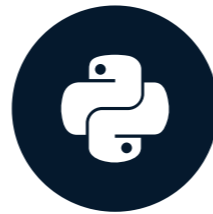


# Convolution operator

INTRODUCTION TO DEEP LEARNING WITH PYTORCH



**Ismail Elezi**

Ph.D. Student of Deep Learning

# Problems with the fully-connected neural networks



- Do you need to consider all the relations between the features?

# Problems with the fully-connected neural networks



- Do you need to consider all the relations between the features?
- Fully-connected neural networks are big and so very computationally inefficient.
- They have so many parameters, and so overfit.

# Main ideas

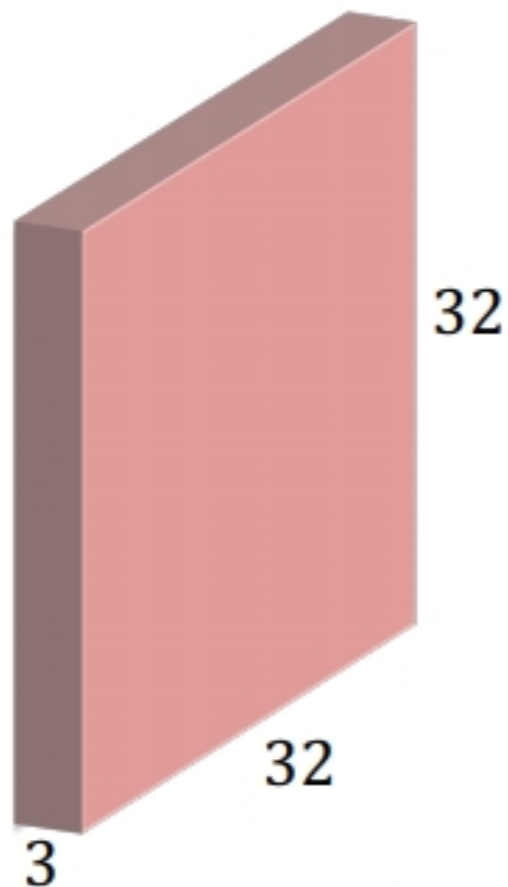


- 1) Units are connected with only a few units from the previous layer.
- 2) Units share weights.

# Convolutions - basic idea

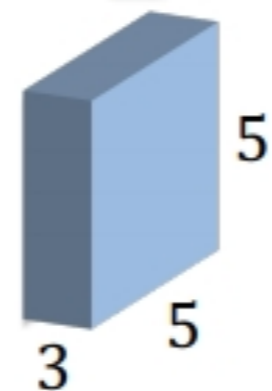
width height depth

32x32x3 image



Depth dimension *must* match;  
i.e., filter extends the full depth of the input

5x5x3 filter

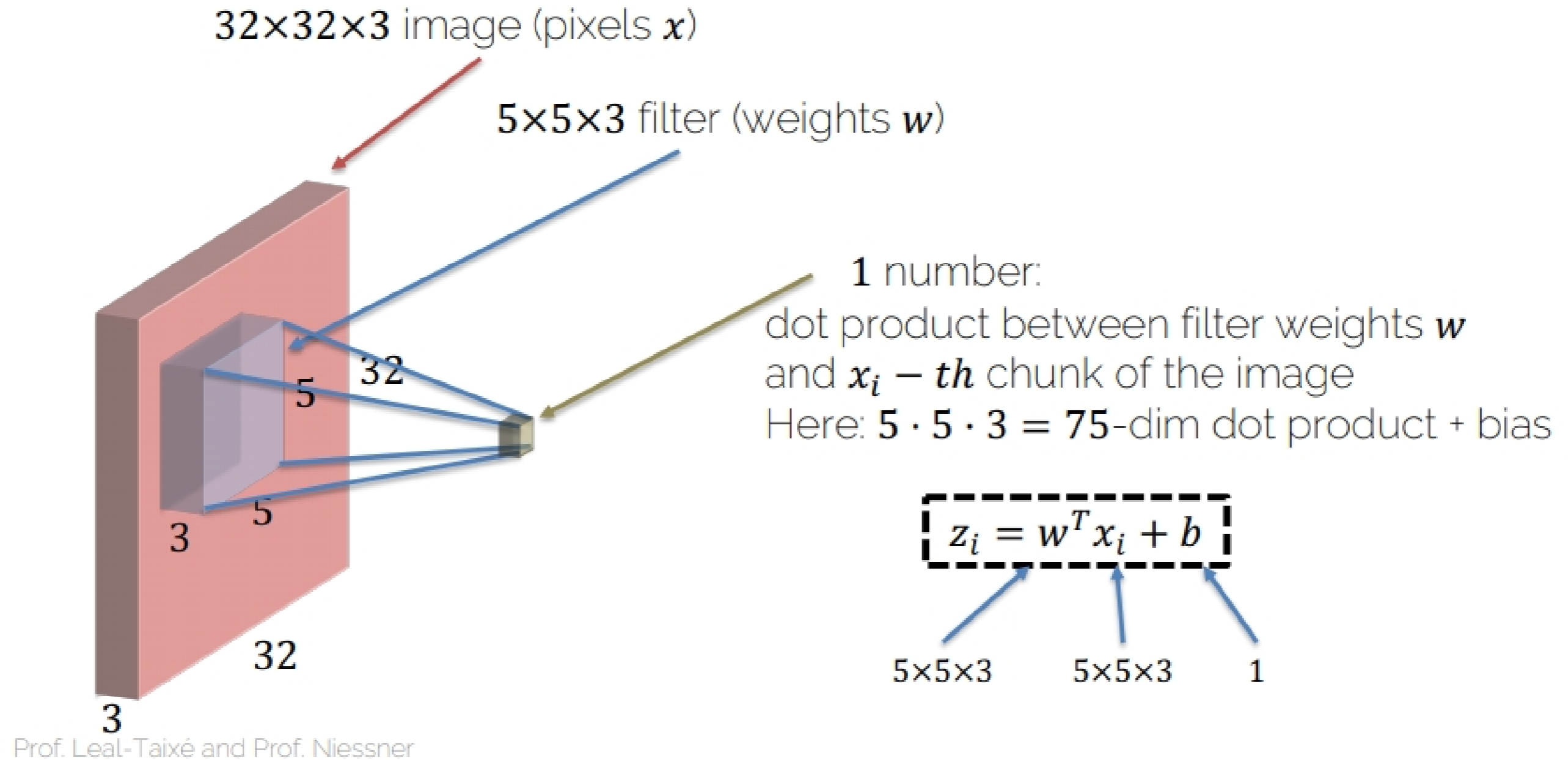


Convolve filter with image  
i.e., 'slide' over it and:  
-> apply filter at each location  
-> dot products

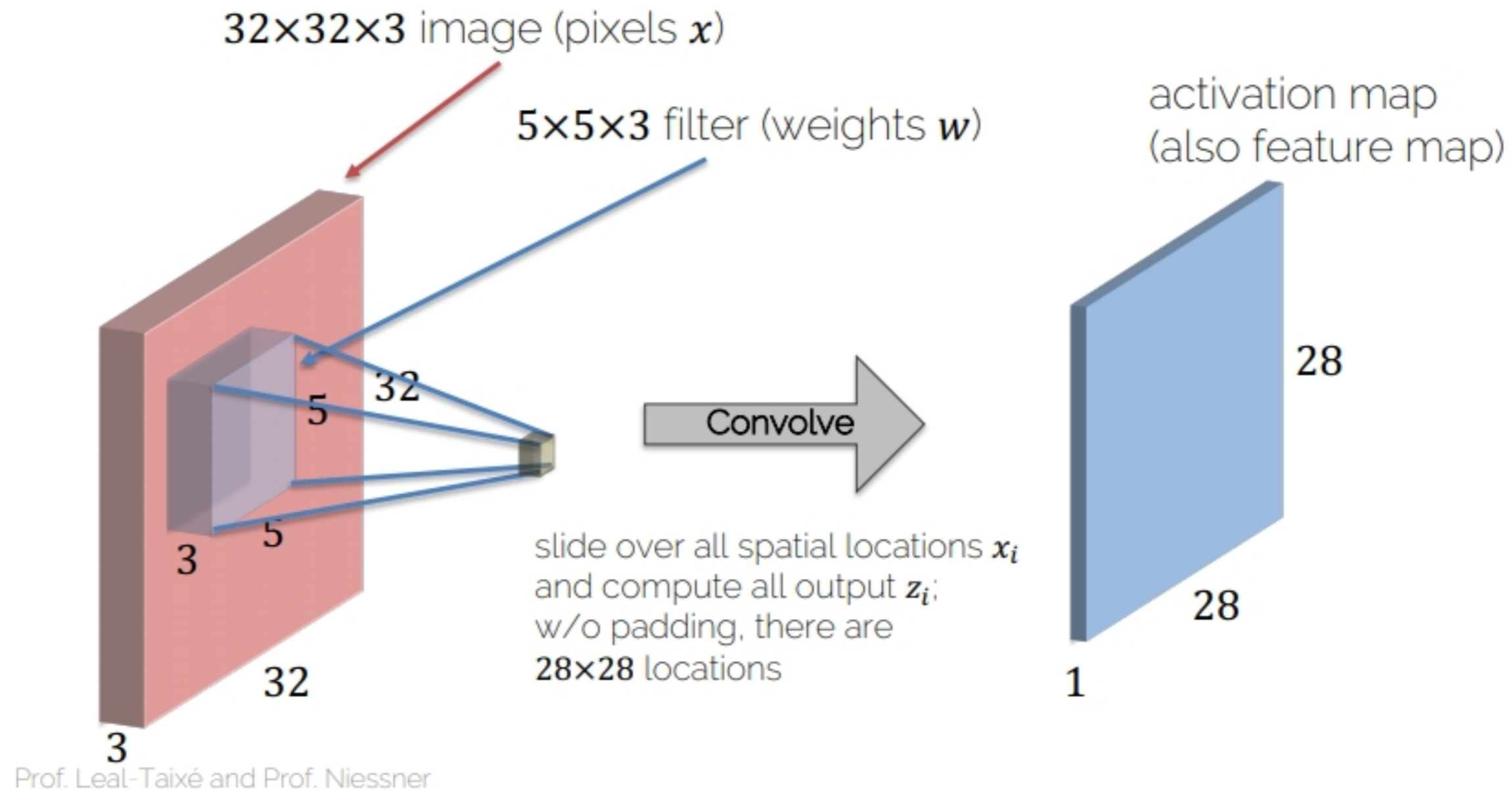
Images have depth: e.g., RGB -> 3 channels

Prof. Leal-Taixé and Prof. Niessner

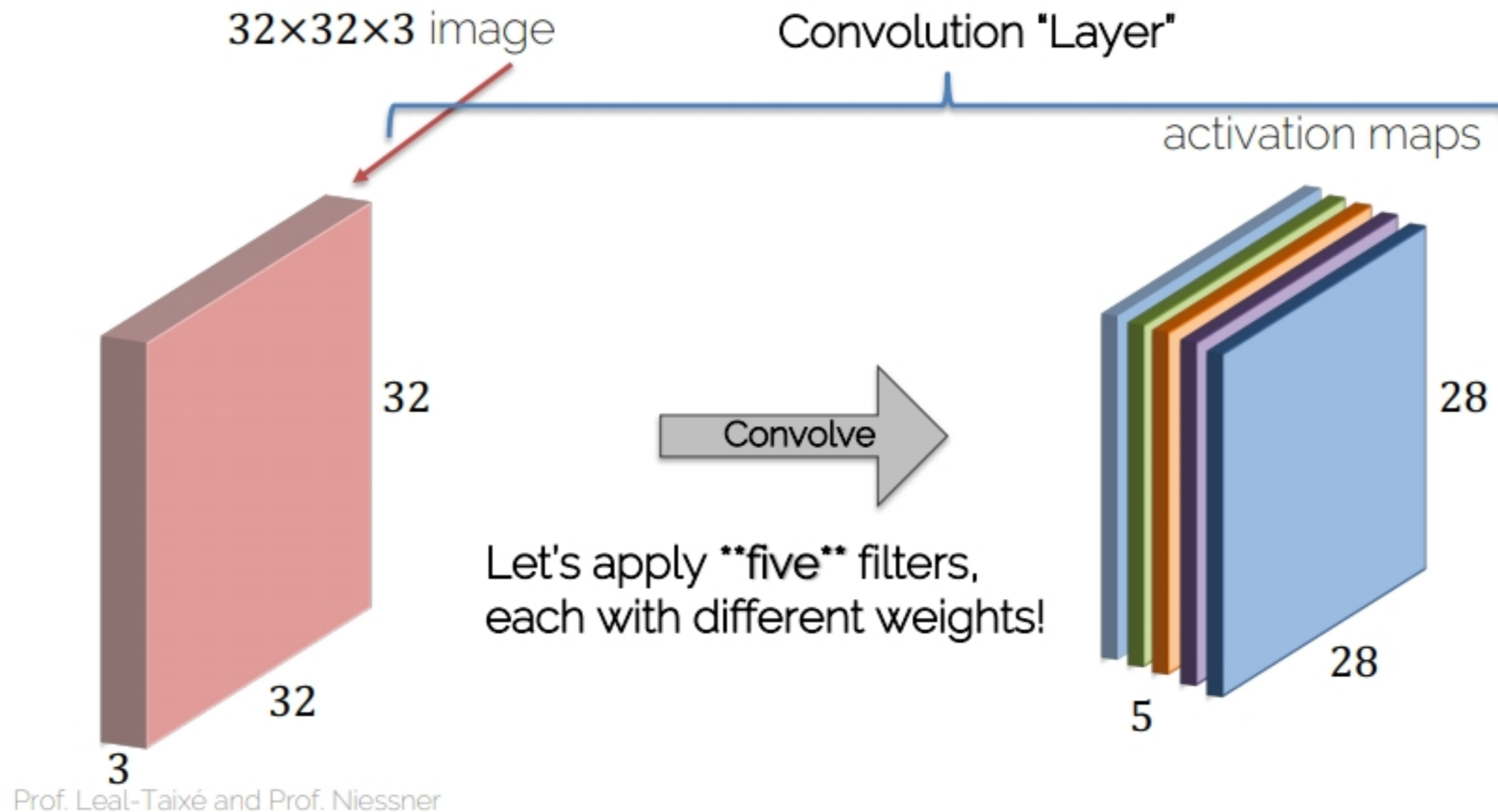
# Convoluting



# Activation map



# Activation map



# Padding

Image 7x7 + zero padding

0	0	0	0	0	0	0	0	0
0								0
0								0
0								0
0								0
0								0
0								0
0	0	0	0	0	0	0	0	0

Why padding:

- Sizes get small too quickly
- Corner pixel is only used once

Prof. Leal-Taixé and Prof. Niessner

# Convolutions in PyTorch

## OOP-based (torch.nn)

`in_channels` (int) – Number of channels in input

`out_channels` (int) – Number of channels produced by the convolution

`kernel_size` (int or tuple) – Size of the convolving kernel

`stride` (int or tuple, optional) – Stride of the convolution. Default: 1

`padding` (int or tuple, optional) – Zero-

## Functional (torch.nn.functional)

`input` – input tensor of shape (minibatch×in\_channels×iH×iW)

`weight` – filters of shape (out\_channels×in\_channels×kH×kW)

`stride` – the stride of the convolving kernel. Can be a single number or a tuple (sH, sW). Default: 1

`padding` – implicit zero paddings on both sides of the input. Can be a single number or a tuple (padH, padW). Default: 0

# Convolutions in PyTorch

```
import torch
import torch.nn

image = torch.rand(16, 3, 32, 32)
conv_filter = torch.nn.Conv2d(in_channels=3,
                              out_channels=1, kernel_size=5,
                              stride=1, padding=0)
output_feature = conv_filter(image)

print(output_feature.shape)
```

```
torch.Size([16, 1, 28, 28])
```

```
import torch
import torch.nn.functional as F

image = torch.rand(16, 3, 32, 32)
filter = torch.rand(1, 3, 5, 5)
out_feat_F = F.conv2d(image, filter,
                      stride=1, padding=0)

print(out_feat_F.shape)
```

```
torch.Size([16, 1, 28, 28])
```

# Convolutions in PyTorch

```
conv_layer = torch.nn.Conv2d(in_channels=3,  
                              out_channels=5, kernel_size=5,  
                              stride=1, padding=1)  
output = conv_layer(image)  
print(output.shape)
```

```
torch.Size([16, 5, 32, 32])
```

```
filter = torch.rand(3, 5, 5, 5)  
output_feature = F.conv2d(image, filter,  
                           stride=1, padding=1)  
print(output_feature.shape)
```

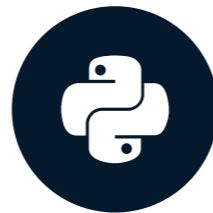
```
torch.Size([16, 5, 32, 32])
```

# Let's practice!

INTRODUCTION TO DEEP LEARNING WITH PYTORCH

# Pooling operators

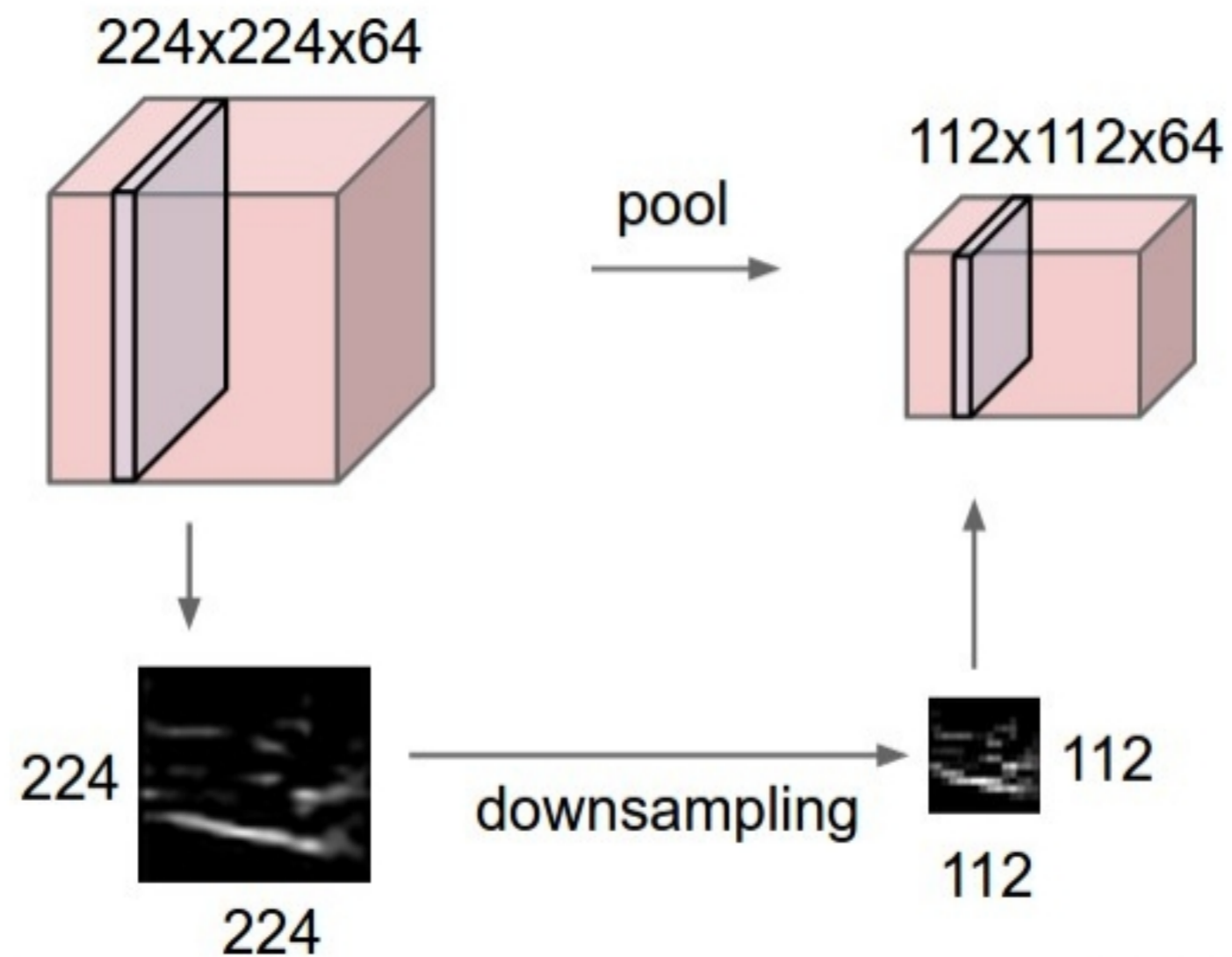
INTRODUCTION TO DEEP LEARNING WITH PYTORCH



**Ismail Elezi**

Ph.D. Student of Deep Learning

# Pooling layer



Prof. Leal-Taixé and Prof. Niessner

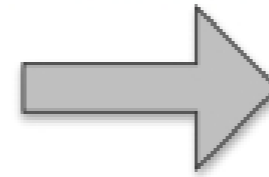
Slide by Li/Karpathy/Johnson

# Max-Pooling

Single depth slice of input

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

Max pool with  
 $2 \times 2$  filters and stride 2



'Pooled' output

6	9
3	4

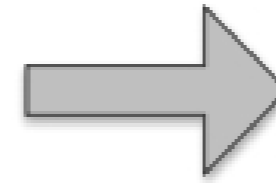
Prof. Leal-Taixé and Prof. Niessner

# Average-Pooling

Single depth slice of input

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

Max pool with  
2x2 filters and stride 2



'Pooled' output

2.5	6
1.75	3

- Typically used deeper in the network

Prof. Leal-Taixé and Prof. Niessner

# Max-pooling in PyTorch

## OOP

```
import torch
import torch.nn

im = torch.Tensor([[[[3, 1, 3, 5], [6, 0, 7, 9],
                    [3, 2, 1, 4], [0, 2, 4, 3]]]])
max_pooling = torch.nn.MaxPool2d(2)
output_feature = max_pooling(im)
print(output_feature)
```

```
tensor([[[[6., 9.],
         [3., 4.]]])])
```

## Functional

```
import torch
import torch.nn.functional as F

im = torch.Tensor([[[[3, 1, 3, 5], [6, 0, 7, 9],
                    [3, 2, 1, 4], [0, 2, 4, 3]]]])

output_feature_F = F.max_pool2d(im, 2)
print(output_feature_F)
```

```
tensor([[[[6., 9.],
         [3., 4.]]])])
```

# Average pooling in PyTorch

## OOP

```
import torch
import torch.nn

im = torch.Tensor([[[[3, 1, 3, 5], [6, 0, 7, 9],
                    [3, 2, 1, 4], [0, 2, 4, 3]]]])

avg_pooling = torch.nn.AvgPool2d(2)
output_feature = avg_pooling(im)
print(output_feature)
```

```
tensor([[[[2.5000, 6.0000],
          [1.7500, 3.0000]]]])
```

## Functional

```
import torch
import torch.nn.functional as F

im = torch.Tensor([[[[3, 1, 3, 5], [6, 0, 7, 9],
                    [3, 2, 1, 4], [0, 2, 4, 3]]]])

output_feature_F = F.avg_pool2d(im, 2)
print(output_feature_F)
```

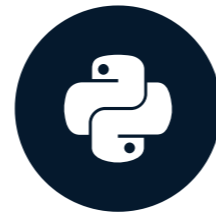
```
tensor([[[[2.5000, 6.0000],
          [1.7500, 3.0000]]]])
```

# Let's practice!

INTRODUCTION TO DEEP LEARNING WITH PYTORCH

# Convolutional Neural Networks

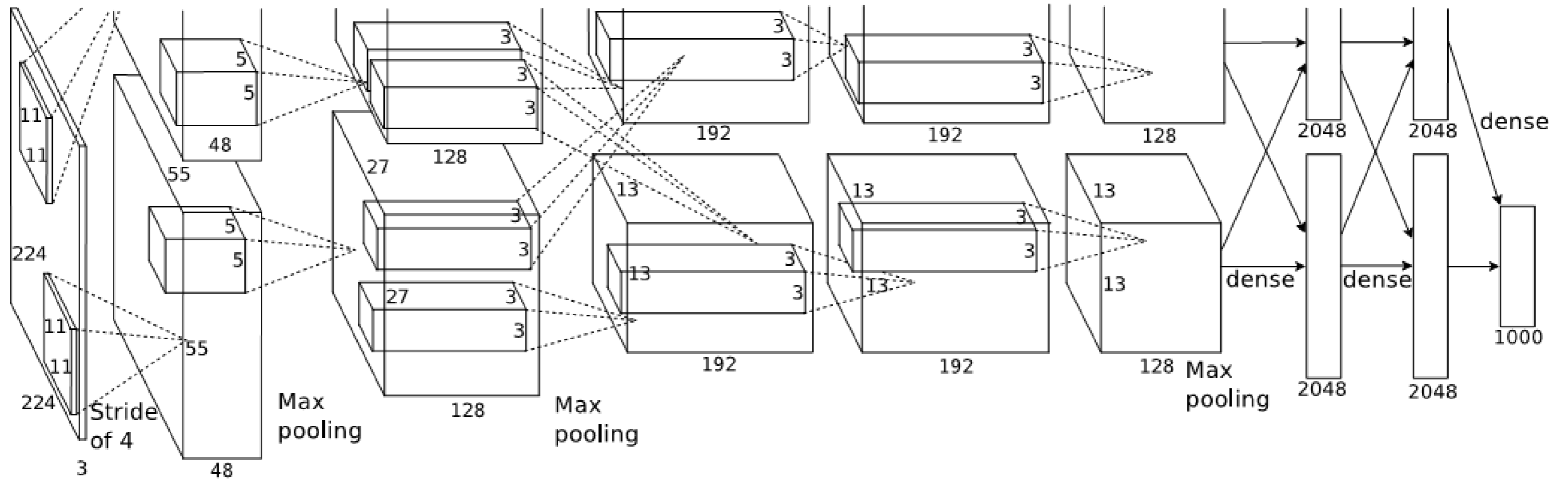
INTRODUCTION TO DEEP LEARNING WITH PYTORCH



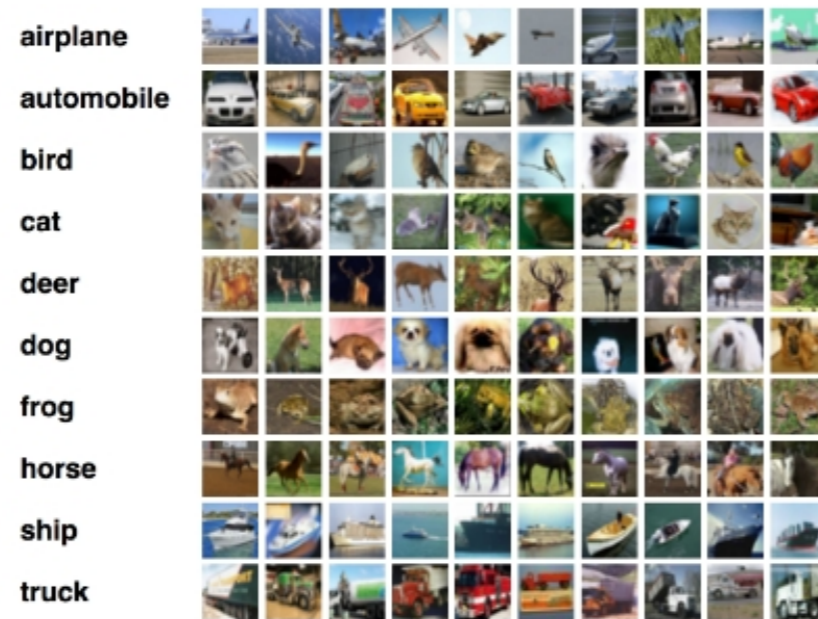
**Ismail Elezi**

Ph.D. Student of Deep Learning

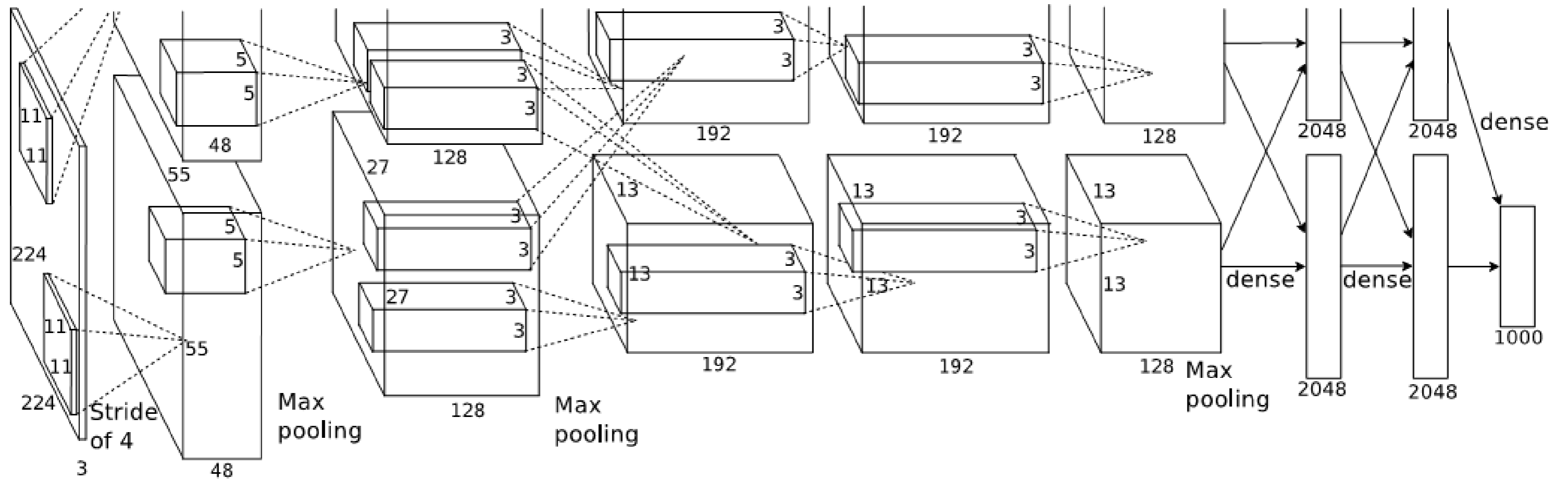
# AlexNet



# Transformation of computer vision



# AlexNet architecture



<sup>1</sup> Alex Krizhevsky, Ilya Sutskever and Geoffrey Hinton; ImageNet Classification with Deep Convolutional Neural Networks, NIPS 2012.

# AlexNet in PyTorch

```
class AlexNet(nn.Module):  
  
    def __init__(self, num_classes=1000):  
        super(AlexNet, self).__init__()  
        self.conv1 = nn.Conv2d(3, 64, kernel_size=11, stride=4, padding=2)  
        self.relu = nn.ReLU(inplace=True)  
        self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2)  
        self.conv2 = nn.Conv2d(64, 192, kernel_size=5, padding=2)  
        self.conv3 = nn.Conv2d(192, 384, kernel_size=3, padding=1)  
        self.conv4 = nn.Conv2d(384, 256, kernel_size=3, padding=1)  
        self.conv5 = nn.Conv2d(256, 256, kernel_size=3, padding=1)  
        self.avgpool = nn.AdaptiveAvgPool2d((6, 6))  
        self.fc1 = nn.Linear(256 * 6 * 6, 4096)  
        self.fc2 = nn.Linear(4096, 4096)  
        self.fc3 = nn.Linear(4096, num_classes)
```

# The forward method

```
def forward(self, x):
    x = self.relu(self.conv1(x))
    x = self.maxpool(x)
    x = self.relu(self.conv2(x))
    x = self.maxpool(x)
    x = self.relu(self.conv3(x))
    x = self.relu(self.conv4(x))
    x = self.relu(self.conv5(x))
    x = self.maxpool(x)
    x = self.avgpool(x)
    x = x.view(x.size(0), 256 * 6 * 6)
    x = self.relu(self.fc1(x))
    x = self.relu(self.fc2(x))
    return self.fc3(x)
```

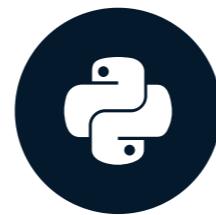
```
net = AlexNet()
```

# Let's practice!

INTRODUCTION TO DEEP LEARNING WITH PYTORCH

# Training Convolutional Neural Networks

INTRODUCTION TO DEEP LEARNING WITH PYTORCH



**Ismail Elezi**

Ph.D. Student of Deep Learning

# Imports

```
import torch
import torchvision
import torchvision.transforms as transforms
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
```

# Dataloaders

```
transform = transforms.Compose(  
    [transforms.ToTensor(),  
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])  
  
trainset = torchvision.datasets.CIFAR10(root='./data', train=True,  
                                       download=True, transform=transform)  
trainloader = torch.utils.data.DataLoader(trainset, batch_size=128,  
                                          shuffle=True, num_workers=2)  
  
testset = torchvision.datasets.CIFAR10(root='./data', train=False,  
                                       download=True, transform=transform)  
testloader = torch.utils.data.DataLoader(testset, batch_size=128,  
                                         shuffle=False, num_workers=2)
```

# Building a CNN

```
class Net(nn.Module):
    def __init__(self, num_classes=10):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=32, kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(in_channels=32, out_channels=64, kernel_size=3, padding=1)
        self.conv3 = nn.Conv2d(in_channels=64, out_channels=128, kernel_size=3, padding=1)
        self.pool = nn.MaxPool2d(2, 2)
        self.fc = nn.Linear(128 * 4 * 4, num_classes)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = self.pool(F.relu(self.conv3(x)))
        x = x.view(-1, 128 * 4 * 4)
        return self.fc(x)
```

# Optimizer and Loss Function

```
net = Net()  
criterion = nn.CrossEntropyLoss()  
optimizer = optim.Adam(net.parameters(), lr=3e-4)
```

# Training a CNN

```
for epoch in range(10):
    for i, data in enumerate(trainloader, 0):
        # Get the inputs
        inputs, labels = data

        # Zero the parameter gradients
        optimizer.zero_grad()

        # Forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

print('Finished Training')
```

# Evaluating the results

```
correct, total = 0, 0
predictions = []
net.eval()
for i, data in enumerate(testloader, 0):
    inputs, labels = data
    outputs = net(inputs)
    _, predicted = torch.max(outputs.data, 1)
    predictions.append(outputs)
    total += labels.size(0)
    correct += (predicted == labels).sum().item()

print('The testing set accuracy of the network is: %d %%' % (
    100 * correct / total))
```

The testing set accuracy of the network is: 68 %

# Let's practice!

INTRODUCTION TO DEEP LEARNING WITH PYTORCH