

Google Classroom Code: mhxgl24

Transfer Learning & Object Detection

Deep Learning (DS-5006)

Dr. Adeel Mumtaz

Lecture 9

Fall, 2022

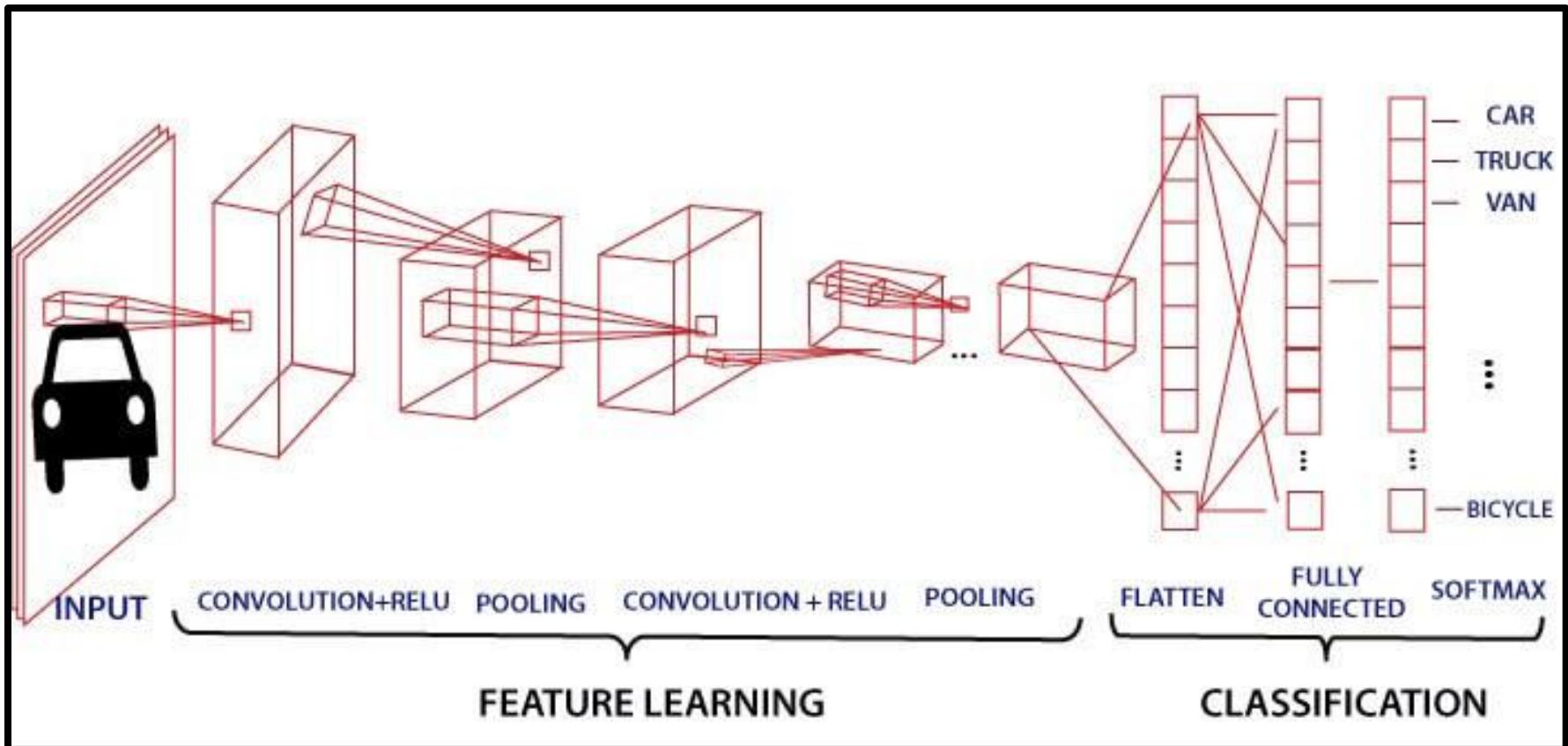


National University
Of Computer and Emerging Sciences

Contents

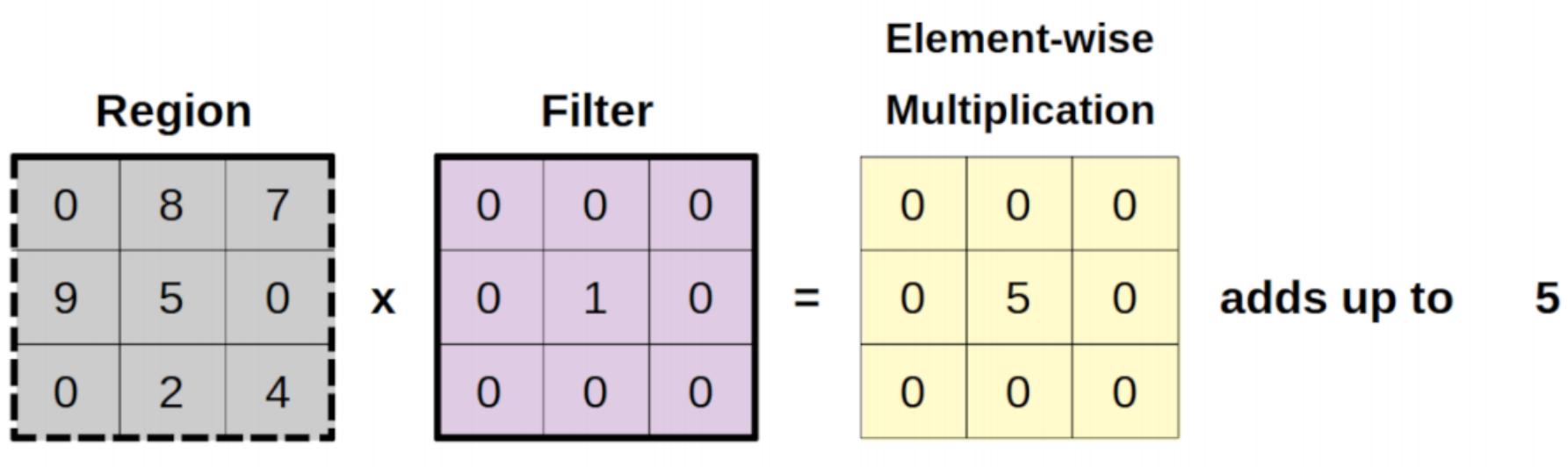
- Last Lecture
 - CNNs / ConvNets Architecture
 - Graded Home Task-5
- Learning Rate
 - Learning Rate Schedulers
- Transfer Learning
 - Introduction
 - Concepts in Large CNN Models
 - Adaptive Pooling
 - Auxiliary classifiers (Side Heads)
 - 1x1 Convolutions
 - Batch Normalization
 - Residual Connections
 - Transfer Learning with AlexNet
 - Step-1 Loading Pre-Trained Model
 - Step-2 Model Freezing
 - Step-3 Standardization Parameters
 - Code for RPS Classification with AlexNet
- Object Detection with CNNs
 - Types of Detection algorithms
 - Sliding Windows Detector
 - Overfeat Detector
 - YOLO Detector
 - IOU
 - NMS
 - Anchor Boxes
 - Training & predicting with YOLO
 - Ultralytics YOLOv5
 - Data Labeling & Preparation of a Custom Dataset
 - Installing YOLOv5
 - Training YOLOv5
 - Inference
- Assignment-2

CNNs / ConvNets Architecture



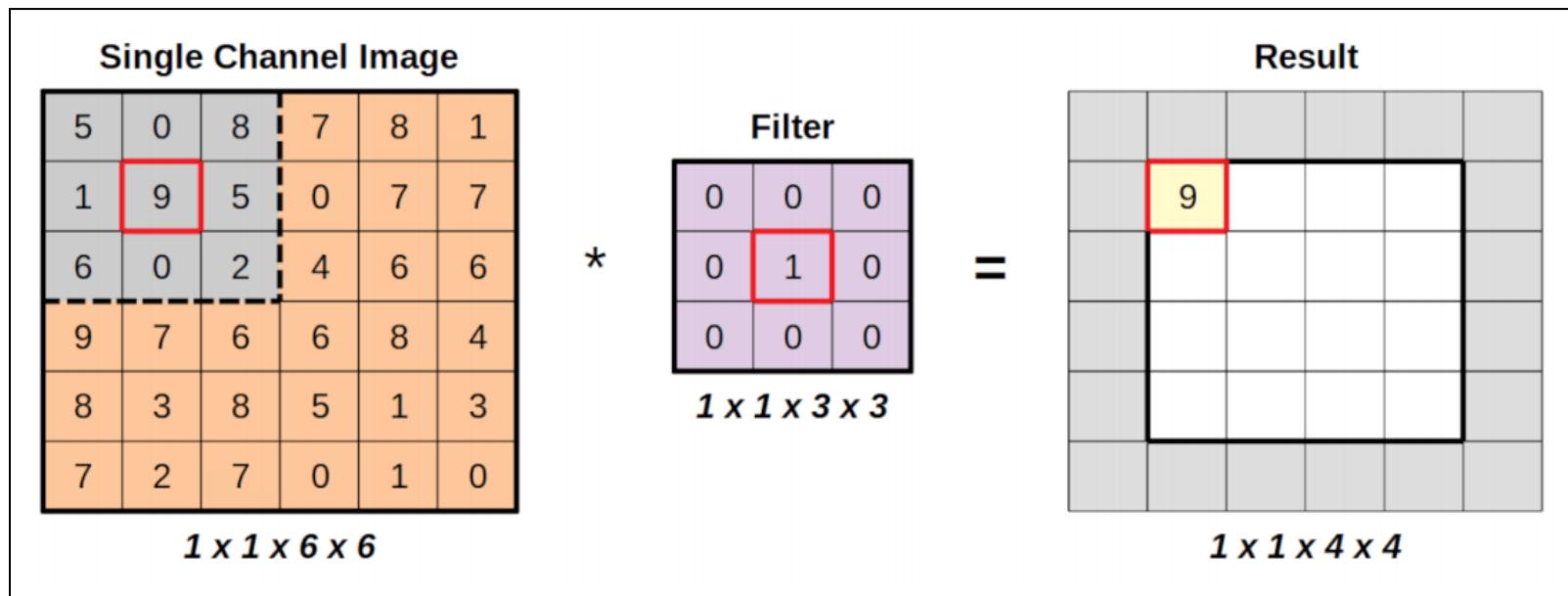
Convolution Layer

- Convolution refers to the **filtering** process



Convolution Layer

- Convolution refers to the filtering process
- Doing a convolution produces an image with a **reduced size**.



Convolution Layer

- Move the region one step to the right, that is, we change the receptive field, and apply the filter again

Single Channel Image

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

$1 \times 1 \times 6 \times 6$

Move Region

1 step

to the right

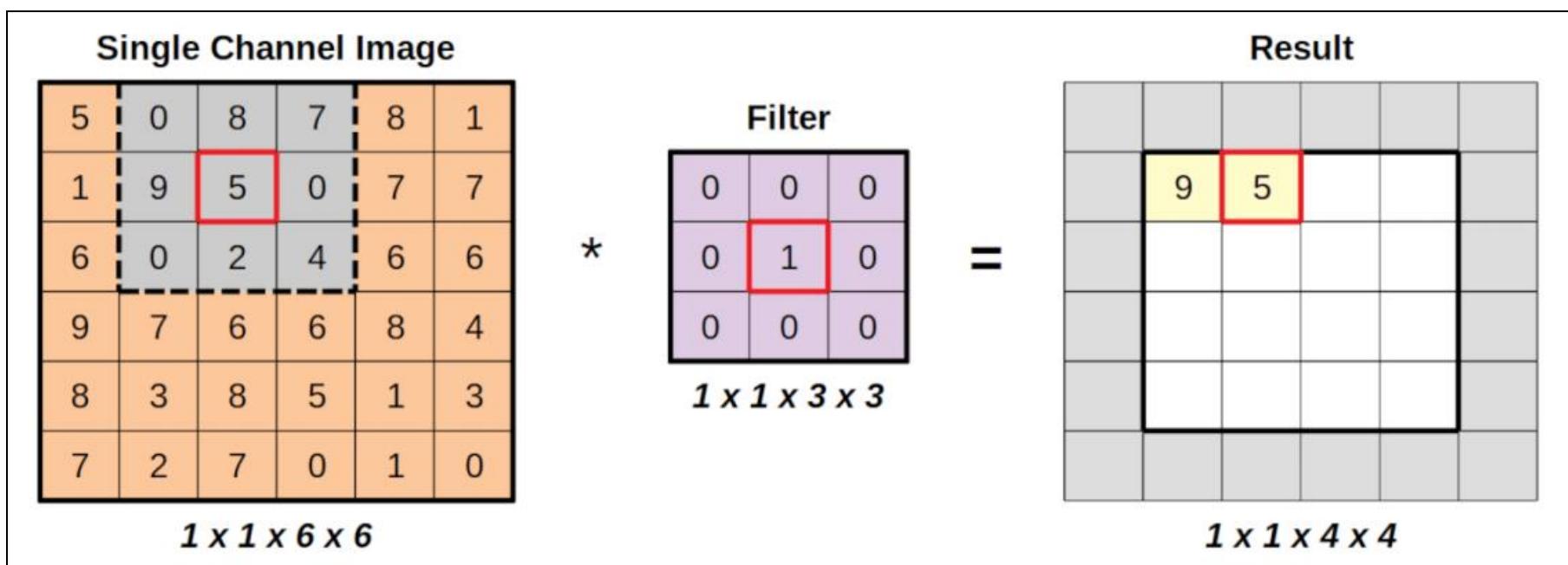
Single Channel Image

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

$1 \times 1 \times 6 \times 6$

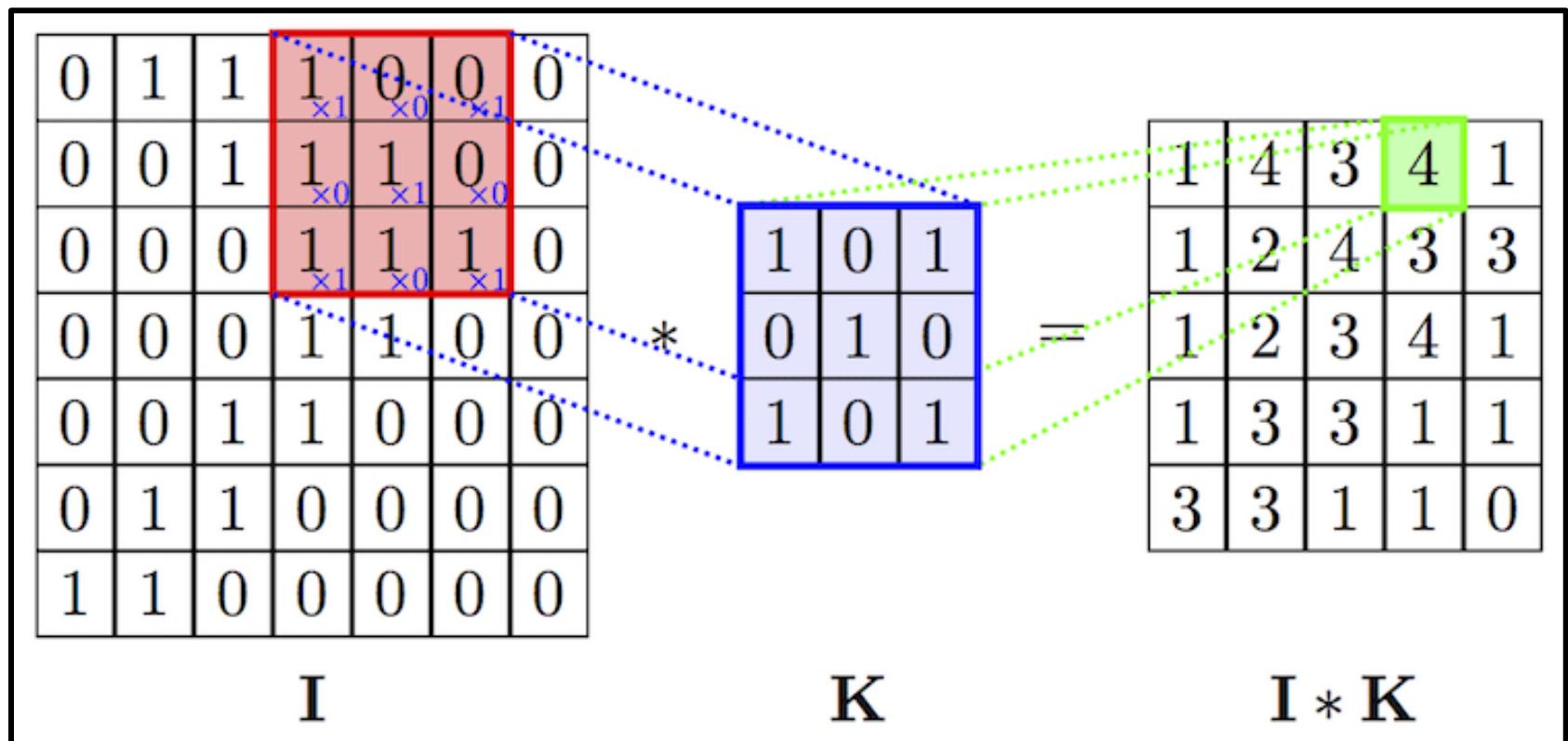
Convolution Layer

- Move the region one step to the right, that is, we change the receptive field, and apply the filter again



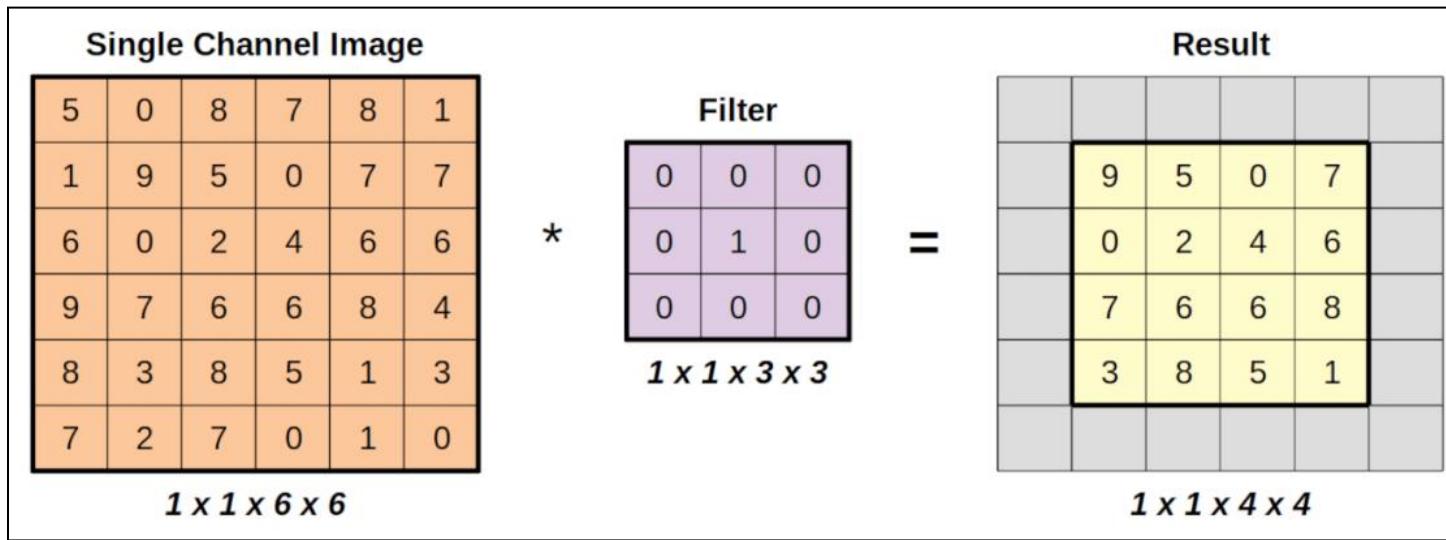
Convolution Layer

- Convolution refers to the filtering process



Convolution Layer

- Fully Convolved
- How much smaller is it going to be?
 - The bigger the filter, the smaller the resulting image
 - **reduction** is equal to the filter size minus one

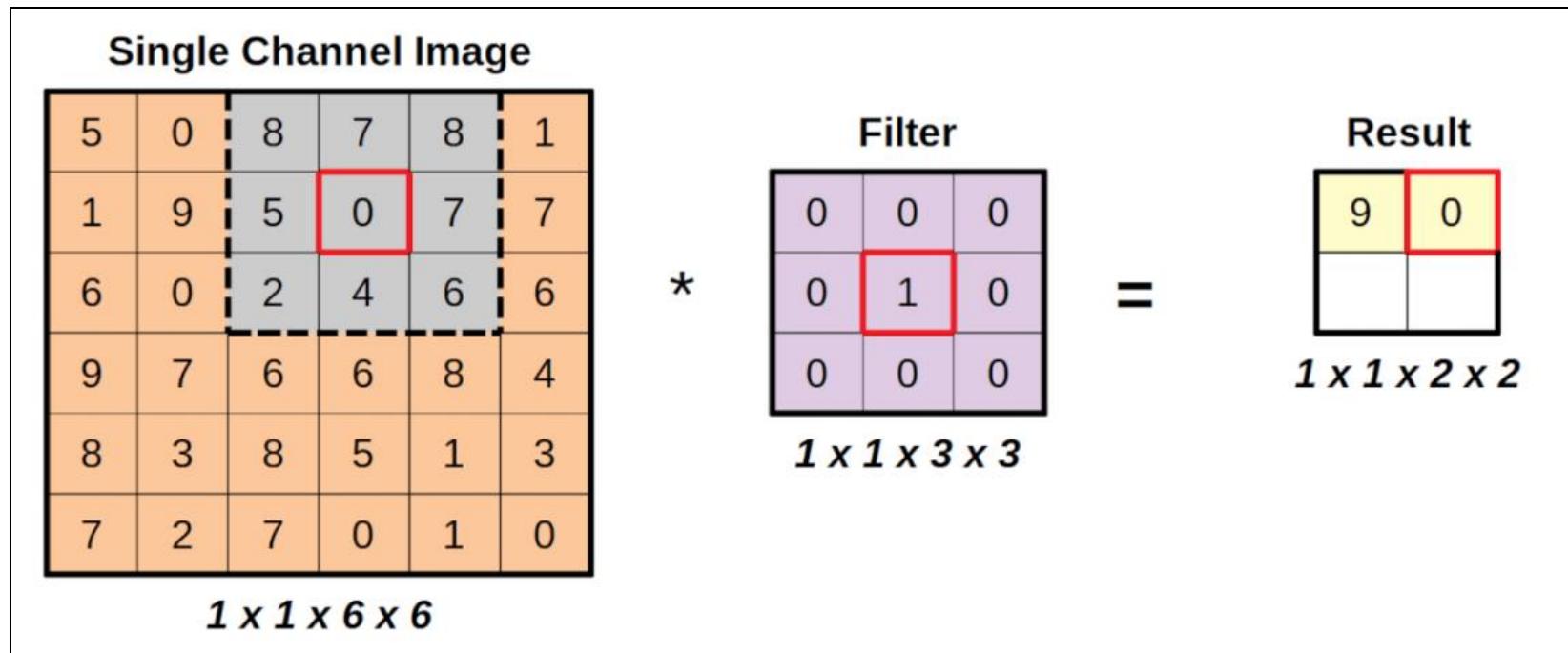


$$(h_i, w_i) * (h_f, w_f) = (h_i - (h_f - 1), w_i - (w_f - 1))$$

$$(h_i, w_i) * f = (h_i - f + 1, w_i - f + 1)$$

Striding

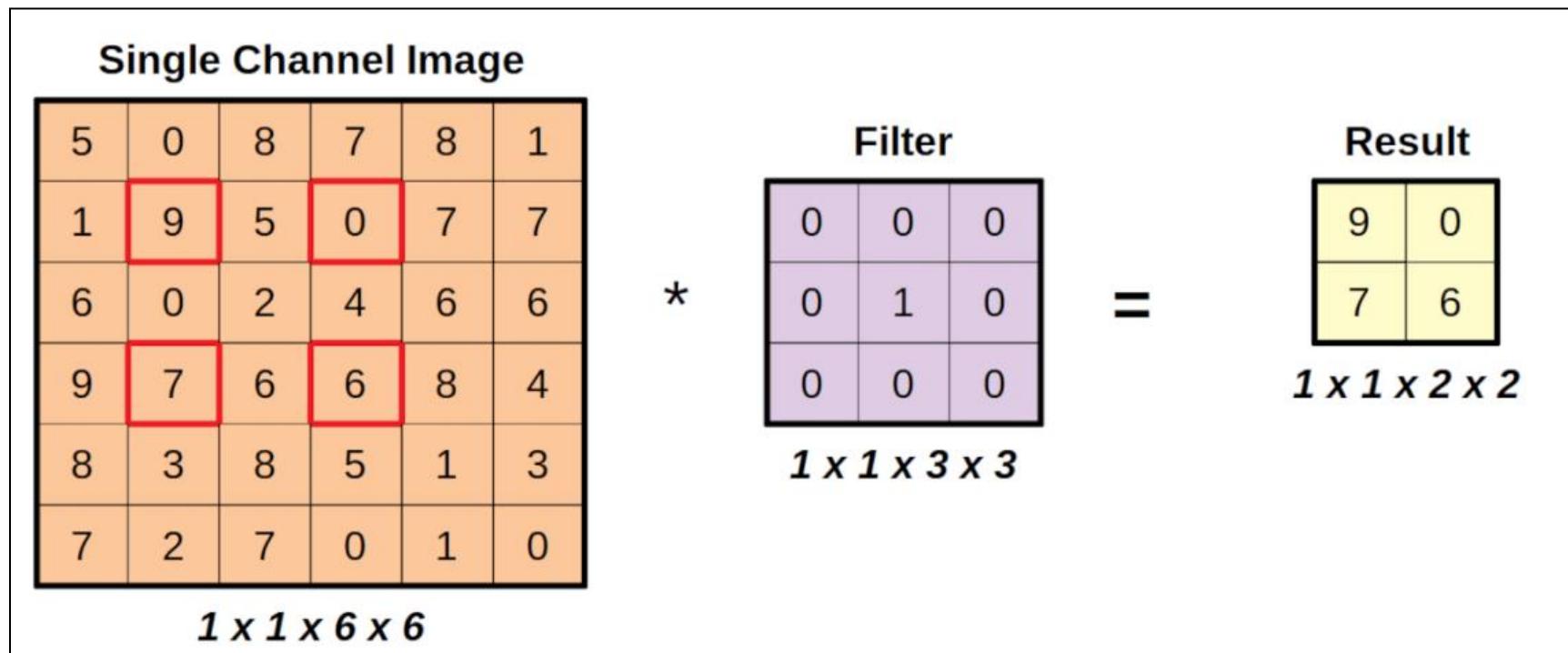
- So far, we've been moving the region of interest one pixel at a time:
 - a stride of one



A stride of 2

Striding

- Stride of 2
 - only **four** valid operations
- The identity kernel may be boring
 - but it highlight the inner workings of the convolutions.
 - It is clear in the figure above where the pixel values in the resulting image come from



A stride of 2

Striding

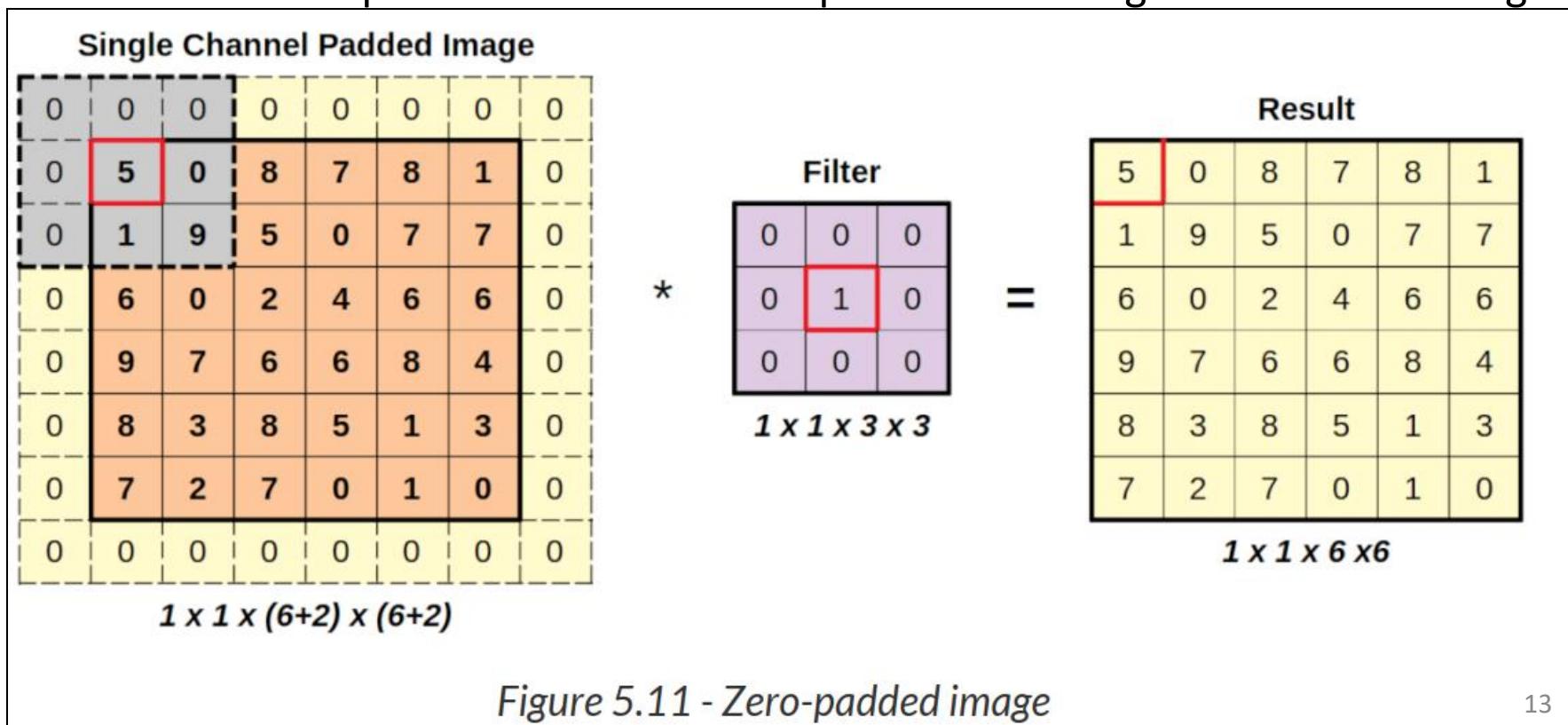
- The bigger the stride, the smaller the resulting image
- So far, the operations we performed have been shrinking the images. What about **restoring them to their original size?**

$$(h_i, w_i) * f = \left(\frac{h_i - f + 1}{s}, \frac{w_i - f + 1}{s} \right)$$

Equation 5.3 - Shape after a convolution with stride

Padding

- Padding means stuffing.
- We need to stuff the original image so it can sustain the "attack" on its size
 - We may simply **add zeros around it**
 - This simple trick can be used to preserve the original size of the image



Padding Modes

- In the **replication** padding, the padded pixels will have the same value as the closest real pixel

Replication Padding									Reflection Padding									Circular Padding								
5	5	0	8	7	8	1	1		9	1	9	5	0	7	7	7	0	7	2	7	0	1	0	7		
5	5	0	8	7	8	1	1		0	5	0	8	7	8	1	8	1	9	5	0	7	7	0	7	1	
1	1	9	5	0	7	7	7		9	1	9	5	0	7	7	7	6	0	2	4	6	6	6	6	6	
6	6	0	2	4	6	6	6		0	6	0	2	4	6	6	6	7	9	7	6	6	8	4	8	8	
9	9	7	6	6	8	4	4		7	9	7	6	6	8	4	8	3	8	3	8	5	1	3	1	3	
8	8	3	8	5	1	3	3		3	8	2	7	0	1	0	1	2	7	2	7	0	1	0	7	0	
7	7	2	7	0	1	0	0		2	7	2	7	0	1	0	1	3	8	3	8	5	1	3	8	1	
7	7	2	7	0	1	0	0		3	8	3	8	5	1	3	1	1	5	0	8	7	8	1	5	1	

Convolution Size after Padding

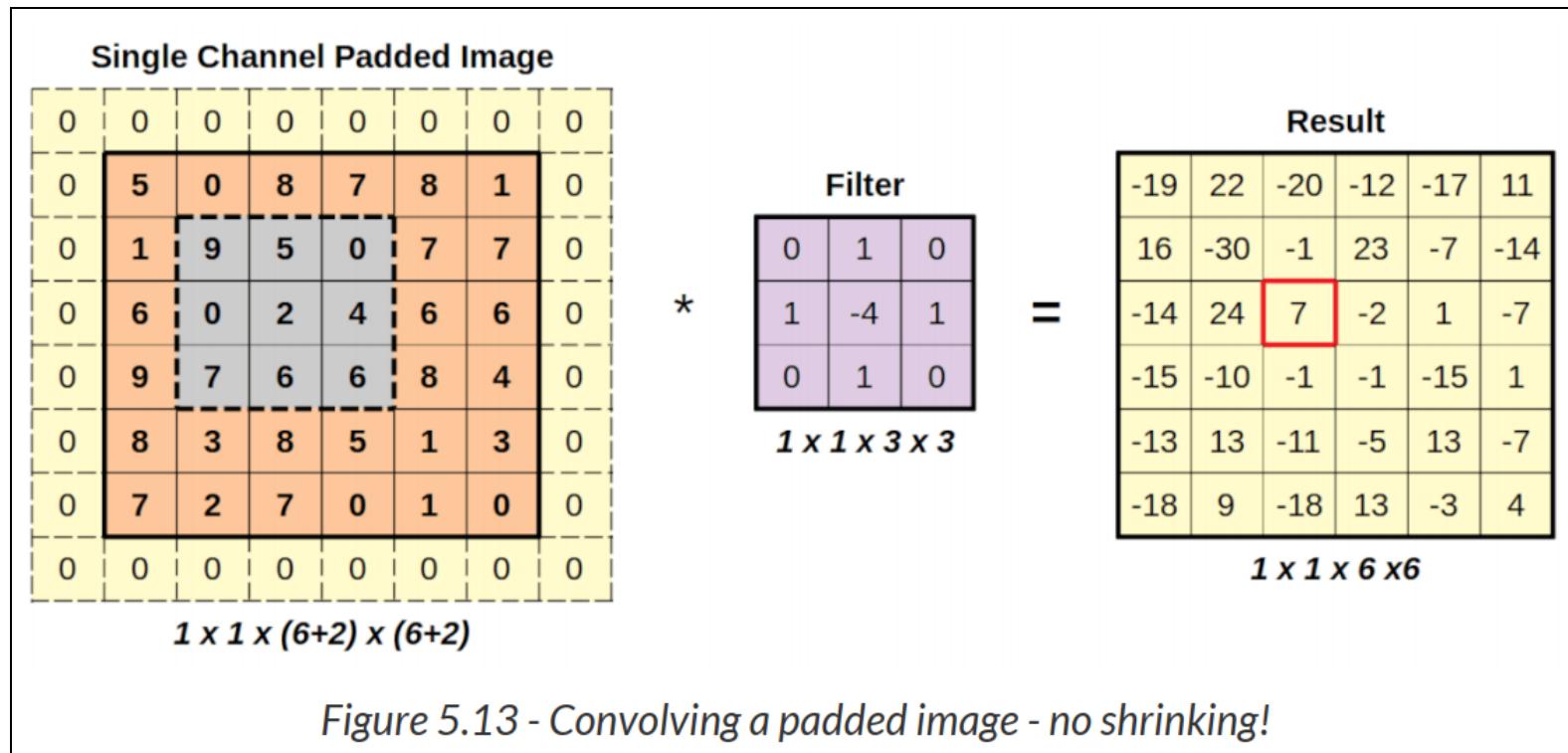
- In the **replication** padding, the padded pixels will have the same value as the closest real pixel

$$(h_i, w_i) * f = \left(\frac{(h_i + 2p) - f}{s} + 1, \frac{(w_i + 2p) - f}{s} + 1 \right)$$

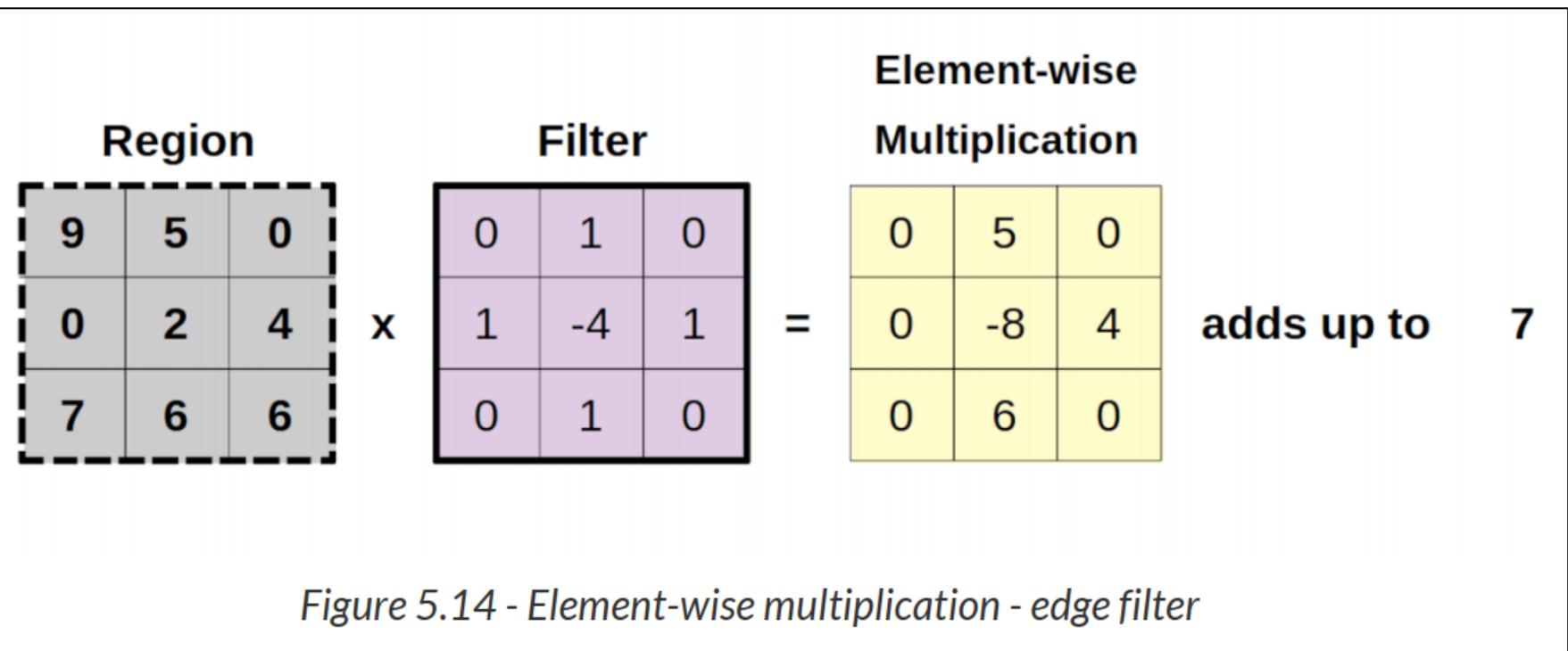
Equation 5.4 - Shape after a convolution with stride and padding

A Real filter

- Edge Detector

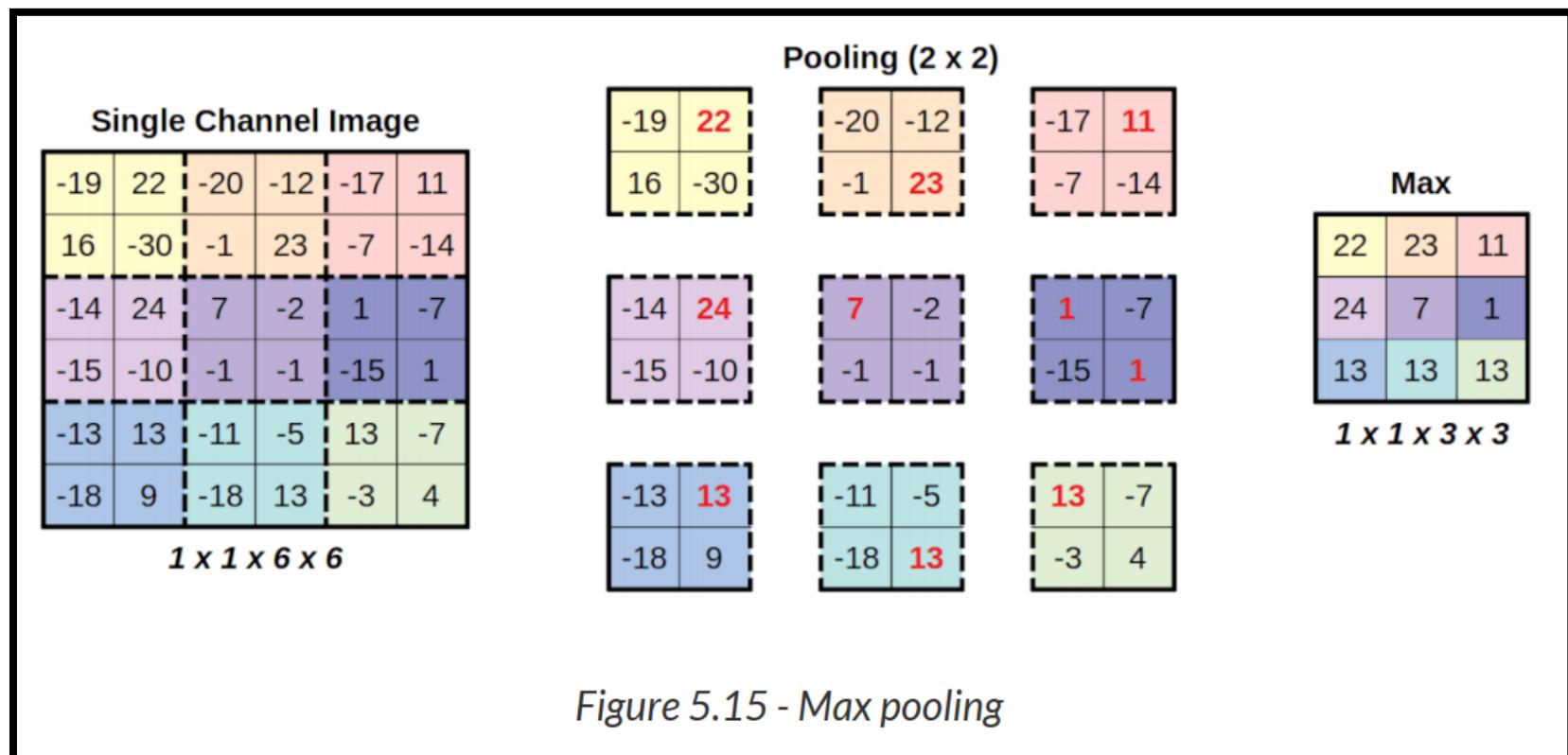


A Real filter



Pooling Layer

- Splits the image into tiny chunks, performs an operation on each chunk
 - Each chunk yields a single value
 - Puts the chunks together as the resulting image and shrink image
- **Max-Pooling** with a kernel size of two (stride is also a parameter of pooling)
- Besides max-pooling, **average pooling** is also fairly common



Flattening Layer

- It simply flattens a tensor by **collapsing** dimensions

1	1	0
4	2	1
0	2	1

Pooled Feature Map

Flattening



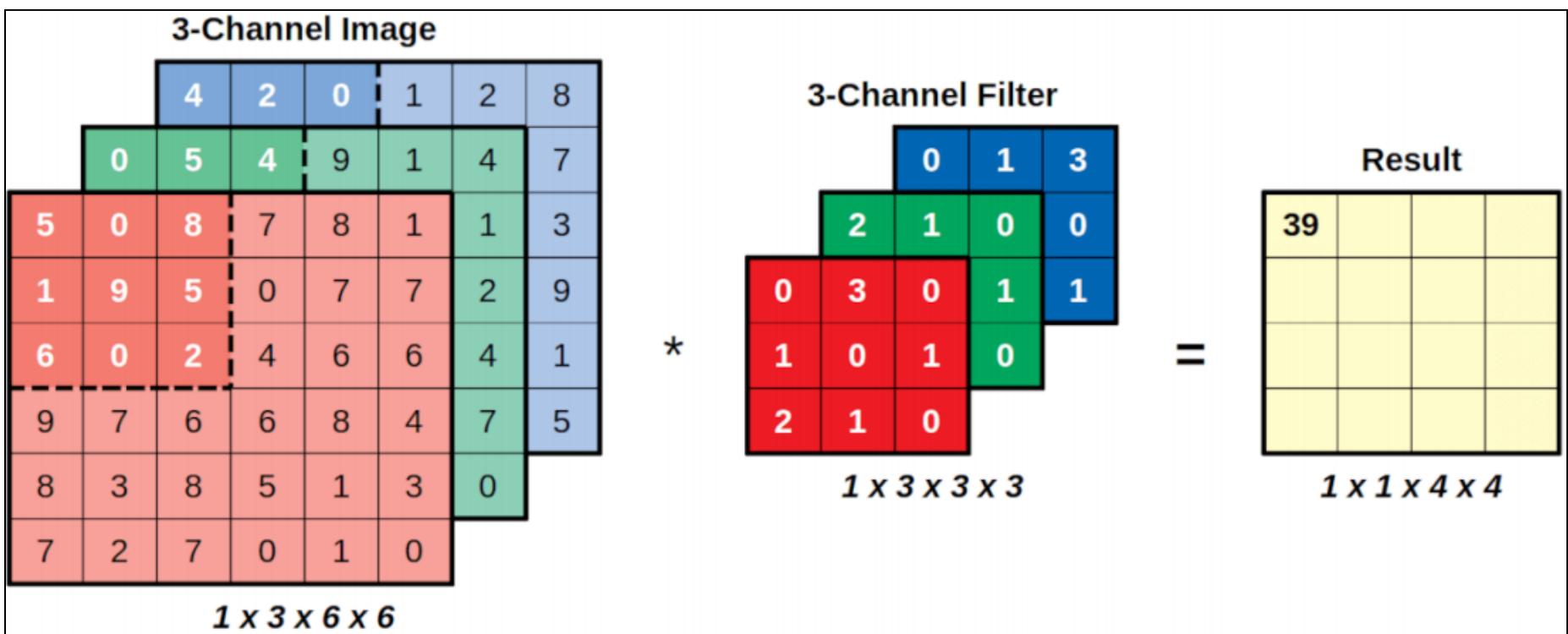
1
1
0
4
2
1
0
2
1

Multi-Channel Convolutions

- So far , there was a single channel image and a single channel filter.
- For three-channel images, a **three-channel filter** is used
- In Generalized 2D convolutions
 - Every filter has as many channels as the image it is convolving
- Convolving a three-channel filter over a three-channel image still produces a single value

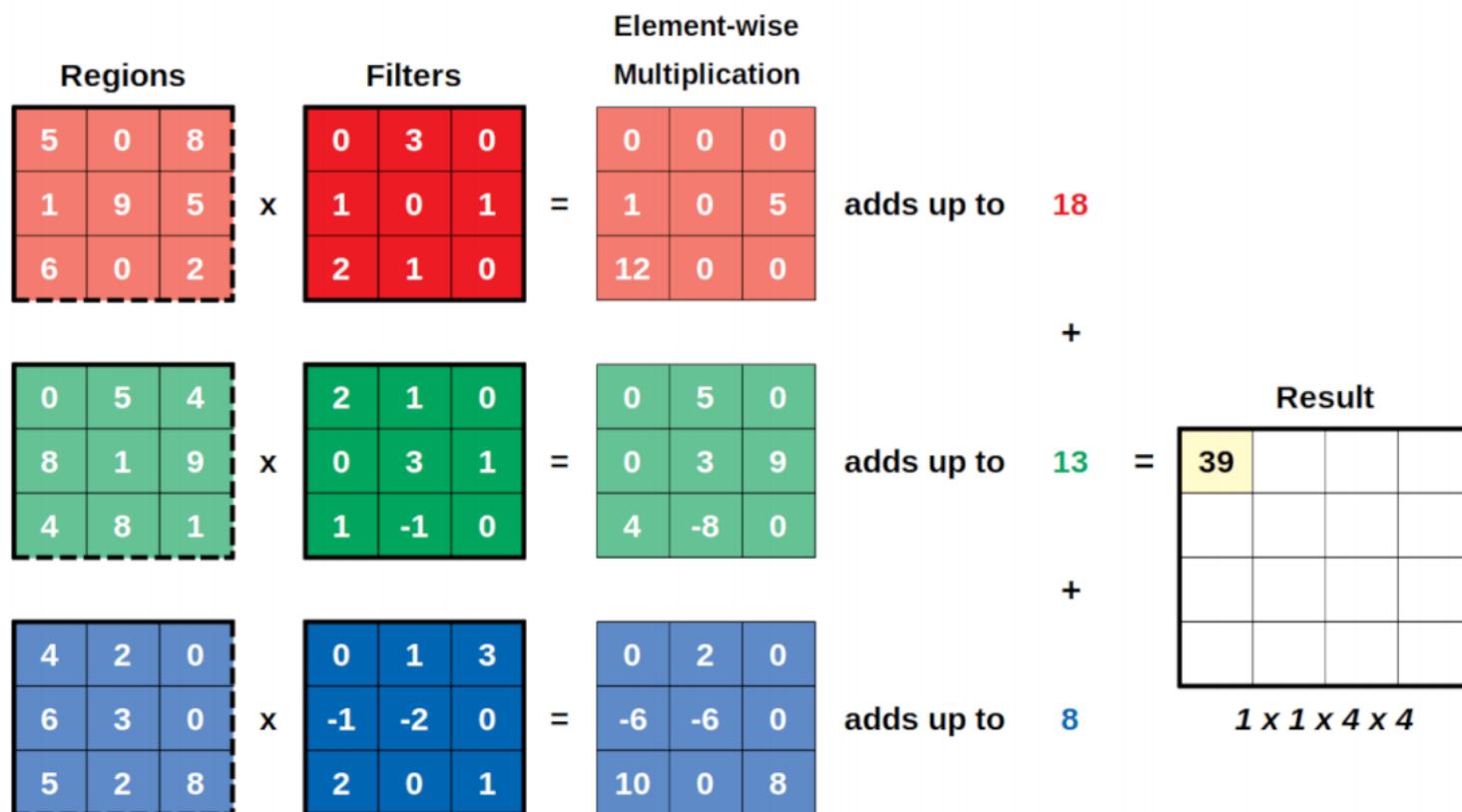
Multi-Channel Convolutions

- Think of it as performing three convolutions, each corresponding to the element-wise multiplication of the matching region/channel and filter/channel,
 - Resulting in three values, one for each channel.
 - **Adding up the results** for each channel produces the expected single value.



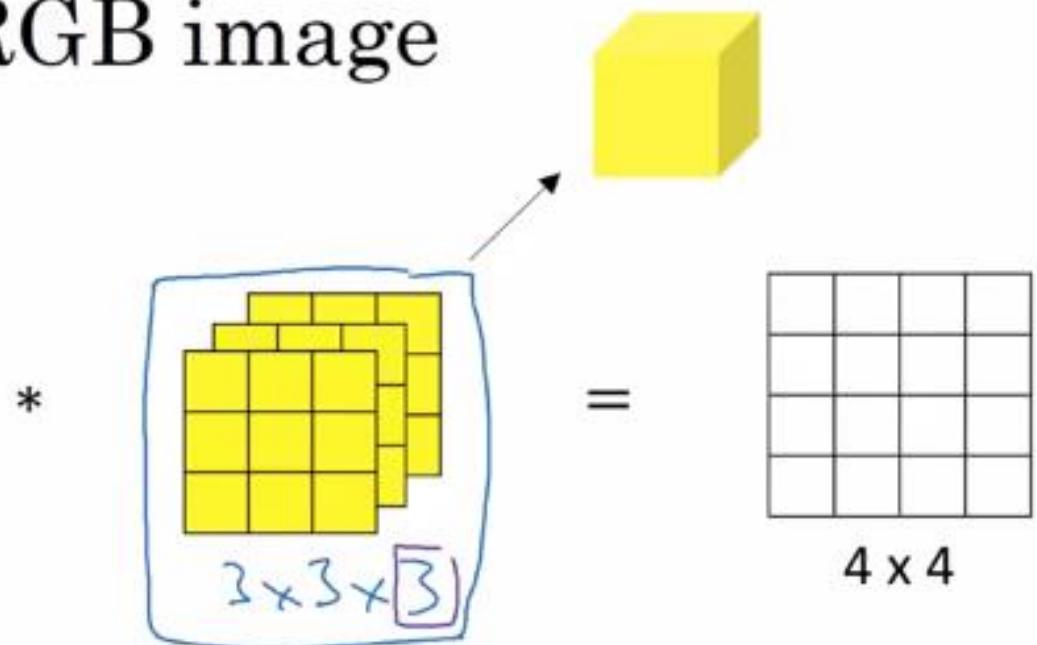
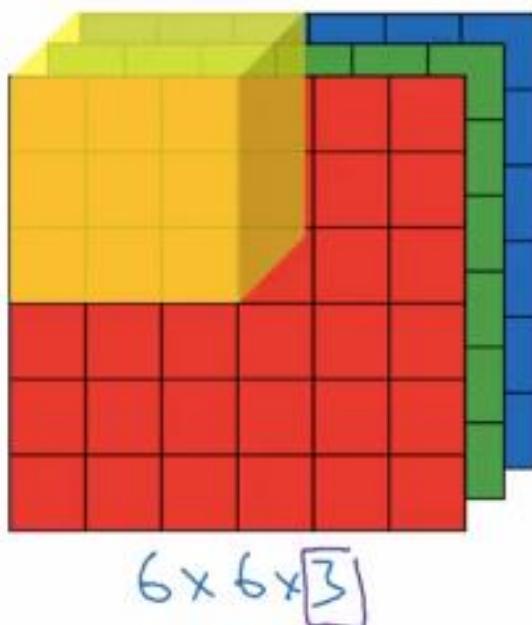
Multi-Channel Convolutions

- Think of it as performing three convolutions, each corresponding to the element-wise multiplication of the matching region/channel and filter/channel,
 - Resulting in three values, one for each channel.
 - Adding up the results for each channel produces the expected single value.



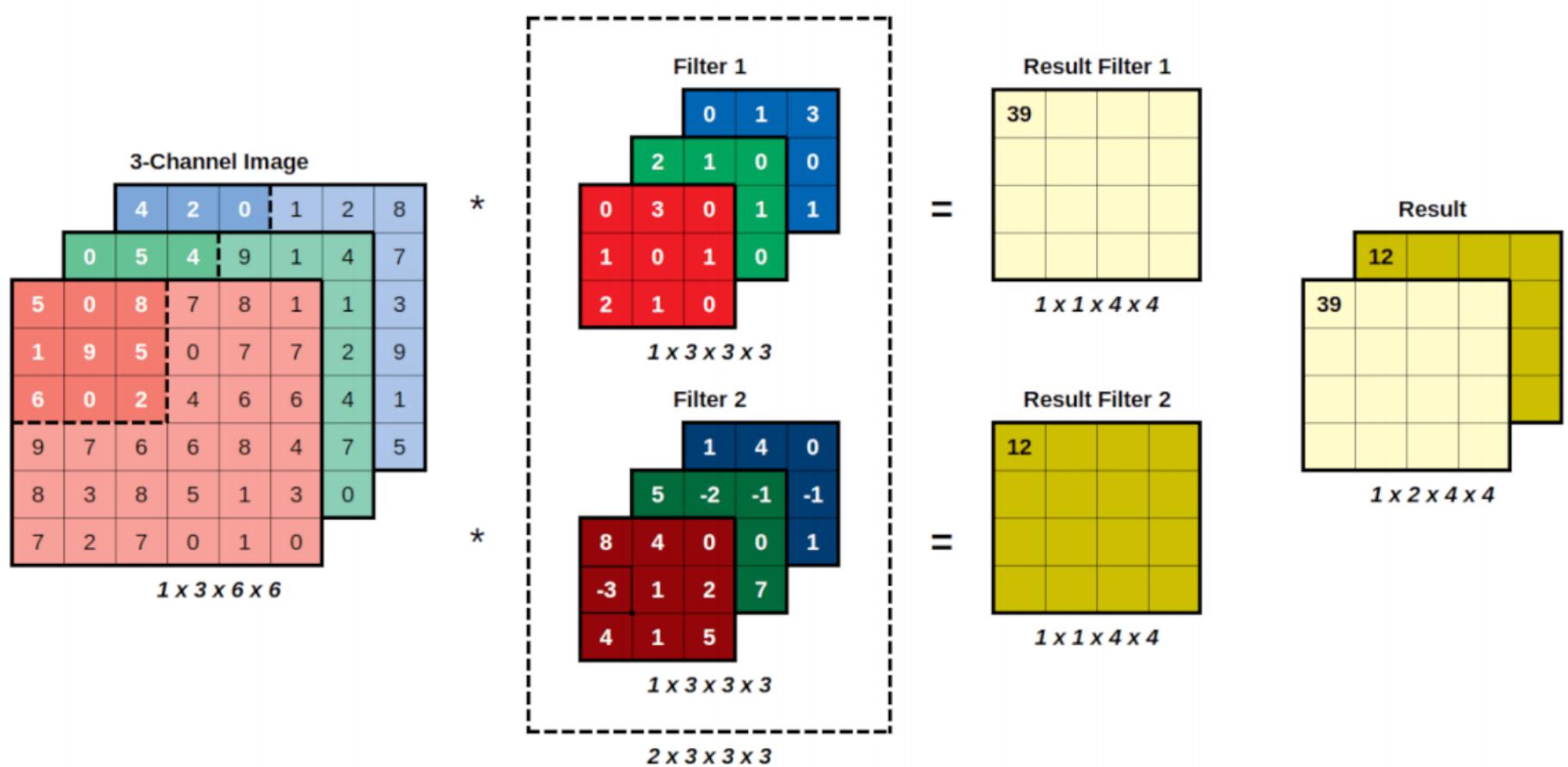
Multi-Channel Convolutions

Convolutions on RGB image



Multiple Filters

- The convolution produces as many channels as there are filters
- Example:
 - Two filters over three channels



PyTorch Conv2D Layer

- PyTorch's convolution module
 - nn.Conv2d
- many arguments
 - **in_channels**: number of channels of the input image
 - **out_channels**: number of channels produced by the convolution
 - **kernel_size**: size of the (square) convolution filter/kernel
 - **stride**: the size of the movement of the selected region
- There is no argument for the kernel/filter itself, there is only a **kernel_size** argument
 - **Learned by the module**
- It is possible to produce multiple channels as output

PyTorch Conv2D Layer

```
conv = nn.Conv2d(  
    in_channels=1, out_channels=1, kernel_size=3, stride=1  
)  
conv(image)
```

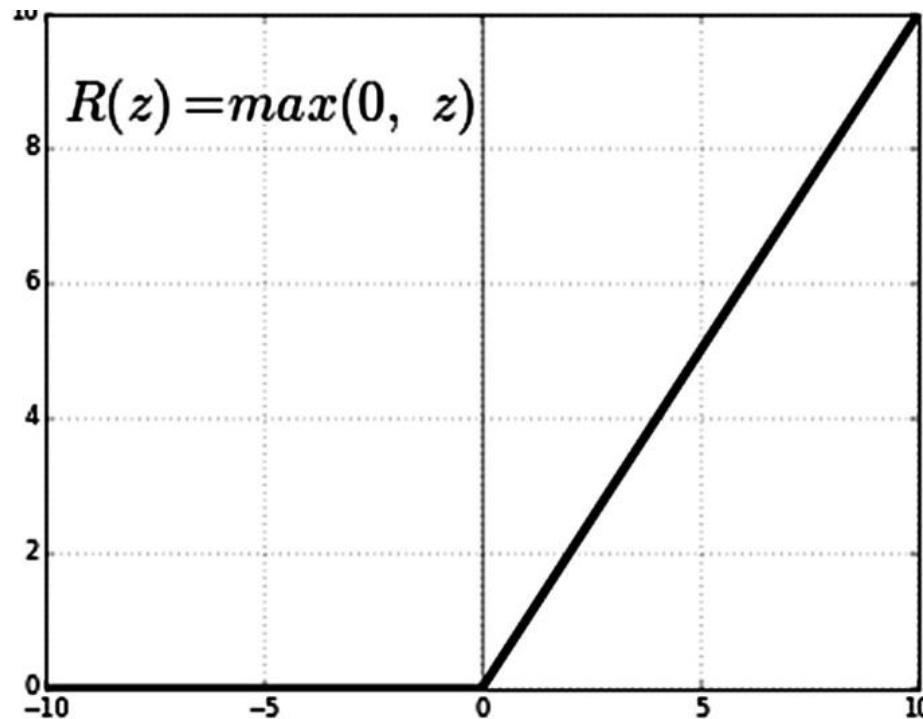
```
conv_multiple = nn.Conv2d(  
    in_channels=1, out_channels=2, kernel_size=3, stride=1  
)  
conv_multiple.weight
```

PyTorch Conv2D Layer

```
# Creates the convolution layers
self.conv1 = nn.Conv2d(
    in_channels=3,
    out_channels=n_filters,
    kernel_size=3
)
self.conv2 = nn.Conv2d(
    in_channels=n_filters,
    out_channels=n_filters,
    kernel_size=3
)
```

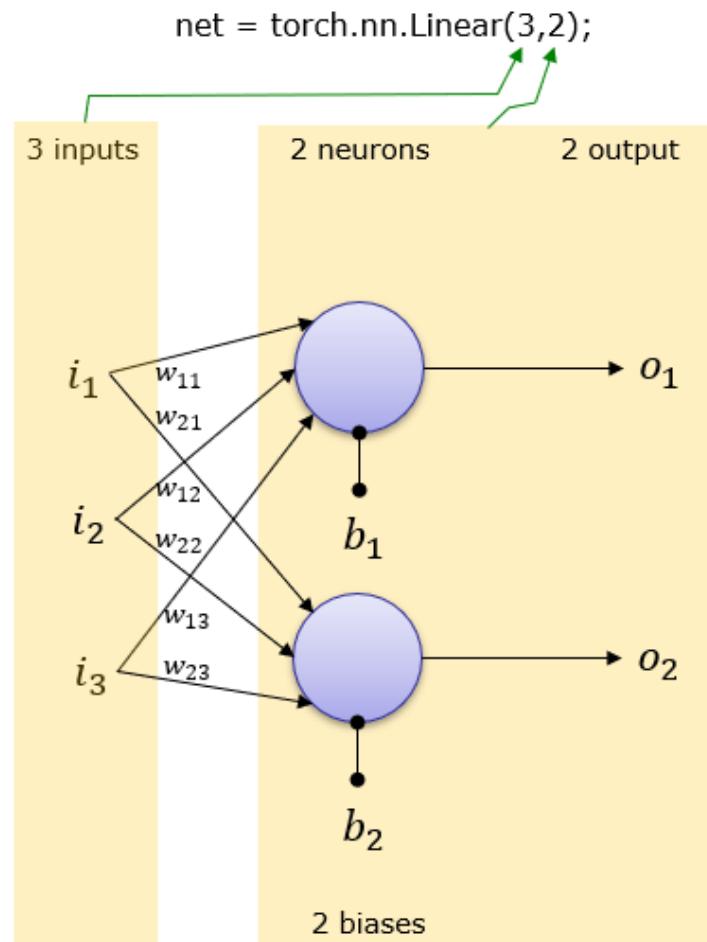
ReLU Activation layer

- Applies the rectified linear unit function element-wise
 - Input: can have any number of dimensions
 - Output: same shape as the input



Fully Connected Layer

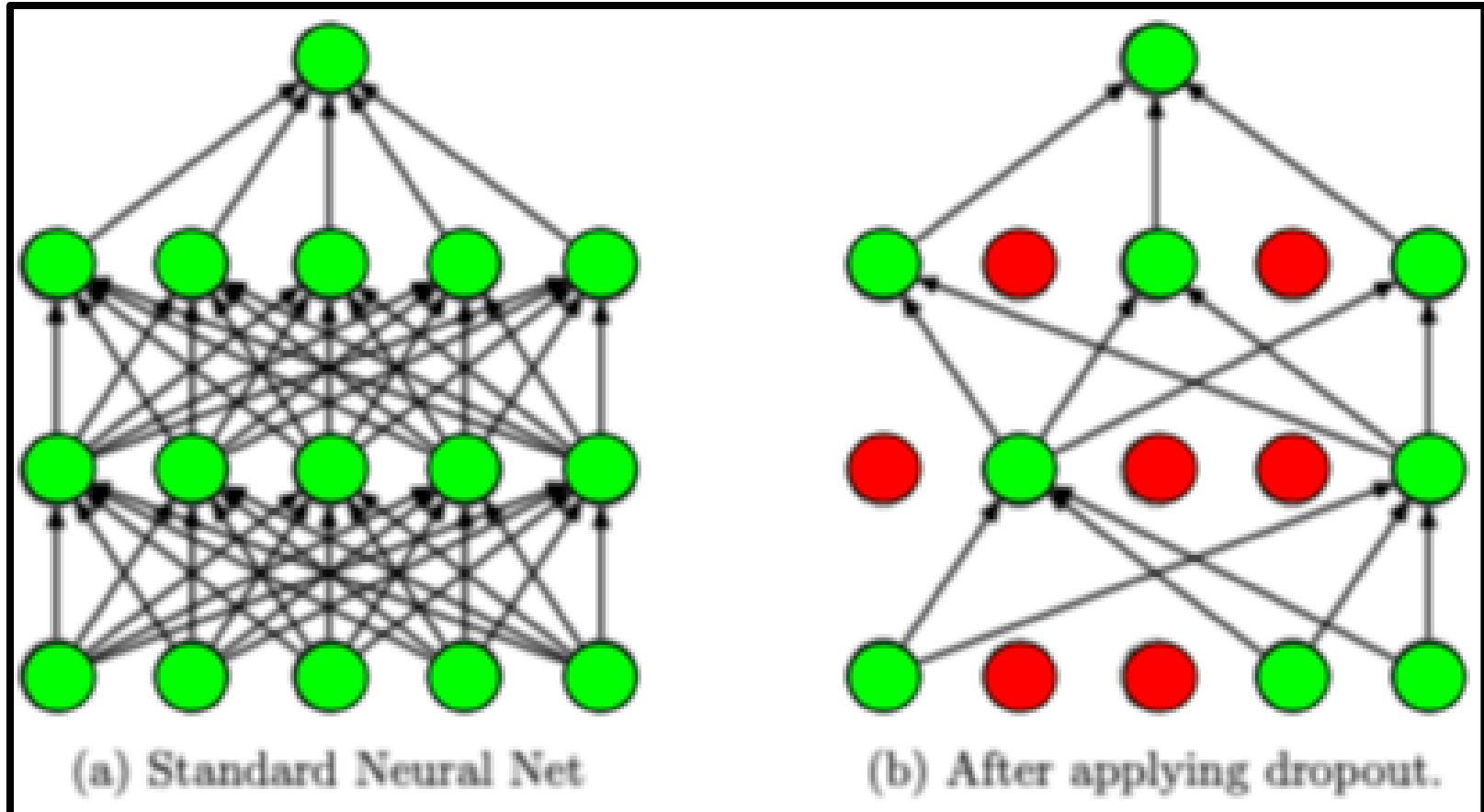
- The linear layer or Dense layer



Dropout Layer

- Dropout is very important deep learning layer
- It is used as a **regularizer**
 - Prevent **Overfitting** by forcing the model to find more than one way to achieve the target
 - Some features are randomly denied to model to achieve the target in a **different** way.
- Without dropout
 - Model may end up relying on a handful of features only because these features were found to be more relevant in the training set

Dropout Layer



(a) Standard Neural Net

(b) After applying dropout.

Dropout Layer (Example)

```
dropping_model = nn.Sequential(nn.Dropout(p=0.5))
```

```
spaced_points = torch.linspace(.1, 1.1, 11)
```

```
tensor([0.1000, 0.2000, 0.3000, 0.4000, 0.5000, 0.6000, 0.7000,  
       0.8000, 0.9000, 1.0000, 1.1000])
```

```
dropping_model.train()  
output_train = dropping_model(spaced_points)  
output_train
```

```
tensor([0.0000, 0.4000, 0.0000, 0.8000, 0.0000, 1.2000, 1.4000,  
       1.6000, 1.8000, 0.0000, 2.2000])
```

Dropout Facts

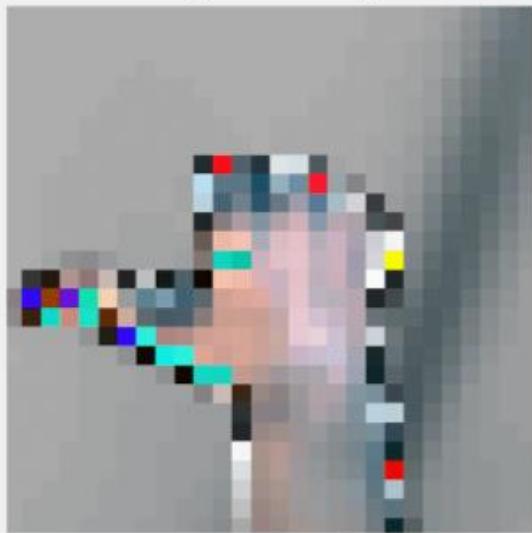
- Works on features not weights
- Works only when model is in training mode
- Drops feature according to the probability
- Rescale the remaining features

2D Dropout Layer

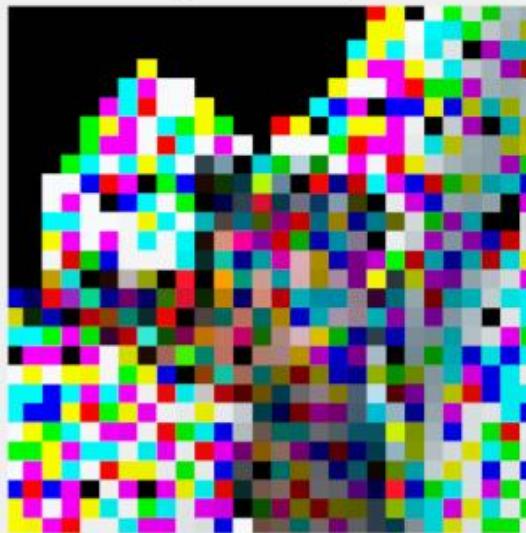
- used with convolutional layers
- It drops entire channels
- **nn.Dropout2d**
 - If a convolutional layer produces 10 channels
 - Two-dimensional dropout with a probability of 50% would drop five filters (on average),
 - Remaining filters would have all their pixel values left untouched

2D Dropout Layer

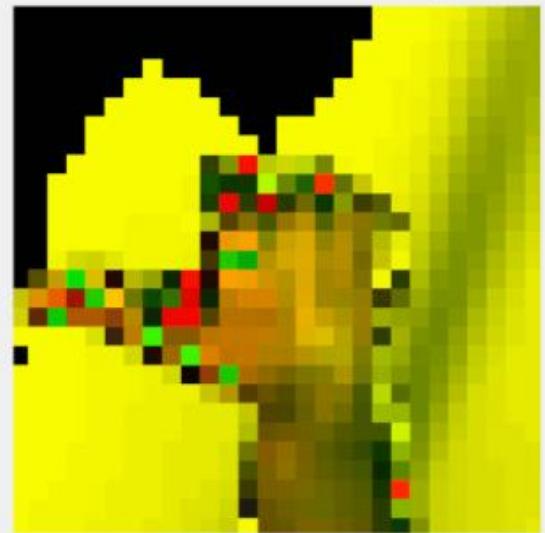
Original Image



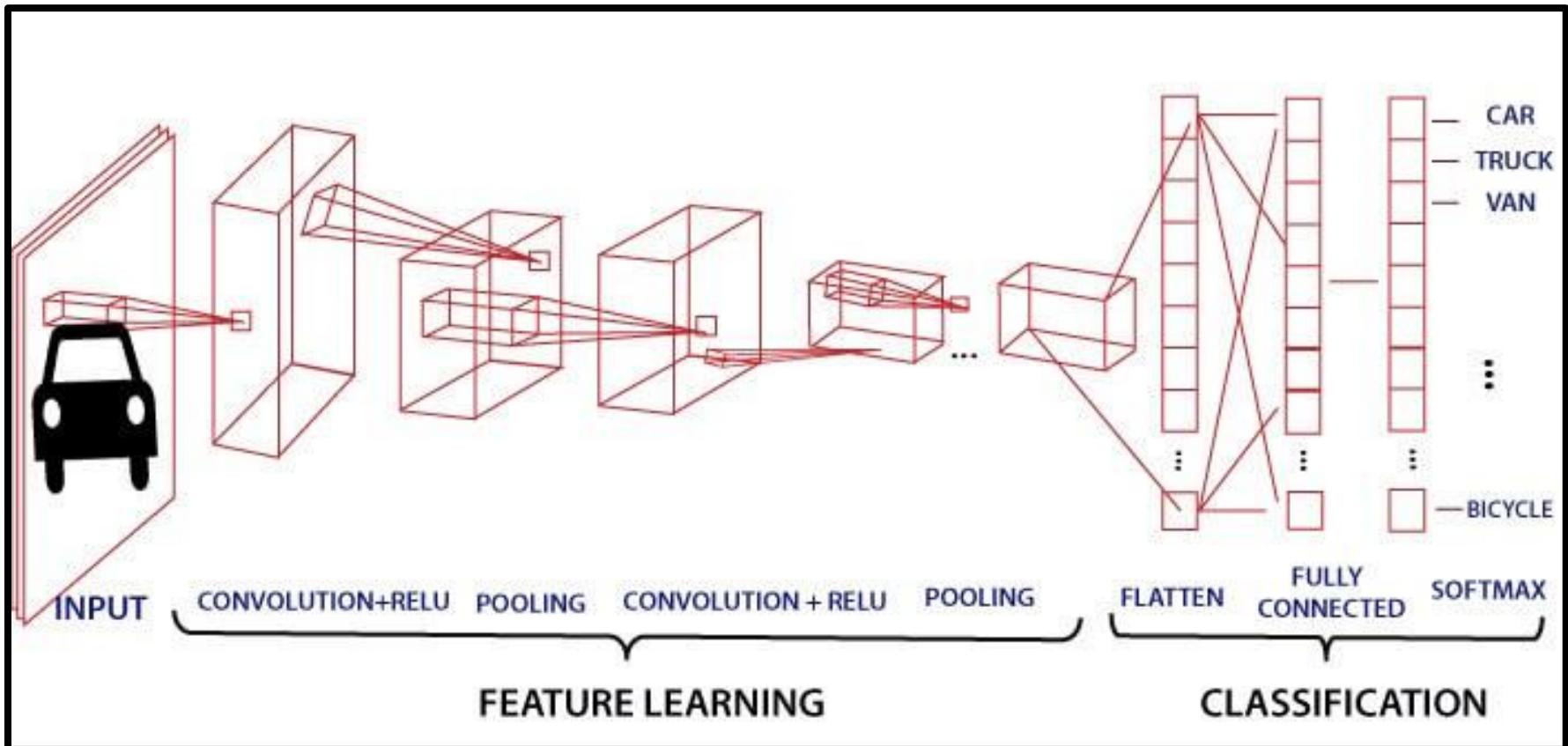
Regular Dropout



Two-Dimensional Dropout



CNNs / ConvNets Typical Architecture



Model Class

```
class RPSModel(nn.Module): #defining your custom model class
    def __init__(self, n_filters):
        super().__init__()

        self.n_filters = n_filters
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=self.n_filters,kernel_size=3)
        self.relu = nn.ReLU()
        self.maxpool = nn.MaxPool2d(kernel_size=2)
        self.conv2 = nn.Conv2d(in_channels=n_filters, out_channels=self.n_filters, kernel_size=3)

        self.flatten = nn.Flatten()
        self.linear1 = nn.Linear(in_features=self.n_filters*5*5, out_features=50)
        self.linear2 = nn.Linear(50, 3)
```

Model Class

```
def forward(self, X):
    #Featurizer
    x = self.conv1(X)
    x = self.relu(x)
    x= self.maxpool(x)
    x = self.conv2(x)
    x = self.relu(x)
    x = self.maxpool(x)
    #Classifier
    x = self.flatten(x)
    x = self.dropout(x)
    x = self.linear1(x)
    x = self.dropout(x)
    x = self.linear2(x)
    return x
```

Model Parameters

Layer (type:depth-idx)	Output Shape	Param #
Conv2d: 1-1	[-1, 16, 26, 26]	448
ReLU: 1-2	[-1, 16, 26, 26]	--
MaxPool2d: 1-3	[-1, 16, 13, 13]	--
Conv2d: 1-4	[-1, 16, 11, 11]	2,320
ReLU: 1-5	[-1, 16, 11, 11]	--
MaxPool2d: 1-6	[-1, 16, 5, 5]	--
Flatten: 1-7	[-1, 400]	--
Dropout: 1-8	[-1, 400]	--
Linear: 1-9	[-1, 50]	20,050
Dropout: 1-10	[-1, 50]	--
Linear: 1-11	[-1, 3]	153

Total params: 22,971

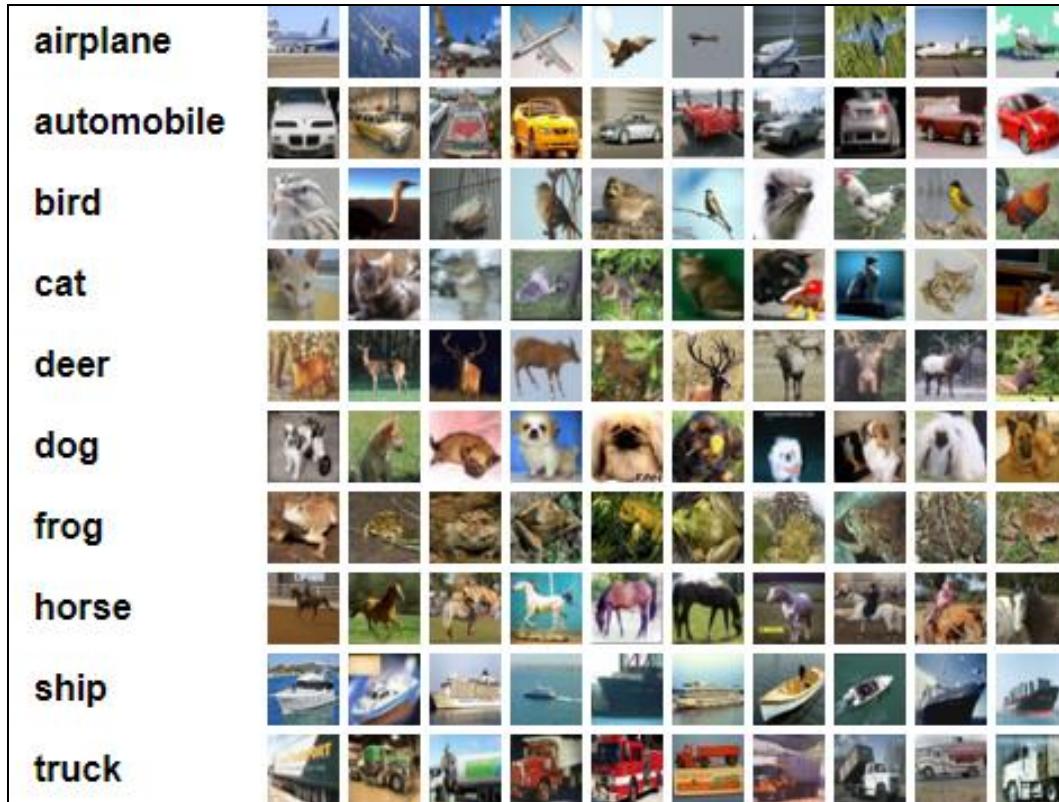
Trainable params: 22,971

Non-trainable params: 0

Total mult-adds (M): 0.59

Graded Home Task-5

- The **CIFAR-10** dataset consists of 60000 32x32 color images in 10 classes, with 6000 images per class
- The dataset is divided into five training batches and one test batch, each with 10000 images

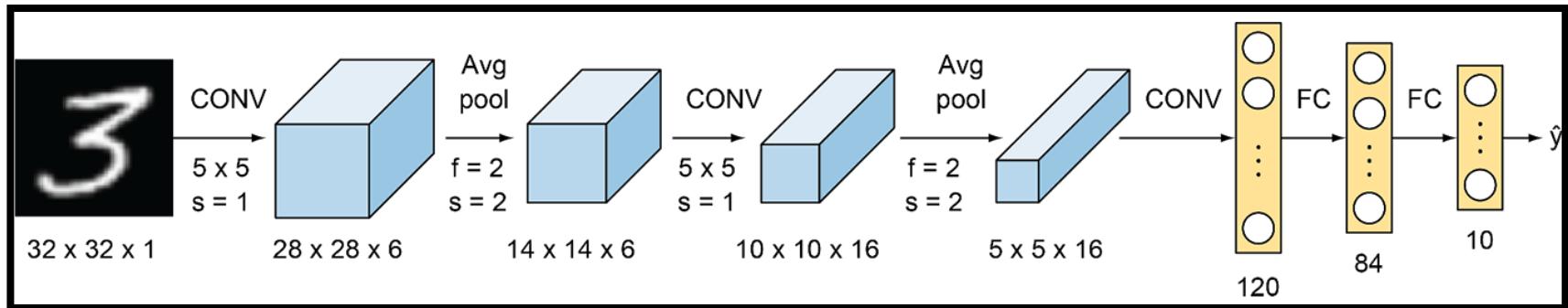


Graded Home Task-5

- Use `torchvision.datasets.CIFAR10` to load the train & validation datasets
- Create normalization transform using CIFAR10 means & std
- Create data augmentation transforms
 - `RandomCrop`
 - `RandomHorizontalFlip`
 - `RandomRotation`

Graded Home Task-5

- Create your own model class for LeNet or AlexNet
 - Use tanh or ReLU activations



- Report results after 50 epochs
 - Train/Val accuracy
 - Confusion matrix

QUIZ-4

- `img = torch.rand(1,3,300,400)`
- `model1= nn.Conv2d(in_channels=3, out_channels=64,kernel_size=3)`
- **# of parameters in model1 = ?**
- `features1 = model1(img)`
- **#Shape of features1=?**
- `model2= nn.Conv2d(in_channels=64, out_channels=128,kernel_size=5)`
- **# of parameters in model1 = ?**
- `features2 = model2(features1.to(device))`
- **#Shape of features2=?**

QUIZ-4 (Solution)

- `img = torch.rand(1,3,300,400)`
- `model1= nn.Conv2d(in_channels=3, out_channels=64,kernel_size=3)`
- **# of parameters in model1 = 1792**
- `features1 = model1(img)`
- **#Shape of features1=1 x 64 x 298 x 398**
- `model2= nn.Conv2d(in_channels=64, out_channels=128,kernel_size=5)`
- **# of parameters in model1 = 204,928**
- `features2 = model2(features1.to(device))`
- **#Shape of features2=1 x 128 x 294 x 394**

TRANSFER LEARNING

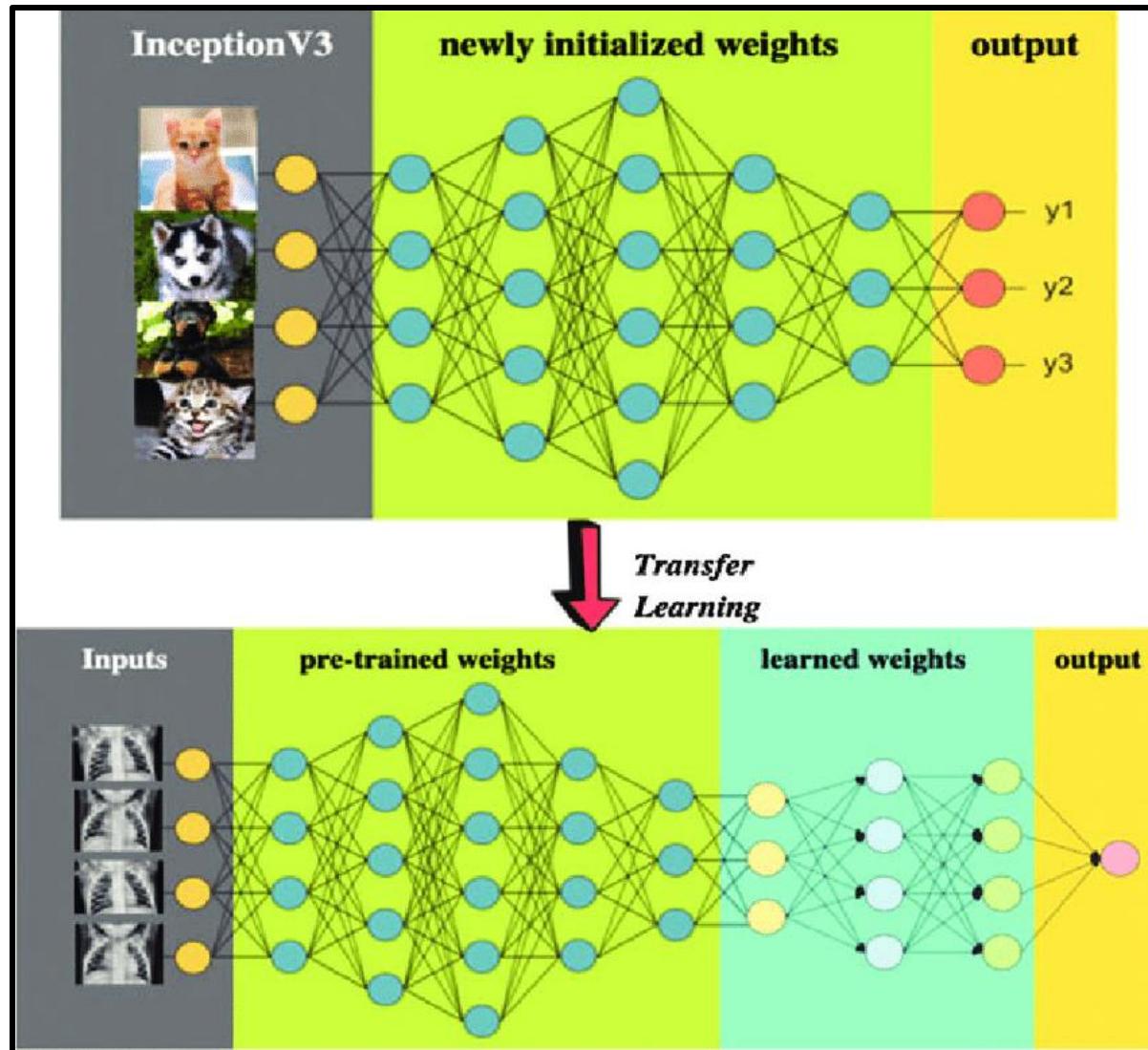
Introduction

- Really powerful fancy CNN models have tens of convolutional blocks
 - Have other **advanced architectural concepts**
 - Have many **million parameters**
 - Require **humongous amounts of data**
 - Thousands of (**expensive**) GPU-hours for training
- So can we use/train them?

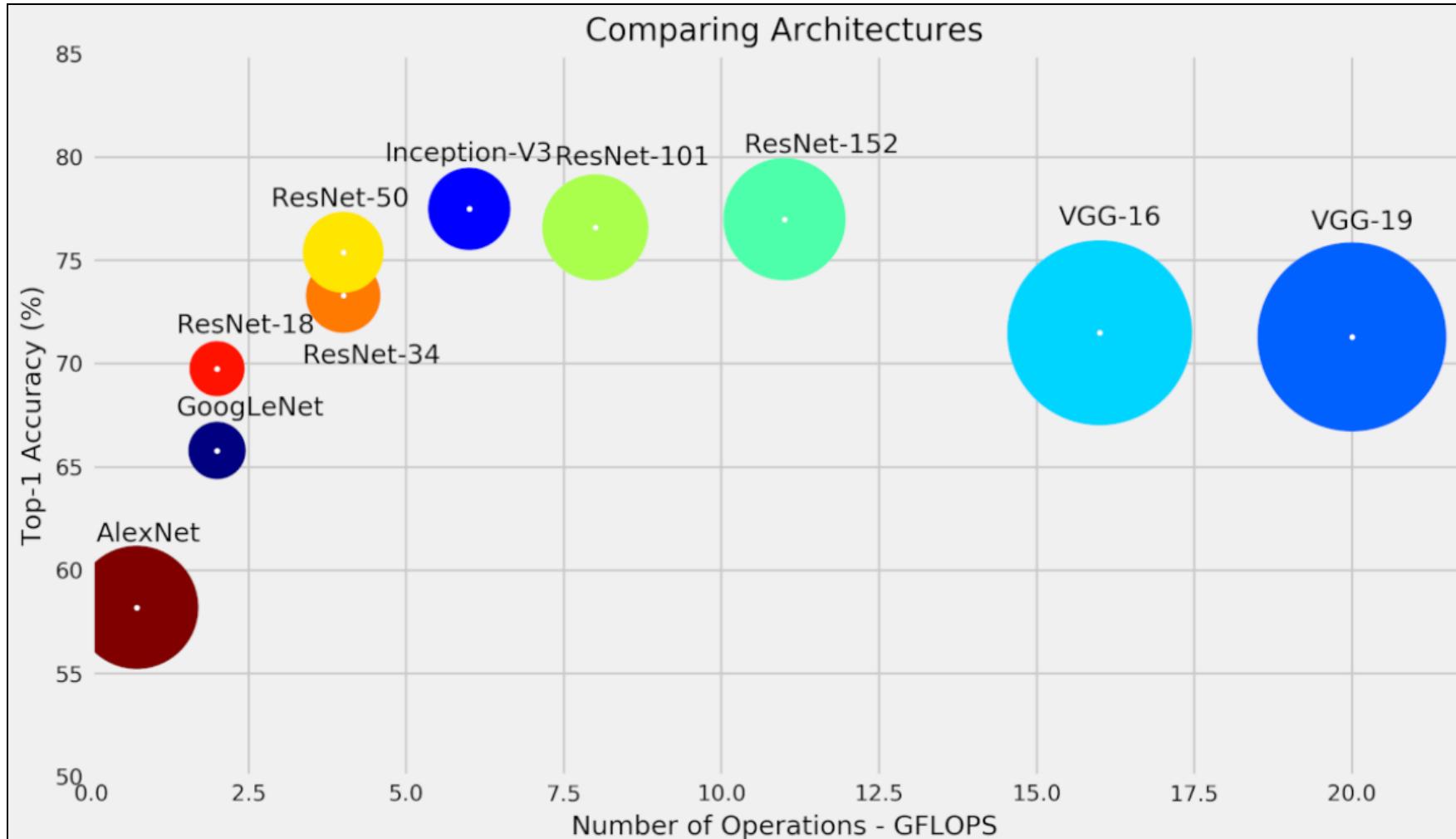
Introduction

- Transfer Learning Process:
 - First, big tech company, which has access to virtually **infinite amounts of data** and **computing power**, develops and trains a huge model for their own purpose.
 - Next, once it is trained, its architecture and the corresponding trained weights (**the pre-trained model**) are released
 - Finally, everyone else can use these weights as a starting point and **fine-tune** it further for a **different (but similar) purpose**

Introduction



Large CNN Models



Which one would you choose for transfer learning?

Some Advanced Concepts

- Adaptive Pooling

```
result1 = F.adaptive_avg_pool2d(  
    torch.randn(16, 32, 32), output_size=(6, 6)  
)  
result2 = F.adaptive_avg_pool2d(  
    torch.randn(16, 12, 12), output_size=(6, 6)  
)  
result1.shape, result2.shape
```

Some Advanced Concepts

- Auxiliary classifiers (Side Heads)

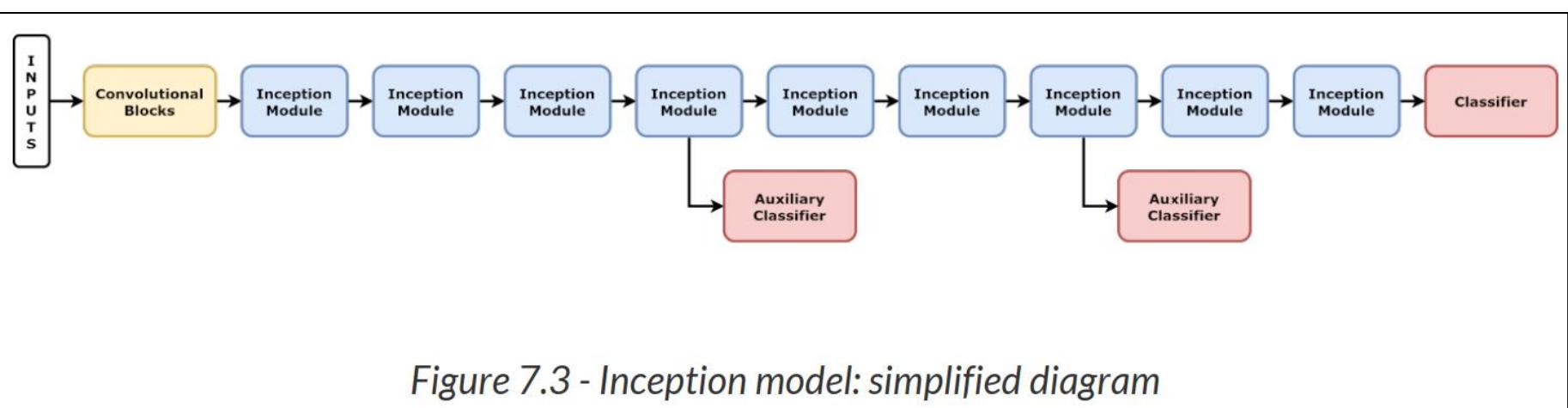


Figure 7.3 - Inception model: simplified diagram

Some Advanced Concepts

- 1×1 Convolutions (**Can replace Linear Layer**)

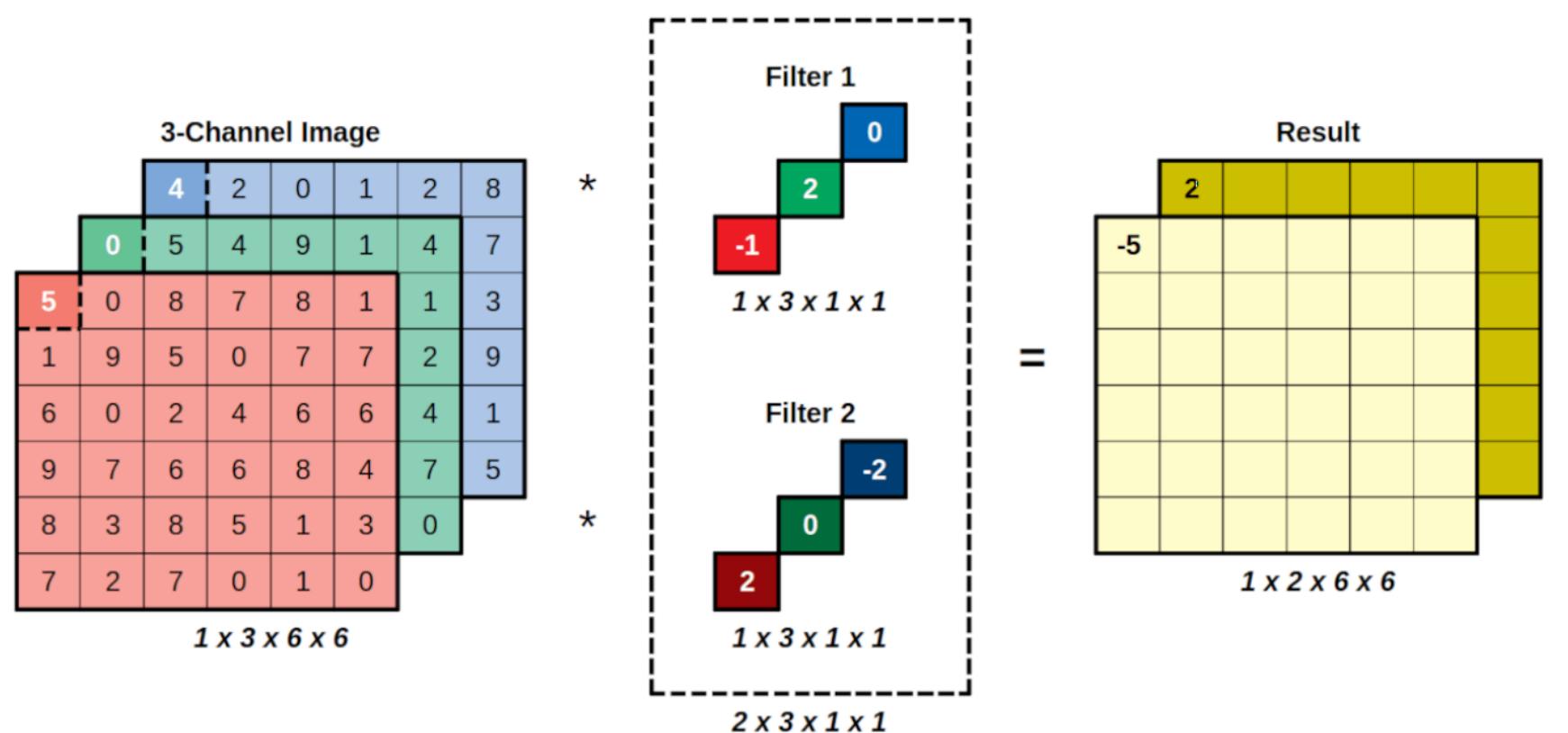
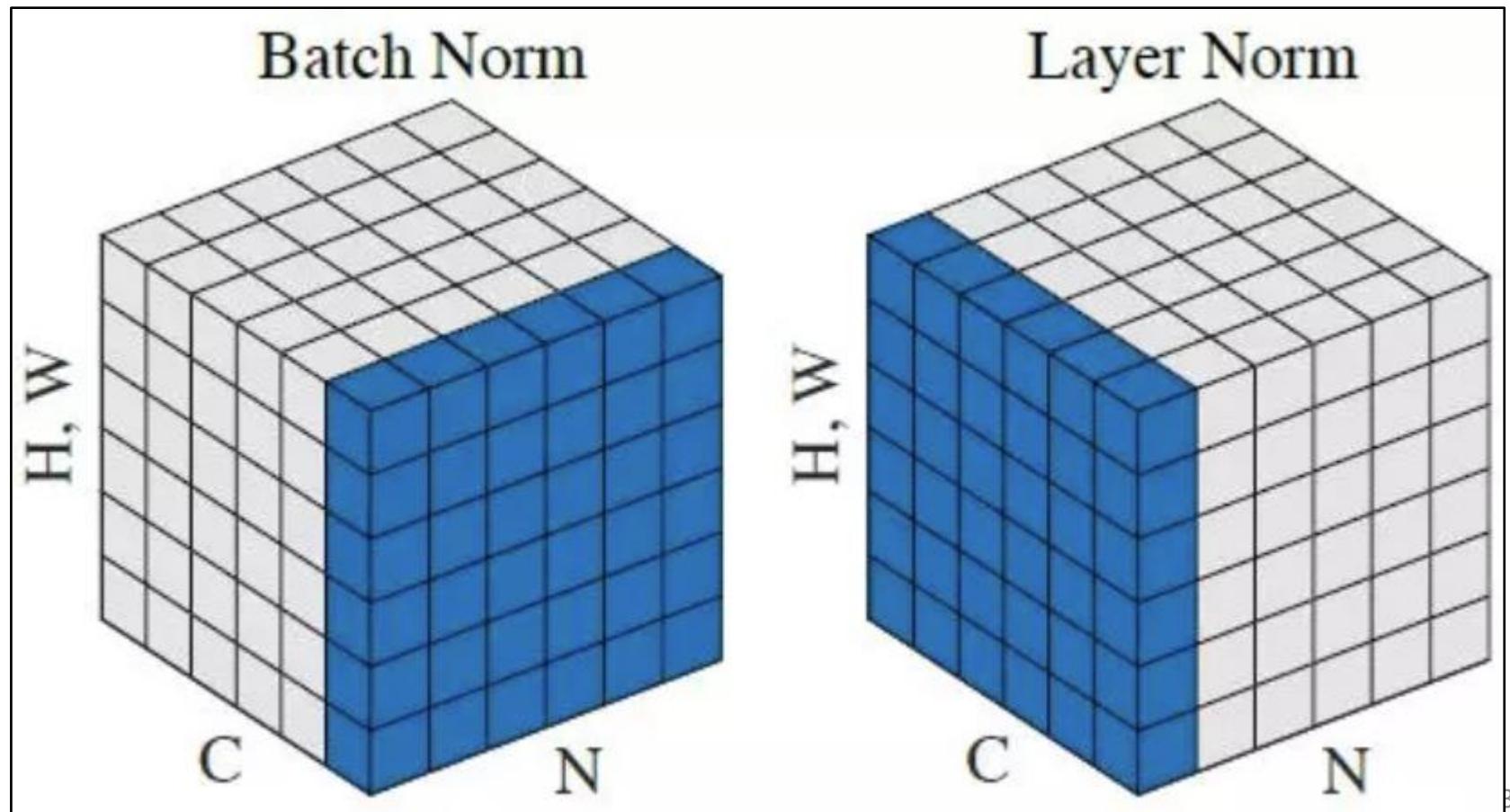


Figure 7.4 - 1×1 Convolution

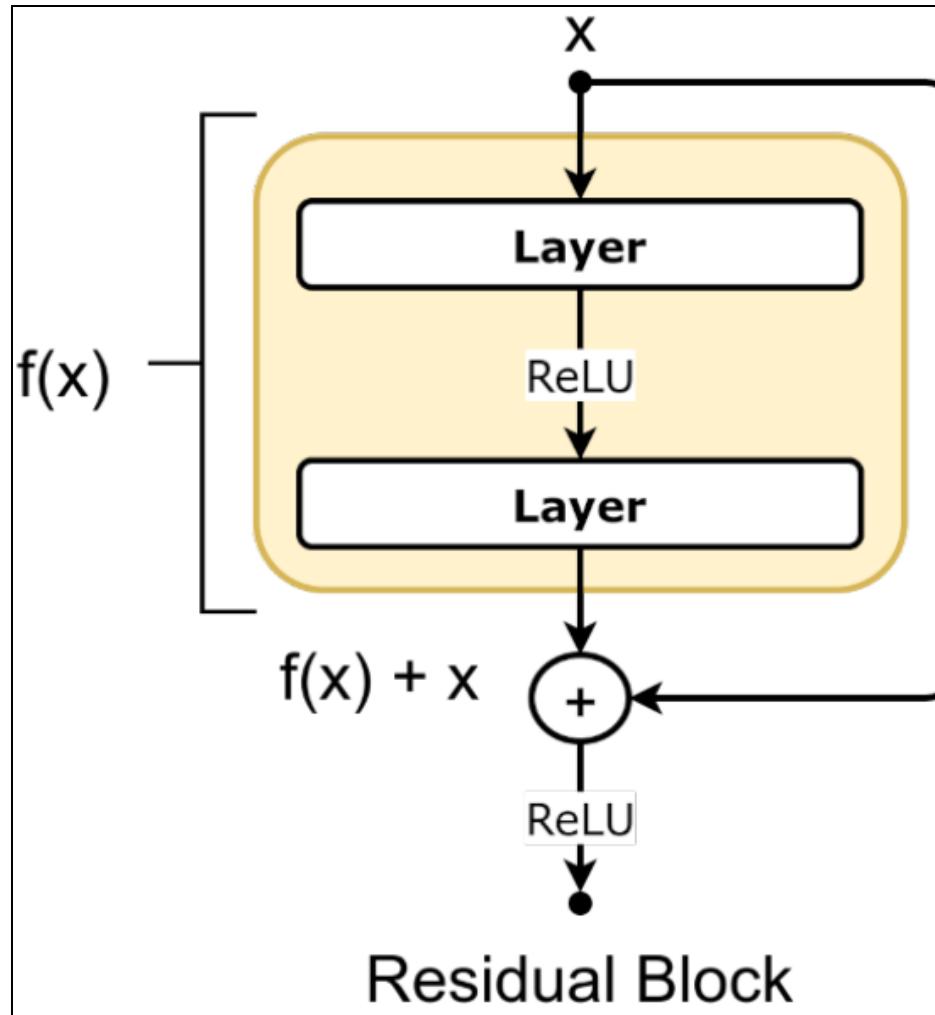
Some Advanced Concepts

- Batch Normalization(**Accelerate the training**)
 - Standardize input after activation



Some Advanced Concepts

- **Residual Connections** (skip connections)
 - More paths of data reaching multiple layers

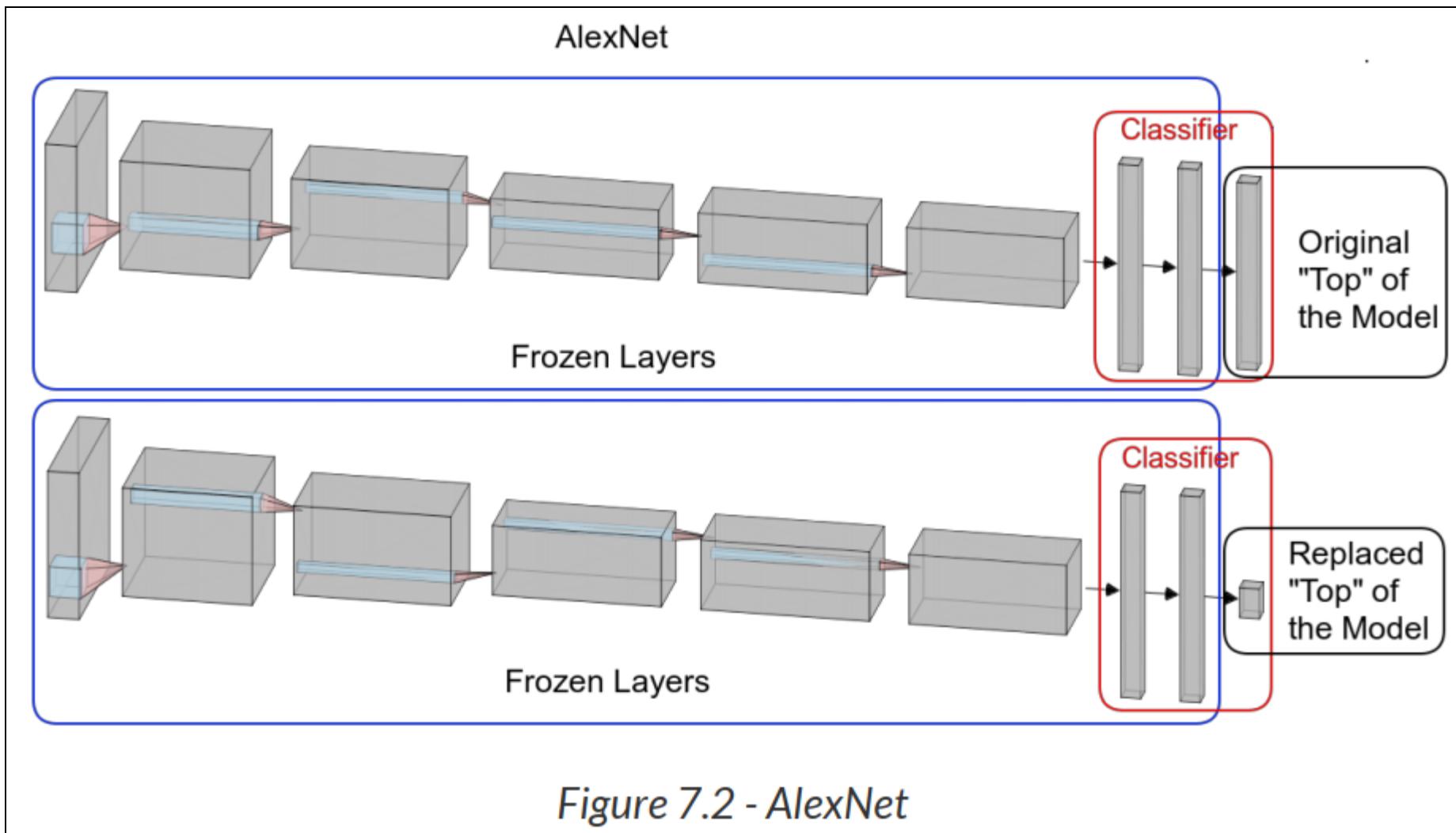


RPS Image Classification

TRANSFER LEARNING WITH ALEXNET

How to do?

From ImageNet **1000 Classes** to RPS **3 Classes**



Step-1 Loading Pre-Trained Model

```
from torchvision.models import alexnet
from torchvision.models.alexnet import model_urls
from torchvision.models.utils import load_state_dict_from_url
```

```
#loading a pretrained model
model = alexnet(pretrained=False)
url = model_urls['alexnet']
state_dict = load_state_dict_from_url(url, model_dir='pretrained', progress=True)
model.load_state_dict(state_dict)
print(model)
summary(model, (3,224,224))
```

- If **pretrained=True** model can be loaded with just one line of code
- But you have to load the weights manually
- If the model weights are from some **third party**

Step-1 Loading Pre-Trained Model

Layer (type:depth-idx)	Output Shape	Param #
Sequential: 1-1	[-1, 256, 6, 6]	--
└ Conv2d: 2-1	[-1, 64, 55, 55]	23,296
└ ReLU: 2-2	[-1, 64, 55, 55]	--
└ MaxPool2d: 2-3	[-1, 64, 27, 27]	--
└ Conv2d: 2-4	[-1, 192, 27, 27]	307,392
└ ReLU: 2-5	[-1, 192, 27, 27]	--
└ MaxPool2d: 2-6	[-1, 192, 13, 13]	--
└ Conv2d: 2-7	[-1, 384, 13, 13]	663,936
└ ReLU: 2-8	[-1, 384, 13, 13]	--
└ Conv2d: 2-9	[-1, 256, 13, 13]	884,992
└ ReLU: 2-10	[-1, 256, 13, 13]	--
└ Conv2d: 2-11	[-1, 256, 13, 13]	590,080
└ ReLU: 2-12	[-1, 256, 13, 13]	--
└ MaxPool2d: 2-13	[-1, 256, 6, 6]	--
AdaptiveAvgPool2d: 1-2	[-1, 256, 6, 6]	--
Sequential: 1-3	[-1, 1000]	--
└ Dropout: 2-14	[-1, 9216]	--
└ Linear: 2-15	[-1, 4096]	37,752,832
└ ReLU: 2-16	[-1, 4096]	--
└ Dropout: 2-17	[-1, 4096]	--
└ Linear: 2-18	[-1, 4096]	16,781,312
└ ReLU: 2-19	[-1, 4096]	--
└ Linear: 2-20	[-1, 1000]	4,097,000
Total params:	61,100,840	
Trainable params:	61,100,840	
Non-trainable params:	0	
Total mult-adds (M):	775.28	

Step-2 Model Freezing

- Freezing the model means it won't learn anymore , that is, its parameters/weights will not be updated anymore
 - Remember **requires_grad**
 - **Can unfreeze/replace whole top also**

```
for parameter in model.parameters():
    parameter.requires_grad = False

model.classifier[6] = nn.Linear(4096, 3)
```

```
for name, param in model.named_parameters():
    if param.requires_grad == True:
        print(name)
```

```
classifier.6.weight  
classifier.6.bias
```

```
num_params = sum(param.numel() for param in model.parameters())
num_trainable_params = sum(param.numel() for param in model.parameters() if param.requires_grad)
```

Step-2 Model Freezing

- Which layer to replace for which model?

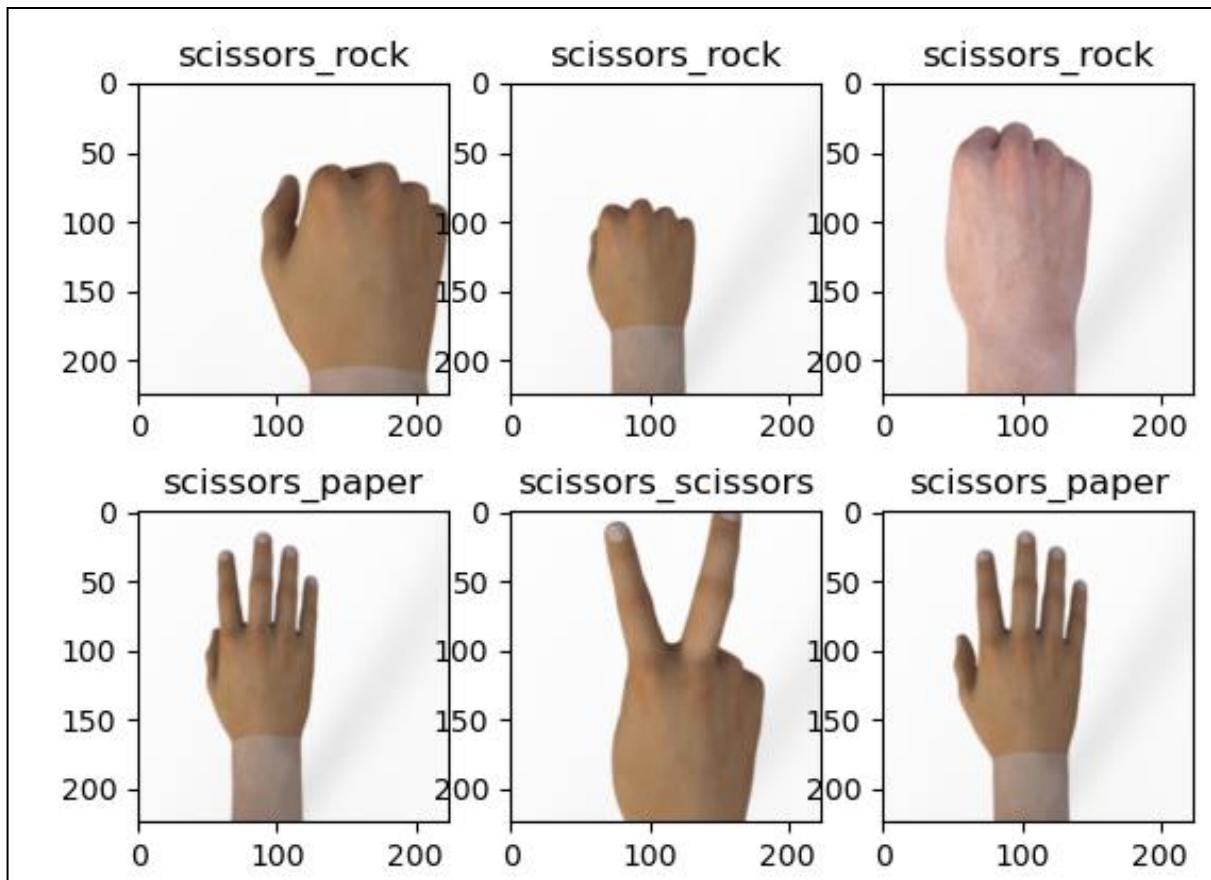
Model	Size	Classifier Layer(s)	Replacement Layer(s)
AlexNet	224	model.classifier[6]	nn.Linear(4096,num_classes)
VGG	224	model.classifier[6]	nn.Linear(4096,num_classes)
InceptionV3	299	model.fc	nn.Linear(2048,num_classes)
		model.AuxLogits.fc	nn.Linear(768,num_classes)
ResNet	224	model.fc	nn.Linear(512,num_classes)
DenseNet	224	model.classifier	nn.Linear(1024,num_classes)
SqueezeNet	224	model.classifier[1]	nn.Conv2d(512,num_classes,kernel_size=1,stride=1)

Step-3 Standardization Parameters

- Use ImageNet statistics (because pretrained model is trained on ImageNet)

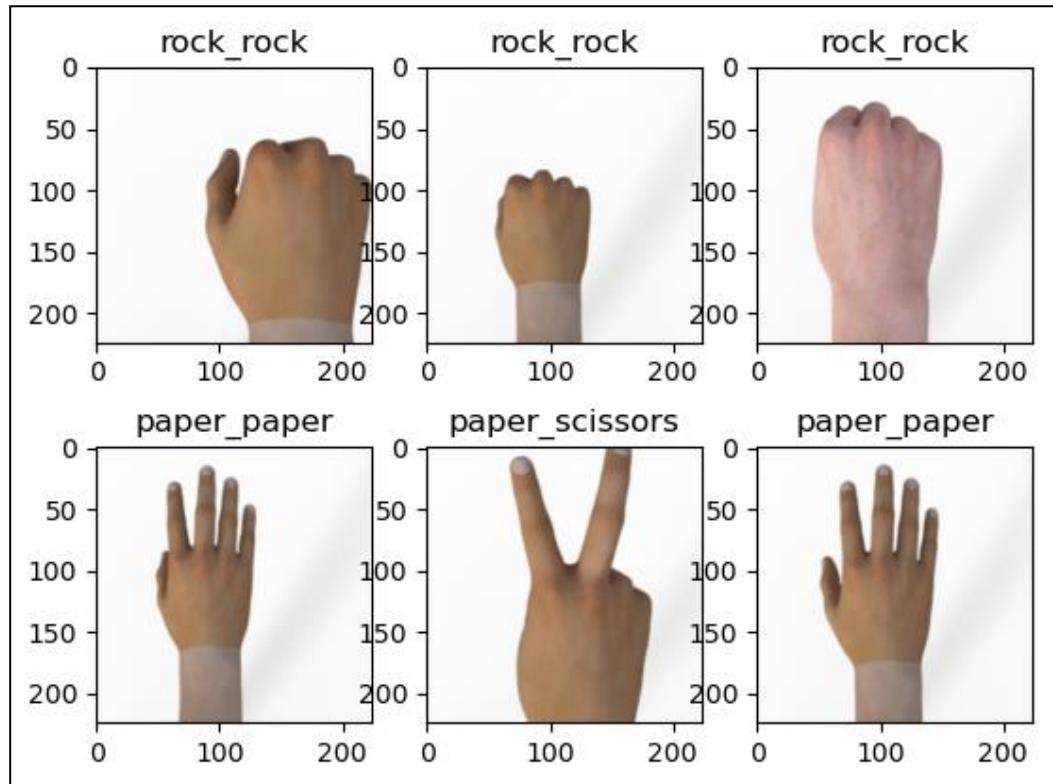
```
#transforms
normTransform=Normalize(mean=torch.Tensor([0.485, 0.456, 0.406]),std=torch.Tensor([0.229, 0.224, 0.225]))
transform=Compose([Resize(256),CenterCrop(224), ToTensor(), normTransform])
#Datasets & Loaders
trainSet=ImageFolder(root='dataset/rps/',transform=transform)
train_loader=DataLoader(trainSet, batch_size=128, shuffle=True)
valSet=ImageFolder(root='dataset/rps-test-set/',transform=transform)
val_loader=DataLoader(valSet, batch_size=128)
```

Results before training



Training Results

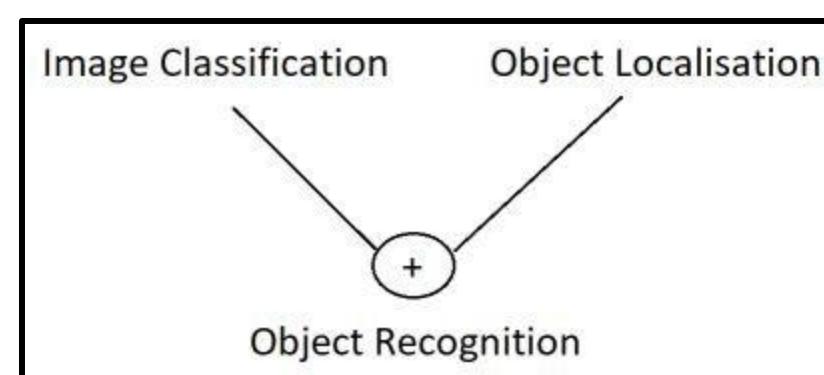
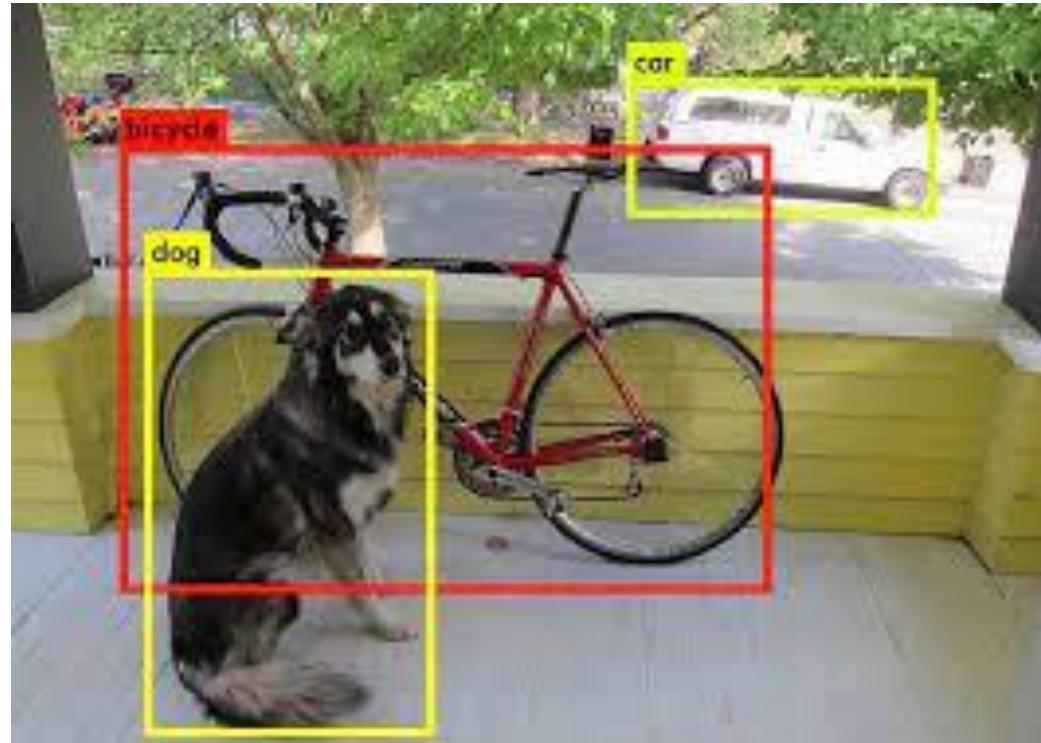
```
[[831  1   8]
 [ 2 837  1]
 [ 8   1 831]]
epoch= 19, accuracyTrain= 0.9916666666666667,
```



OBJECT DETECTION

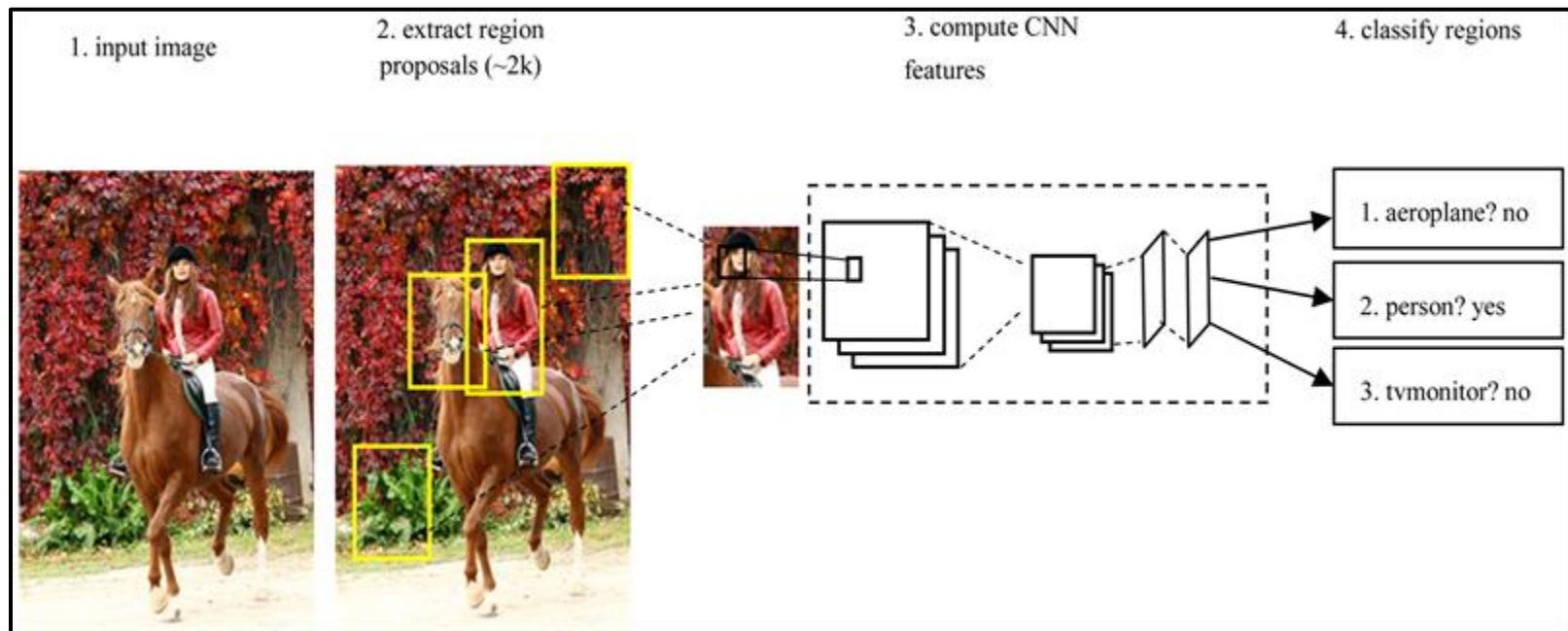
Introduction

- Object Detection (Object Recognition)
 - Bounding Box — ROI (Region of interest)



Types of Detectors

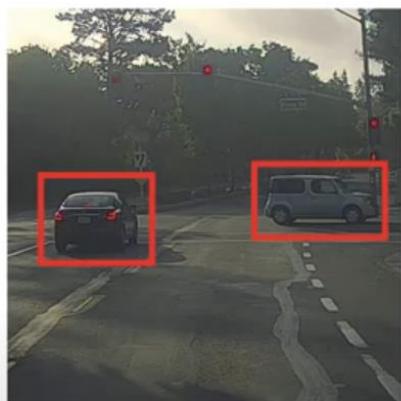
- Multi-Stage Detectors
 - RCNN, Fast RCNN, Faster RCNN



Can you see problems with this approach?

Sliding Window Detector (Stage-1)

Training set:

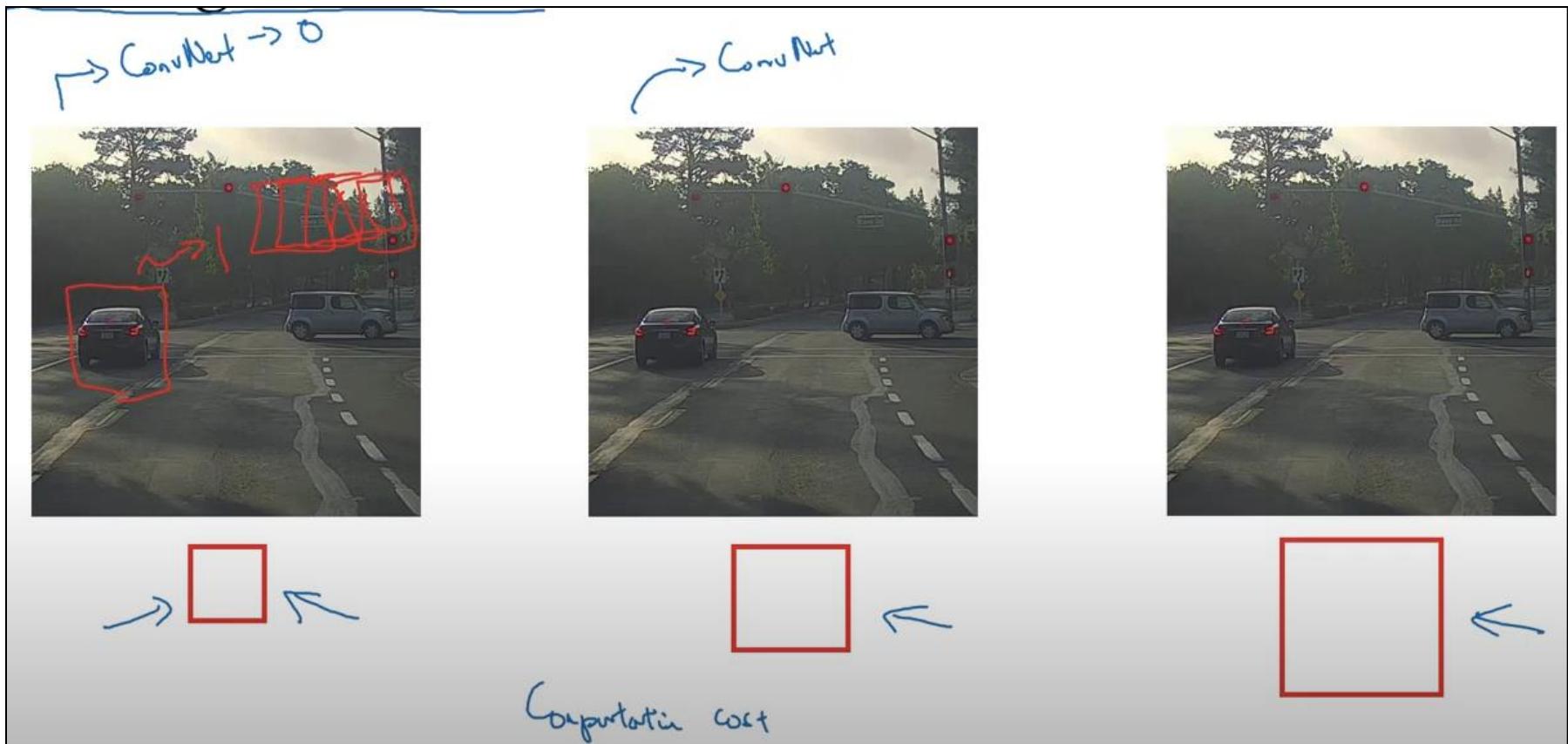


x	y
	1
	1
	1
	0
	0

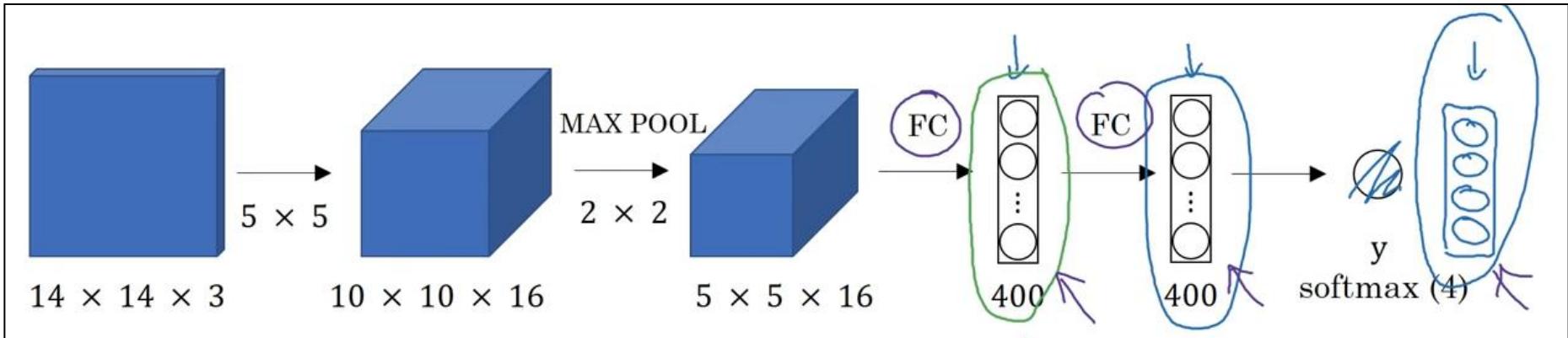


→ ConvNet → y

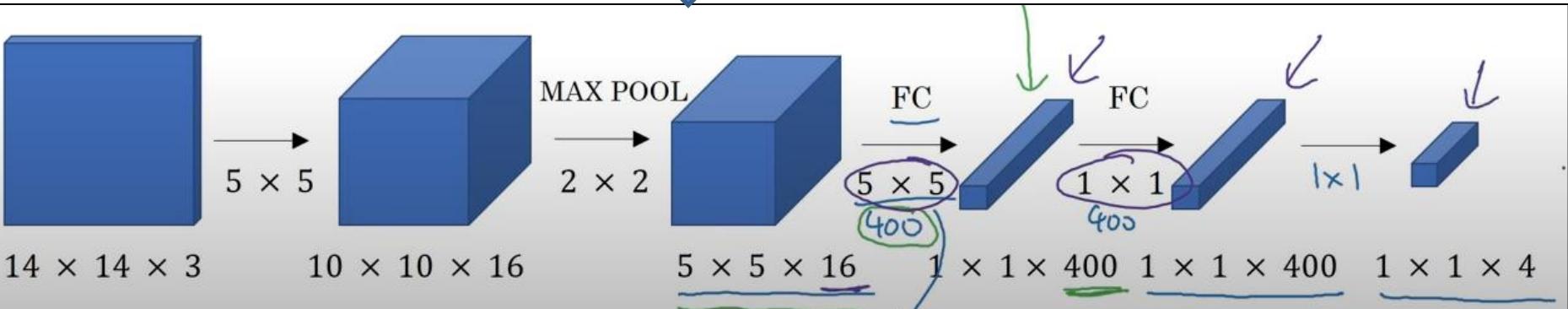
Sliding Window Detector (Stage-2)



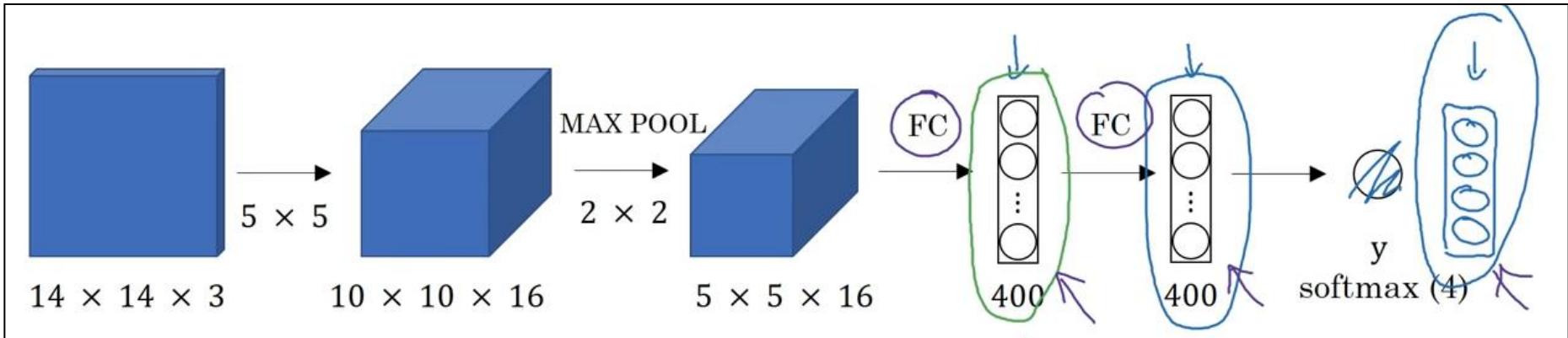
CNN Implementation of Sliding Window



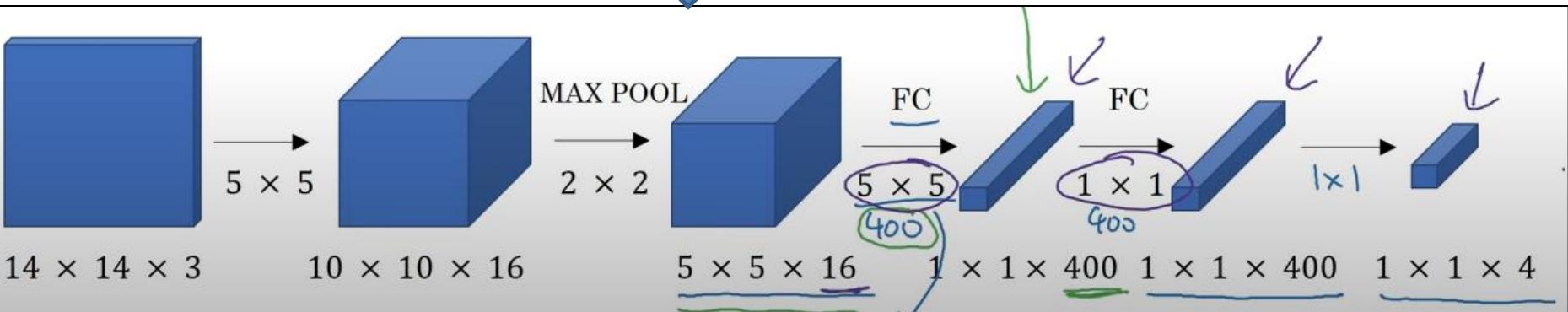
Power of 1×1 convolutions



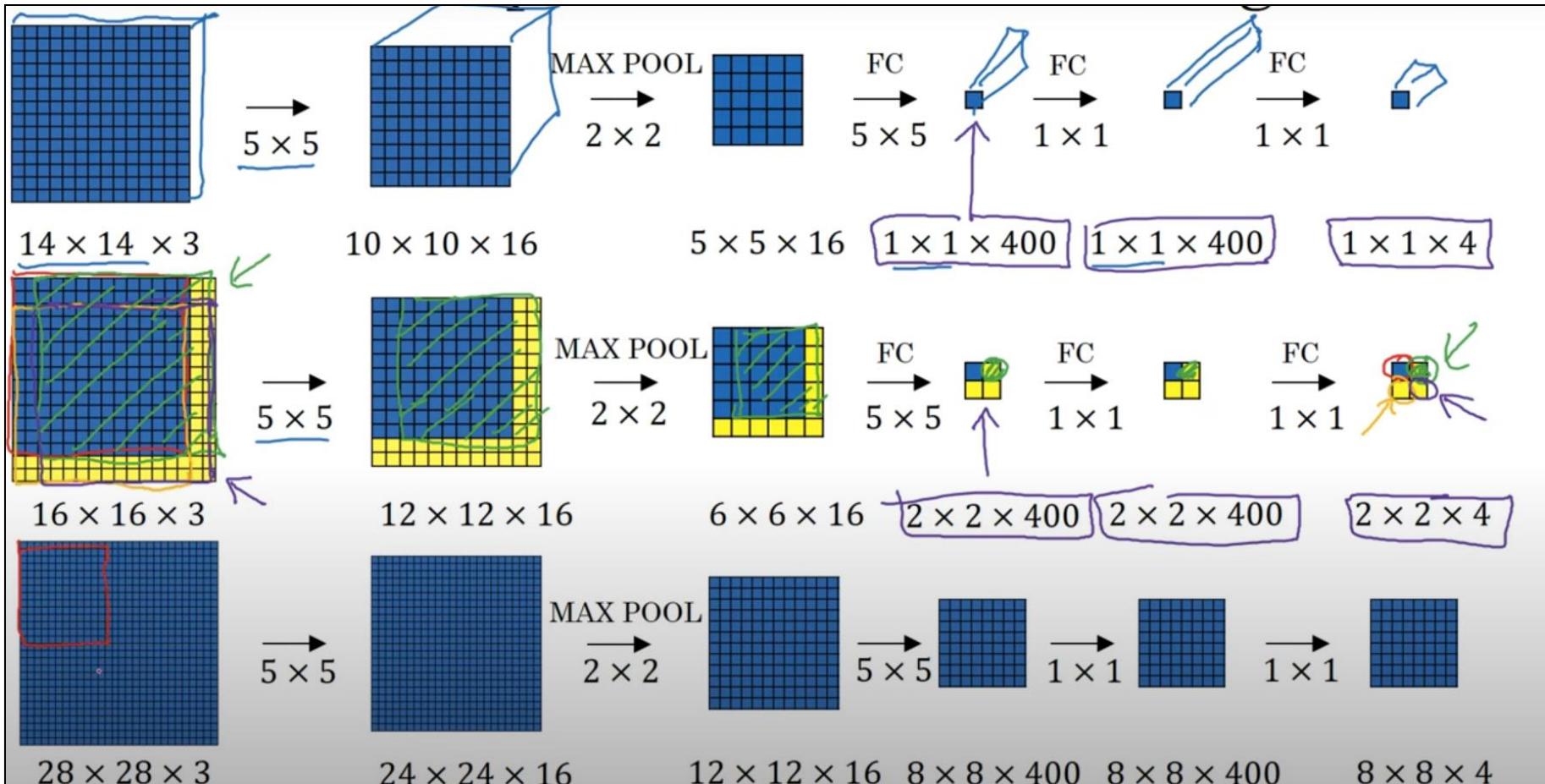
CNN Implementation of Sliding Window



Power of 1×1 convolutions

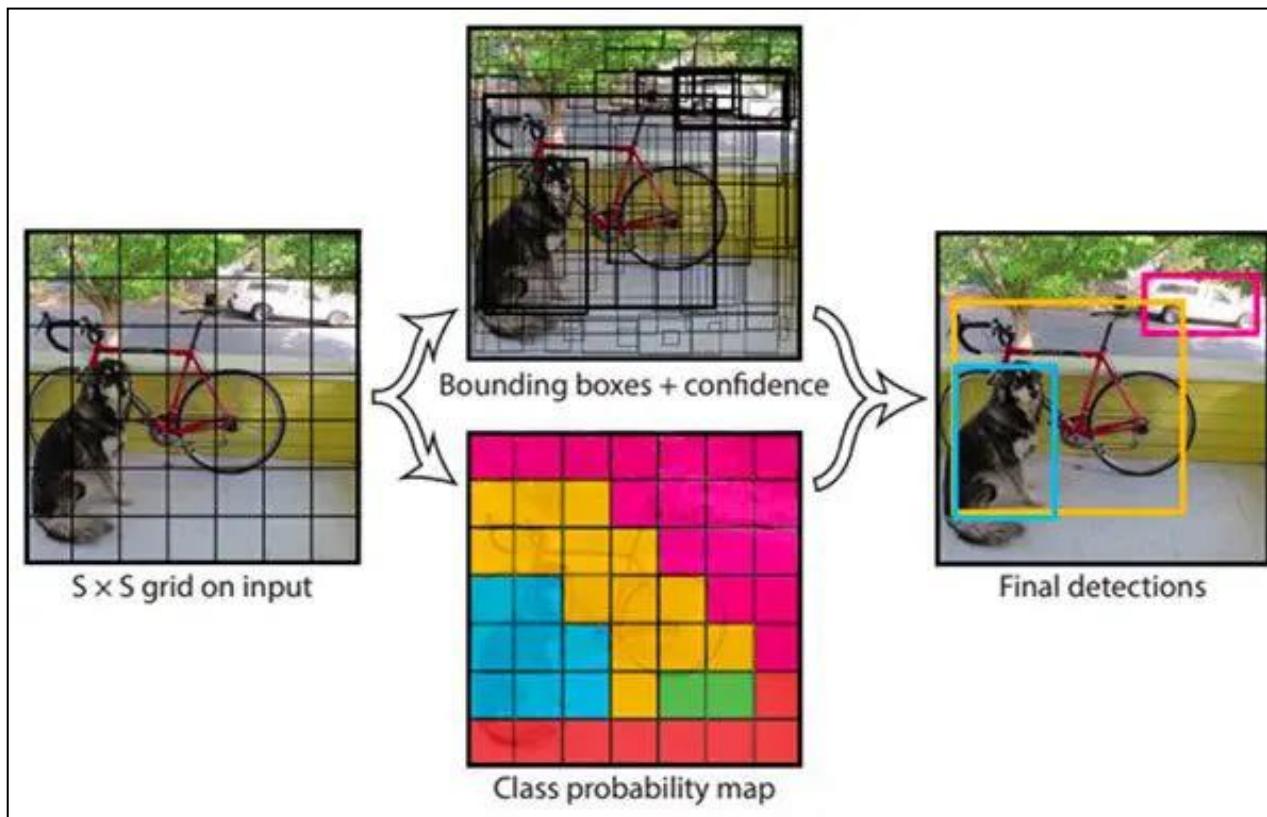


Overfeat algorithm



Types of Detectors

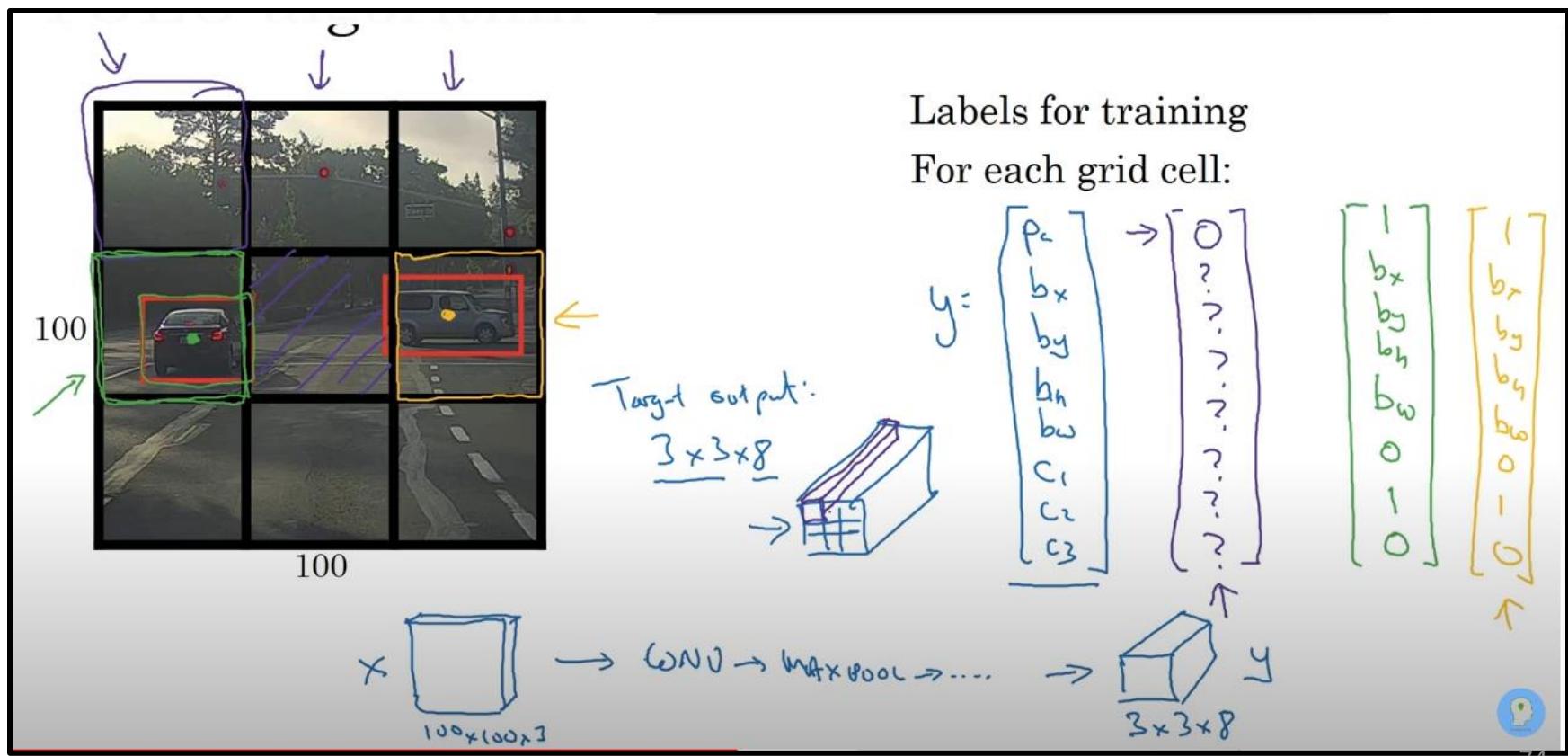
- Single-Stage Detectors
 - SSD, YOLO
- A standard pre-trained neural network is used as a feature extractor like VGG16, VGG19, or ResNet.



YOLO OBJECT DETECTOR

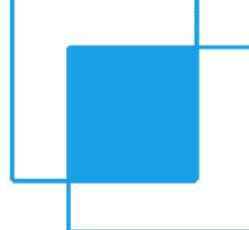
YOLO Idea

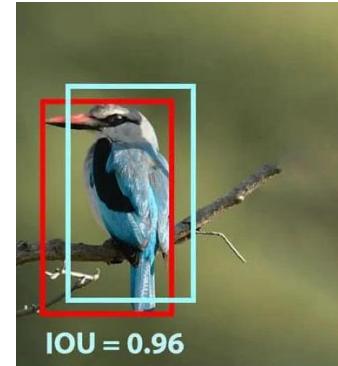
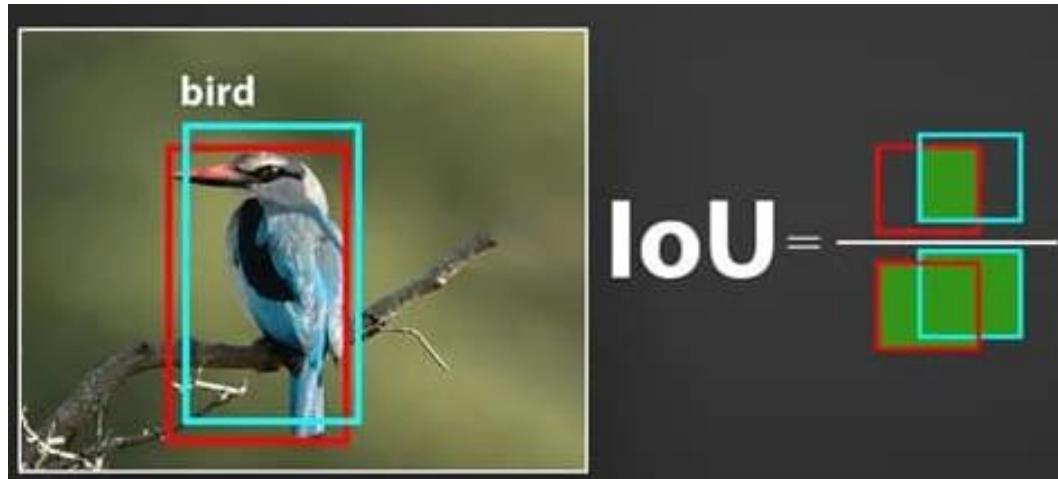
- Pass the image from a CNN and get the output volume
- For each Grid Cell:
- Assign the label (which include class probabilities & bounding box coordinates)
- Train the CNN as a regression problem



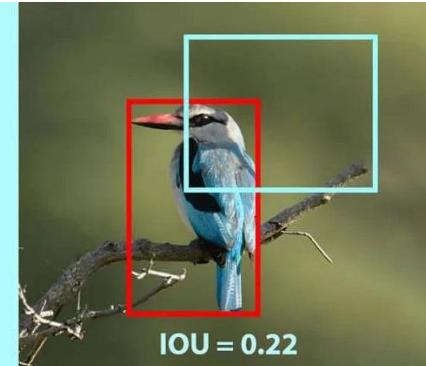
Intersection Over Union (IOU)

- Evaluating object localization
- IOU is a measure of overlap between two boxes
- Generally $\text{IOU} > 0.5$ is used for True Positive

$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$




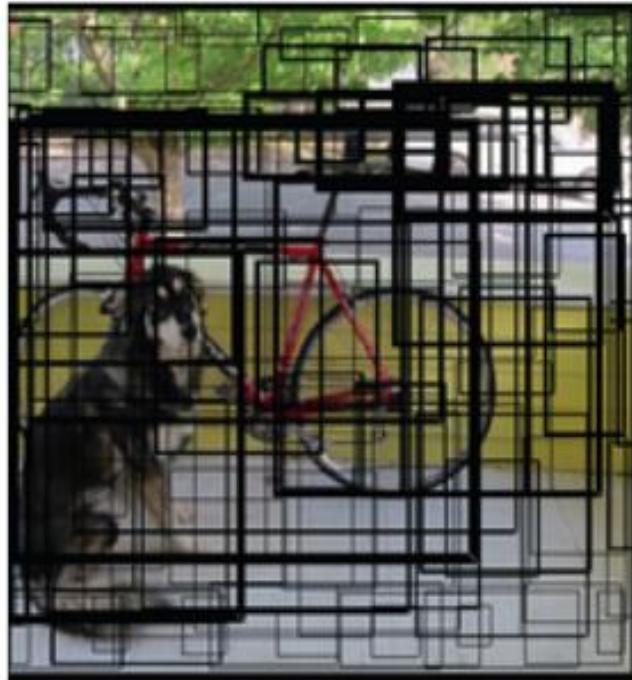
True Positive



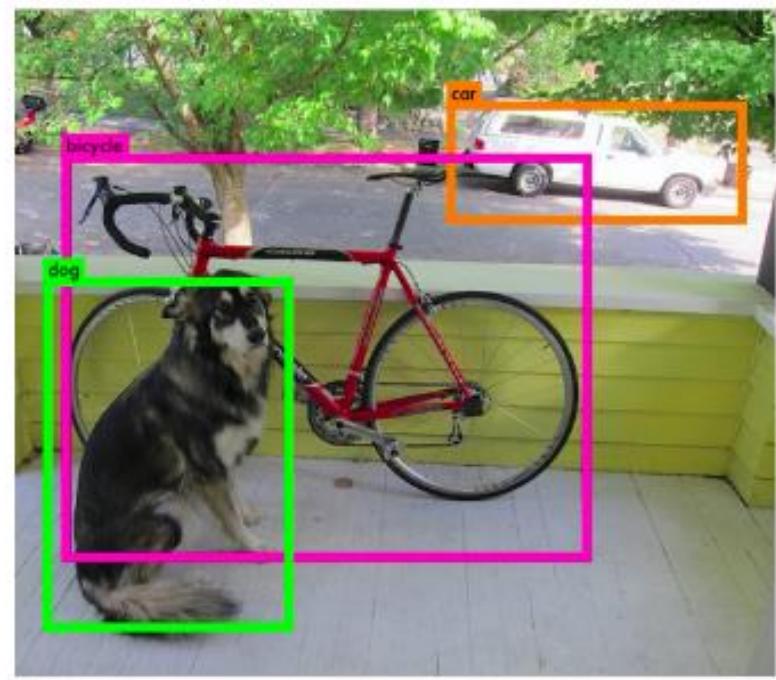
False Positive

Nonmax Suppression (NMS)

- Object detection algorithms create multiple bounding boxes.
- Ideally, for each object in the image, we must have a single bounding box



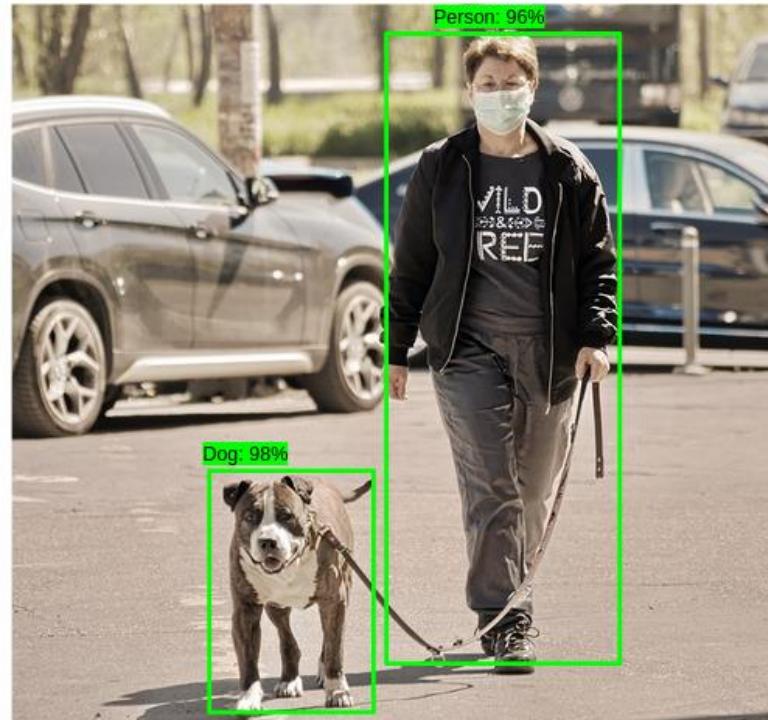
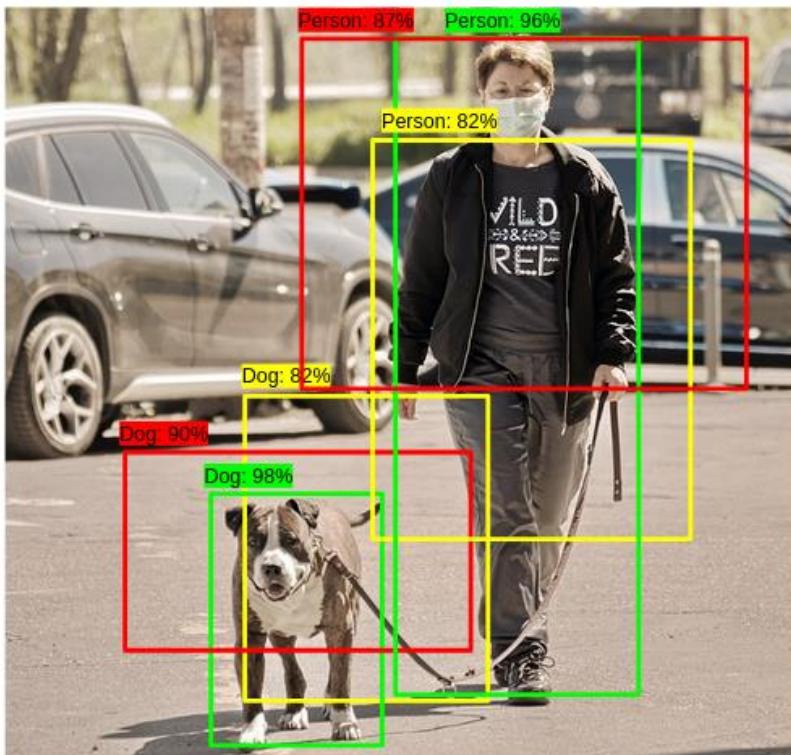
Multiple Bounding Boxes



Final Bounding Boxes

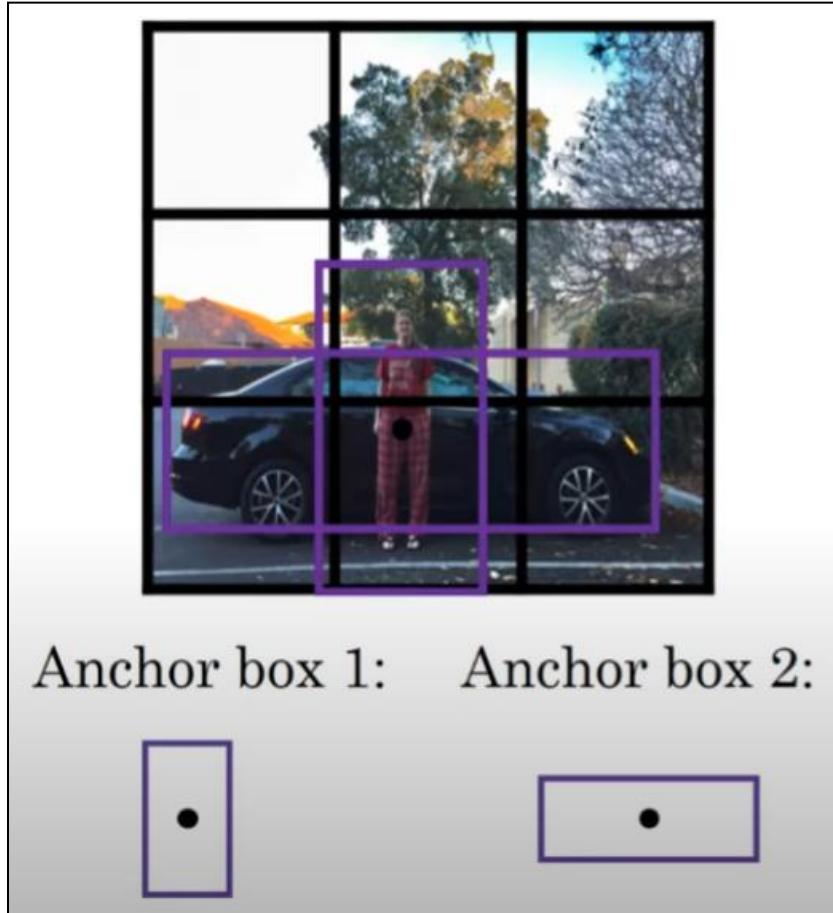
Nonmax Suppression (NMS)

- Procedure (Run independent for each class)
 - First select the bounding box with the highest objectiveness score
 - Remove all the other boxes with high overlap (using IOU) with the selected one
 - Freeze the selected box
 - Repeat until all boxes freeze



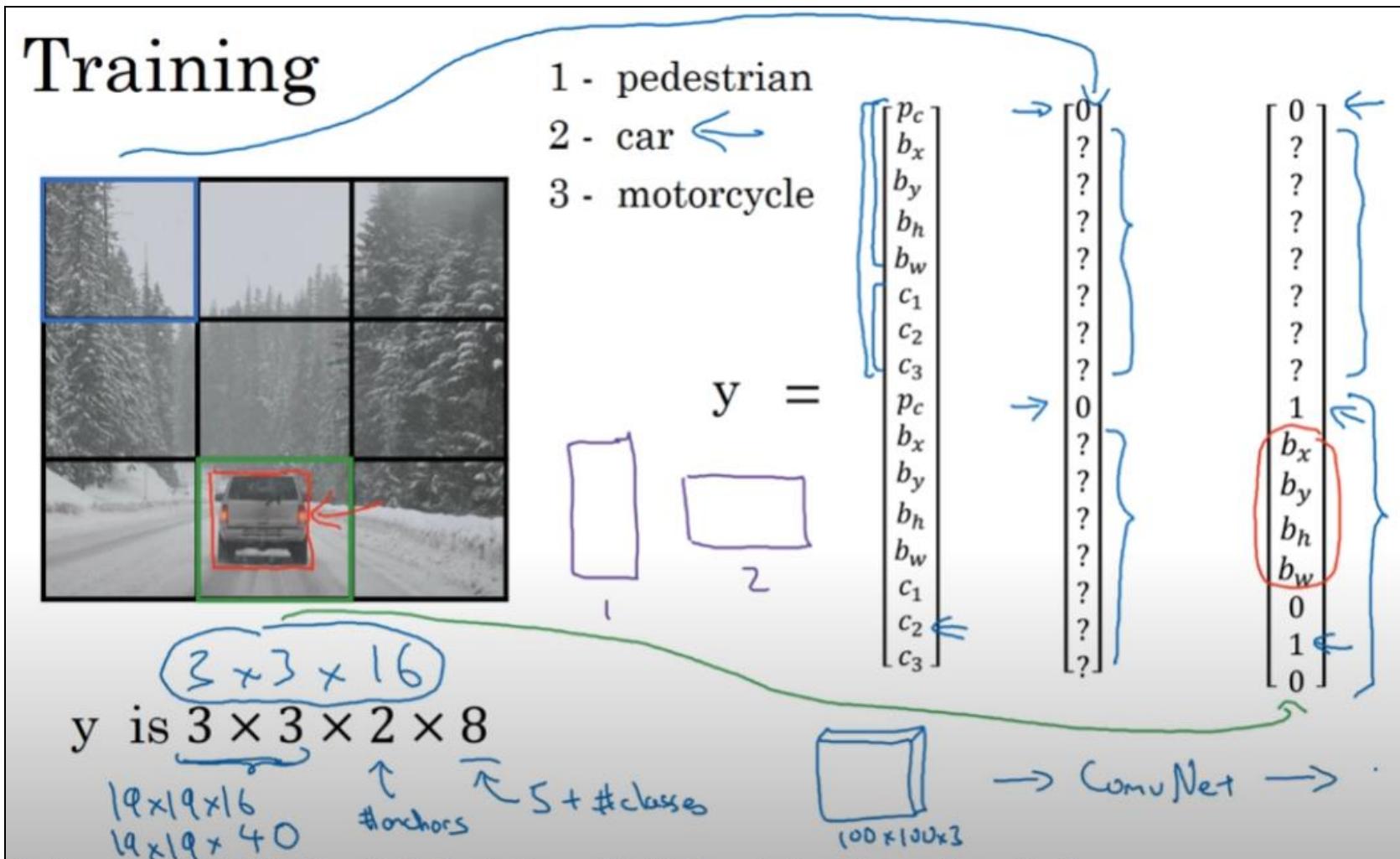
Anchor Boxes

- What if center of multiple objects fall in same grid cell



$$\mathbf{y} = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \\ p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

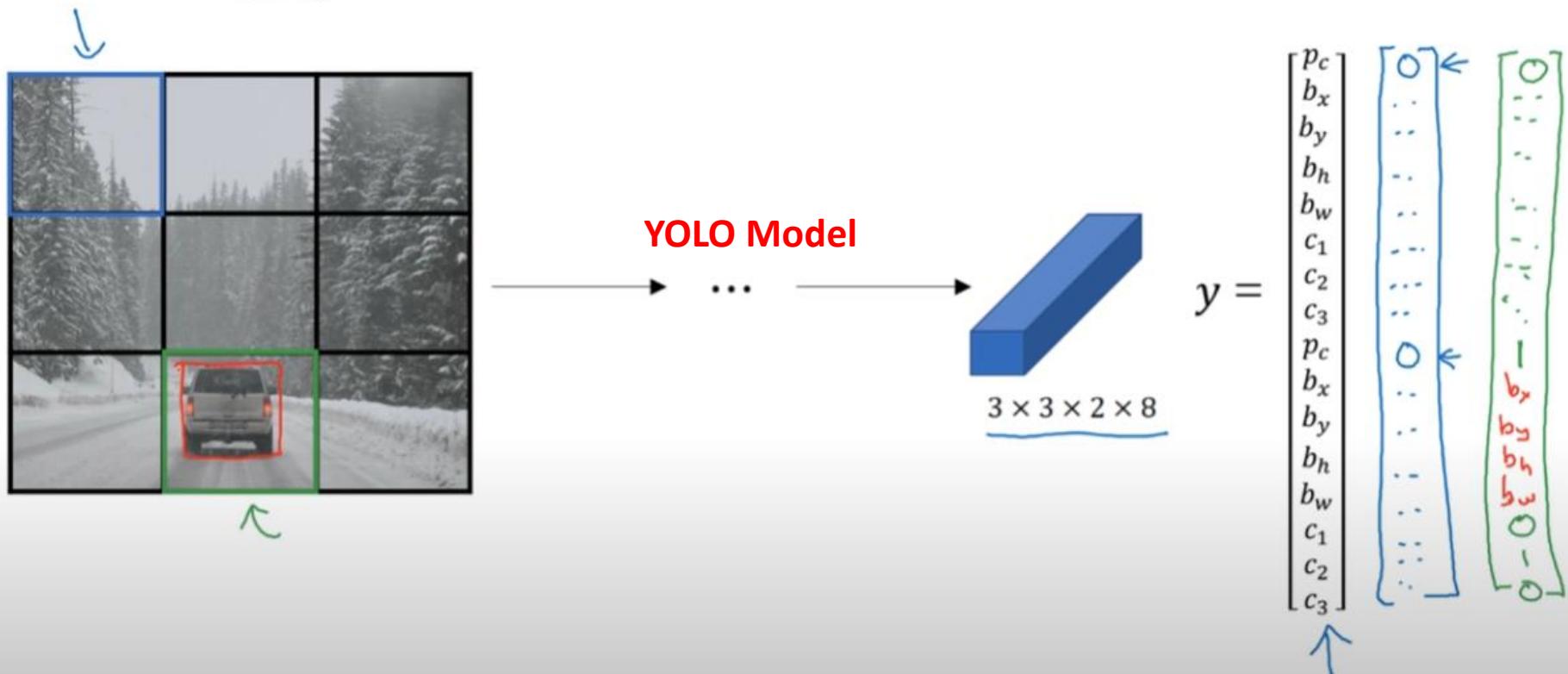
Putting it together (YOLO)



Think of the Loss function?

Putting it together (YOLO)

Making predictions



Putting it together (YOLO)

Outputting the non-max suppressed outputs



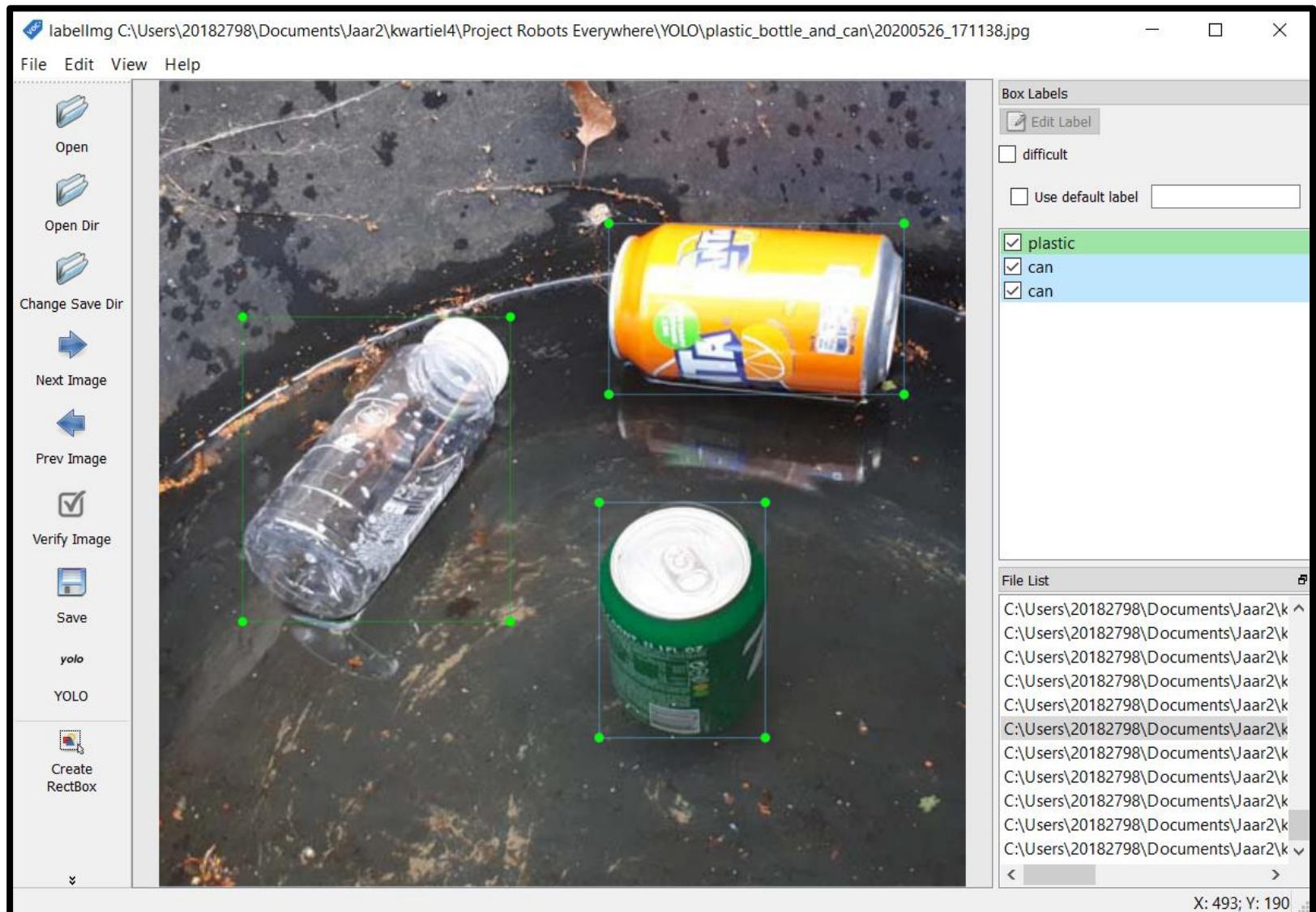
- For each grid cell, get 2 predicted bounding boxes.
- Get rid of low probability predictions.
- For each class (pedestrian, car, motorcycle) use non-max suppression to generate final predictions.

ULTRALYTICS YOLOV5

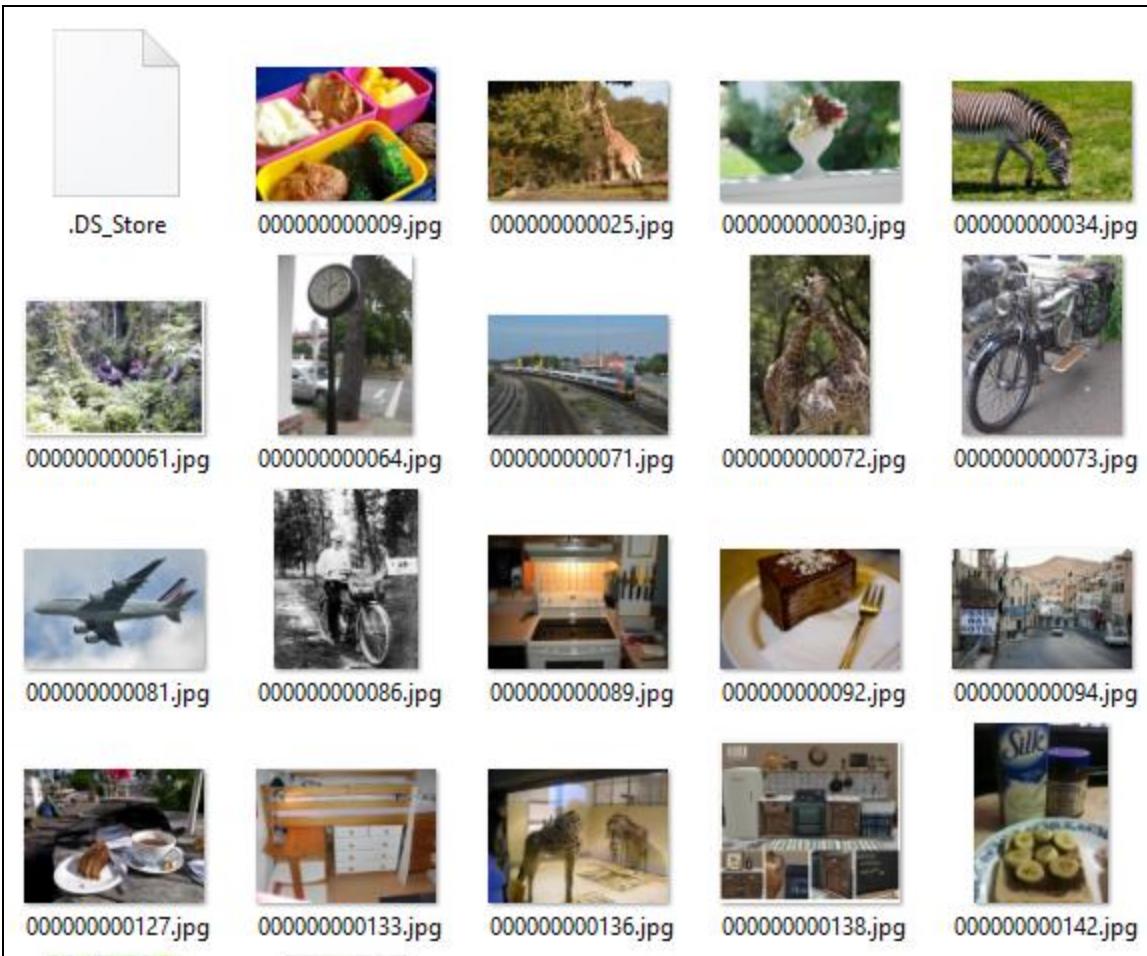
Ultralytics YOLOv5

- Family of object detection architectures and models pretrained on the COCO dataset

Data Labeling (LabelImg tool)



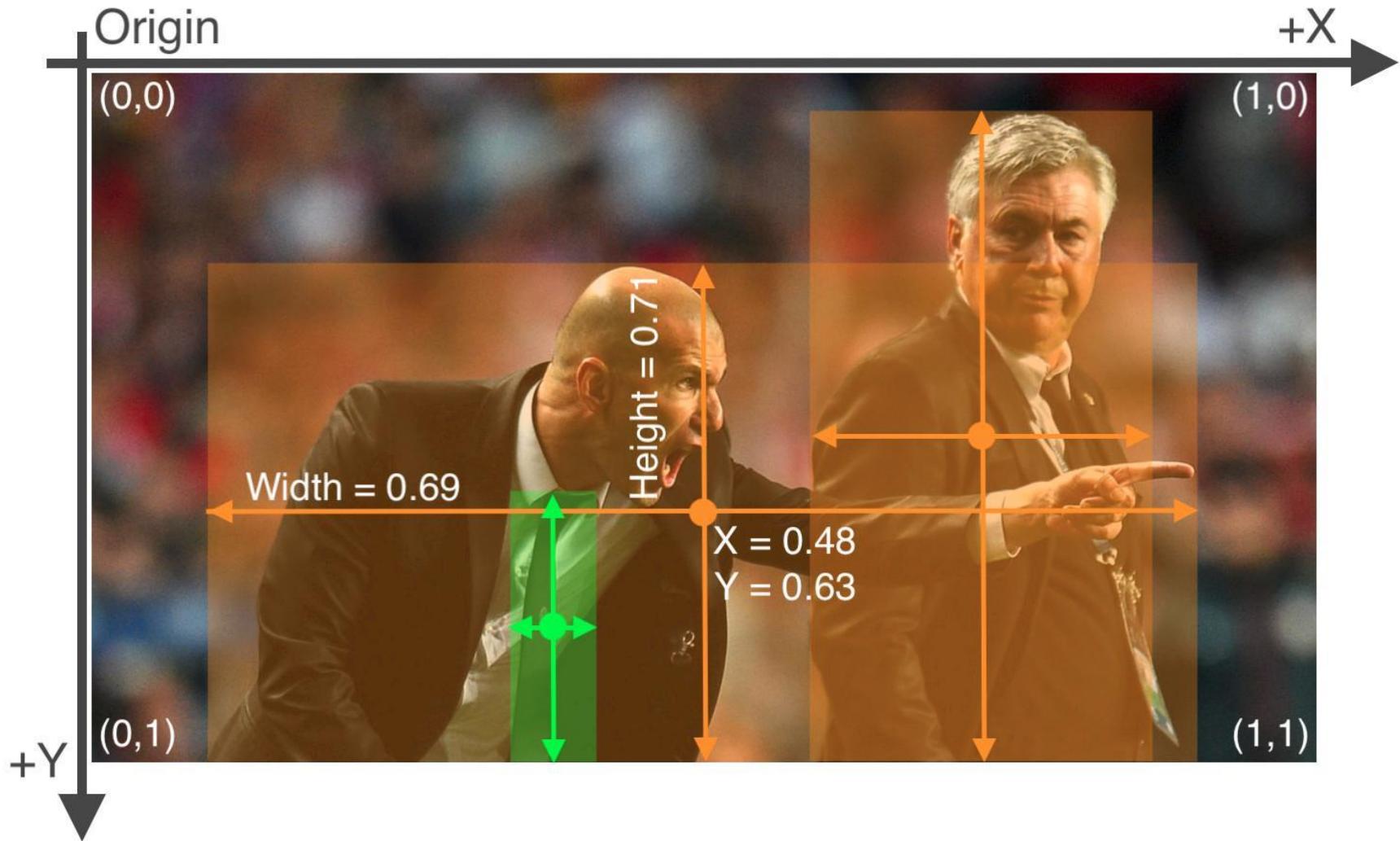
COCO-128 Dataset



Label File:

000000000061.txt - Notepad
File Edit Format View Help
0 0.445688 0.480615 0.075125 0.117295
0 0.640086 0.471742 0.050828 0.081434
20 0.643211 0.558852 0.129828 0.097623
20 0.459703 0.592121 0.22175 0.159242
0 0.435383 0.45832 0.053453 0.111025

YOLO Annotation Format



Dataset YAML File

```
# Train/val/test sets as 1) dir: path/to/imgs, 2) file: path/to/imgs.txt,
path: ../datasets/coco128 # dataset root dir
train: images/train2017 # train images (relative to 'path') 128 images
val: images/val2017 # val images (relative to 'path') 128 images
test: # test images (optional)

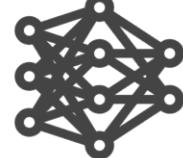
# Classes
names:
  0: person
  1: bicycle
  2: car
  3: motorcycle
  4: airplane
  5: bus
  6: train
  7: truck
  8: boat
  9: traffic light
  10: fire hydrant
```

Installing YOLOv5 Repository

- git clone <https://github.com/ultralytics/yolov5>
- cd yolov5
- pip install -r requirements.txt # install

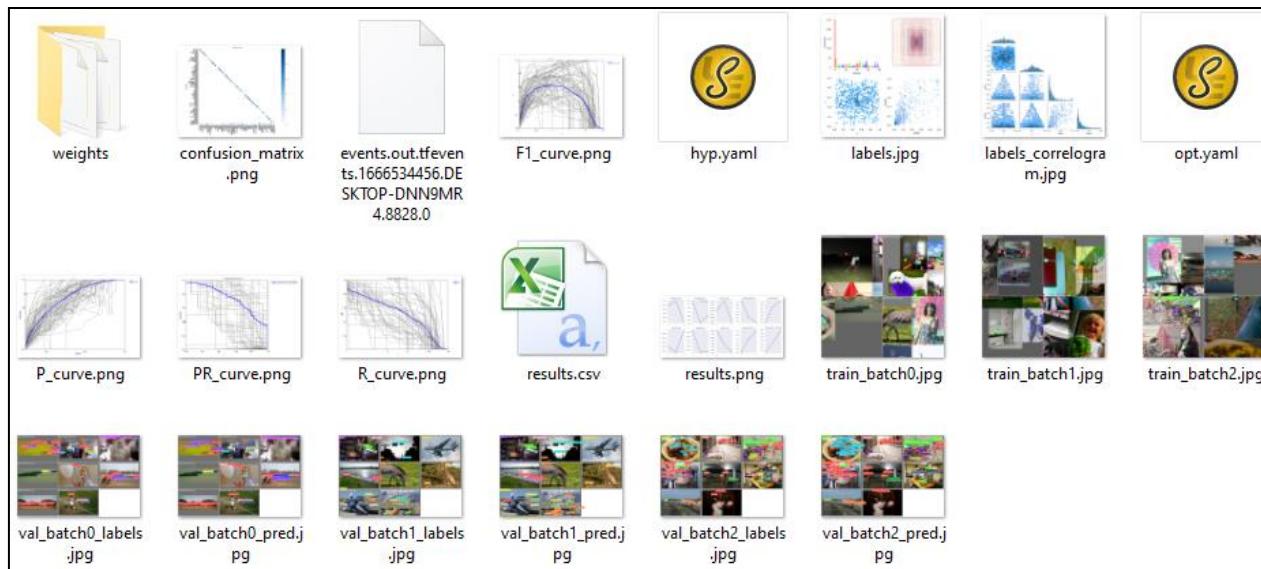
Training Model

- >> python train.py --workers 1 --img 640 --batch 16 --epochs 3 --data coco128.yaml **--weights yolov5s.pt**

				
Nano YOLOv5n	Small YOLOv5s	Medium YOLOv5m	Large YOLOv5l	XLarge YOLOv5x
4 MB _{FP16} 6.3 ms _{V100} 28.4 mAP _{COCO}	14 MB _{FP16} 6.4 ms _{V100} 37.2 mAP _{COCO}	41 MB _{FP16} 8.2 ms _{V100} 45.2 mAP _{COCO}	89 MB _{FP16} 10.1 ms _{V100} 48.8 mAP _{COCO}	166 MB _{FP16} 12.1 ms _{V100} 50.7 mAP _{COCO}

Training Model

Epoch	GPU_mem	box_loss	obj_loss	cls_loss	Instances	Size	
0/2	0.994G	0.04644	0.07389	0.02304	84	640: 100% ██████████ 32/32 [00:40<00:00, 1.26s/it]	
	Class all	Images 128	Instances 929	P 0.763	R 0.609	mAP50 0.722	mAP50-95: 100% ██████████ 16/16 [00:09<00:00, 1.66it/s]
						0.474	
Epoch	GPU_mem	box_loss	obj_loss	cls_loss	Instances	Size	
1/2	1.41G	0.04705	0.06827	0.02385	44	640: 100% ██████████ 32/32 [00:36<00:00, 1.15s/it]	
	Class all	Images 128	Instances 929	P 0.735	R 0.634	mAP50 0.73	mAP50-95: 100% ██████████ 16/16 [00:09<00:00, 1.76it/s]
						0.477	
Epoch	GPU_mem	box_loss	obj_loss	cls_loss	Instances	Size	
2/2	1.41G	0.04454	0.07496	0.02411	38	640: 59% █████ 19/32 [00:21<00:14, 1.15s/it]	



Training Model



Inference Script

```
class YOLOV5Model:
    def __init__(self) -> None:
        self.yoloPath='yolov5-master/'
        self.model=None
        self.inferSizeValue=640
    def loadModel(self,modelPath='yolov5-master/runs/train/exp/weights/best.pt'):
        self.modelPath=modelPath
        try:
            self.model = torch.hub.load(self.yoloPath, 'custom', path=self.modelPath, source='local',force_reload=True) # local repo
            self.classnames = self.model.module.names if hasattr(self.model, 'module') else self.model.names
            return True, ''
        except Exception as e:
            return False, e
    def inferImage(self,img,inferSize=640,conf=0.4,iou=0.45):
        inferSize=self.inferSizeValue
        self.model.iou=iou
        self.model.conf=conf
        results = self.model(img,size=inferSize) # includes NMS
        boxes = results.pandas().xyxy[0]

        bboxes=[]
        for index, row in boxes.iterrows():
            xyxy=[row['xmin'],row['ymin'],row['xmax'],row['ymax'],row['name'], row['confidence'],row['class']]
            bboxes.append(xyxy)

        results.show(labels=True)

        return results.ims[0], bboxes, results
```

Inference Script

```
class YOLOV5Model:
    def __init__(self) -> None:
        self.yoloPath='yolov5-master/'
        self.model=None
        self.inferSizeValue=640
    def loadModel(self,modelPath='yolov5-master/runs/train/exp/weights/best.pt'):
        self.modelPath=modelPath
        try:
            self.model = torch.hub.load(self.yoloPath, 'custom', path=self.modelPath, source='local',force_reload=True) # local repo
            self.classnames = self.model.module.names if hasattr(self.model, 'module') else self.model.names
            return True, ''
        except Exception as e:
            return False, e
    def inferImage(self,img,inferSize=640,conf=0.4,iou=0.45):
        inferSize=self.inferSizeValue
        self.model.iou=iou
        self.model.conf=conf
        results = self.model(img,size=inferSize) # includes NMS
        boxes = results.pandas().xyxy[0]

        bboxes=[]
        for index, row in boxes.iterrows():
            xyxy=[row['xmin'],row['ymin'],row['xmax'],row['ymax'],row['name'], row['confidence'],row['class']]
            bboxes.append(xyxy)

        results.show(labels=True)

        return results.ims[0], bboxes, results
```

Inference Script

```
model = YOLOV5Model()
model.loadModel()

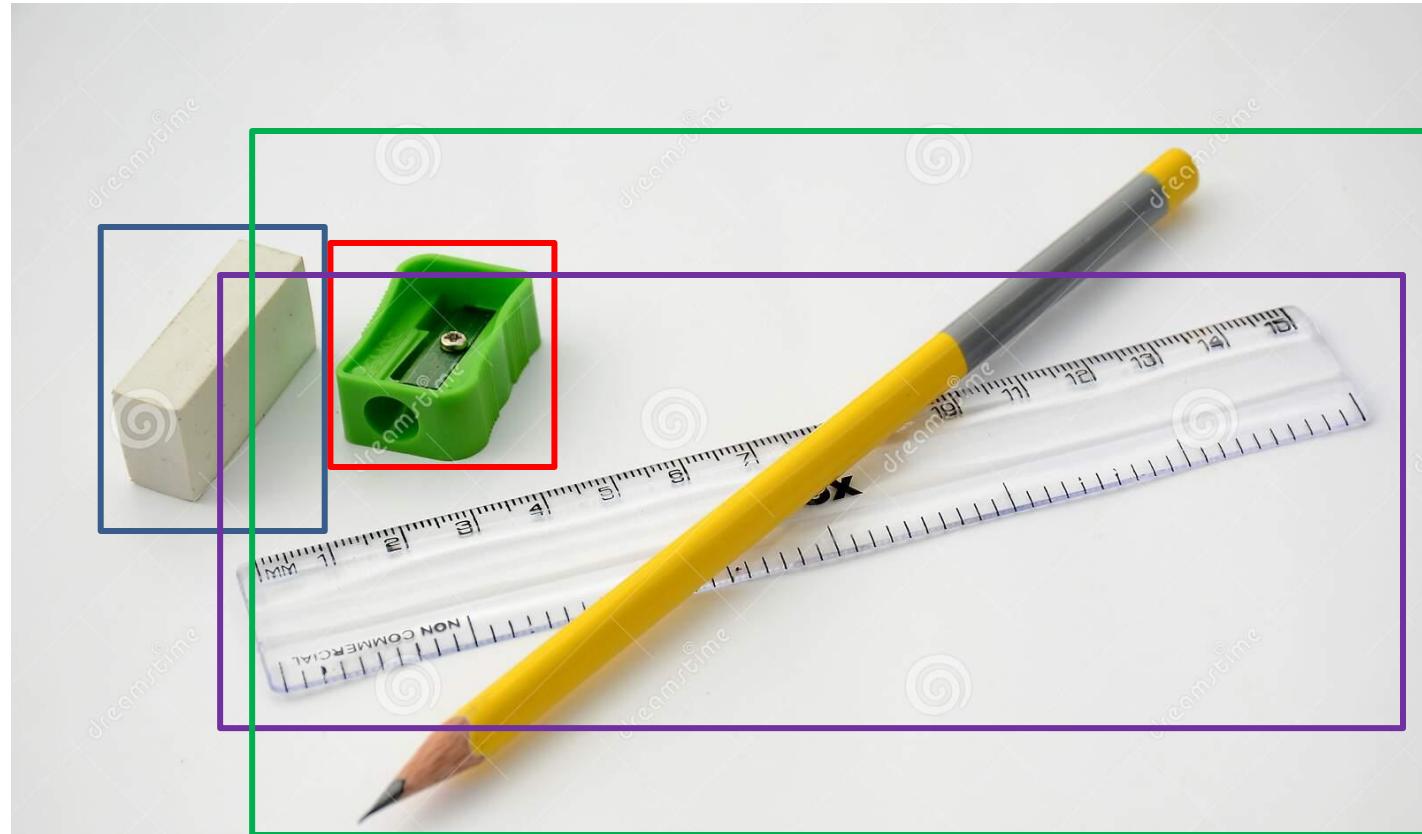
pillowIm=Image.open('yolov5-master/datasets/coco128/images/train2017/000000000510.jpg')
img, bboxes, results=model.inferImage(pillowIm)
```

Inference Script



Assignment 2

AI Based Automatic Stationery Billing System



Assignment 2

AI Based Automatic Stationery Billing System

- In this assignment you will implement a vision based billing system for a stationery items purchased in a shop
- Step-1: Collect yourself a data set of at least 500 images having following items
 - **Eraser, scale, pencil, sharpener**
 - Place different types of items in different combinations and take photos or get images from Google
 - Label the dataset using labelimage tool and save annotations in yolo format
 - Create folders for train (60%), validation (20%) and test(20%) according to YOLO training directory structure

Assignment 2

AI Based Automatic Stationery Billing System

- Step-2: **Train a YOLO model** (you can choose any size)
- Step-3: Develop a GUI application which can take
 - Image of a purchased item as input and detects the items with your trained model and **counts them and display a formatted bill** (assume any price for each item)
 - Video having images of different purchases. Your program must process each frame and annotate the bill on top and output the annotated video