AIminify

BIG GEODATA TALK

AIminify

- Matthijs Plat (founder)
- Steven van Blijderveen (AI engineer)

Goal of today:
Learn about smart ways to compress neural networks

AIminify

- Spin-off from tinify (TinyPNG.com)
- Innovating to make the world more easy
- Compression without settings
- Automated pipeline
- Focus on computer vision

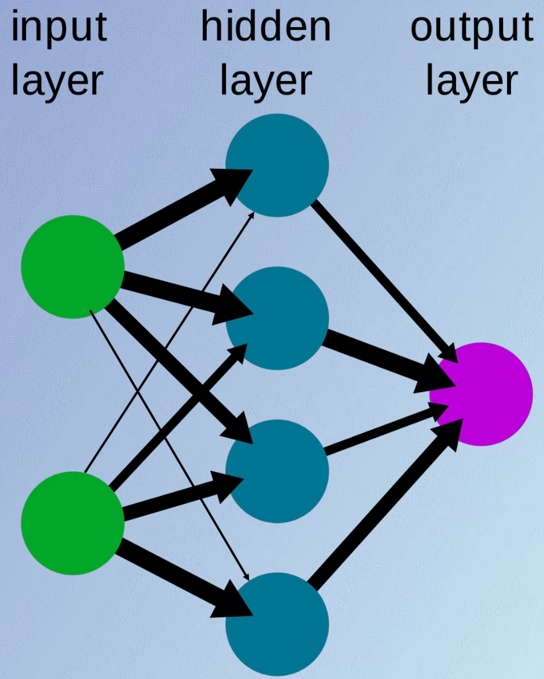
Neural Networks: the basics

- DNN: Dense Neural Network
  - Also called Fully Connected Neural Network
  - Input is 1-dimensional: a number or a list of numbers
  - Output can be:
    - An absolute number: Estimate the price of a car based on the features of that car
    - A percentage: Estimate the odds of someone having a disease based on test results
- CNN: Convolutional Neural Network
  - Input is 2-dimensional: usually an image
  - Output can be:
    - An absolute number: Estimate someone's age based on an image of their face
    - A percentage: What are the odds this image is a dog?
    - An image: Turn this image into a Van Gogh-style image

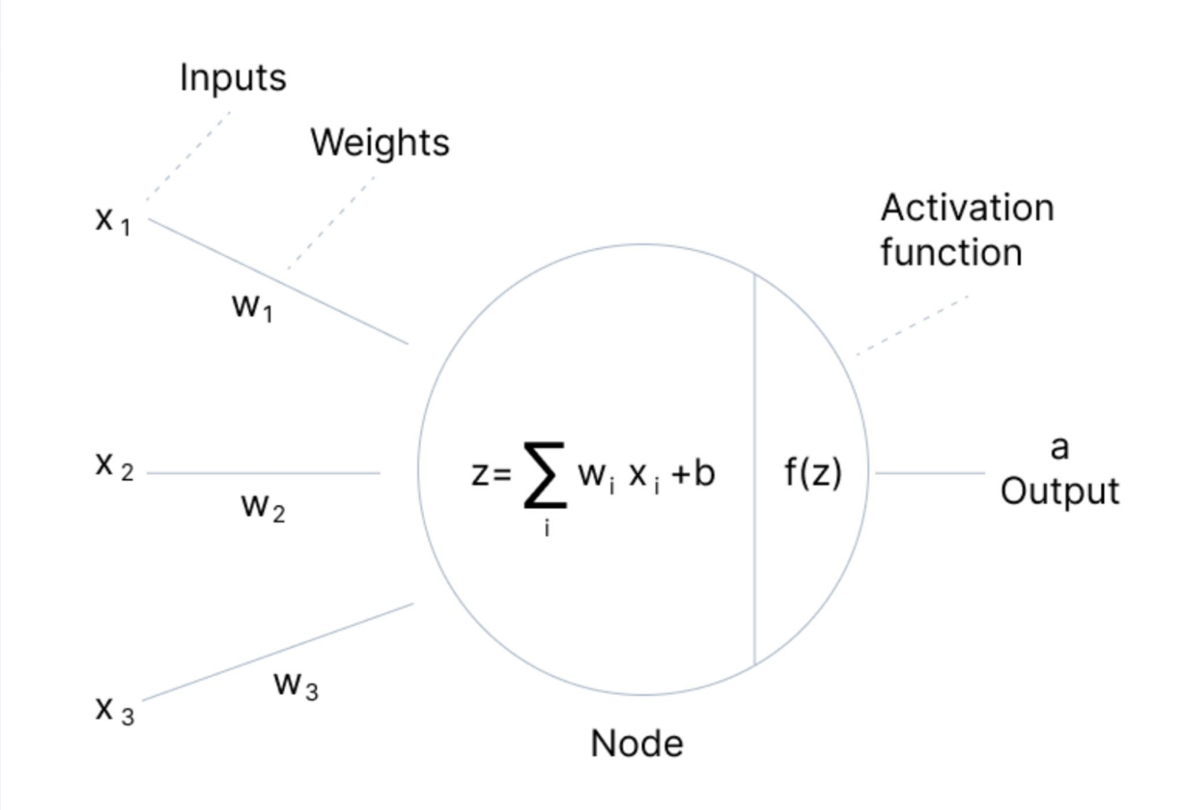
Neural Networks: the basics

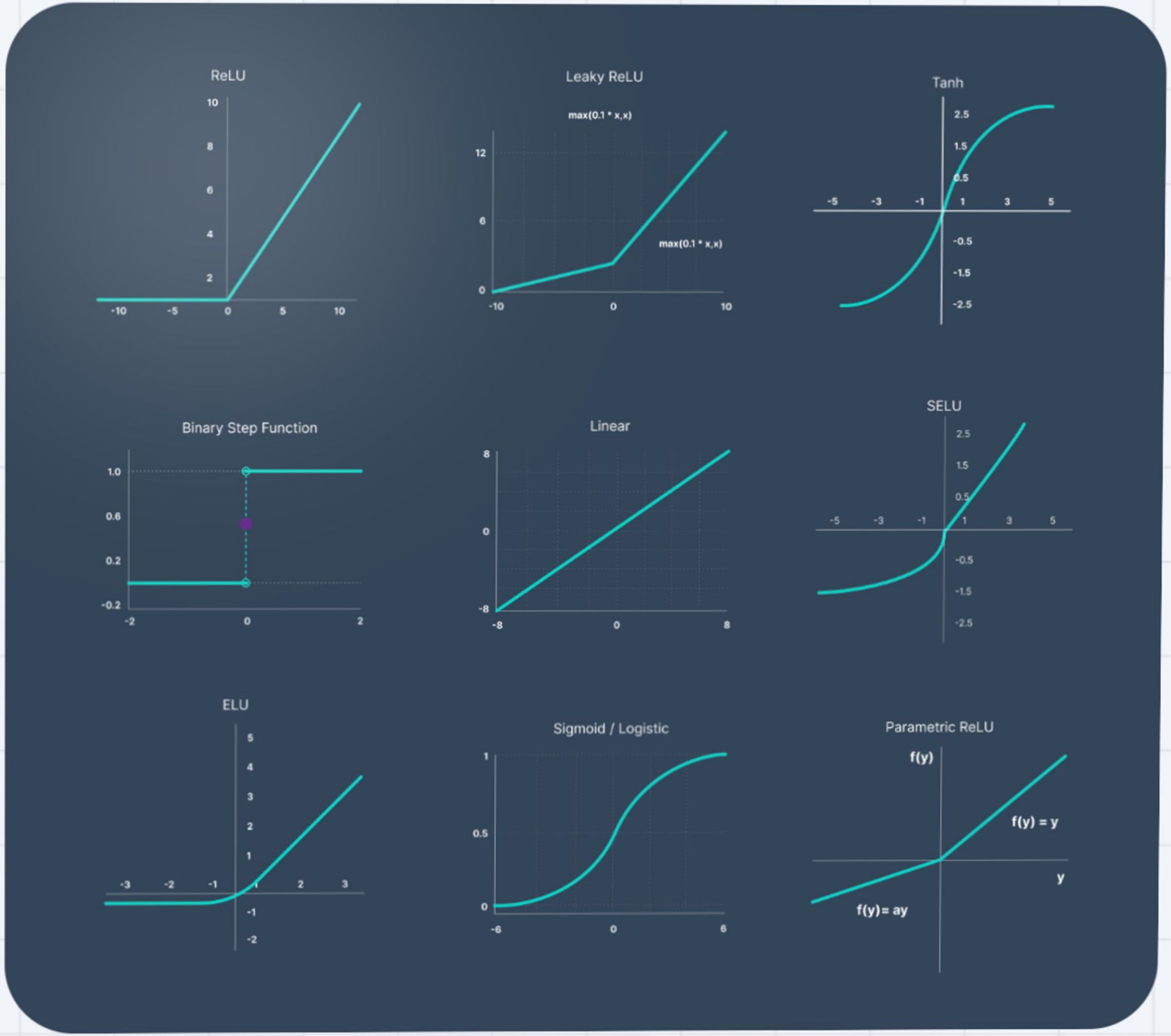
- A Dense Neural Network looks like this image, but probably with a lot more hidden layers
- All nodes are connected to every node in the next layer
- All connections have **weights** (the thickness)
- Inside a node the **activation function** decides what value to propagate to the next node(s)

A simple neural network



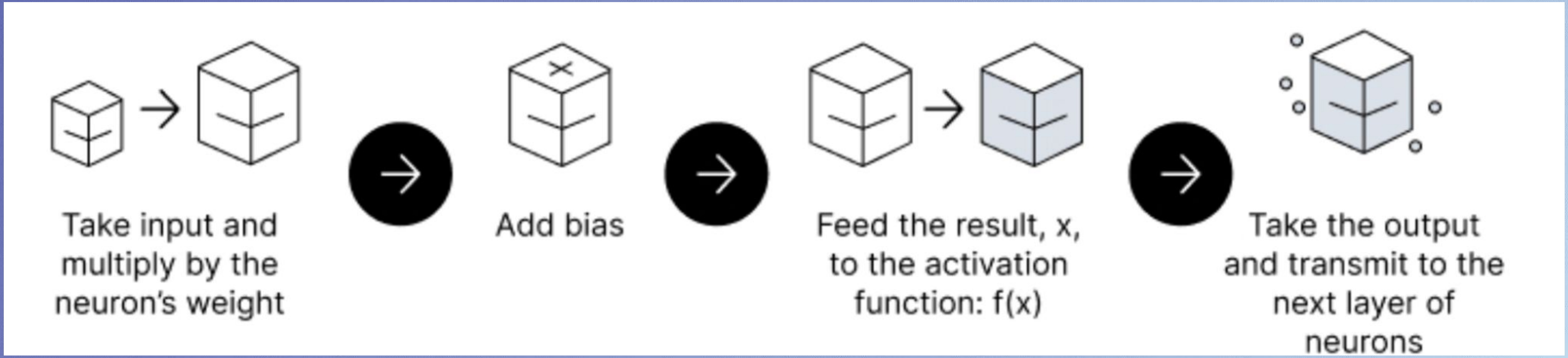
Activation functions





ReLU
Leaky ReLU
max(0.1 * x,x)
Tanh
Binary Step Function
Linear
SELU
ELU
Sigmoid / Logistic
Parametric ReLU
f(y)
f(y) = y
y
f(y)= ay

So what happens inside a neuron?



How does a network learn the weights + bias?

- The keyword is **backpropagation**
- Your network starts with some basic weights and biases
- You propagate your training data through the network to get expected output 'y' and real output 'ŷ'
- Using a **loss function** you calculate the loss, for example a simple one is Mean Squared Error:

$$\mathrm{MSE} = \frac{1}{n} \sum_{i=1}^{n} (\hat{Y}_i - Y_i)^2$$

How does a network learn the weights + bias?

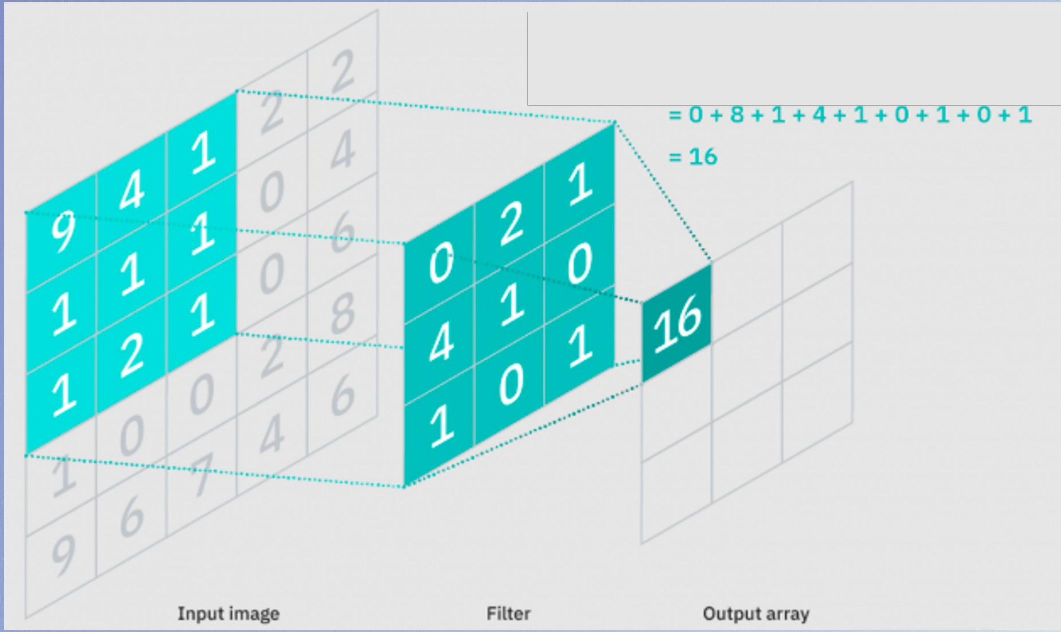
- The calculated loss of the network is given back to the previous layer of nodes and using the *chain rule of calculus* the weights are altered to minimize the loss
- The speed of adaptation of the weights is determined by a **learning rate** which is applied to an **optimization algorithm**.
- Pull your training data a few times (**epochs**) through your network until your loss is minimal and then your network hopefully works!

CNN’s, what’s different?

- Not that much, except for 2-dimensional data
- Next to that, some new layers get introduced:
  - Convolutional layers (hence **C**NN)
  - Pooling
  - Flattening
  - Etc.

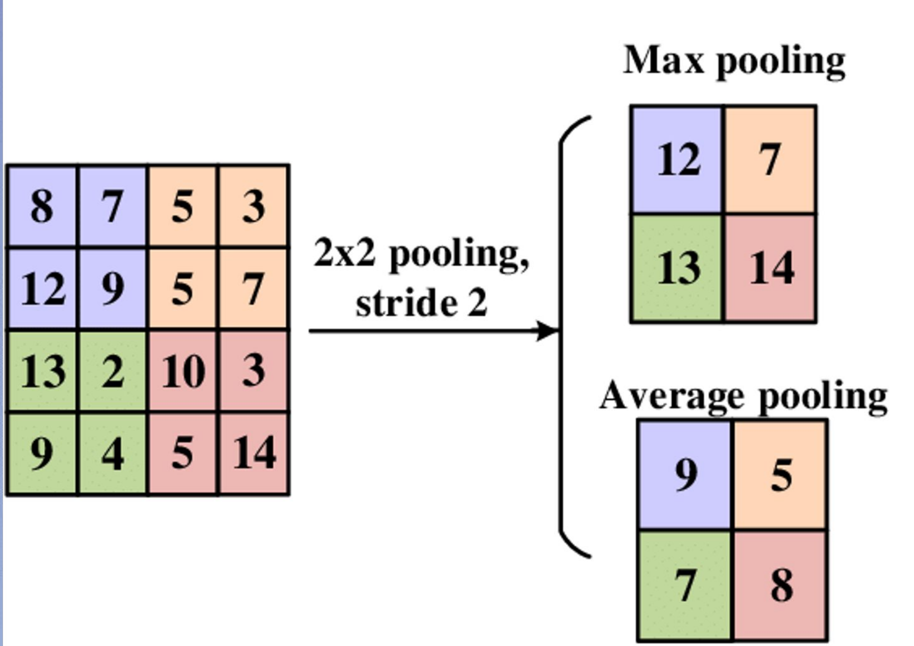
Convolutional layers

- A filter or kernel is slid over the input matrix
- You can decide:
  - Filter size
  - Stride (stepsize)
  - Padding (adding numbers to the borders of the input matrix)
  - Amount of filters
- All these parameters influence the size of the output matrix
- Usually a shrinks the matrix, but adds in the third dimension



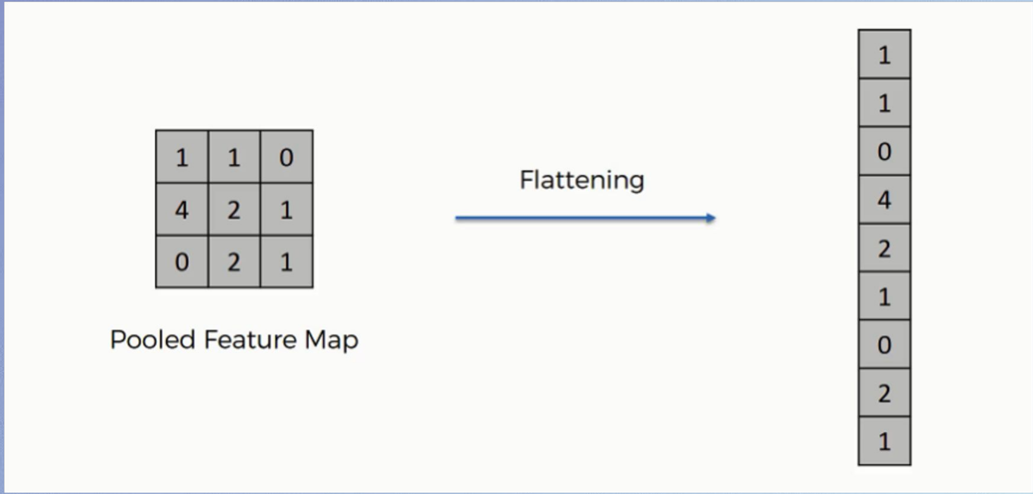
Pooling Layer

- Pooling is used to prevent overfitting by generalizing the input
- It has two parameters:
    - Filter size
    - Stride (stepsize)
- There are many different pooling layers: Min/Max/Average etc.
- Pooling shrinks the matrix without adding in the third dimension

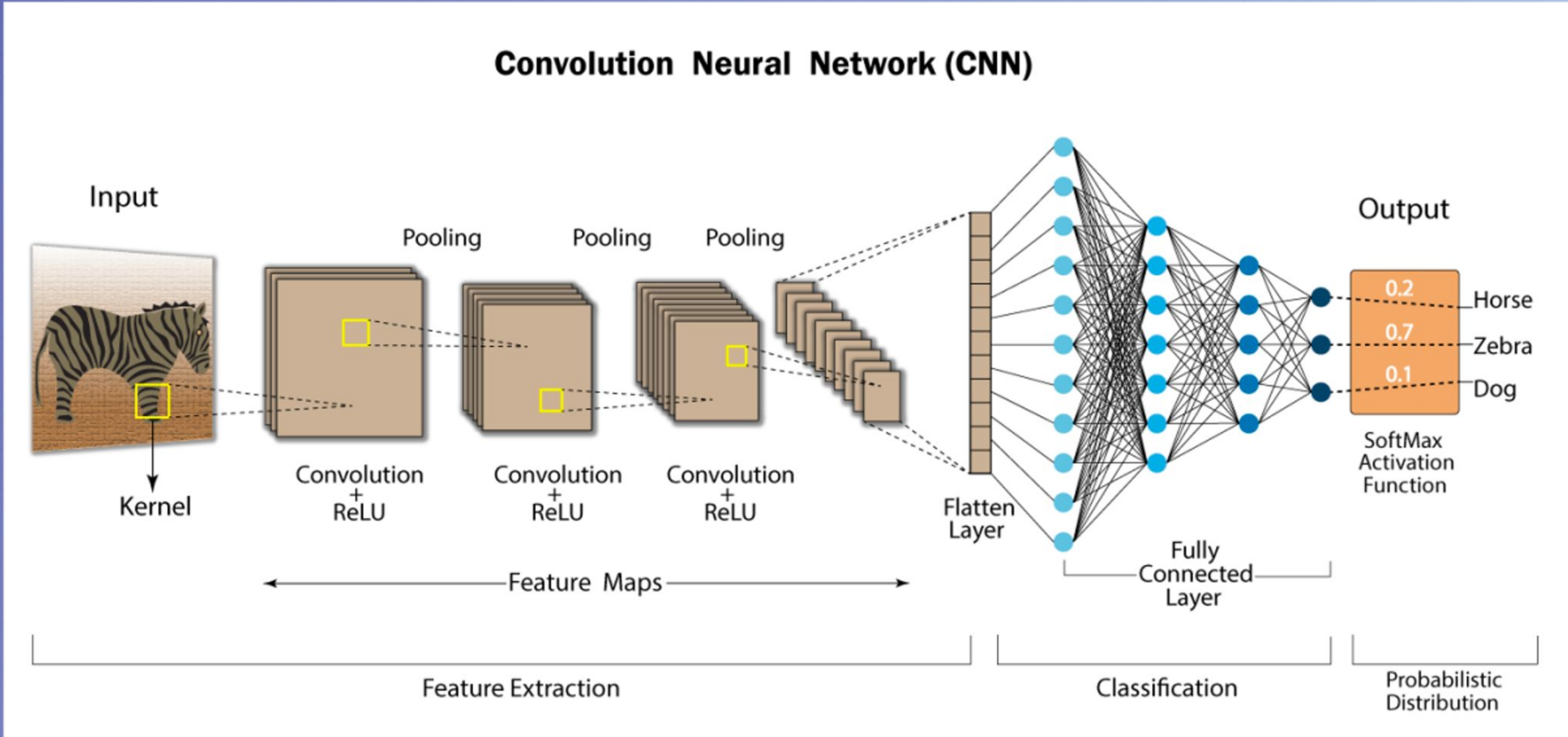


Flattening layer

- Very simple, turns your data from a 2-dimensional matrix into a 1-dimensional list
- After a flattening layer you can use Dense layers again



What does a complete CNN look like?

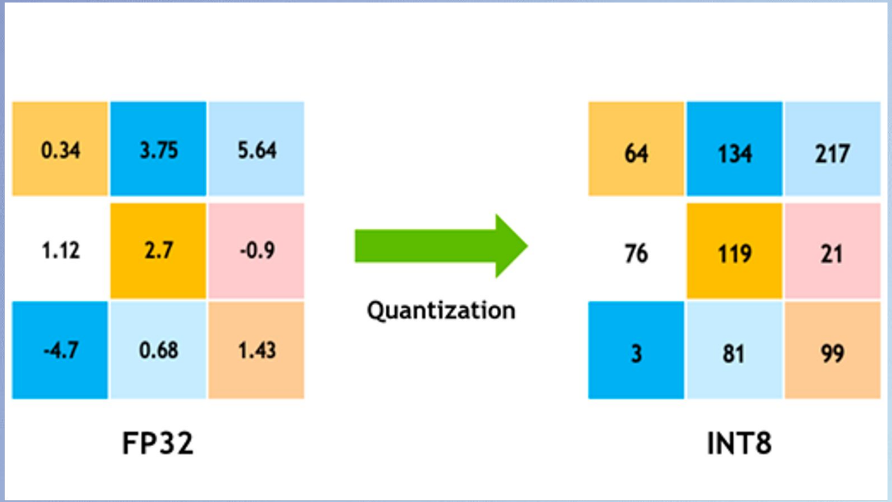


Compression

- Quantization
- Pruning
- Knowledge Distillation

Quantization

- Changing weights, biases and activations from 32-bits floats to 16-bits floats or 8-bits integers
- Mostly a storage saver, but can also increase speed
- Highly depending on hardware
- Can cause accuracy loss



Quantization - a bit of math

- Basic int-8 quantization involves mapping all your weights to the range [-128, 127]
- Do this by finding the scale *s*: $$s = \frac{max(f) - min(f)}{256}$$
    - Where *f* are your weights
- Then find the zero-point *z*: $$z = round(\frac{-min(f)}{s})$$
- Finally quantize your weights to *q* by doing: $$q = round(\frac{f}{s}) + z$$

Quantization in AIminify

TensorFlow

- Dynamic range quantization
- Full integer quantization
- Float-16 quantization

PyTorch

- Dynamic quantization
- Static quantization

Dynamic quantization

- Easiest form of quantization
- Quantizes the weights to 8-bits integers
- Converts activations to 8-bits integers dynamically, making int-8 matrix multiplications possible
- But activations are both read and written in 32-bit float format
- Saves a lot of space (~75%), but only improves latency a bit

Full integer quantization / static quantization

- Converting weights, bias and activations to int-8
- Needs small representative data set to correctly convert activations to keep the distribution of your activations in mind:
  - Uniformly distributed activations can simply be mapped to [-128, 127] directly
  - For non uniformly distributed activations we need something smarter

Float 16 quantization (Tensorflow only)

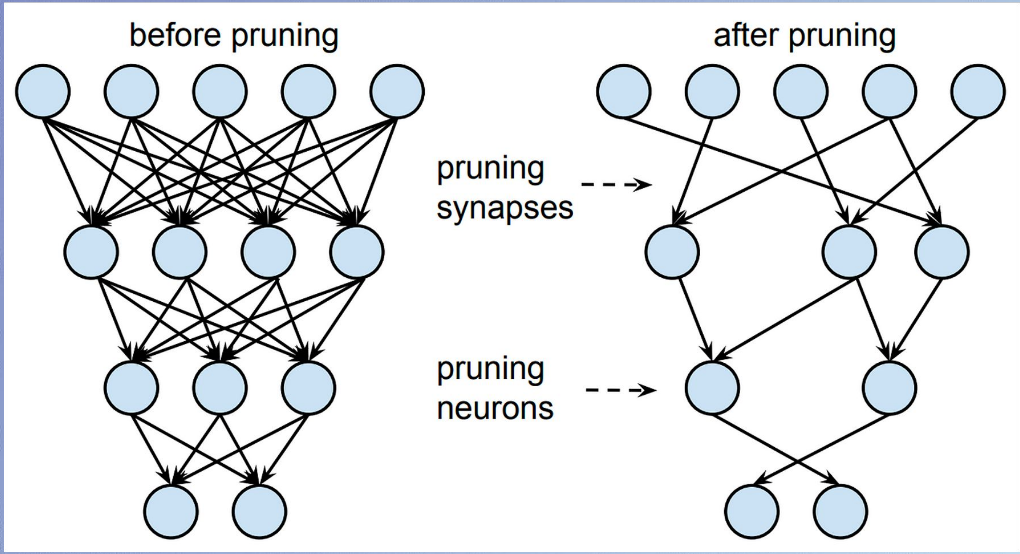
- Does what is says: quantize to 16-bits floats
- Halfs your model size
- Doesn't improve speed on CPU's
  - Internally all weights have to be dequantized to 32-bits since CPU's can't work with 16-bits
- Improves speed a lot on certain GPU's

Pruning in AIminify

- Pruning is removing unnecessary parts of the network
- In AIminify we use various pruning techniques, depending on the input network
- Besides looking at the complete network, we determine the pruning strength per layer

Pruning in DNN’s

- Quite simple in theory:
  - Remove nodes or synapses that are not important
- Difficulties:
  - What nodes are unimportant?
  - CPU’s/GPU’s have a hard time with sparse networks
  - Removing a node also impacts the next node

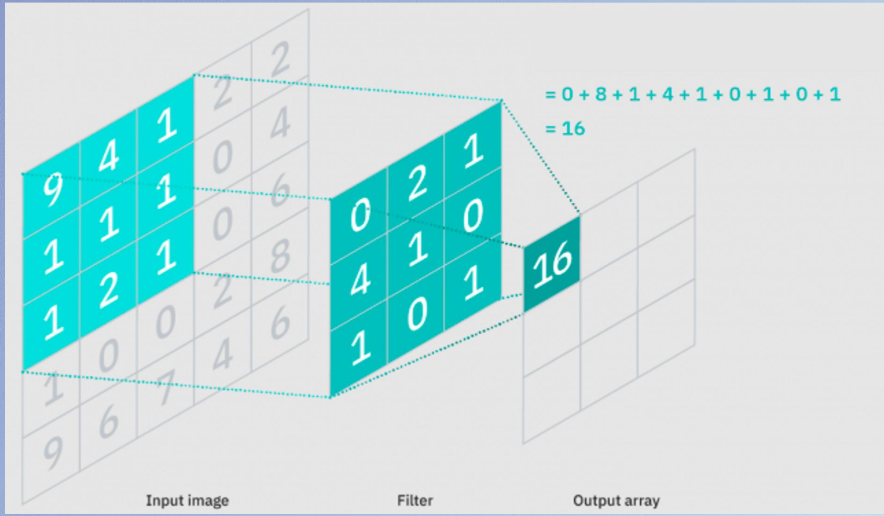


Pruning in DNN's

- What nodes/synapses are unimportant?
  - Intuïtively: synapses with weights close to 0 impact the network minimally
- But pruning synapses creates a sparse network, which doesn't improve latency, so we need to prune whole nodes
- How to calculate nodes with a low *magnitude*?
  - L_2-norm: Taking the square root of all incoming weights squared (Euclidean distance)

Pruning in CNN’s

- Theoretically we want to prune weights from a filter
- But CPU’s/GPU’s have a hard time with that
- So we prune **whole filters**
- Less calculations → speed improvement!



What filters to prune?

- Similar to pruning in DNN's, remove filters with the smallest norm?
- Only really works well if two criteria are met:
  - The norm deviation of the filters should be large

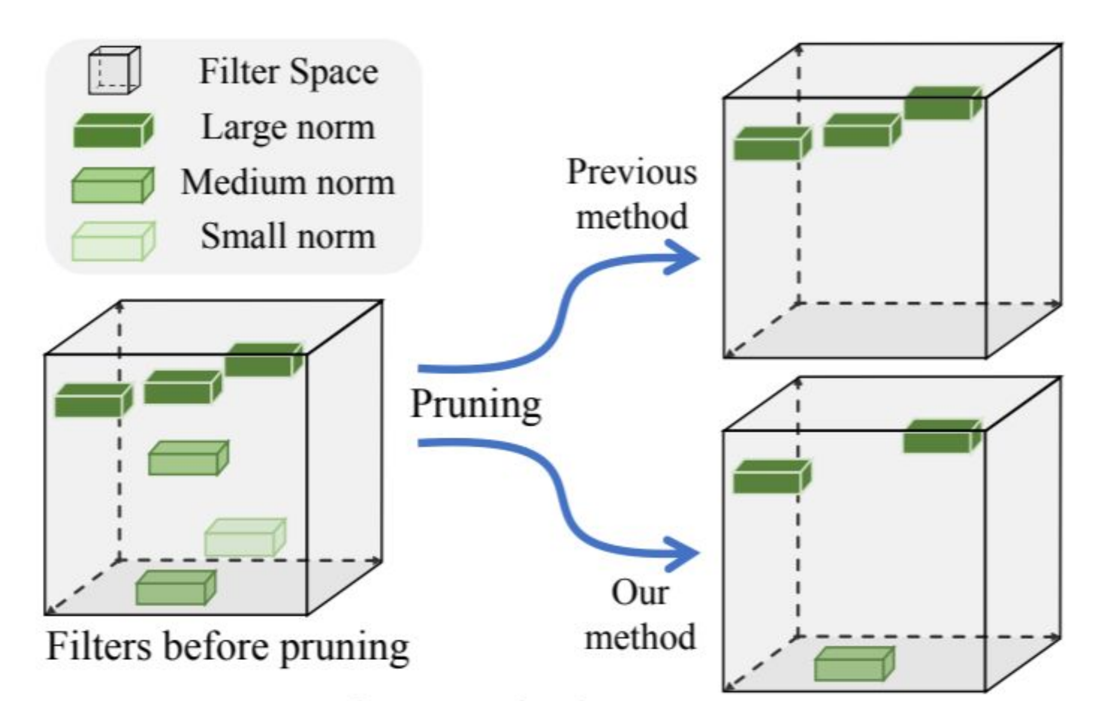
    →Otherwise it's hard to find a good threshold to prune
  - The minimum norm of the filters should be small

    →Otherwise you're still pruning away filters with a lot of influence on the outcome
- These criteria are not always met at all

FPGM pruning

Filter Pruning via Geometric Median

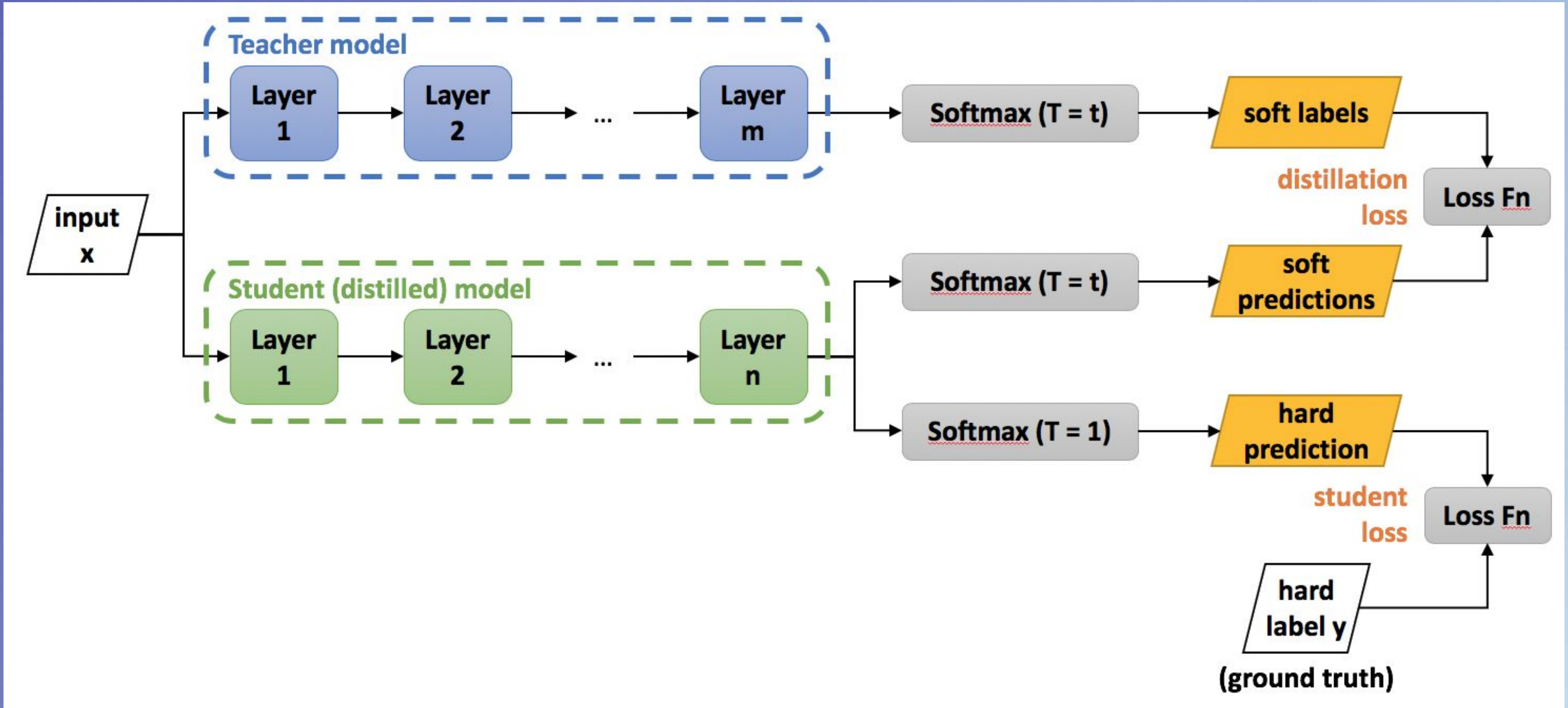
- The norm tells us how much the filter influences the outcome
- But what if a small norm has a very critical small influence?
- So we prune the 'geometric median'



He, Y., Liu, P., Wang, Z., Hu, Z., & Yang, Y. (2019). Filter pruning via geometric median for deep convolutional neural networks acceleration. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 4340-4349)

Knowledge Distillation (Teacher - Student model)

- The original model (teacher) is used to train a smaller model (student)
- The model outputs something like: ‘A: 86%, B: 1%, C: 5%, D: 8%’
- By using these ‘soft labels’ as an extra in the loss function of the student network, it knows a lot more about preferred outcomes.
- With all this extra info, the student will need a smaller architecture to learn the same thing



Teacher model
Layer 1
Layer 2
...
Layer m
Softmax (T = t)
soft labels
distillation loss
Loss Fn
input x
Student (distilled) model
Layer 1
Layer 2
...
Layer n
Softmax (T = t)
soft predictions
Softmax (T = 1)
hard prediction
student loss
Loss Fn
hard label y
(ground truth)

DistilBERT

- BERT is one of the predecessors of ChatGPT
- Researchers used Knowledge Distillation and:
    - Reduced the size of BERT by 40%
    - Retrained to 97% of the original model's language understanding capabilities
    - Made the model 60% faster

Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2019). DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*.

AIminify

- Matthijs Plat (founder)
- Steven van Blijderveen (AI engineer)

Goal of today:
Learn about smart ways to compress neural networks