

# The sequential module

INTRODUCTION TO DEEP LEARNING WITH PYTORCH



Ismail Elezi

Ph.D. Student of Deep Learning

# AlexNet - declaring the modules

```
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)
```

# AlexNet - forward() methods

```
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()
```

# The sequential module - declaring the modules

```
class AlexNet(nn.Module):

    def __init__(self, num_classes=1000):
        super(AlexNet, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=11, stride=4, padding=2), nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2), nn.Conv2d(64, 192, kernel_size=5, padding=2),
            nn.ReLU(inplace=True), nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(192, 384, kernel_size=3, padding=1), nn.ReLU(inplace=True),
            nn.Conv2d(384, 256, kernel_size=3, padding=1), nn.ReLU(inplace=True),
            nn.Conv2d(256, 256, kernel_size=3, padding=1), nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),)
        self.avgpool = nn.AdaptiveAvgPool2d((6, 6))
        self.classifier = nn.Sequential(
            nn.Dropout(), nn.Linear(256 * 6 * 6, 4096), nn.ReLU(inplace=True),
            nn.Dropout(), nn.Linear(4096, 4096), nn.ReLU(inplace=True), nn.Linear(4096, num_classes))
```

# The sequential module - forward() method

```
def forward(self, x):  
    x = self.features(x)  
    x = self.avgpool(x)  
    x = x.view(x.size(0), 256 * 6 * 6)  
    x = self.classifier(x)  
    return x
```

# Let's practice!

INTRODUCTION TO DEEP LEARNING WITH PYTORCH

# The problem of overfitting

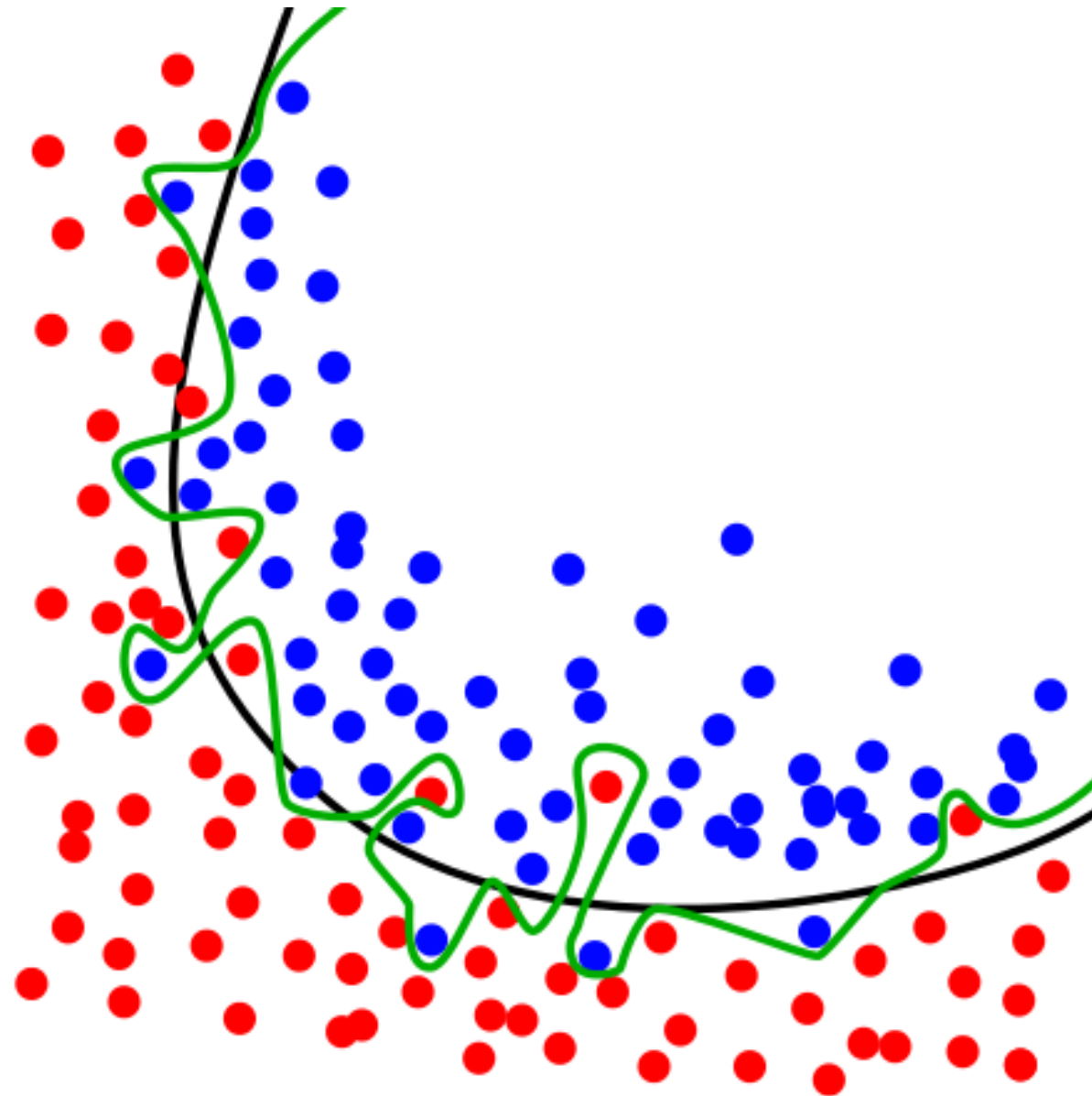
INTRODUCTION TO DEEP LEARNING WITH PYTORCH



Ismail Elezi

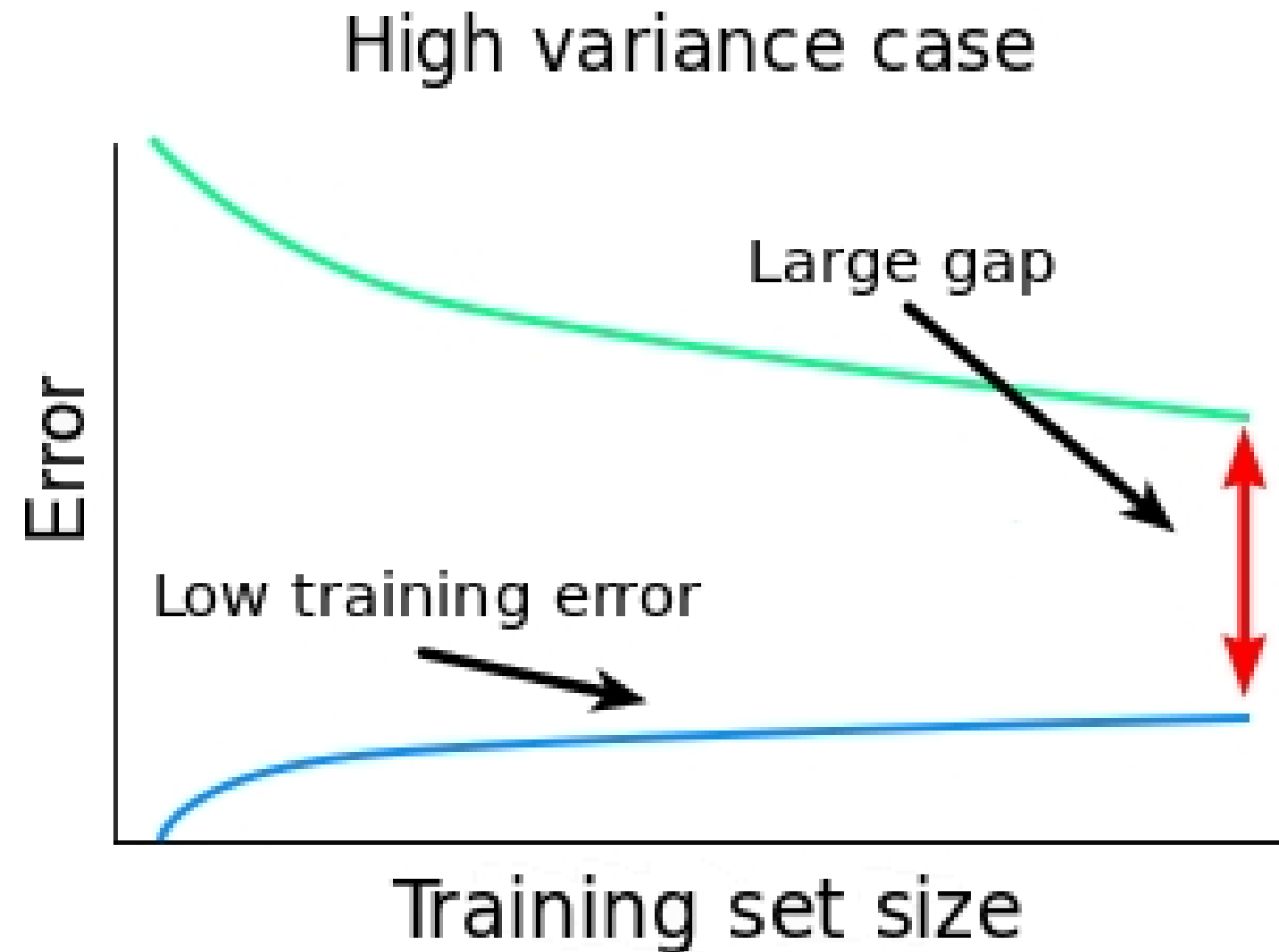
Ph.D. Student of Deep Learning

# Overfitting

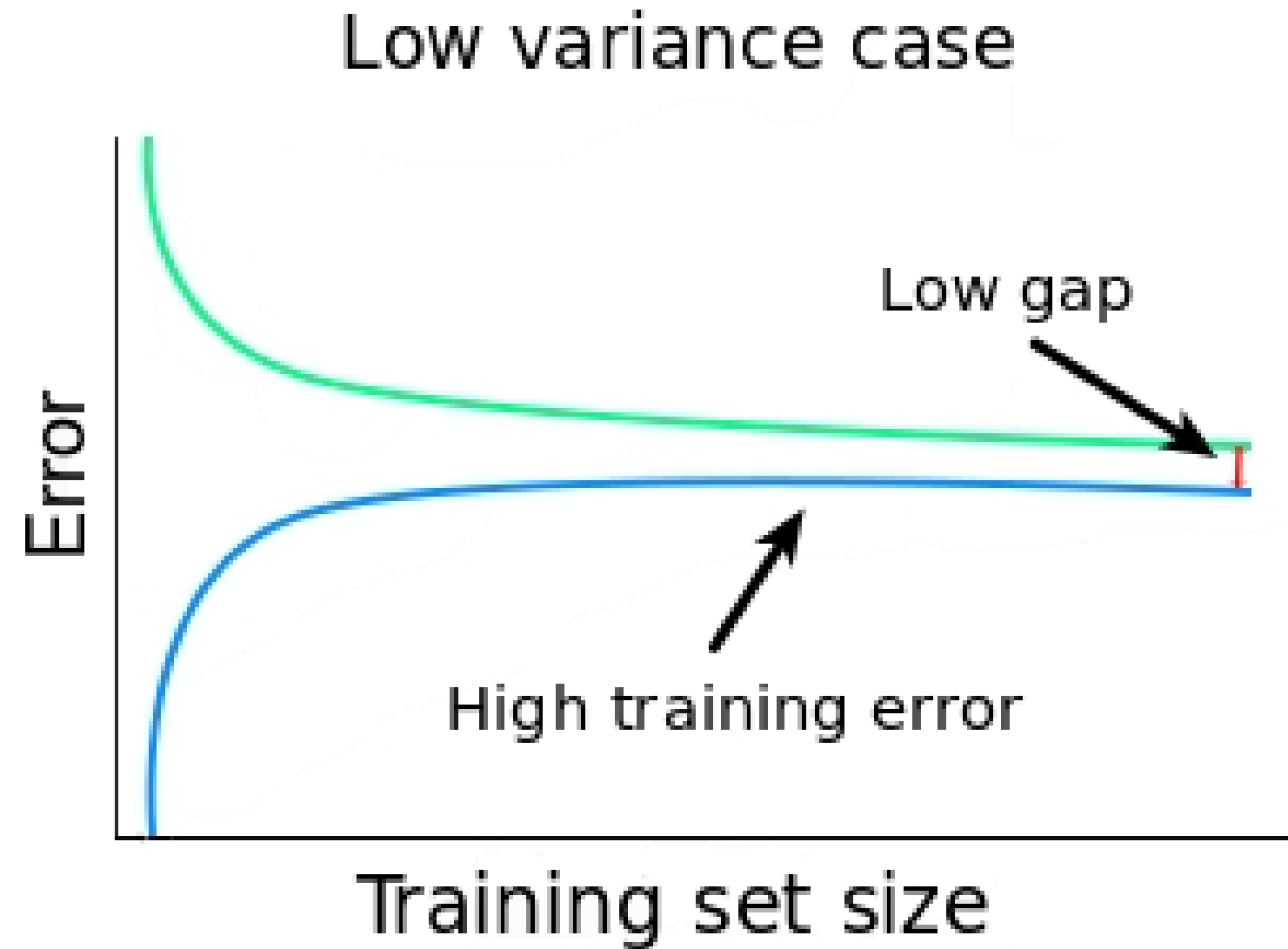




# Detecting overfitting



# Detecting overfitting



# Overfitting in the testing set

- Training set
- Validation set
- Testing set

# Validation set

- Training set: train the model
- Validation set: select the model
- Testing set: test the model

# Using validation sets in PyTorch

```
indices = np.arange(50000)
np.random.shuffle(indices)

train_loader = torch.utils.data.DataLoader(
    datasets.CIFAR10(root='./data', train=True, download=True,
                     transform=transforms.Compose([transforms.ToTensor(),
                                                    transforms.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225))])),
    batch_size=1, shuffle=False, sampler=torch.utils.data.SubsetRandomSampler(indices[:45000]))

val_loader = torch.utils.data.DataLoader(
    datasets.CIFAR10(root='./data', train=True, download=True,
                     transform=transforms.Compose([transforms.ToTensor(),
                                                    transforms.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225))])),
    batch_size=1, shuffle=False, sampler=torch.utils.data.SubsetRandomSampler(indices[45000:50000]))
```

# Let's practice!

INTRODUCTION TO DEEP LEARNING WITH PYTORCH

# Regularization techniques

INTRODUCTION TO DEEP LEARNING WITH PYTORCH



Ismail Elezi

Ph.D. Student of Computer Science

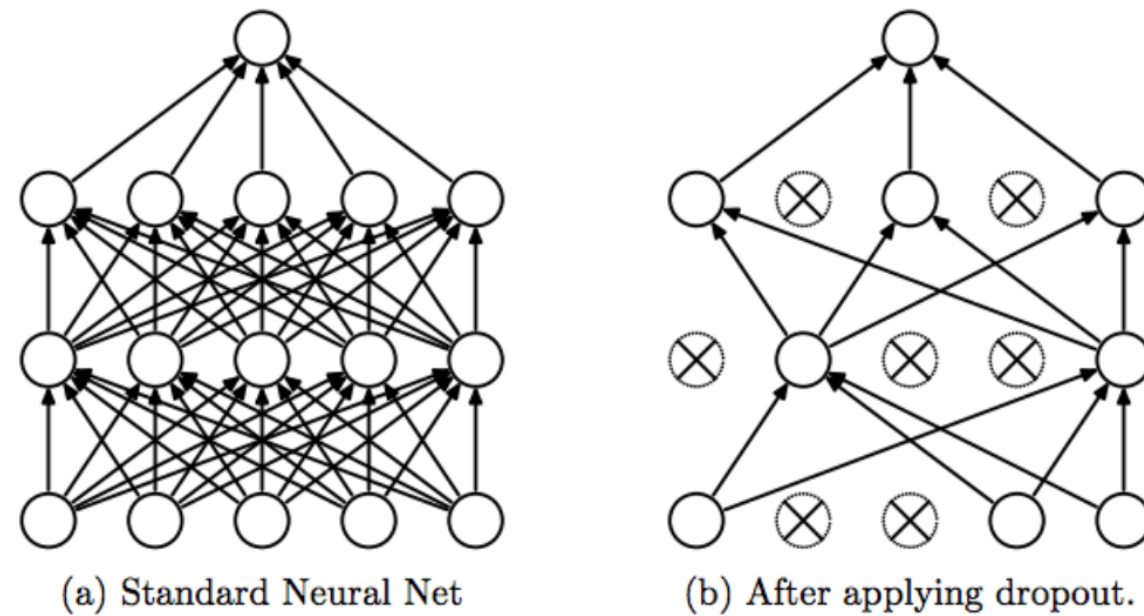
# L2-regularization

$$C = -\frac{1}{n} \sum_{x_j} \left[ y_j \ln a_j^L + (1 - y_j) \ln(1 - a_j^L) \right] + \frac{\lambda}{2n} \sum_w w^2$$

```
optimizer = optim.Adam(net.parameters(), lr=3e-4, weight_decay=0.0001)
```



# Dropout



<sup>1</sup> Srivastava, Hinton, Krizhevsky, Sutskever, Salakhutdinov; Dropout: a simple way to prevent neural networks from overfitting, JMLR 2014

# Dropout in AlexNet - PyTorch code

```
self.classifier = nn.Sequential(  
    nn.Dropout(p=0.5),  
    nn.Linear(256 * 6 * 6, 4096),  
    nn.ReLU(inplace=True),  
    nn.Dropout(p=0.5),  
    nn.Linear(4096, 4096),  
    nn.ReLU(inplace=True),  
    nn.Linear(4096, num_classes),  
)
```

# Batch-normalization

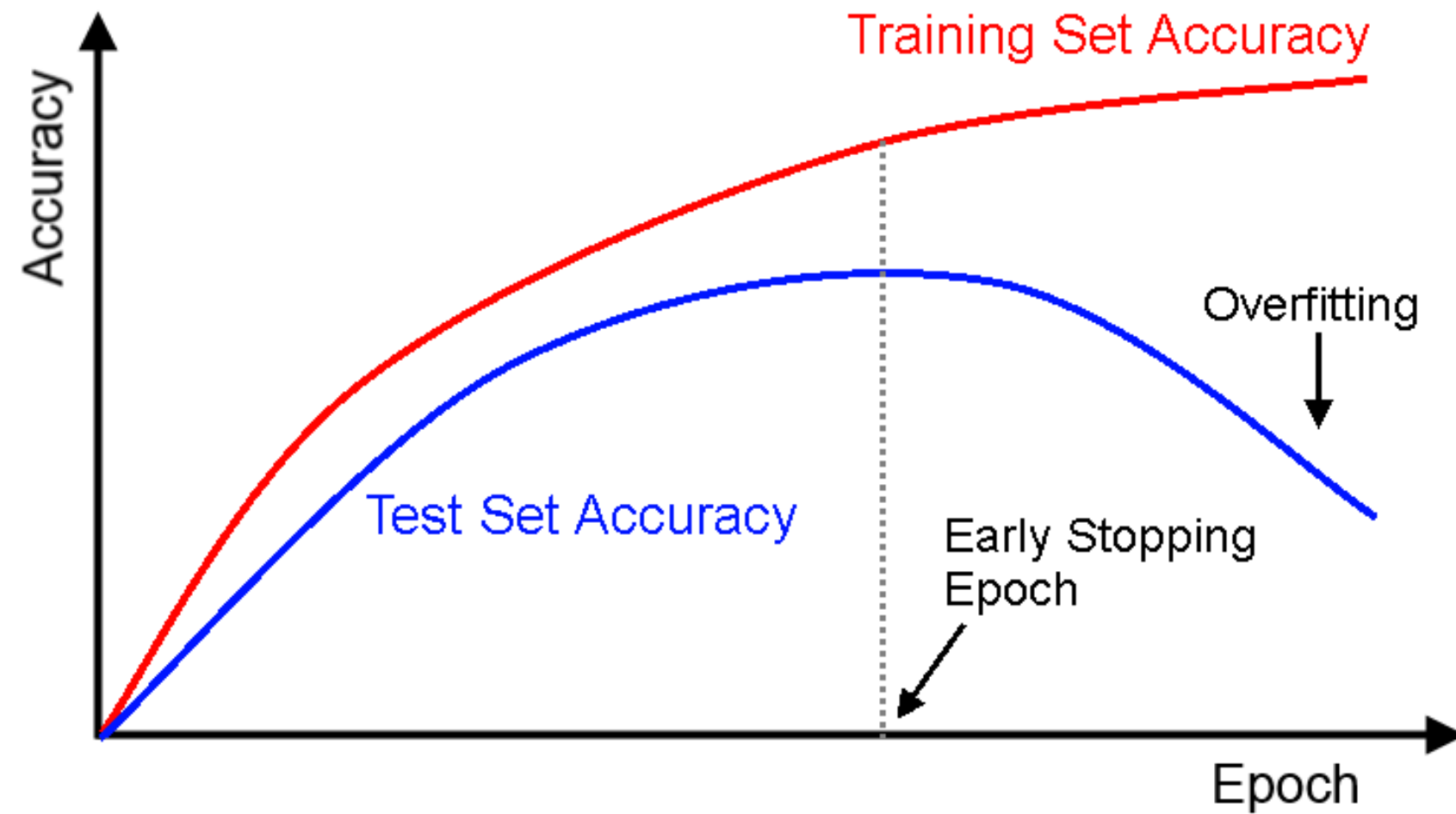
**Input:** Values of  $x$  over a mini-batch:  $\mathcal{B} = \{x_{1..m}\}$ ;  
Parameters to be learned:  $\gamma, \beta$   
**Output:**  $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad // \text{ mini-batch mean}$$
$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad // \text{ mini-batch variance}$$
$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad // \text{ normalize}$$
$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) \quad // \text{ scale and shift}$$

```
self.bn = nn.BatchNorm2d(num_features=64, eps=1e-05, momentum=0.9)
```

<sup>1</sup> Ioffe and Szegedy; Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, ICML 2015

# Early-stopping



# Hyperparameters

Question: How to choose all these hyperparameters (l2 regularization, dropout parameter, optimizers (Adam vs gradient descent etc), batch-norm momentum and epsilon, number of epochs for early stopping etc)?

Answer: Train many networks with different hyperparameters (typically use random values for them), and test them in the validation set. Then use the best performing net in the validation set to know the expected accuracy of the network in new data.

# Eval() mode

```
# Sets the net in train mode  
model.train()
```

```
# Sets the net in evaluation mode  
model.eval()
```

# Let's practice!

INTRODUCTION TO DEEP LEARNING WITH PYTORCH

# Transfer learning

INTRODUCTION TO DEEP LEARNING WITH PYTORCH

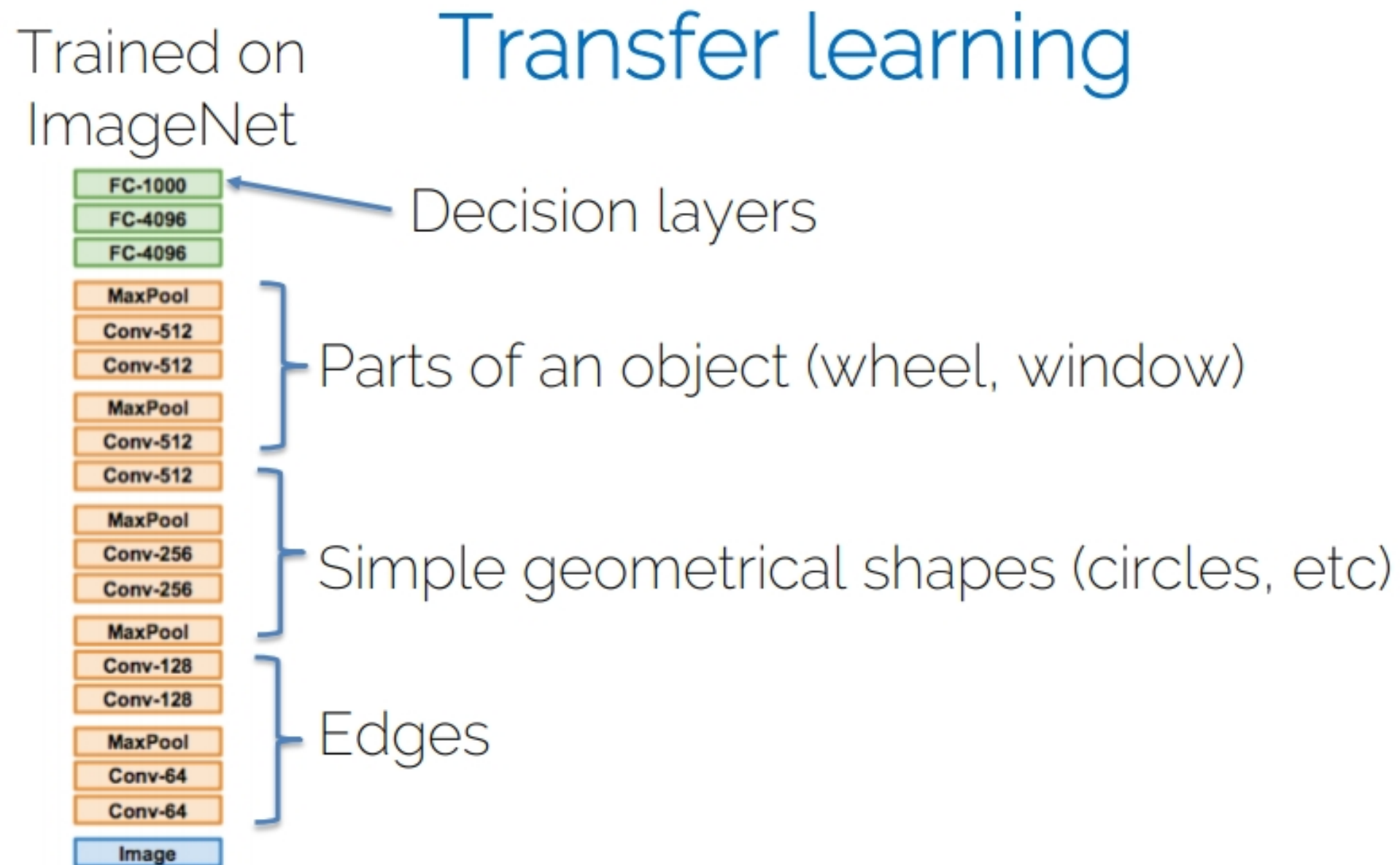


Ismail Elezi

Ph.D. Student of Deep Learning



# Features in CNN

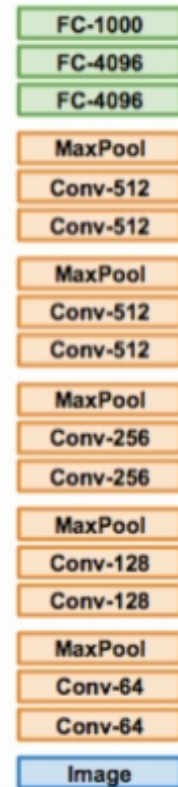


Prof. Leal-Taixé and Prof. Niessner

Donahue 2014, Razavian 2014

# Transfer Learning

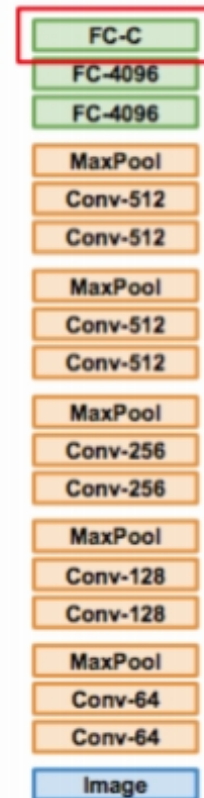
Trained on  
ImageNet



Prof. Leal-Taixé and Prof. Niessner

Transfer learning

TRAIN



New dataset  
with C classes

FROZEN

Donahue 2014, Razavian 2014

# Finetuning in PyTorch

```
# Instantiate the model
model = Net()

# Load the parameters from the old model
model.load_state_dict(torch.load('cifar10_net.pth'))

# Change the number of out channels
model.fc = nn.Linear(4 * 4 * 1024, 100)

# Train and evaluate the model
model.train()
```

# Freezing the layers

```
# Instantiate the model
model = Net()

# Load the parameters from the old model
model.load_state_dict(torch.load('cifar10_net.pth'))

# Freeze all the layers bar the final one
for param in model.parameters():
    param.requires_grad = False

# Change the number of output units
model.fc = nn.Linear(4 * 4 * 1024, 100)

# Train and evaluate the model
model.train()
```

# Torchvision library

```
import torchvision  
  
model = torchvision.models.resnet18(pretrained=True)  
  
model.fc = nn.Linear(512, num_classes)
```

# Let's practice!

INTRODUCTION TO DEEP LEARNING WITH PYTORCH

# Congratulations!

INTRODUCTION TO DEEP LEARNING WITH PYTORCH



**Ismail Elezi**

Ph.D. Student of Deep Learning

# What you have learned

- Differences between neural networks and other types of classifiers.
- Building, training and testing basic neural networks.
- Building, training, testing and using convolutional neural networks.
- Detecting and preventing overfitting.
- Using the finetuning technique.
- Using the torchvision module to use pre-trained networks.



# Thank you!

INTRODUCTION TO DEEP LEARNING WITH PYTORCH