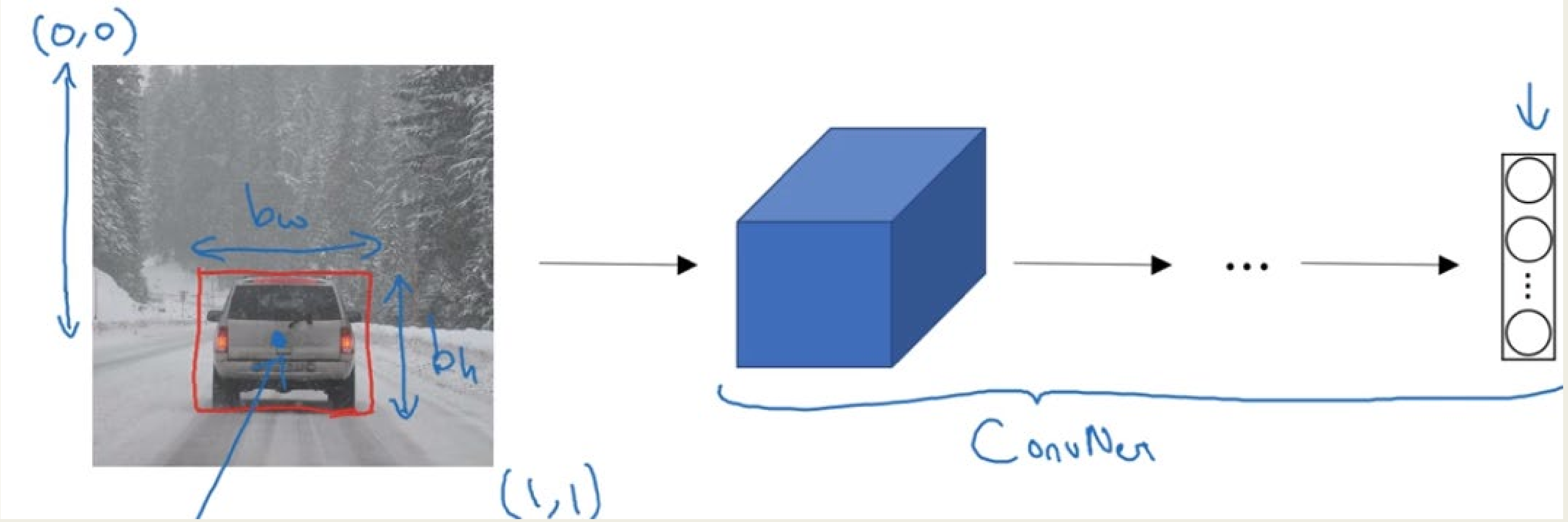
1. Create the model by calling the function above
2. Compile the model by calling model.compile(optimizer=“”, loss=“”, metrics=[“accuracy”])
3. Train the model on train data by calling model.fit(X=,Y=,epochs=,batch_size=)
4. Test the model on test data by calling model.evaluate(X=,Y=)

- <u>Useful tips</u>:
  - model.summary(): prints the details of your layers in a table with the sizes of its inputs/outputs
  - plot.model(): plots your graph in a nice layout. You can even save it as “.png” using SVG()

Classification, Localization, Detection



Classification with Localization



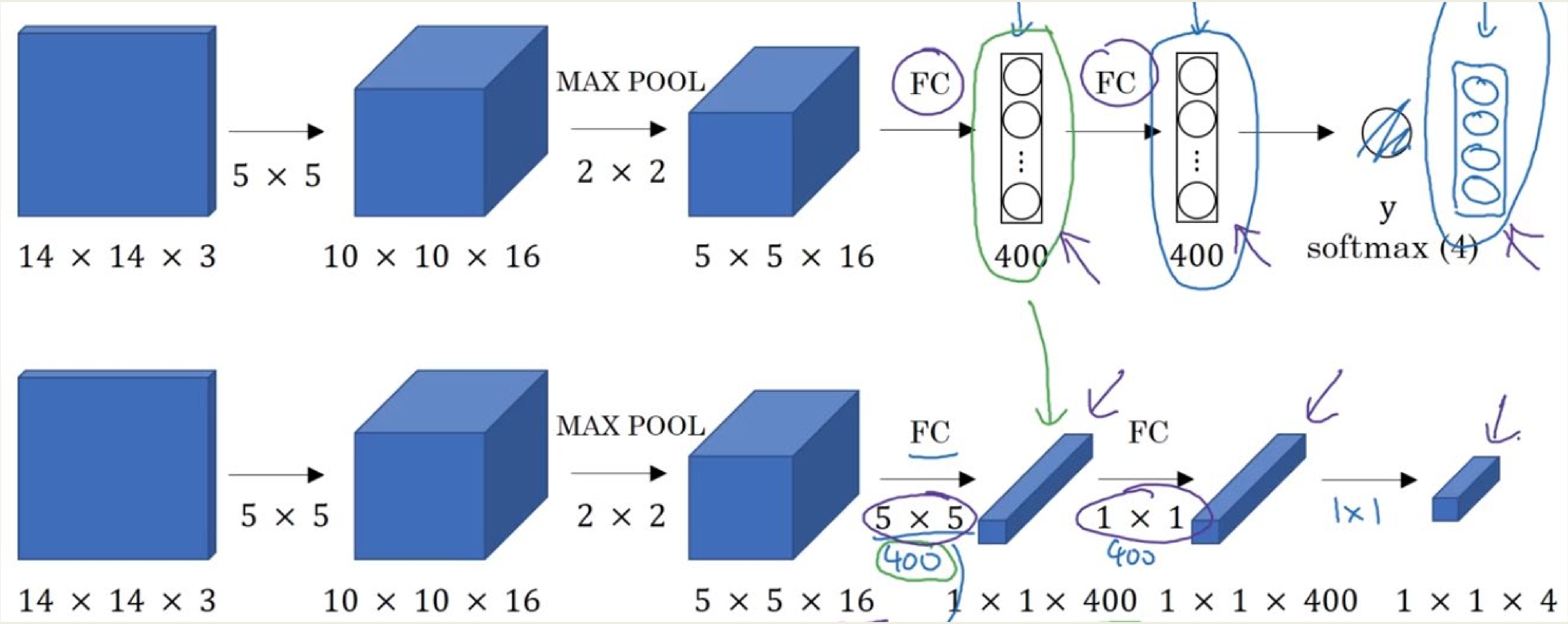
- Output layer contains: $\begin{pmatrix} p_c \rightarrow 1\ if\ object\ exist \\ b_x \rightarrow box\ x\_coordinate \\ b_y \rightarrow box\ y\_coordinate \\ b_h \rightarrow box\ height \\ b_w \rightarrow box\ width \\ c_1 \rightarrow 1st\ object\ class \\ c_2 \rightarrow 2nd\ object\ class \\ c_3 \rightarrow 3rd\ object\ class \end{pmatrix}$; PS: If pc=0, we don't care about other values

Landmark Detection & Non-Convolution Implementation of Sliding Windows

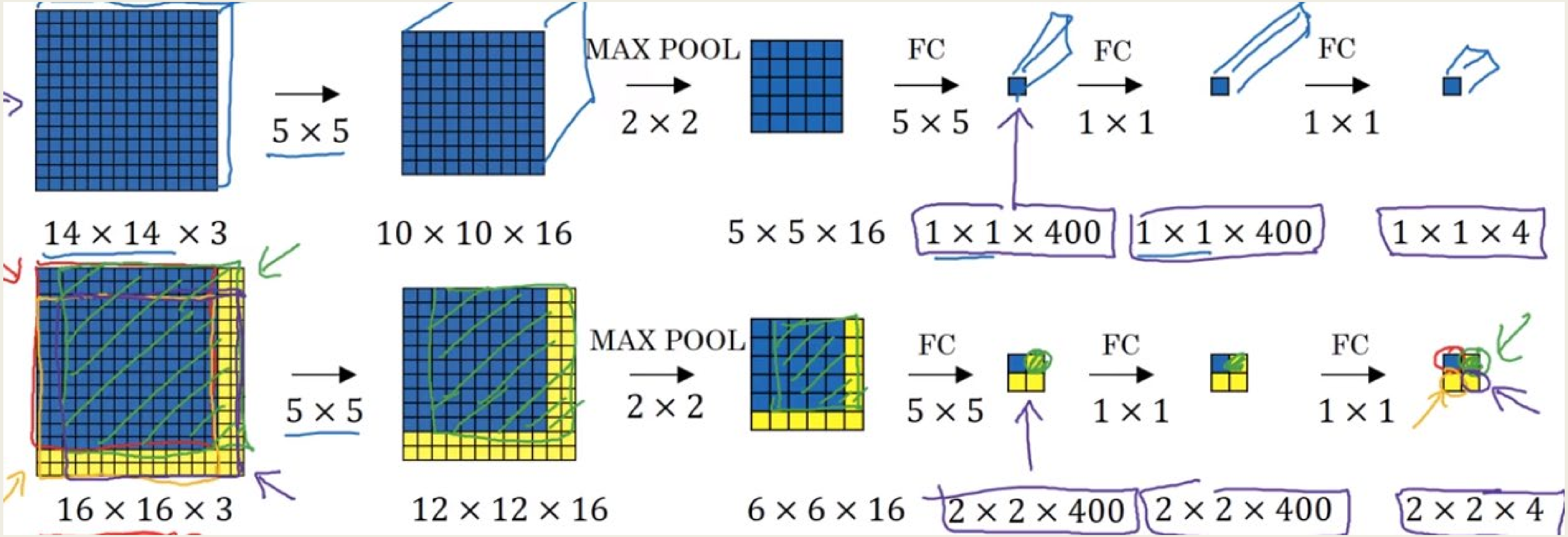
- **Goal:** Detect points (coordinates) instead of boxes (with height and width)

- Output layer contains: $\begin{pmatrix} p_c & \rightarrow 1\ if\ object\ exist \\ l_{1x} & \rightarrow landmardk1\ x\_coordinate \\ l_{1y} & \rightarrow landmardk1\ y\_coordinate \\ l_{2x} & \rightarrow landmardk2\ x\_coordinate \\ l_{2y} & \rightarrow landmardk2\ y\_coordinate \\ & \vdots \end{pmatrix}$

Turning FC layer into Convolutional Layer



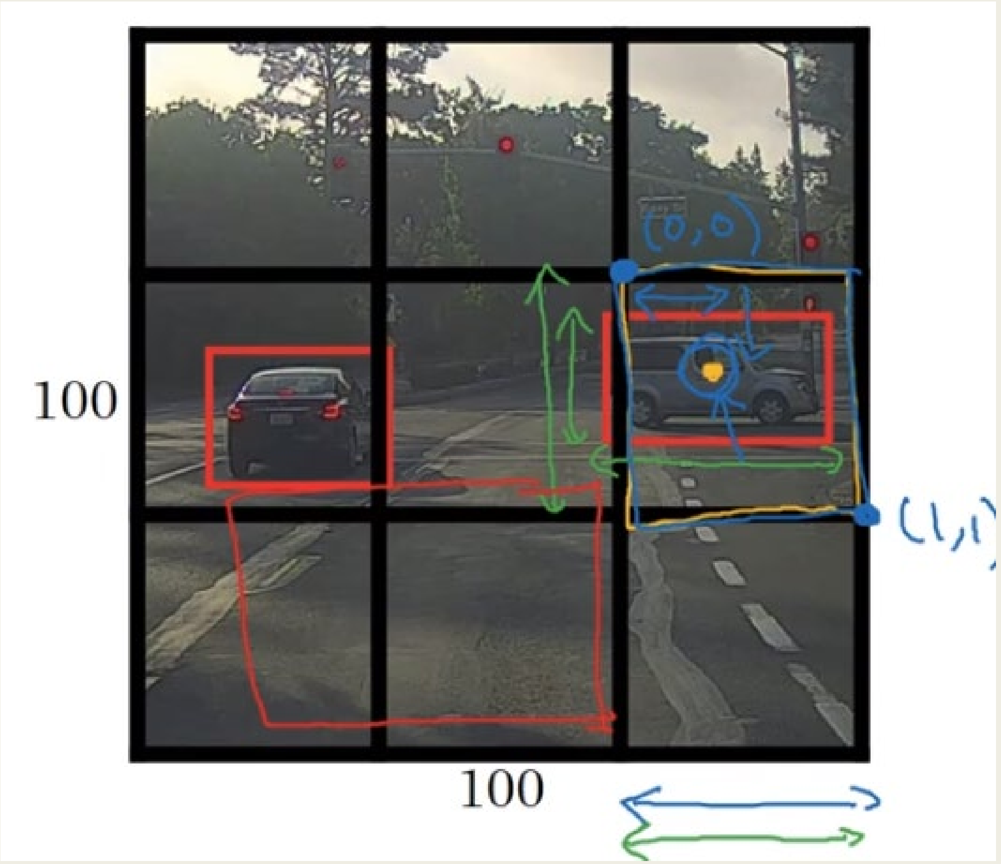
Convolution Implementation of Sliding Windows



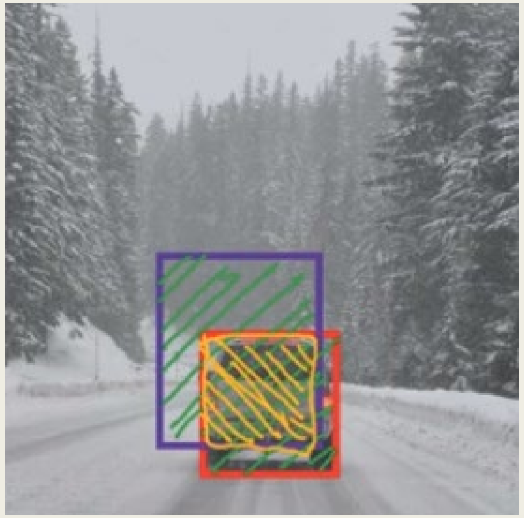
- PS: Instead of working on each window by itself (non-convolution), we will work on all the windows together using convolutional implementation

YOLO Algorithm

- YOLO = You Only Look Once
- Output size → $\left(\sqrt{\#grids}, \sqrt{\#grids}, 5 + \#classes\right), 5 = pc + b_x + b_y + b_h + b_w$
- PS: If P_c=0, other values don't matter
- PS1: $0 \leq b_x and\ b_y \leq 1 \rightarrow Relative\ to\ the\ grid$
- PS2: $b_h and b_w can\ be > 1$

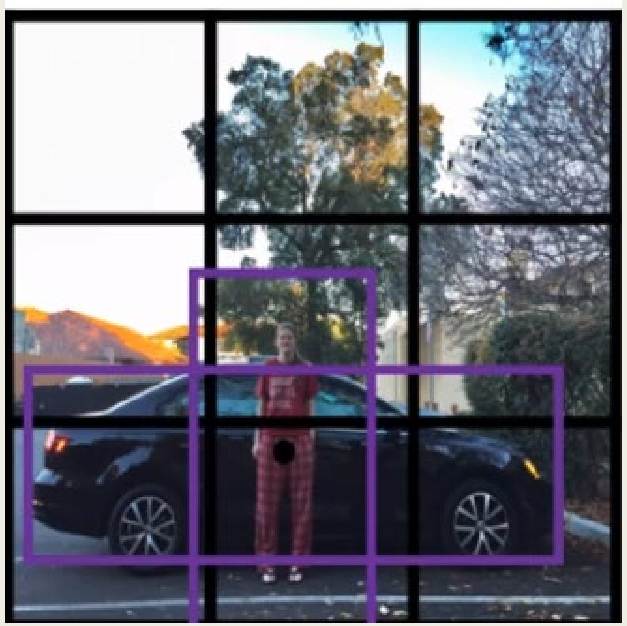


Non-Max Suppression Algorithm



- **Intersection over Union:** $IoU = \frac{size\ of\ intersection}{size\ of\ union} \rightarrow Correct\ if\ IoU \geq 0{,}5$
  - *IoU is measure of the overlap between two bounding boxes*
- **Algorithm:**
  1. *Each output prediction is* $\begin{pmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \end{pmatrix}$
  2. *Discard all boxes with* $p_c \leq 0{,}6$
  3. *While there are any remaining boxes:*
     1. Pick the box with the largest p_c and output that as a prediction
     2. Discard any remaining box with $IoU \geq 0{,}5$ with the box output in the previous step

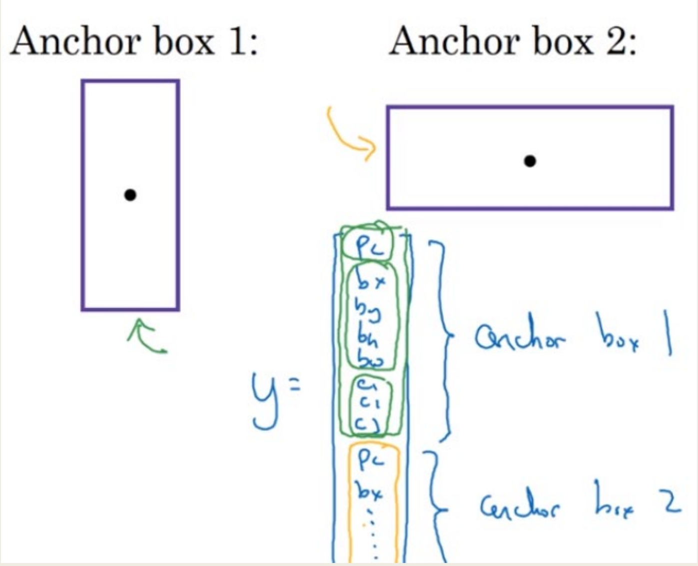
Anchor Box Algorithm



- **Goal:** Find solution to overlapping objects

PS: 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

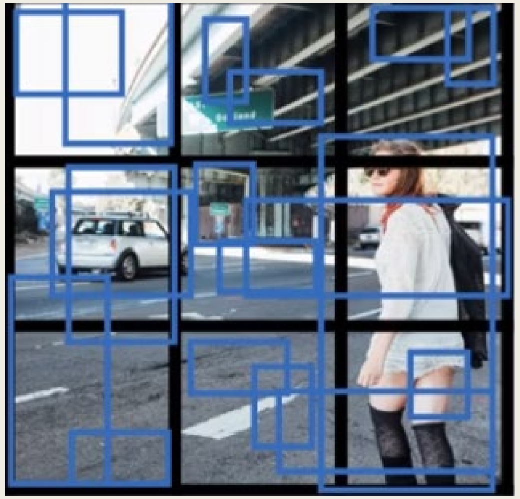
- Output size → $\left(\sqrt{\#grids}, \sqrt{\#grids}, (5 + \#classes) * \#AnchorBoxes\right), 5 = pc + b_x + b_y + b_h + b_w$
- PS: If one P_c=0, other values with same anchor box don't matter

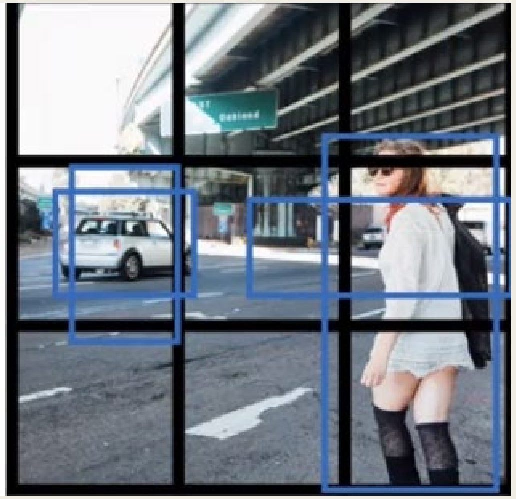


- **Drawbacks:**
  - *It is bad when different objects have the same shape*
  - *It is bad if 3 objects appear when having 2 anchor boxes*

Final Algorithm

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

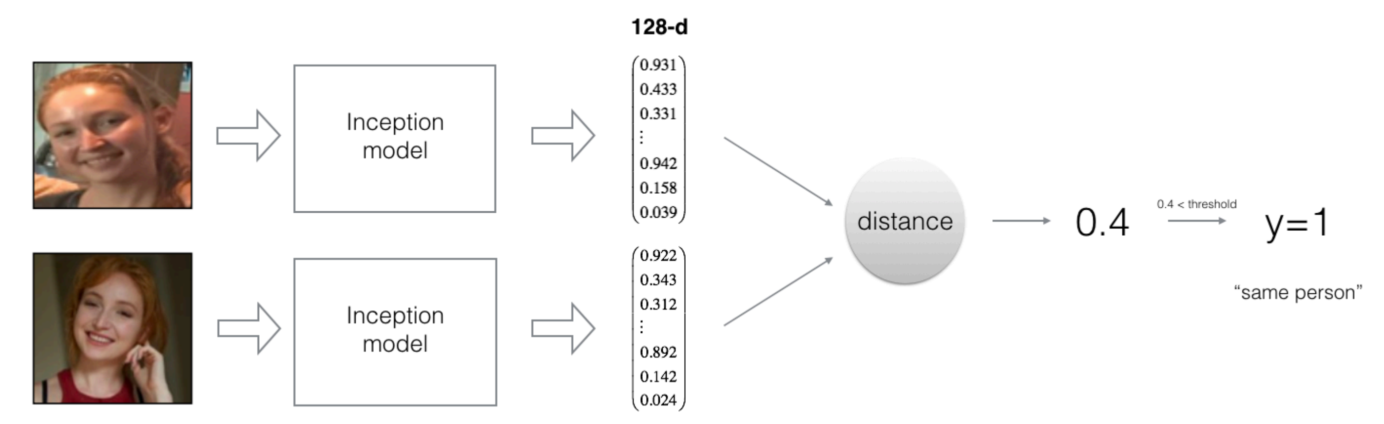




Region Proposal: R-CNN

- It is a segmentation algorithm to reduce the calculations
  - *R-CNN: Propose regions. Classify proposed regions one at a time. Output level + bounding box*
  - *Fast 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*

One-shot Learning (1)



- Face verification:
  - *Input image, name/ID*
  - *Output whether the input image is that of the claimed person*
- Face recognition:
  - *Has a database of K persons*
  - *Input image*
  - *Output ID if the image is any of the K persons (or "not recognized")*
- **One-shot learning**: Learning from one example to recognize the person again

  → *Learning a "**similarity**" function:* $d(img1, img2) = degree\ of\ difference\ between\ images$

$$d\left(x^{(1)}, x^{(2)}\right) = \left\|f\left(x^{(1)}\right) - f\left(x^{(2)}\right)\right\|_2^2$$

$$if\ d(img1, img2) \le \tau\ \rightarrow \text{same}$$

$$if\ d(img1, img2) > \tau\ \rightarrow different$$

  → ***Siamese Network***:
  - Person 1 → CNN → Softmax output (128 units) = encoding of person 1
  - Person 2 → Same CNN → Softmax output = encoding of person 2

  → Learn parameters so that is person 1/2 are the same person, d is small, and large otherwise
- Famous Networks: FaceNet, DeepFace

One-shot Learning (2)

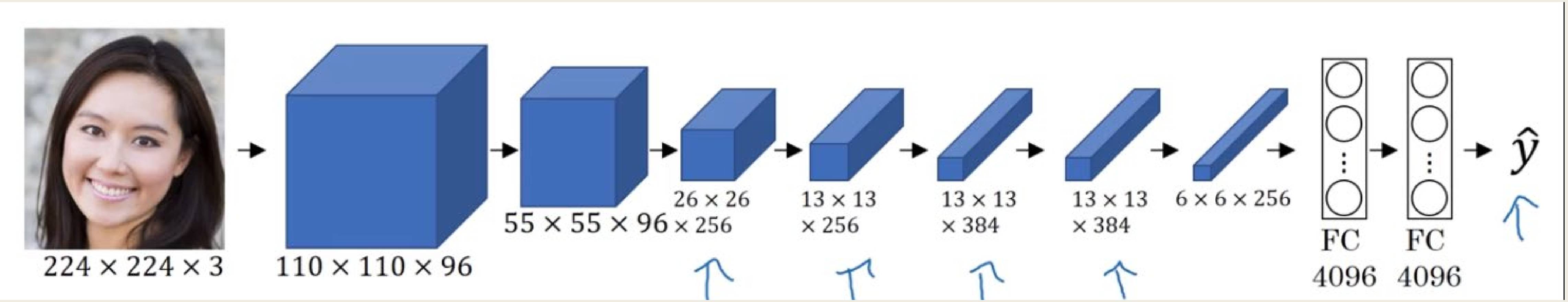
- We want $\left|\left|f(A) - f(P)\right|\right|^2 + \alpha \leq \left|\left|f(A) - f(N)\right|\right|^2 \Leftrightarrow d(A,P) + \alpha \leq d(A,N)$
  - *A: Anchor image, P: Positive, N: Negative, α: margin to make difference larger*
- Loss function: $L(A,P,N) = \max\left(\left|\left|f(A) - f(P)\right|\right|^2 - \left|\left|f(A) - f(N)\right|\right|^2 + \alpha, 0\right)$
- $J = \sum_{i=1}^{n} L\left(A^{(i)}, P^{(i)}, N^{(i)}\right)$; Training set: 10k pictures of 1k persons for example
- **PS**: if A, P, N are chosen randomly, the inequality is easily satisfied

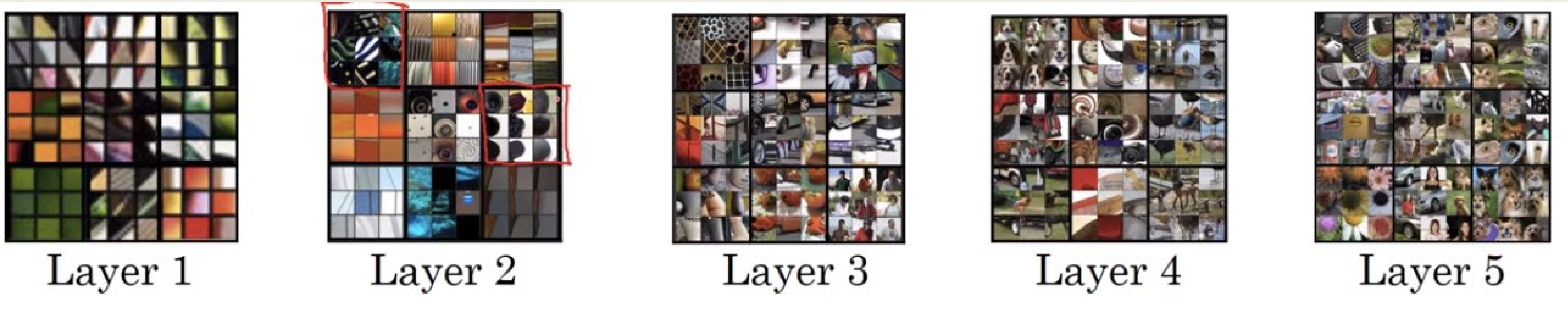
  → Choose triplets that are "hard" to train on ⇔ $d(A,P) \approx d(A,N)$
- **Learning the similarity function**: If a new person is added to the dataset, its ouput f($x^{(new)}$) will be compared to the already precomputed f($x^{(i)}$) of the dataset

$$\hat{\boldsymbol{y}} = \boldsymbol{\sigma}\left(\sum_{k=1}^{128} \boldsymbol{w_i}\left|\boldsymbol{f}\left(\boldsymbol{x}^{(i)}\right)_{\boldsymbol{k}} - \boldsymbol{f}\left(\boldsymbol{x}^{(j)}\right)_{\boldsymbol{k}}\right| + \boldsymbol{b}\right)$$

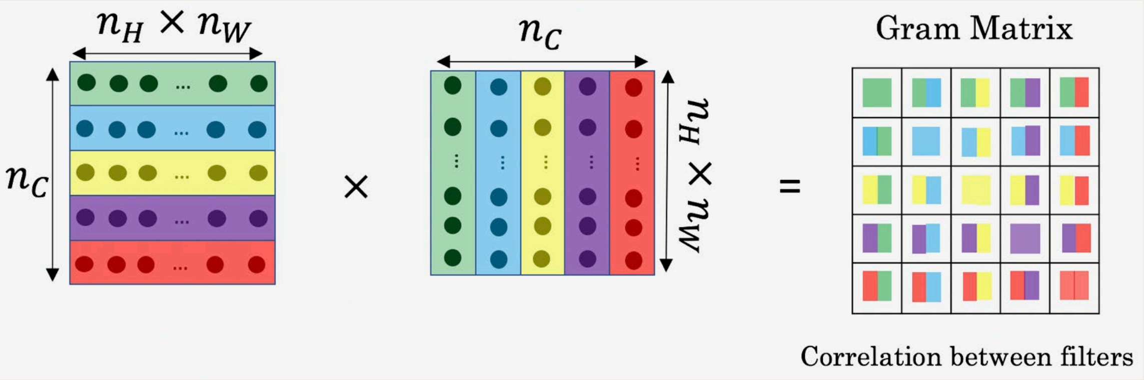
Visualizing what a deep network is learning

- **Tip**: Pick a unit in layer l. Find the nine image patches that maximize the unit's activation.

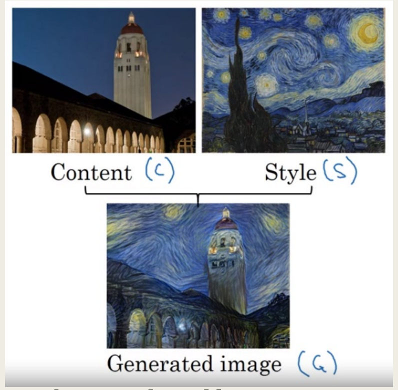


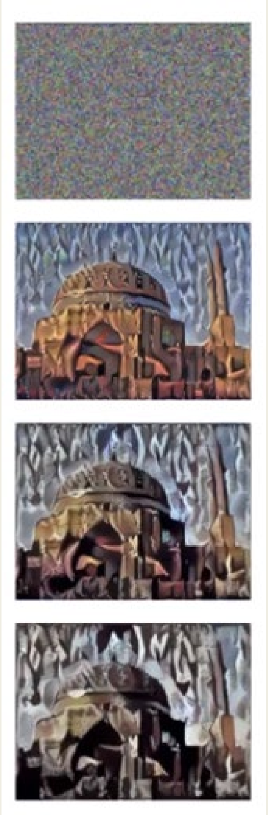


Neural Style Transfer



- $J(G) = \alpha\, J_{content}(C,G) + \beta\, J_{style}(S,G)$
- <u>Algorithm</u>:
  1. *Initialize G randomly*
  2. *Use gradient descent to minimize J(G):* $G \coloneqq G - \frac{d}{dG}J(G)$





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

→ The difference between the activation of the two layers. We want them to be similar

- **Style** = Correlation between activations across channels
- **Style matrix: = Gram matrix:** $G_{kk'}^{[l]} = \sum_{i=1}^{n_H^{[l]}} \sum_{j=1}^{n_W^{[l]}} a_{ijk}^{[l]} a_{ijk'}^{[l]}$
- $J_{style}^{[l]}(S,G) = \frac{1}{\left(2n_H^{[l]}n_W^{[l]}n_C^{[l]}\right)^2} \sum_{k=1}^{n_c^{[l]}} \sum_{k'=1}^{n_c^{[l]}} (G_{kk'}^{[l](S)} - G_{kk'}^{[l](G)}) = \frac{1}{\left(2n_H^{[l]}n_W^{[l]}n_C^{[l]}\right)^2} \left|\left|G^{[l](S)} - G^{[l](G)}\right|\right|_F^2$
- $J_{style}(S,G) = \sum_l \lambda^{[l]}.J_{style}^{[l]}(S,G)$