ETH zürich | University of Zurich[UZH] | EXCITE ZURICH IMAGING

Bastian Wittmann PhD candidate

Research Interest: Graph Neural Networks, Vision Transformers, Blood Vessel Graphs
Google Scholar: https://scholar.google.com/citations?user=0mxtU-cAAAAJ&hl=en&oi=ao
GitHub: https://github.com/bwittmann
LinkedIn: https://ch.linkedin.com/in/bastian-wittmann-7830731b4

Medical Image **Classification** and Segmentation with Deep Learning - Part 1

Prof. Dr. Bjoern Menze
Bastian Wittmann

Biomedical Image Analysis and Machine Learning @ UZH

Agenda

PART 1: Medical Image Classification - Bastian Wittmann

1) Brief Introduction to Machine Learning and Deep Learning
2) E1: Hands-On Session
   a) Introduction to PyTorch (see notebook section 0)
   b) Preparing Data (see notebook section 1)
   c) Implementation of a Fully Connected Network (see notebook section 2)

3) Convolutional Neural Networks and ResNet
4) E2: Hands-On Session
   a) Implementation of a simple Convolutional Neural Network (see notebook section 3)
   b) Experiments with Deep Residual Networks (see notebook section 4)
   c) Optional: Interpretability of Predictions (see notebook section 5)

12am - 1pm: Lunch Break

PART 2: Medical Image Segmentation - Paul Bueschl

4pm: End of Practical Session

Colab Notebooks

E1
0) Introduction to PyTorch
1) Preparing the Data
2) Model 1 - Fully Connected Network (FCN/MLP)

E2
3) Model 2 - Simple Convolutional Neural Network (CNN)
4) Model 3 - Deep Residual Neural Network (ResNet)
5) Interpretability of Predictions

Please ask whenever you have any open questions!

We provide **two** versions of our notebook
1) version without solutions: here (recommended version)
2) version with solutions: here (solely if you feel super overwhelmed*)

**This should be considered the 'last resort'. Please ask us for help first.*

Big Picture

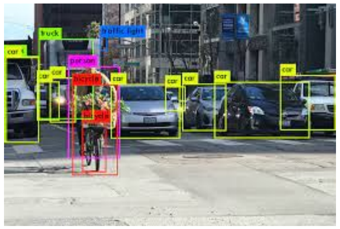
various different input formats

...

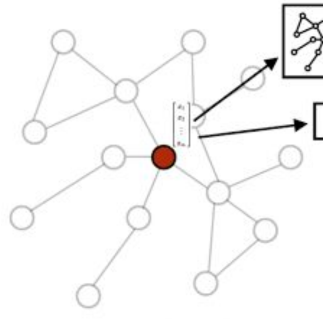
blackbox

Inputs → Neural Network → Outputs

various different output formats



dog: 0.9



...

Classification

x-ray image of lung

x_i

Inputs

blackbox $\boldsymbol{\theta}$

Neural Network

$\hat{y}_i$

Outputs

predict whether patient has bacterial pneumonia

0.9 or 90%

Linear Classification

blackbox $\boldsymbol{\theta}$

x_i

Inputs → Linear Classifier → Outputs

$\hat{y}_i$

0.9 or 90%

image i

predicted class corresponding to image i

how to divide image into different features that we can use for classification?

how can we create a model representative of our neural network?

how can we ensure that our network delivers predictions in the range of [0, 1]?

three main questions we have to solve for any classifier

Linear Classification

blackbox $\boldsymbol{\theta}$

x_i → Inputs → Linear Classifier → Outputs $\hat{y}_i$ 0.9 or 90%

image i

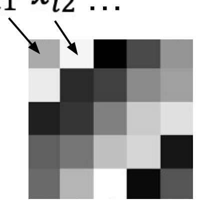
predicted class corresponding to image i

how to divide image into different features that we can use for classification?

how can we create a model representative of our neural network?

how can we ensure that our network delivers predictions in the range of [0, 1]?

x_{i1} x_{i2} ...



Linear Classification

blackbox θ

x_i

Inputs

image i

Linear Classifier

$\hat{y}_i$

Outputs

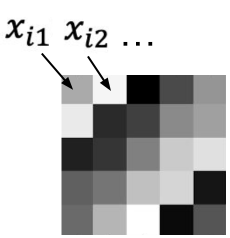
0.9 or 90%

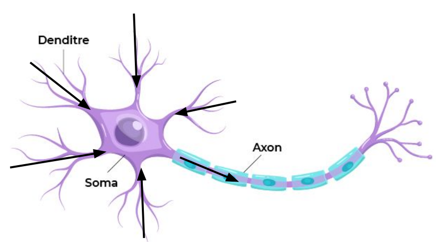
predicted class corresponding to image i

how to divide image into different features that we can use for classification?

how can we create a model representative of our neural network?

how can we ensure that our network delivers predictions in the range of [0, 1]?





Linear Classification

blackbox $\boldsymbol{\theta}$

x_i Inputs → Linear Classifier → $\hat{y}_i$ Outputs 0.9 or 90%

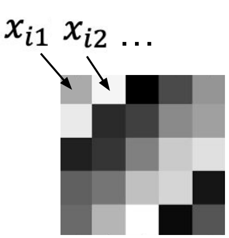
image i

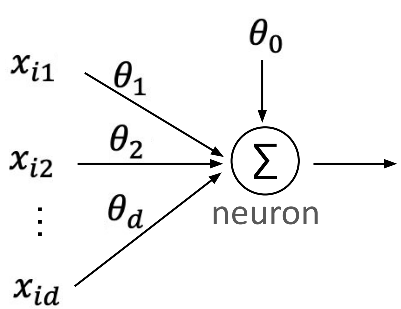
predicted class corresponding to image i

how to divide image into different features that we can use for classification?

how can we create a model representative of our neural network?

how can we ensure that our network delivers predictions in the range of [0, 1]?





single layer perceptron

Linear Classification

blackbox $\boldsymbol{\theta}$

x_i

Inputs → Linear Classifier → Outputs

$\hat{y}_i$

0.9 or 90%

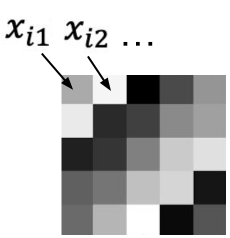
image i

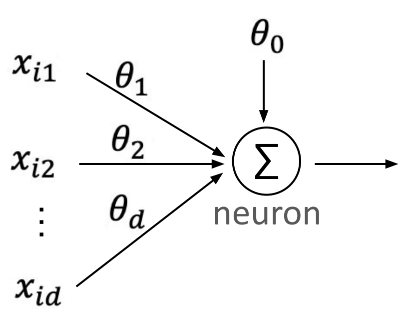
predicted class corresponding to image i

how to divide image into different features that we can use for classification?

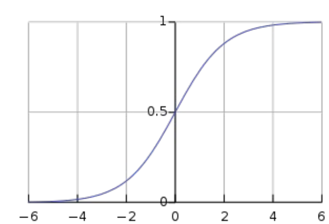
how can we create a model representative of our neural network?

how can we ensure that our network delivers predictions in the range of [0, 1]?





single layer perceptron



$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Linear Classification

blackbox $\boldsymbol{\theta}$

x_i → **Inputs** → **Linear Classifier** → **Outputs** $\hat{y}_i$ → 0.9 or 90%

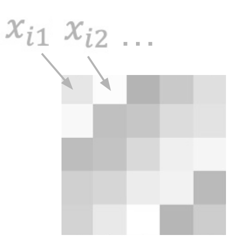
image i

predicted class corresponding to image i

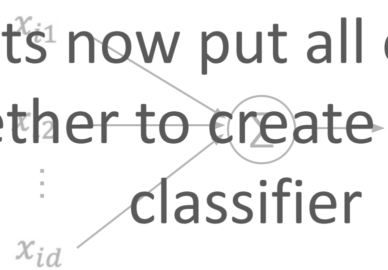
how to divide image into different features that we can use for classification?

how can we create a model representative of our neural network?

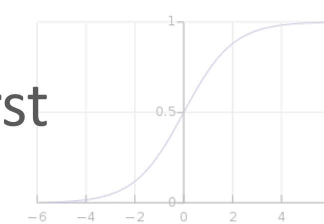
how can we ensure that our network delivers predictions in the range of [0, 1]?



Lets now put all of this together to create our first classifier

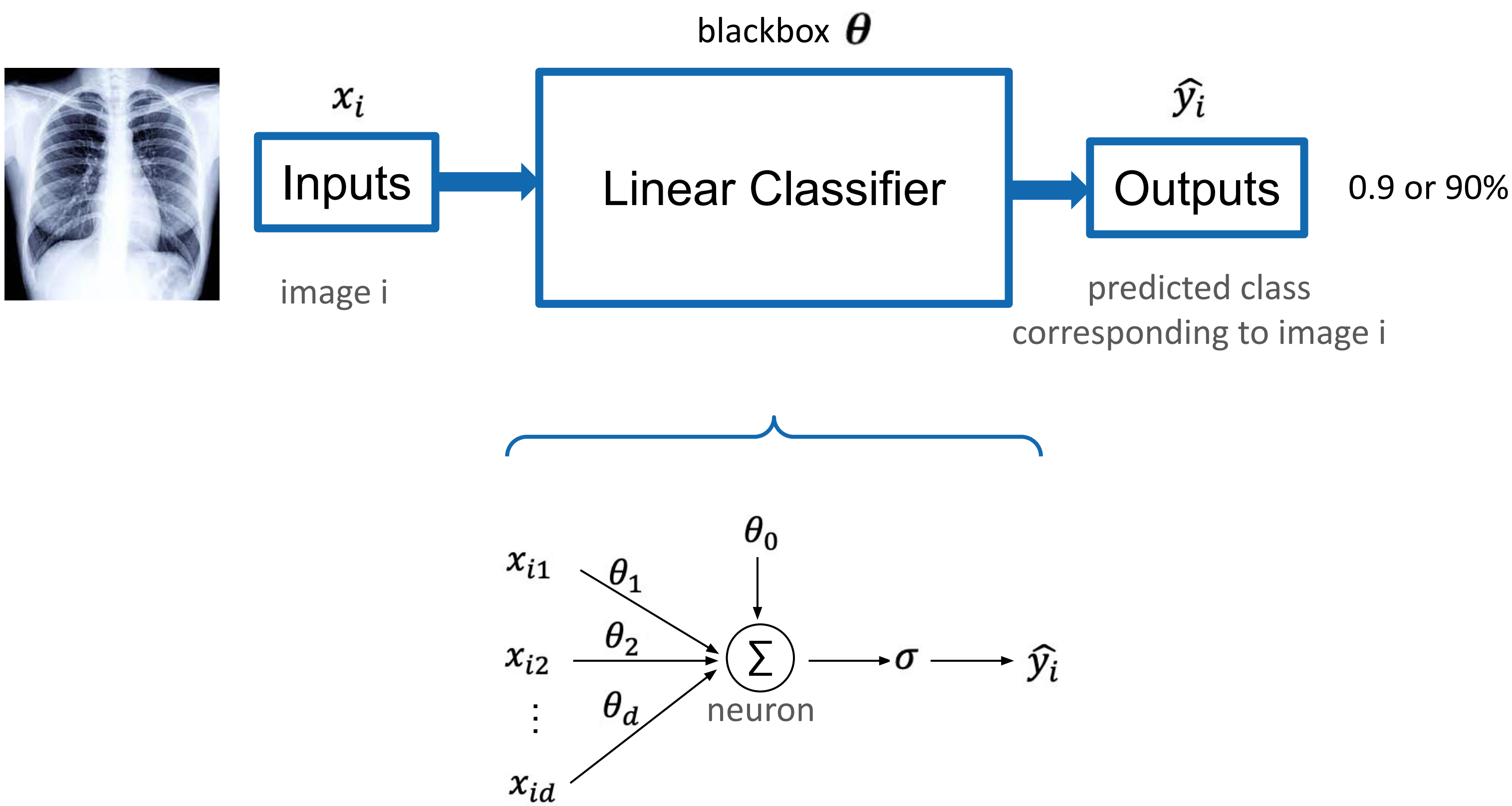


single layer perceptron



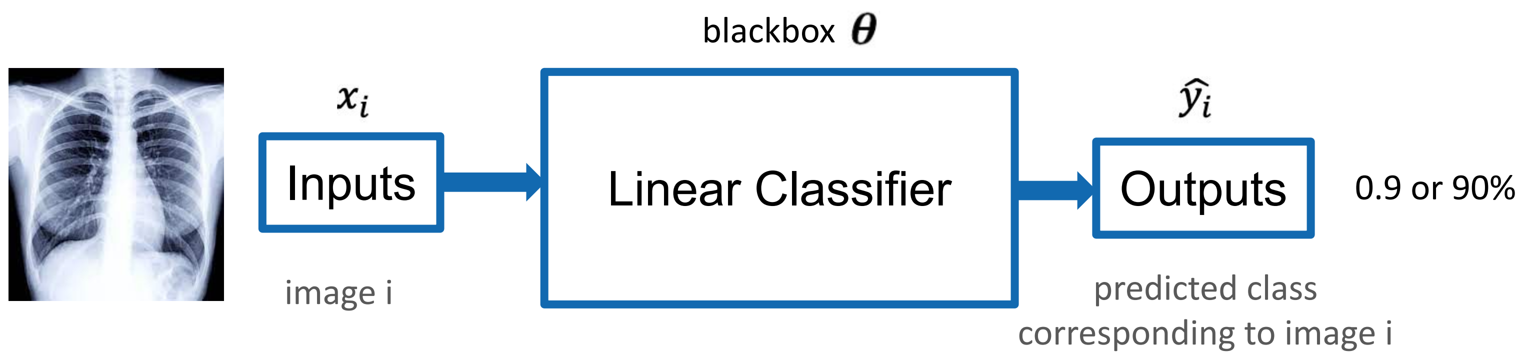
$$\sigma(x) = \frac{1}{1+e^{-x}}$$

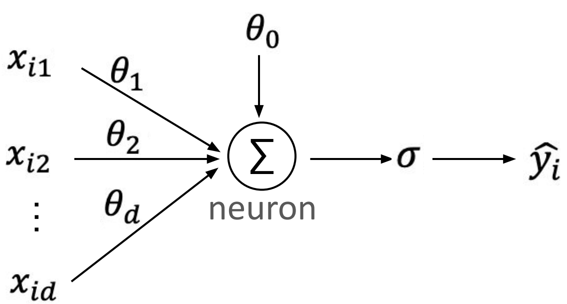
Linear Classification



Logistic Regression:
a 1 layer, 1 neuron neural network

Linear Classification





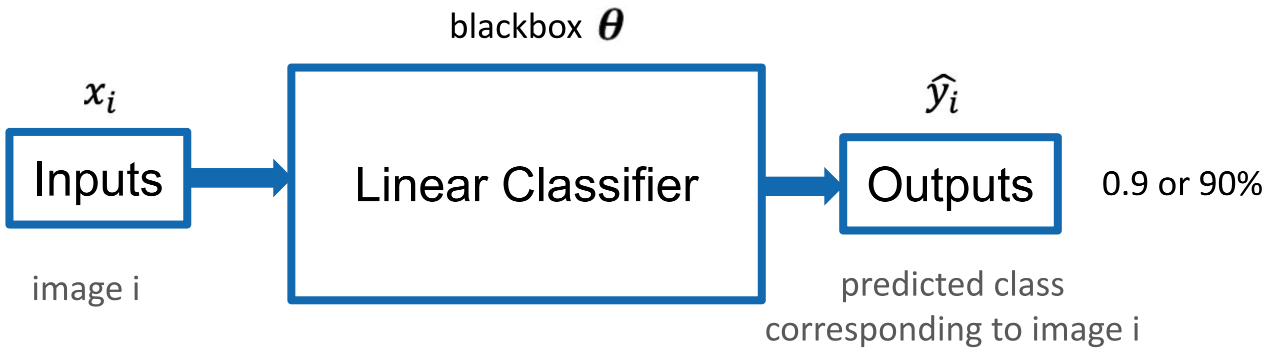
Logistic Regression:
a 1 layer, 1 neuron neural network

$$\hat{y}_i = \sigma(\underbrace{x_i\boldsymbol{\theta}+\theta_0})$$

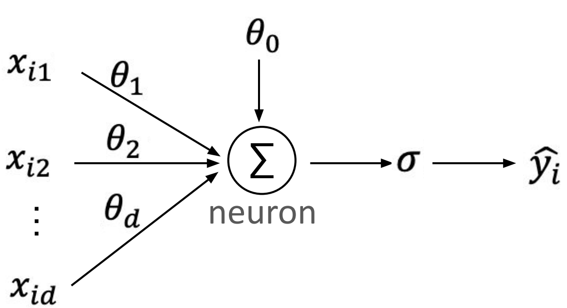
$$\theta_0 + \sum_{j=1}^{d} x_{ij}\theta_j = \theta_0 + x_{i1}\theta_1 + x_{i2}\theta_2 + \cdots + x_{id}\theta_d$$

what are now all of
these parameters ϴ?

Linear Classification



the models parameters ϴ are what we can actually modify to obtain better performances for our classification task!



Logistic Regression:
a 1 layer, 1 neuron neural network

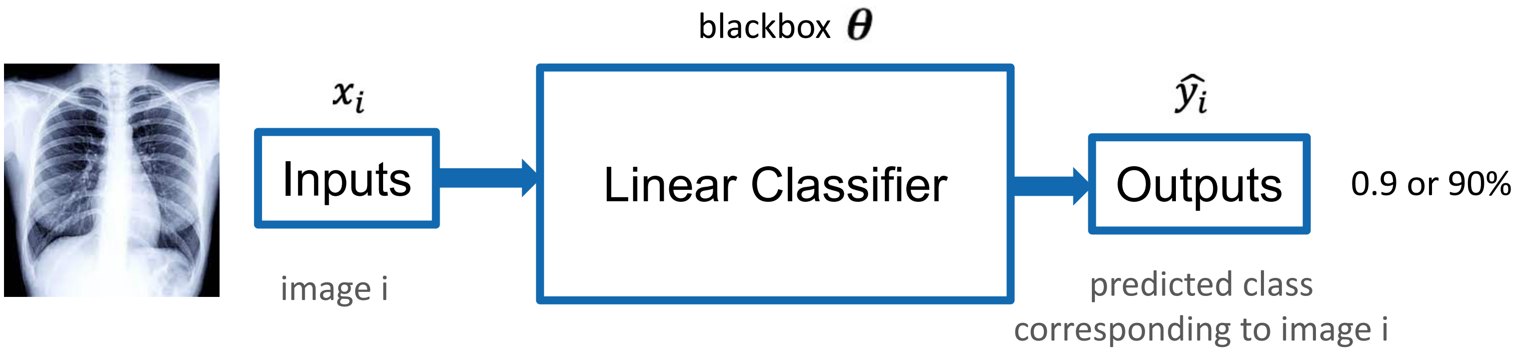
$$\hat{y}_i = \sigma(\underbrace{x_i\boldsymbol{\theta}+\theta_0})$$

$$\theta_0 + \sum_{j=1}^{d} x_{ij}\theta_j = \theta_0 + x_{i1}\theta_1 + x_{i2}\theta_2 + \cdots + x_{id}\theta_d$$

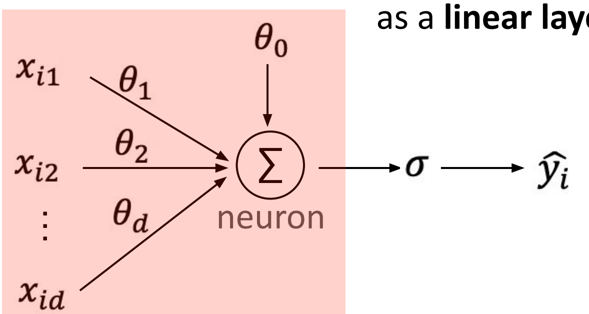
this is what is commonly referred to as **training** a neural network.

more on this later…

Linear Classification



this is, therefore, typically referred to as a **linear layer**



$$\hat{y}_i = \sigma(\underbrace{x_i\boldsymbol{\theta}+\theta_0})$$

$$\theta_0 + \sum_{j=1}^{d} x_{ij}\theta_j = \theta_0 + x_{i1}\theta_1 + x_{i2}\theta_2 + \cdots + x_{id}\theta_d$$

Logistic Regression:
a 1 layer, 1 neuron neural network

Linear Classification

blackbox $\boldsymbol{\theta}$

x_i

Inputs

image i

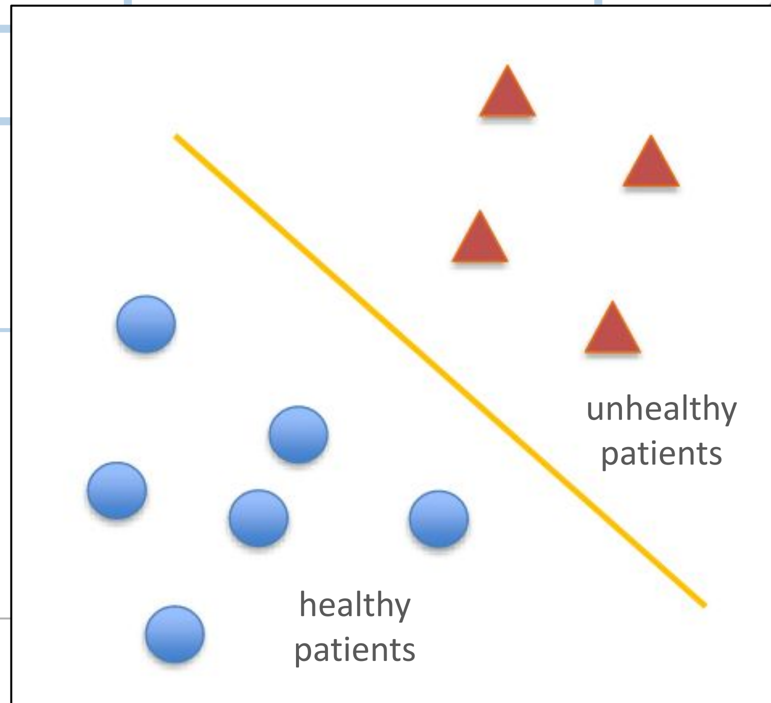
$\hat{y}_i$

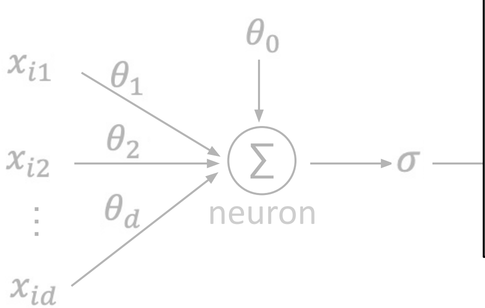
puts

0.9 or 90%

ed class

ng to image i





$x_i\boldsymbol{\theta})$

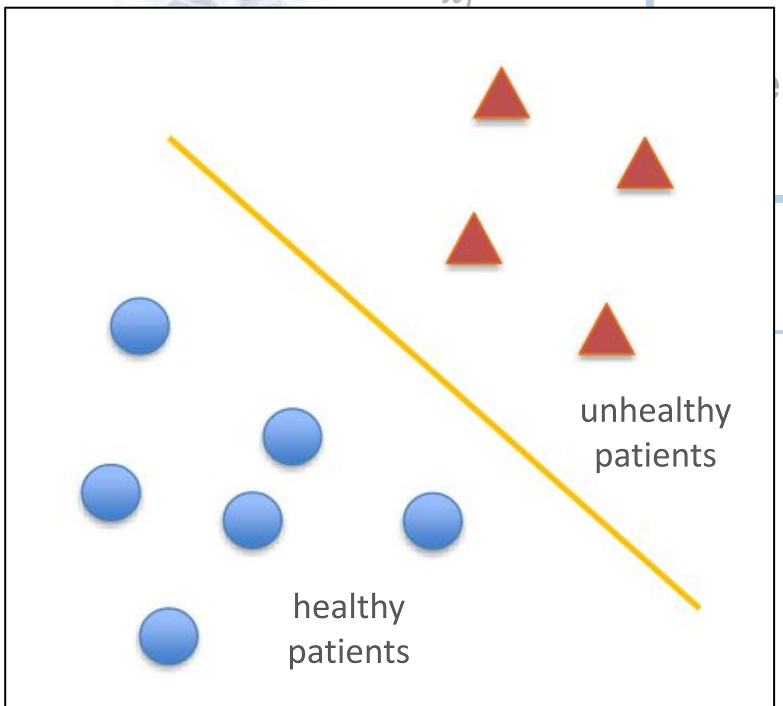
$+ x_{i1}\theta_1 + x_{i2}\theta_2 + \cdots + x_{id}\theta_d$

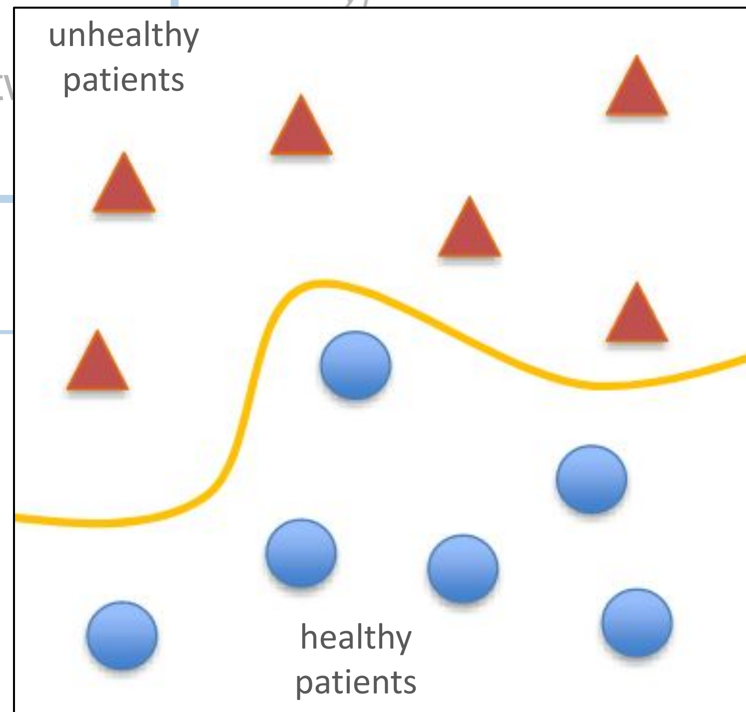
$j=1$

Logistic Regression:
a 1 neuron neural network

Linear Classification

blackbox θ

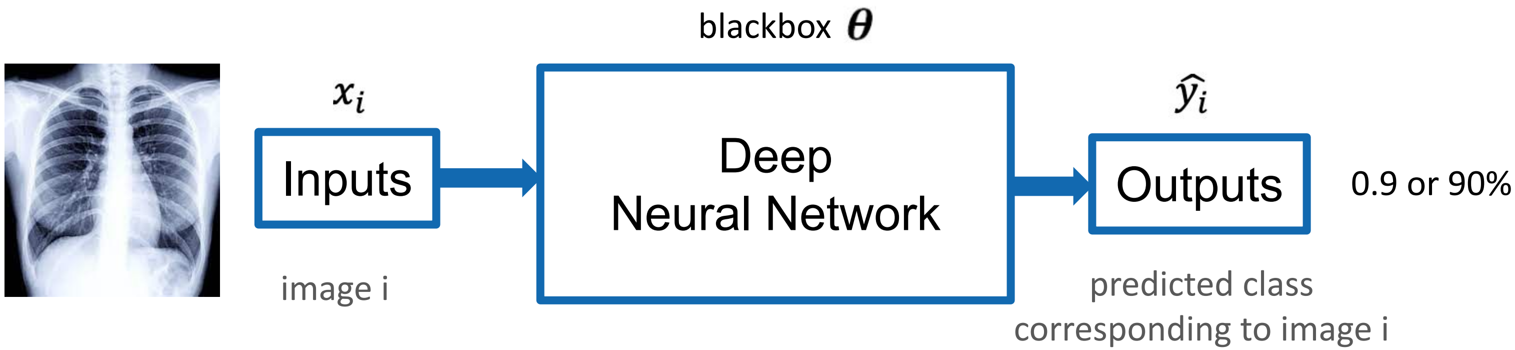




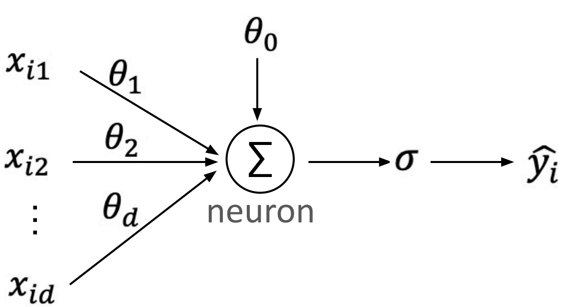
A linear classifier can solely learn a linear decision boundary!

Logistic Regression:
a 1 neuron neural network

Neural Networks



$$\hat{y}_i = \sigma(x_i \boldsymbol{\theta})$$



Logistic Regression:
a 1 layer, 1 neuron neural network

how can we
transform our simple
linear classifier to
learn complex
non-linear functions?

Neural Networks

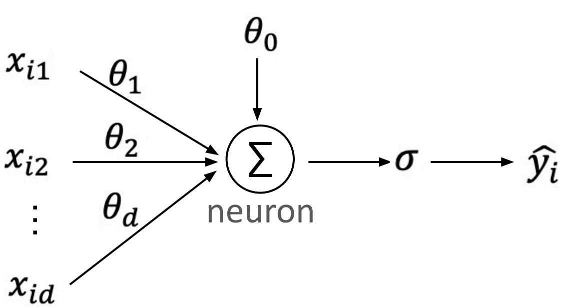
blackbox $\boldsymbol{\theta}$

x_i Inputs → Deep Neural Network → Outputs $\hat{y}_i$ 0.9 or 90%

image i

predicted class corresponding to image i

$$\hat{y}_i = \sigma(x_i\boldsymbol{\theta})$$



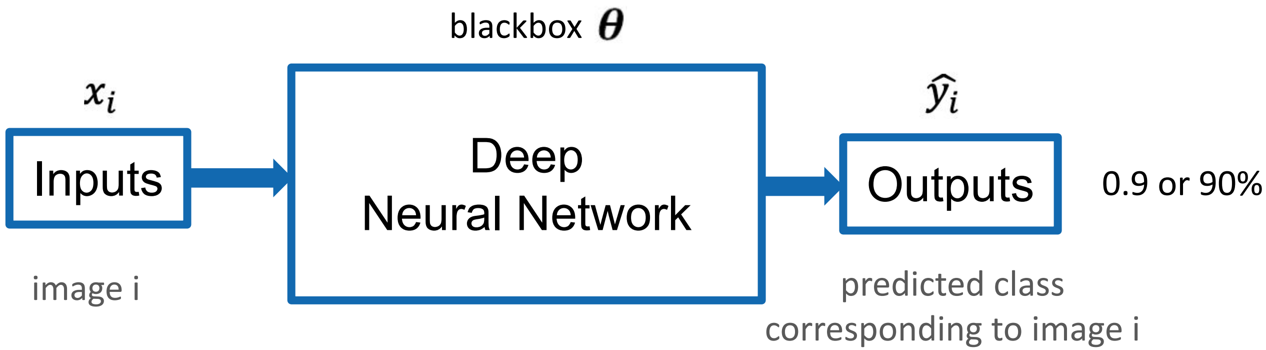
Logistic Regression:
a 1 layer, 1 neuron neural network

separate the neural network's parameters via non-linearities to enable us to learn a more complex classifier!

$$\hat{y}_i = \sigma(x_i\boldsymbol{\theta}) \longrightarrow \hat{y}_i = \sigma(\sigma(x_i\boldsymbol{\theta})\boldsymbol{\theta})$$

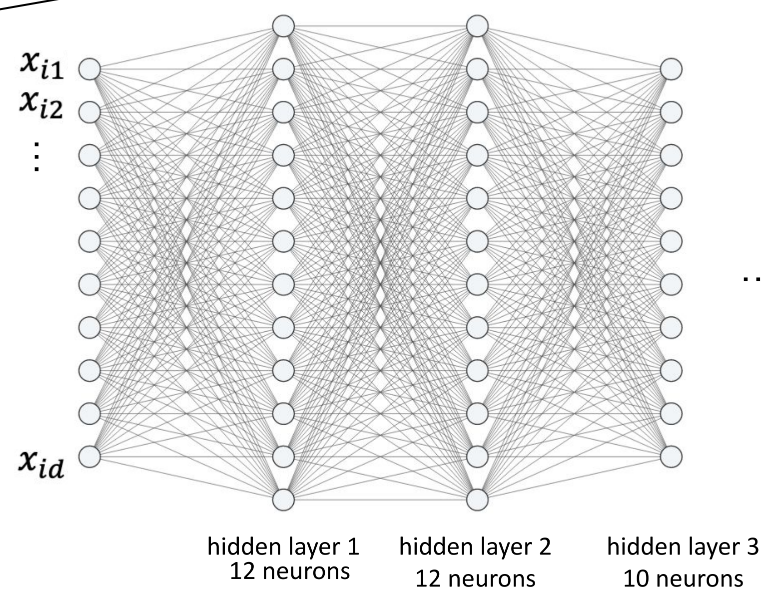
Non-linear Classifier:
a 2 layer neural network

Neural Networks



$$\hat{y}_i = \boldsymbol{W}_5 \sigma(\boldsymbol{W}_4 \tanh(\boldsymbol{W}_3, \max(\boldsymbol{0}, \boldsymbol{W}_2 \max(\boldsymbol{0}, \boldsymbol{W}_1 x_i))))$$

lernable
parameters $\boldsymbol{\theta}$



a neural network is comprised of multiple layers separated by non-linearities

this results in **deep neural networks**
(that's why it's called deep learning)

Neural Networks

blackbox $\boldsymbol{\theta}$

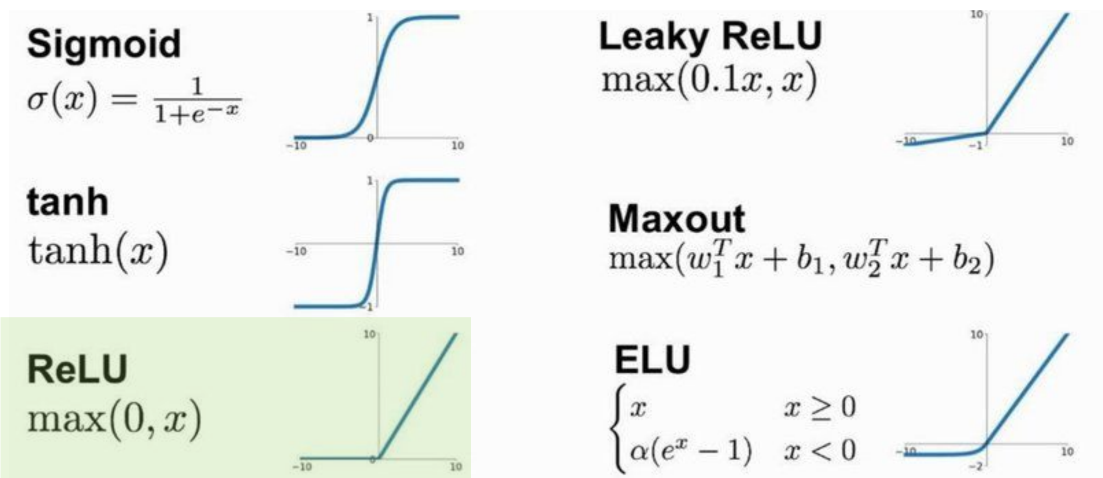
x_i → **Inputs** → **Deep Neural Network** → **Outputs** $\hat{y}_i$ 0.9 or 90%

image i

predicted class corresponding to image i

there exists a whole zoo of non-linearities (or activations).

ReLU is typically the standard choice.
(best convergence, no vanishing gradients)



rectified linear unit

The Dataset

blackbox $\boldsymbol{\theta}$

x_i

Inputs

image i

Deep Neural Network

$\hat{y}_i$

Outputs

0.9 or 90%

predicted class corresponding to image i

$\{x_{1:n}, y_{1:n}\}$

whole annotated dataset (images + ground truth)

first, we split the dataset into three splits

the training, validation, and test set

60% train	20% validation	20% test
train the neural network's parameters $\boldsymbol{\theta}$	validate our design choices	test performance on hidden test set

we don't use this data to optimize the parameters!!!

How to Train the Neural Network

blackbox $\boldsymbol{\theta}$

x_i Inputs → Deep Neural Network → $\hat{y}_i$ Outputs 0.51 | y_i Target 1

prediction should be as close as possible to ground truth!

we need a way to mathematically describe how close our predictions are to the target!

all images in our train data

binary cross-entropy (BCE): $$\underbrace{L(\boldsymbol{y}, \hat{\boldsymbol{y}};\ \boldsymbol{\theta})}_{\text{train loss}} = -\sum_{i=1}^{n} (y_i \cdot \log \hat{y}_i + (1 - y_i) \cdot \log[1 - \hat{y}_i])$$

prediction of image i — ground truth

How to Train the Neural Network

blackbox $\boldsymbol{\theta}$

x_i Inputs → Deep Neural Network → $\hat{y}_i$ Outputs | y_i Target

0.01 | 1

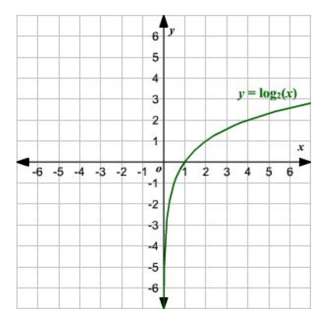
prediction should be as close as possible to ground truth!

we want a **large loss** for **bad performances**

here we predict the complete opposite

this should be penalized with a high loss

$$\mathcal{L}(\hat{y}_i, y_i) = \overbrace{-[\underset{\uparrow \atop 1}{y_i} \log \underset{\uparrow \atop 0.01}{\hat{y}_i}}^{2.0000} + \cancel{(1 - y_i)\log(1 - \hat{y}_i)}]$$



How to Train the Neural Network

blackbox $\boldsymbol{\theta}$

x_i Inputs → Deep Neural Network → Outputs $\hat{y}_i$ | Target y_i

Outputs: 0.99 Target: 1

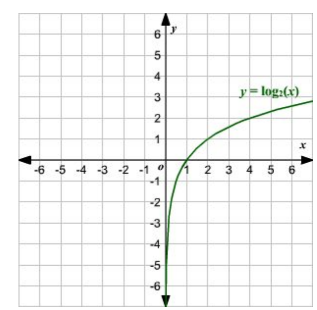
prediction should be as close as possible to ground truth!

we want a **small loss** for **good performances**

here we predict correct

this should be rewarded with a low loss

$$\mathcal{L}(\hat{y}_i, y_i) = \overbrace{-[\underset{1}{y_i} \log \underset{0.99}{\hat{y}_i}}^{0.0044} + \cancel{(1 - y_i)\log(1 - \hat{y}_i)}]$$



How to Train the Neural Network

blackbox $\boldsymbol{\theta}$

x_i Inputs → Deep Neural Network → $\hat{y}_i$ Outputs | y_i Target

Outputs: 0.99 | Target: 1

prediction should be as close as possible to ground truth!

the smaller the loss, the better the performance of our classifier on the train set

$$L(\boldsymbol{y}, \hat{\boldsymbol{y}};\ \boldsymbol{\theta}) = -\sum_{i=1}^{n}(y_i \cdot \log \hat{y}_i + (1 - y_i) \cdot \log[1 - \hat{y}_i])$$

⇒ we aim to minimize the loss with respect to the trainable parameters ϴ

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} L(\boldsymbol{y}, \hat{\boldsymbol{y}};\ \boldsymbol{\theta})$$

we **train our NN** by updating its parameters to fit the task at hand

However, no closed-form solution ⇒ iterative optimization methods: stochastic gradient descent

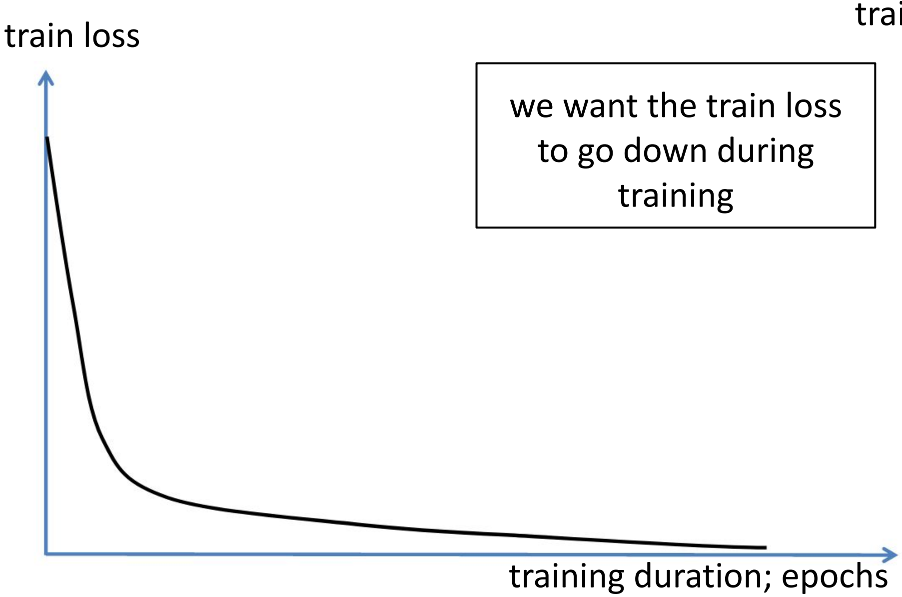
How to Train the Neural Network

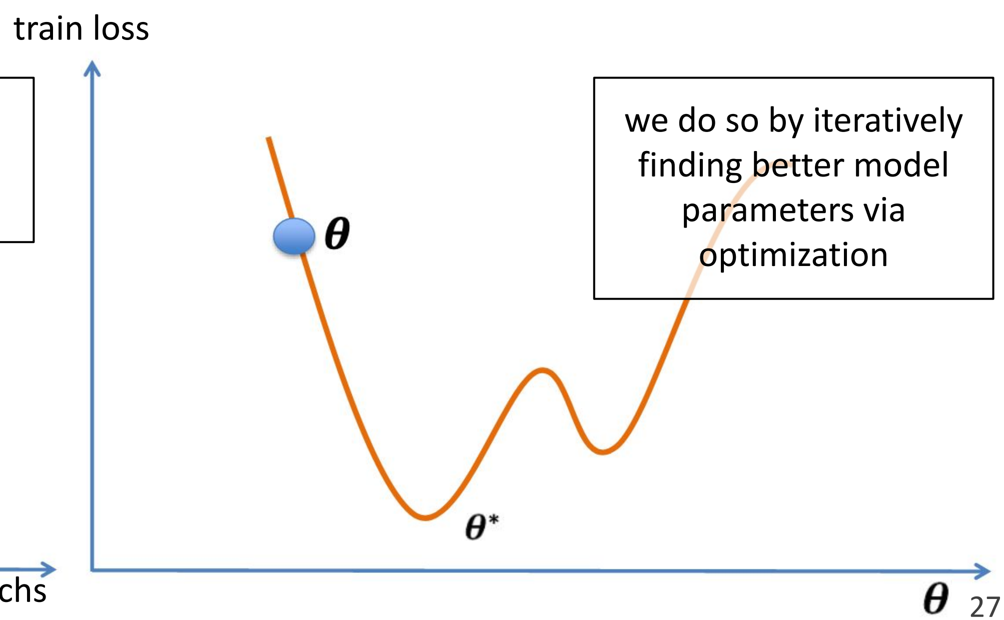
blackbox $\boldsymbol{\theta}$

x_i **Inputs** → **Deep Neural Network** → $\hat{y}_i$ **Outputs** | y_i **Target**

Outputs: 0.99 | Target: 1

prediction should be as close as possible to ground truth!





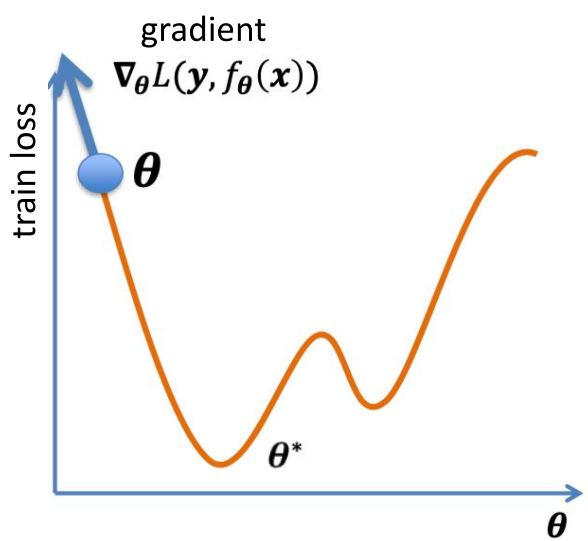
How to Train the Neural Network

blackbox $\boldsymbol{\theta}$

x_i Inputs → Deep Neural Network → $\hat{y}_i$ Outputs, y_i Target

0.99 | 1

prediction should be as close as possible to ground truth!



gradient points towards the direction of greatest increase

we iteratively minimize the loss $L(\boldsymbol{y}, f_{\boldsymbol{\theta}}(\boldsymbol{x}))$ wrt. $\boldsymbol{\theta}$ using **stochastic gradient descent**!

learning rate (each time step we just make a small update)

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha \nabla_{\boldsymbol{\theta}} L(\boldsymbol{y}, f_{\boldsymbol{\theta}_t}(\boldsymbol{x}))$$

we iteratively update our parameters until convergence

$$\nabla_{\boldsymbol{\theta}} L = \frac{1}{m} \sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} L_i$$

we approximate the gradient over a mini-batch for faster convergence

(*stochastic* gradient descent)

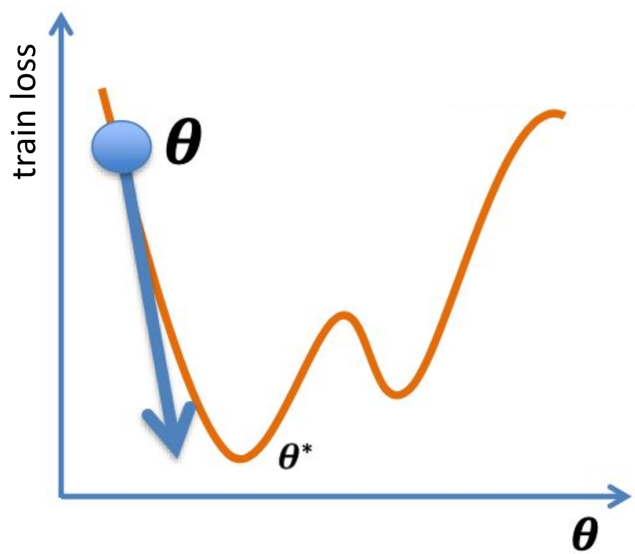
How to Train the Neural Network

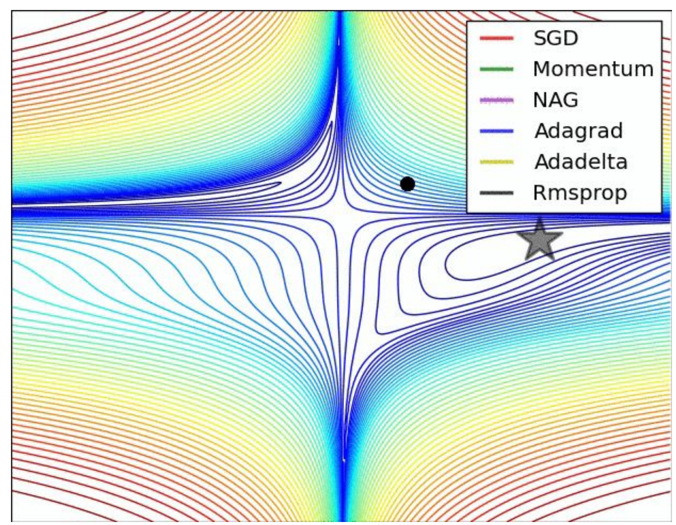
blackbox $\boldsymbol{\theta}$

x_i Inputs → Deep Neural Network → Outputs $\hat{y}_i$ | Target y_i

0.99 | 1

prediction should be as close as possible to ground truth!





$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} L(\boldsymbol{y}, \hat{\boldsymbol{y}};\ \boldsymbol{\theta})$$

How to Train the Neural Network

blackbox $\boldsymbol{\theta}$

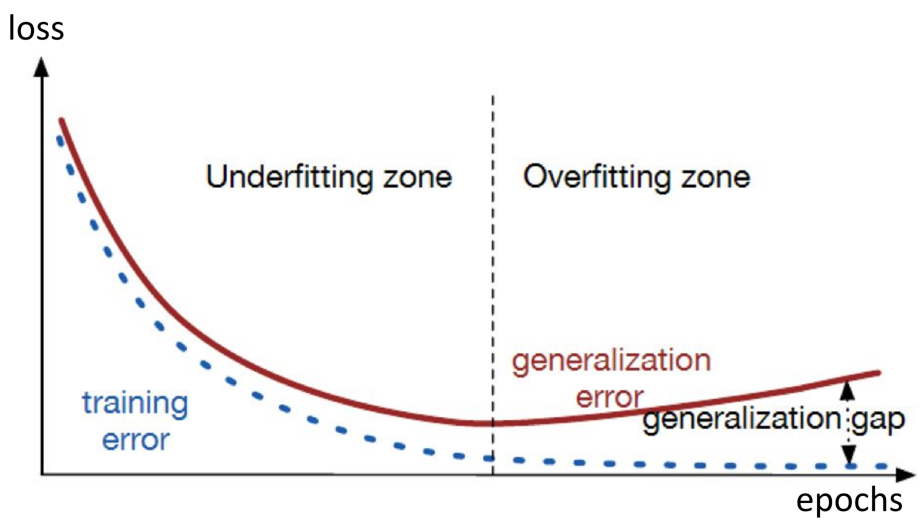
x_i Inputs → Deep Neural Network → $\hat{y}_i$ Outputs | y_i Target

0.99 | 1

prediction should be as close as possible to ground truth!

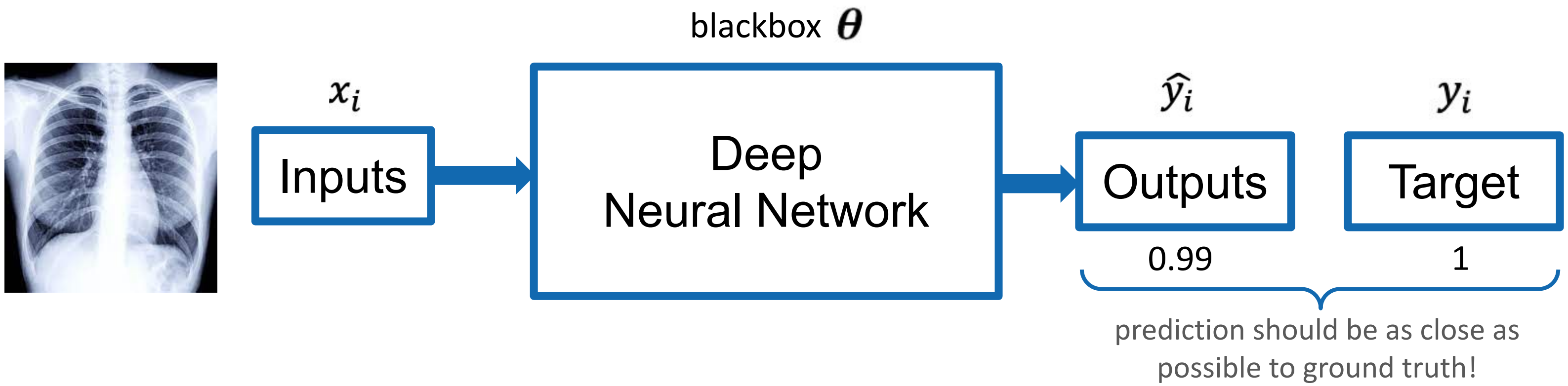
please remember that we split our dataset:

60% train	20% validation	20% test

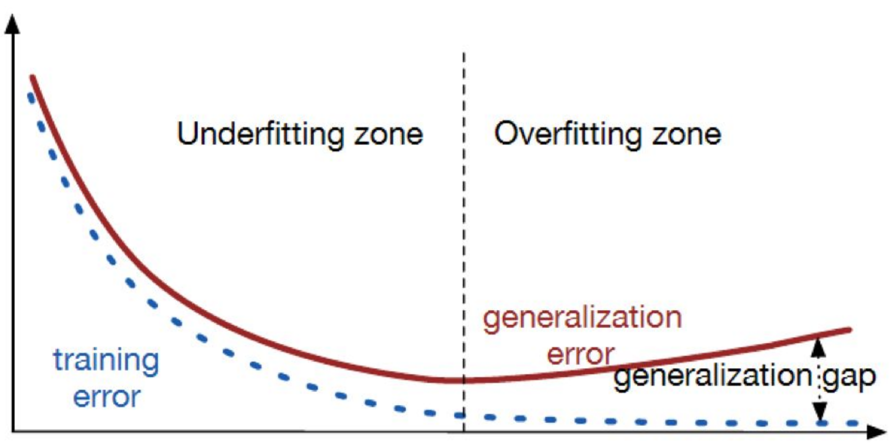


to ensure that our model does not just **memorize samples** seen during training, we always have to check whether the **model generalizes to data from the validation set**.

How to Train the Neural Network



to **avoid overfitting (reduce the generalization gap)** of our model to the training dataset we typically utilize **regularization** techniques:



lower
validation error

increased
training error

- more training data
- less model parameters
- L1 regularization
- L2 regularization
- dropout
- early stopping
- data augmentation

How to Train the Neural Network

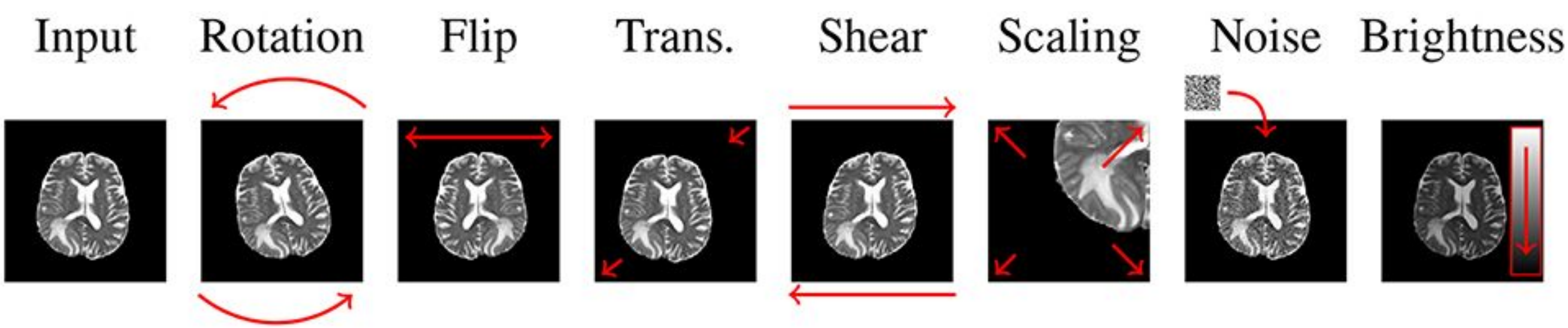
blackbox θ

x_i Inputs → Deep Neural Network → $\hat{y}_i$ Outputs | y_i Target

0.99 | 1

prediction should be as close as possible to ground truth!

We aim to **artificially increase the size of the training set** by modifying training data in a meaningful manner. Some prominent data augmentation techniques in medical image analysis:



E1: Time for Some Practical Exercises!

one epoch represents one run through all samples present in the train set

```
for epoch in range(2):  # loop over the dataset multiple times

    running_loss = 0.0
    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 statistics
        running_loss += loss.item()
        if i % 2000 == 1999:    # print every 2000 mini-batches
            print('[%d, %5d] loss: %.3f' %
                  (epoch + 1, i + 1, running_loss / 2000))
            running_loss = 0.0

print('Finished Training')
```

x_i, y_i → inputs, labels = data

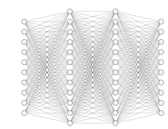
$\hat{y}_i$ → outputs = net(inputs)

loss = criterion(outputs, labels) ← $\mathcal{L}(\hat{y}_i, y_i)$

optimizer.step() ← $\theta_{k+1} = \theta_k - \alpha \nabla_\theta L(\theta_k, x_{\{1..m\}}, y_{\{1..m\}})$

Please have a look at your notebook and work on:

1) Introduction to PyTorch
2) Preparing Data
3) Implementation of a FCN/MLP



The image to the left may help you to connect the dots between the learned theory and the practical exercises you will have to tackle in the notebook.

Again, we would like to remind you to please ask whenever you have any question!

Neural Networks

blackbox $\boldsymbol{\theta}$

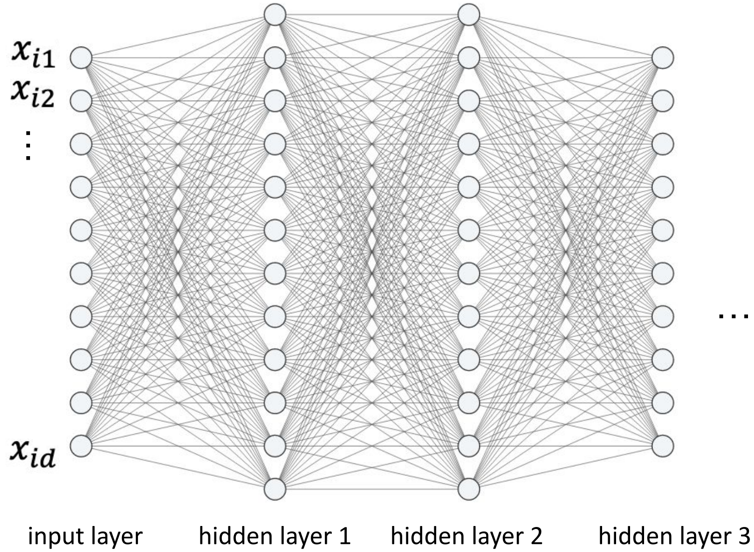
x_i → **Inputs** → **Deep Neural Network** → **Outputs** $\hat{y}_i$ 0.9 or 90%

image i

predicted class corresponding to image i

$$\hat{y}_i = \boldsymbol{W}_5 \sigma(\boldsymbol{W}_4 \tanh(\boldsymbol{W}_3, \max(\mathbf{0}, \boldsymbol{W}_2 \max(\mathbf{0}, \boldsymbol{W}_1 x_i))))$$

lernable parameters $\boldsymbol{\theta}$



we organize a neural network into multiple layers separated by non-linearities.

Neural Networks

blackbox $\boldsymbol{\theta}$

x_i → Inputs → Neural Network → Outputs $\hat{y}_i$ → 0.9 or 90%

image i

predicted class corresponding to image i

$$\hat{y}_i = \boldsymbol{W}_5\,\sigma(\boldsymbol{W}_4 \tanh(\boldsymbol{W}_3, \max(\mathbf{0}, \boldsymbol{W}_2 \max(\mathbf{0}, \boldsymbol{W}_1 x_i))))$$

we organize a neural network into multiple layers separated by non-linearities.

let us explore more advanced Neural Networks!

Neural Networks

blackbox $\boldsymbol{\theta}$

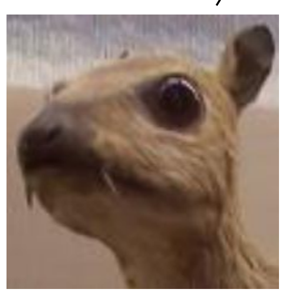
x_i Inputs → Neural Network → Outputs $\hat{y}_i$ 0.9 or 90%

image i

predicted class corresponding to image i

convolution: $$f * g = \int_{-\infty}^{\infty} f(\tau) g(t - \tau) d\tau$$

kernel weights



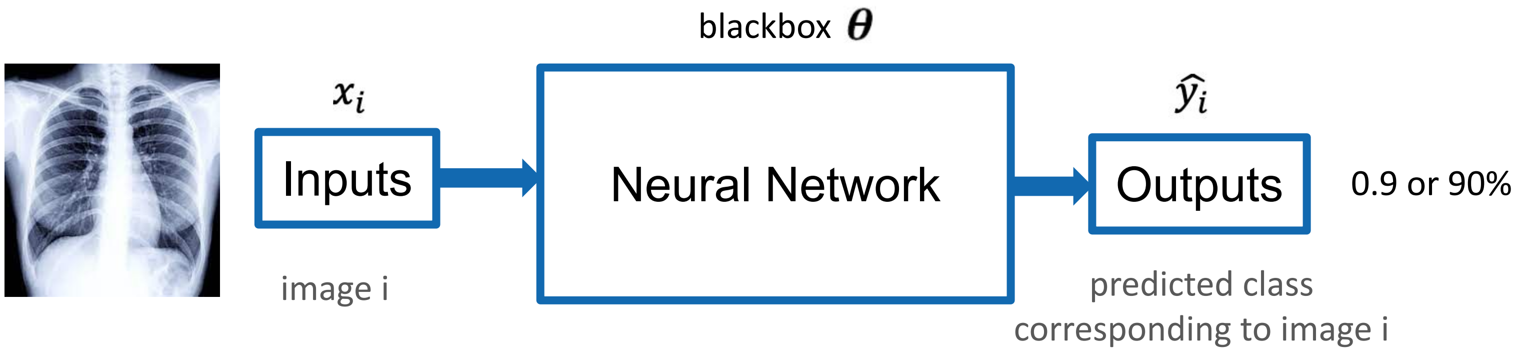
input image

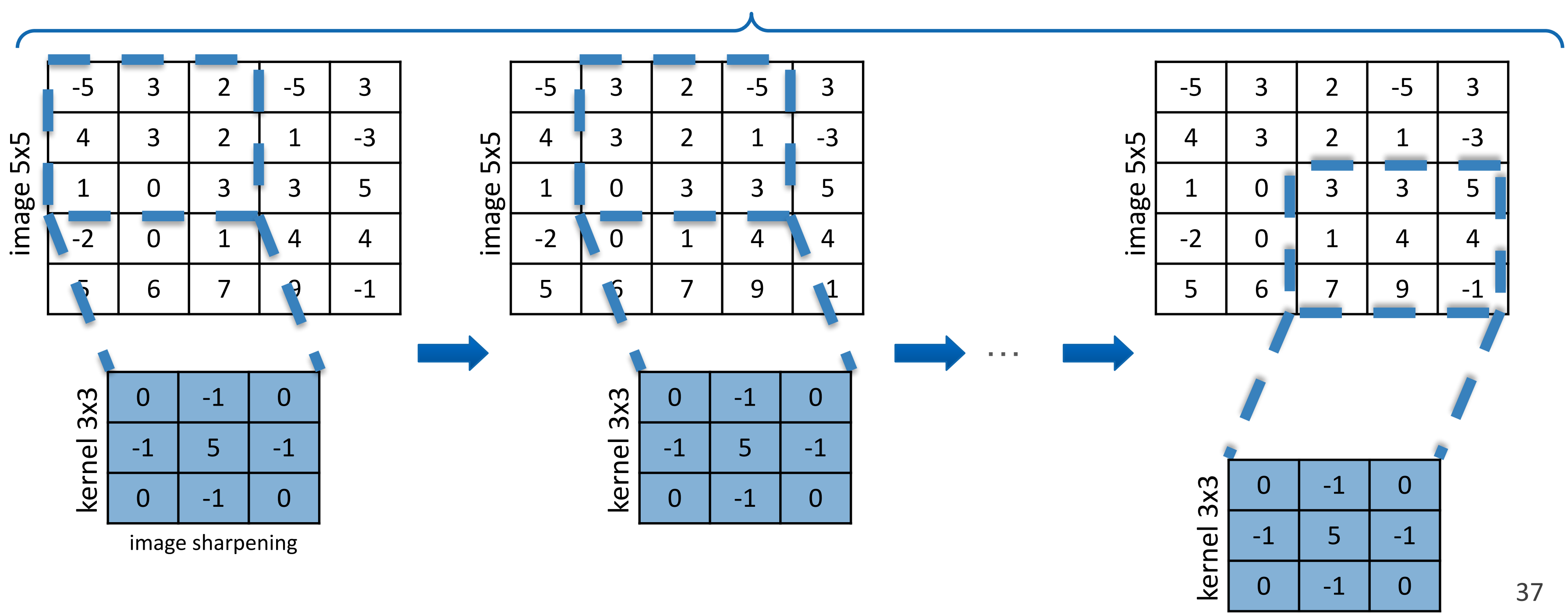
$$* \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} =$$

kernel for edge detection

processed output image

Neural Networks





Neural Networks

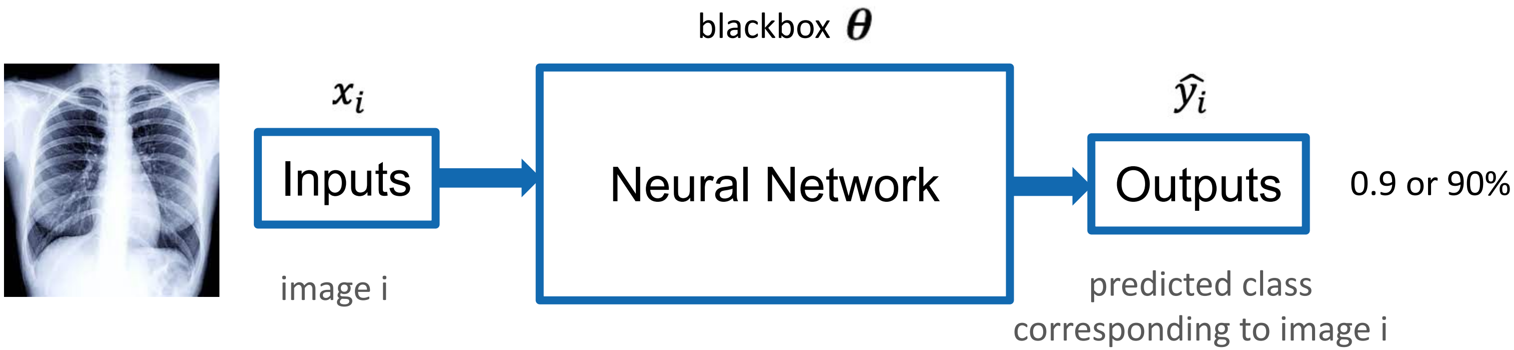


image 5x5

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

kernel 3x3

0	-1	0
-1	5	-1
0	-1	0

output 3x3

6		

$5 \cdot 3 + -1 \cdot 3 + -1 \cdot 2 + -1 \cdot 0 + -1 \cdot 4$
$= 15 - 9 = 6$

Neural Networks

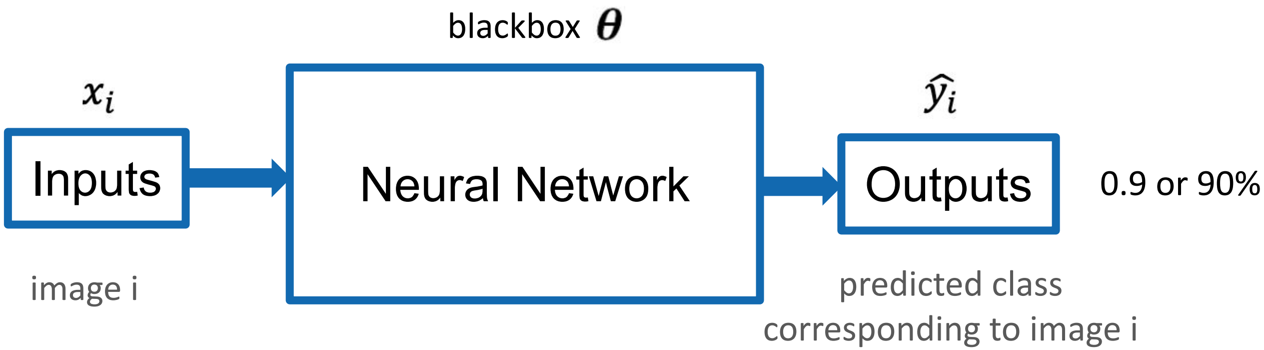


image 5x5

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

kernel 3x3

0	-1	0
-1	5	-1
0	-1	0

$5 \cdot 3 + -1 \cdot 3 + -1 \cdot 2 + -1 \cdot 0 + -1 \cdot 4 = 15 - 9 = 6$

image 5x5

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

kernel 3x3

0	-1	0
-1	5	-1
0	-1	0

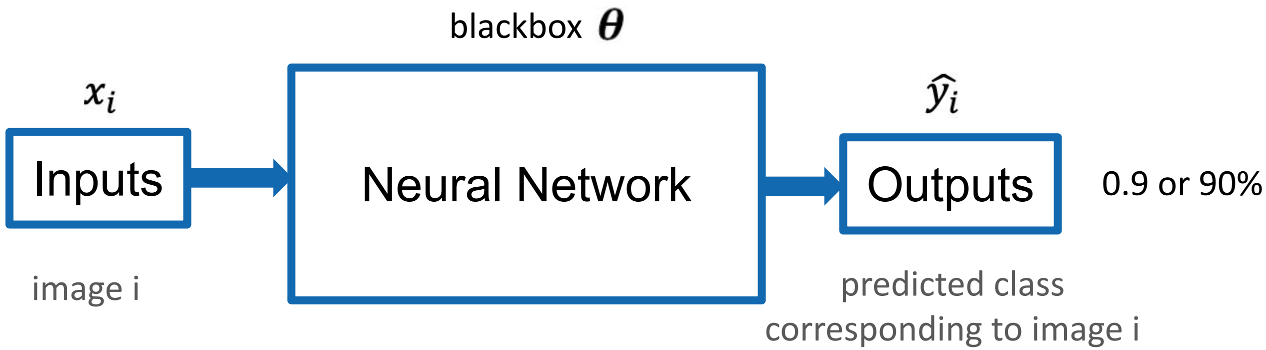
$5 \cdot 2 + -1 \cdot 2 + -1 \cdot 1 + -1 \cdot 3 + -1 \cdot 3 = 10 - 9 = 1$

...

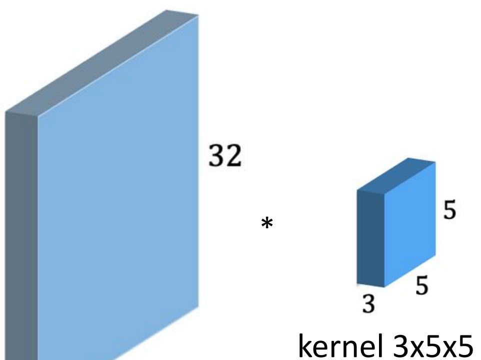
output 3x3

6	1	

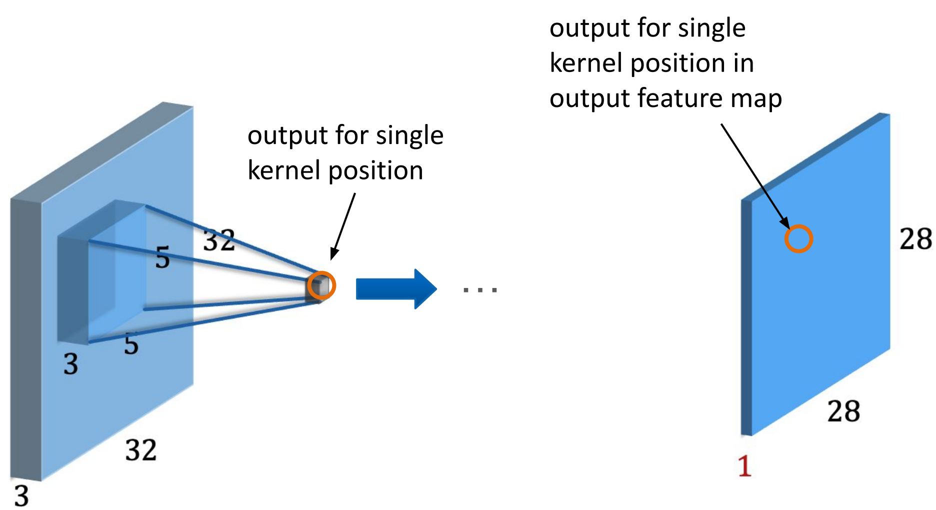
Neural Networks



filter must extend to all image channels!

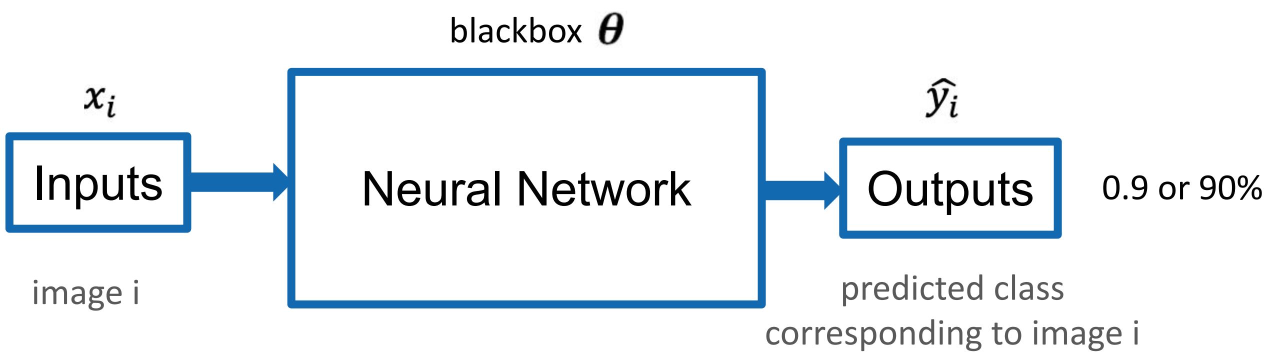


RGB image 3x32x32

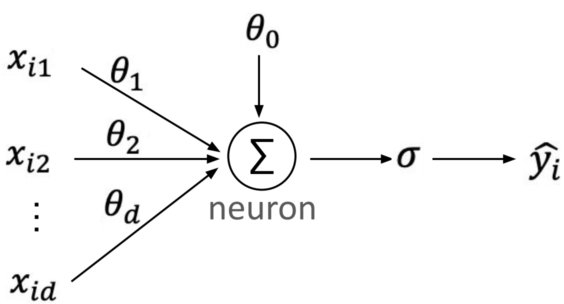


output 1x28x28

Neural Networks

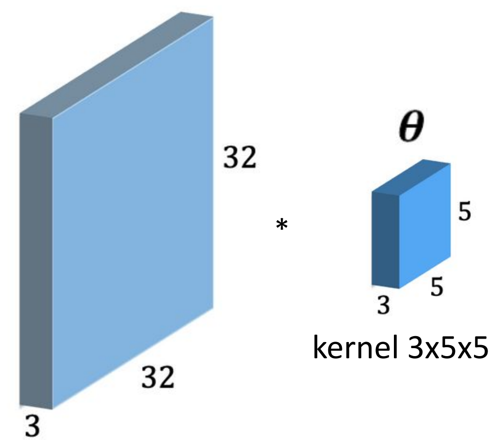


how is can we use this for our neural networks?



Logistic Regression:
a 1 layer, 1 neuron neural network

we **learn the kernel weights** and optimize them as parameters during training!



RGB image 3x32x32

Convolutional Neural Networks (CNN)

blackbox $\boldsymbol{\theta}$

x_i **Inputs** → **Convolutional Neural Network** → $\hat{y}_i$ **Outputs** 0.9 or 90%

image i

predicted class corresponding to image i

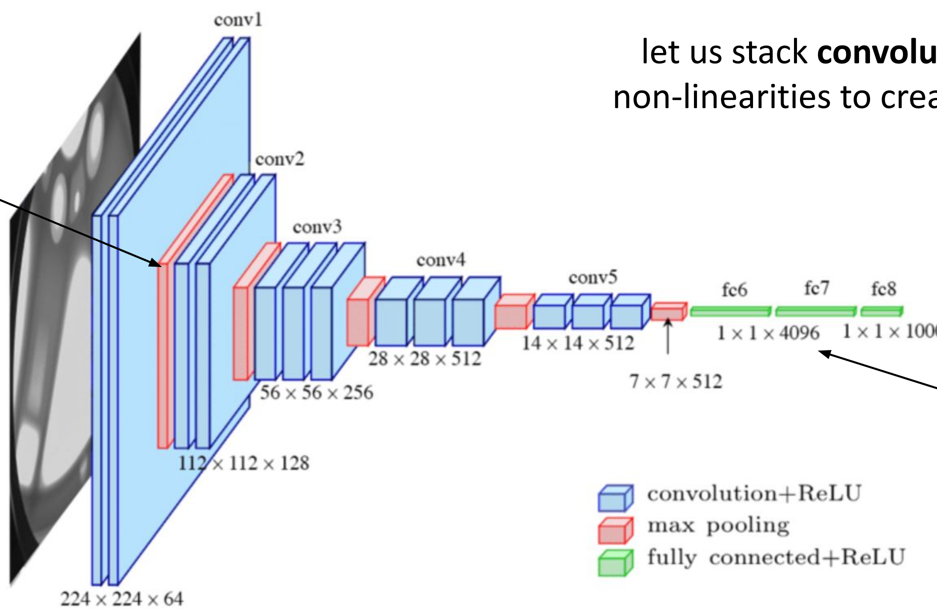
downsampling via max-pooling
(lower comp. cost; higher receptive field)

Max Pooling

29	15	28	184
0	100	70	38
12	12	7	2
12	12	45	6

2 x 2
pool size

100	184
12	45



let us stack **convolutional layers** followed by non-linearities to create our first CNN (VGG-16)!

final linear layers
(for classification;
here they want to predict 1000 classes)

Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." *arXiv preprint arXiv:1409.1556* (2014).

Convolutional Neural Networks (CNN)

blackbox $\boldsymbol{\theta}$

x_i → Inputs → Convolutional Neural Network → Outputs $\hat{y}_i$ 0.9 or 90%

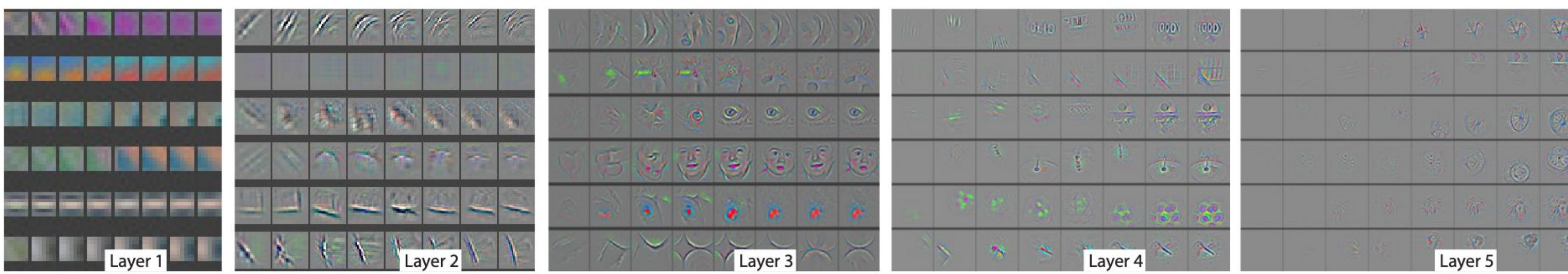
image i

predicted class corresponding to image i

the intuition behind CNNs:

conv1 conv2 conv3 conv4 conv5

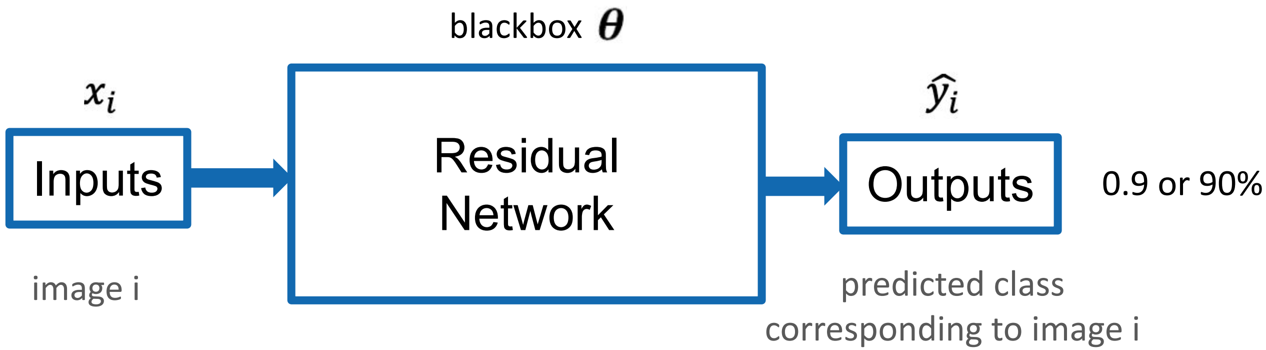
each filter learns different image characteristics



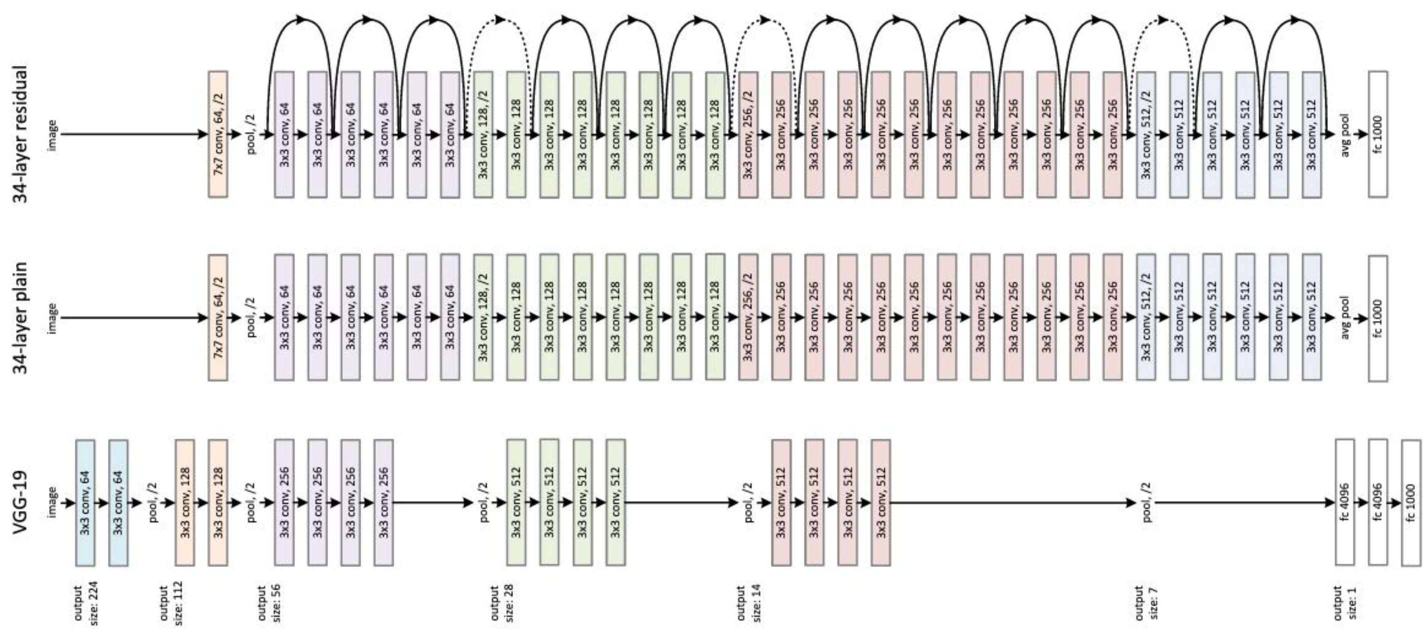
Zeiler, Matthew D., and Rob Fergus. "Visualizing and understanding convolutional networks." *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I 13*. Springer International Publishing, 2014.

later layers learn more complex features

Residual Networks (ResNet)



a quite deep CNN



He, Kaiming, et al. "Deep residual learning for image recognition." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016.

ETH zürich | University of Zurich

E2: Time for More Practical Exercises!

```
for epoch in range(2):  # loop over the dataset multiple times

    running_loss = 0.0
    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 statistics
        running_loss += loss.item()
        if i % 2000 == 1999:    # print every 2000 mini-batches
            print('[%d, %5d] loss: %.3f' %
                  (epoch + 1, i + 1, running_loss / 2000))
            running_loss = 0.0

print('Finished Training')
```

Please have a look at your notebook and work on:

1) Implementation of simple CNN
2) Experiments with ResNet
3) Interpretability of Predictions

Again, we would like to remind you to please ask whenever you have any question!

Please note that the last exercise on interpretability is optional and ment for students with prior experience.