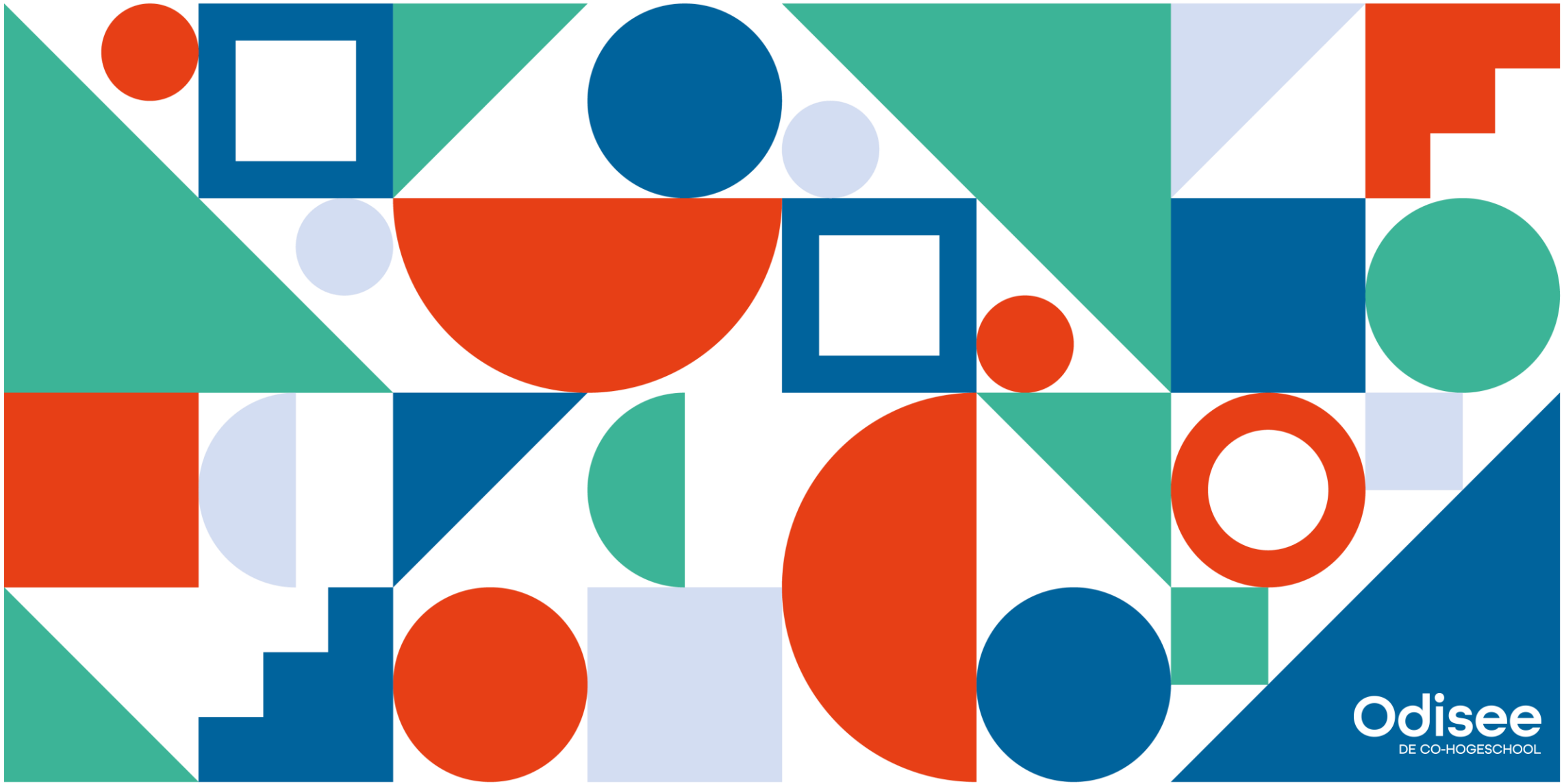

Odisee
DE CO-HOGESCHOOL

# Machine learning – Week 3

Jens Baetens

# How to participate?

1. Go to wooclap.com
2. Enter the event code in the top banner

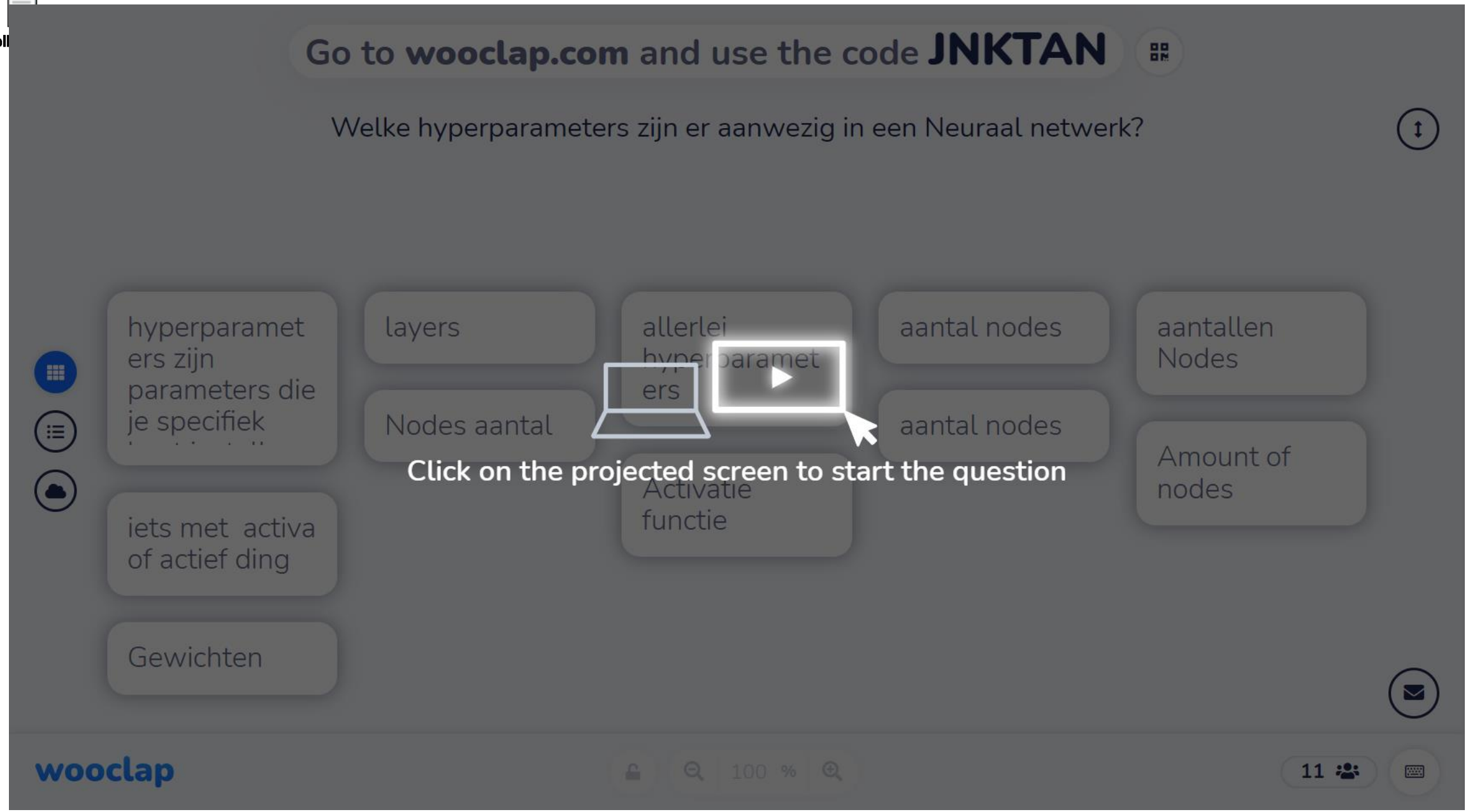
Event code
**JNKTAN**

Click on the projected screen to start the question

1. Send @JNKTAN to 0460 200 711
2. You can participate

Copy participation link

wooclap 100 % 17


Go to wooclap.com and use the code JNKTAN
Welke hyperparameters zijn er aanwezig in een Neuraal netwerk?
hyperparamet ers zijn parameters die je specifiek
layers
allerlei hyperparamet ers
aantal nodes
aantallen Nodes
Nodes aantal
aantal nodes
Amount of nodes
Activatie functie
iets met activa of actief ding
Gewichten
Click on the projected screen to start the question
wooclap
100 %
11

Go to **wooclap.com** and use the code **JNKTAN**

Welke lossfunctie kan gebruikt worden voor classificatie van 3 klassen?

1. Binary Cross Entropy — 8% — 1
2. Mean Squared Error — 31% — 4
3. Categorical Cross Entropy — 62% — 8
4. Mean Absolute Error — 0% — 0

Click on the projected screen to start the question

wooclap 100 % 13 / 17

You cannot vote anymore

# In welke volgorde moet je een Neuraal Netwerk trainen met tensorflow

The correct combination was:

- Kiezen tussen sequentieel model / functional API
- Toevoegen van preprocessing lagen
- Opstellen van de architectuur van het neuraal netwerk
- Compileren van het model met keuze loss-functie en learning rate optimizers
- Trainen van de gewichten in het neuraal netwerk
- Evaluation

# Data generation

# Data generators

- Meer informatie vind je hier: https://www.tensorflow.org/guide/data
- Vooral gebruikt als de data niet eenvoudig in het geheugen kan ingeladen worden
  - Beelden, figuren of images
  - Gestreamde data
  - Data van verschillende bronnen
- Door het gebruik van een generator kan deze data batch per batch ingeladen en verwerkt worden

# Beelden inladen

- Bron: https://www.tensorflow.org/tutorials/load_data/images
- tf.keras.utils.image_dataset_from_directory
  - Alle figuren van een klasse in een aparte directory

```
main_directory/
...class_a/
......a_image_1.jpg
......a_image_2.jpg
...class_b/
......b_image_1.jpg
......b_image_2.jpg
```

- Andere soorten datasets:
  - list_files om alle bestanden te lezen uit een directory
  - Gebruik de .map() functie om de bestanden uit te lezen en te verwerken

# Beelden inladen

- In sommige gevallen kan het nodig zijn om een Sequence klasse te maken om data in te lezen
  - Dit werkt met meerdere Python processen ipv meerdere C++ threads
  - Normaal dus trager
  - Maar als je data inleest met python library kan het sneller zijn

# Tekst inlezen

- Bron: https://www.tensorflow.org/tutorials/load_data/text
- tf.keras.utils.text_dataset_from_directory
  - Strikte datastructuur vereist
- list_files
  - Als elk element in een aparte file staat
- tf.data.TextLineDataset
  - Elke lijn in een bestand is een apart element/voorbeeld

```
train/
...csharp/
......1.txt
......2.txt
...java/
......1.txt
......2.txt
...javascript/
......1.txt
......2.txt
...python/
......1.txt
......2.txt
```

# Data augmentation

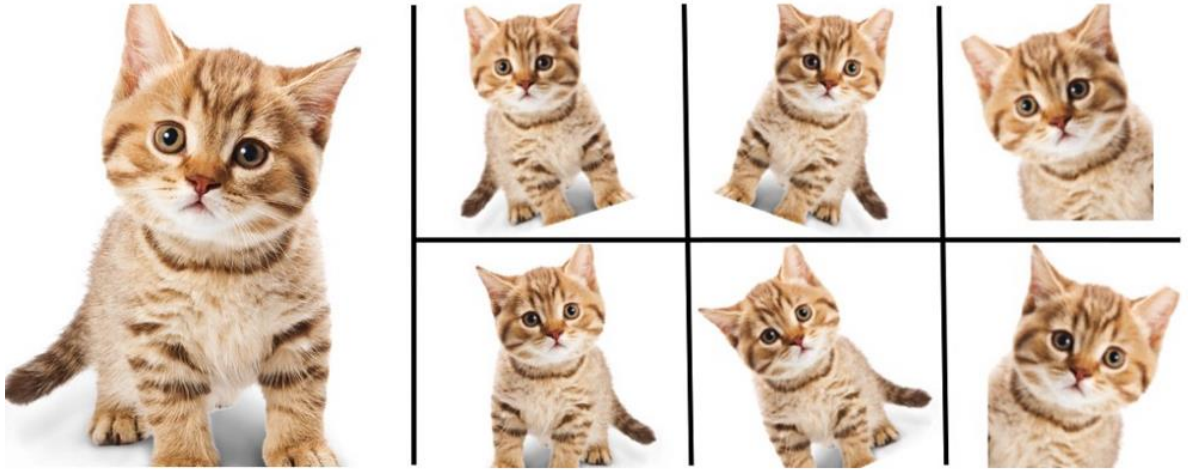
# Data augmentation

- Vaak gebruikt in Computer Visie
  - Kan ook in audio
- Genereer extra data door kleine wijzigingen toe te passen
  - Random crop, resize, rotate, spiegelen horizontaal of verticaal, …
  - Aanpassen van de kleuren (elk kleur apart of combinaties)
  - Toevoegen van ruis
  - Versnellen of vertragen van audio
  - Verhogen of verlagen van de stemtoon
  - …

# Data augmentation: voorbeeld

```
from tensorflow import keras
from tensorflow.keras import layers

# Create a data augmentation stage with horizontal flipping, rotations, zooms
data_augmentation = keras.Sequential(
    [
        layers.RandomFlip("horizontal"),
        layers.RandomRotation(0.1),
        layers.RandomZoom(0.1),
    ]
)
```



# Data augmentation: Live demo

- Zie notebook voor demo

- Lagen die hiervoor gebruikt kunnen worden:

  `tf.keras.layers.RandomCrop`

  `tf.keras.layers.RandomFlip`

  `tf.keras.layers.RandomTranslation`

  `tf.keras.layers.RandomRotation`

  `tf.keras.layers.RandomZoom`

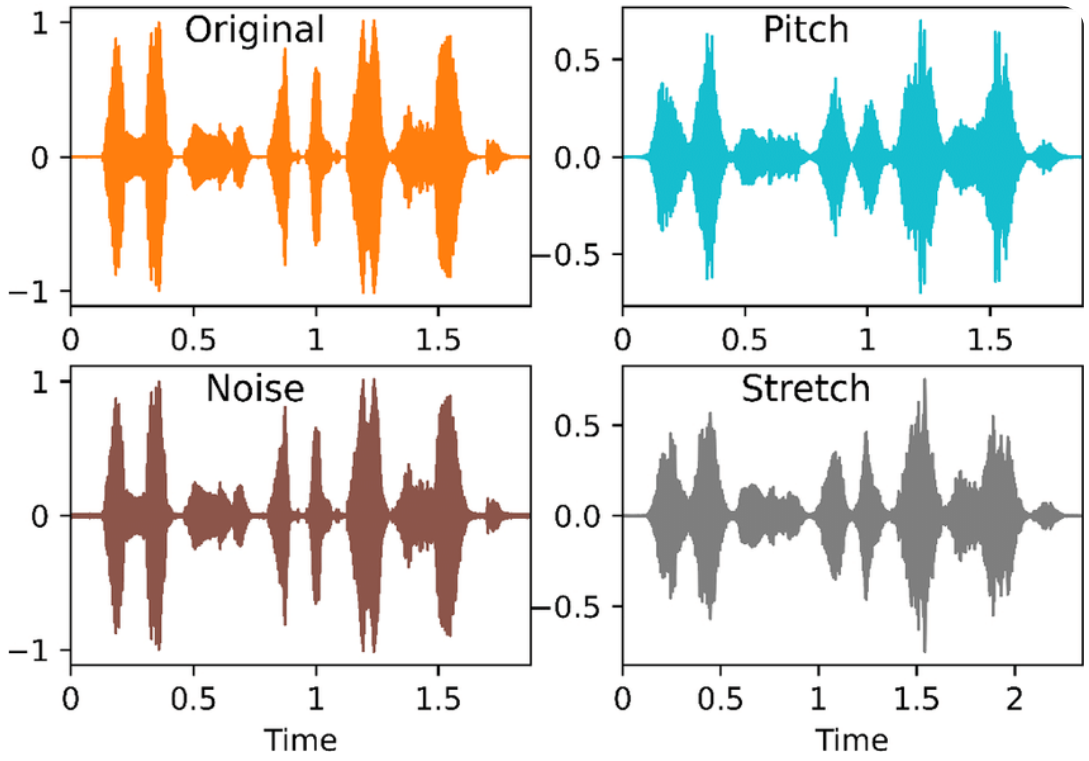
  `tf.keras.layers.RandomHeight`

  `tf.keras.layers.RandomWidth`

  `tf.keras.layers.RandomContrast`

# Enkel voor images?

- Audio
  - Ruis toevoegen
  - Toon wijzigen
  - Versnellen/vertragen
- Video
  - Reeks van beelden
  - Zelfde transformatie nodig voor alle beelden



# Wat je zelf doet, doe je (soms) beter!

- Niet alle mogelijke augmentaties bestaan
- Je kan er zelf toevoegen
  - Op basis van een lambda laag met een functie

```
def random_invert_img(x, p=0.5):
  if  tf.random.uniform([]) < p:
    x = (255-x)
  else:
    x
  return x
```

```
def random_invert(factor=0.5):
  return layers.Lambda(lambda x: random_invert_img(x, factor))

random_invert = random_invert()
```

  - Overerving van een Layer

```
class RandomInvert(layers.Layer):
  def __init__(self, factor=0.5, **kwargs):
    super().__init__(**kwargs)
    self.factor = factor

  def call(self, x):
    return random_invert_img(x)
```

# Wanneer voer je data augmentatie uit?

- Voor de training
  - Asynchroon (non-blocking) op de CPU
  - Meer opslag nodig om de verschillende varianten bij te houden
  - Lagen niet standaard mee geëxporteerd maar kunnen wel toegevoegd worden
- Voeg de preprocessing lagen toe aan het model
  - Augmentaties uitgevoerd synchroon met de andere lagen
  - Kan uitgevoerd worden op de GPU
  - Mee geëxporteerd bij bewaren model
    - Standaardisaties worden automatisch uitgevoerd (Resizing, Cropping, …)
    - Augmentaties (Random-Rotation, …) enkel uitgevoerd bij .fit()

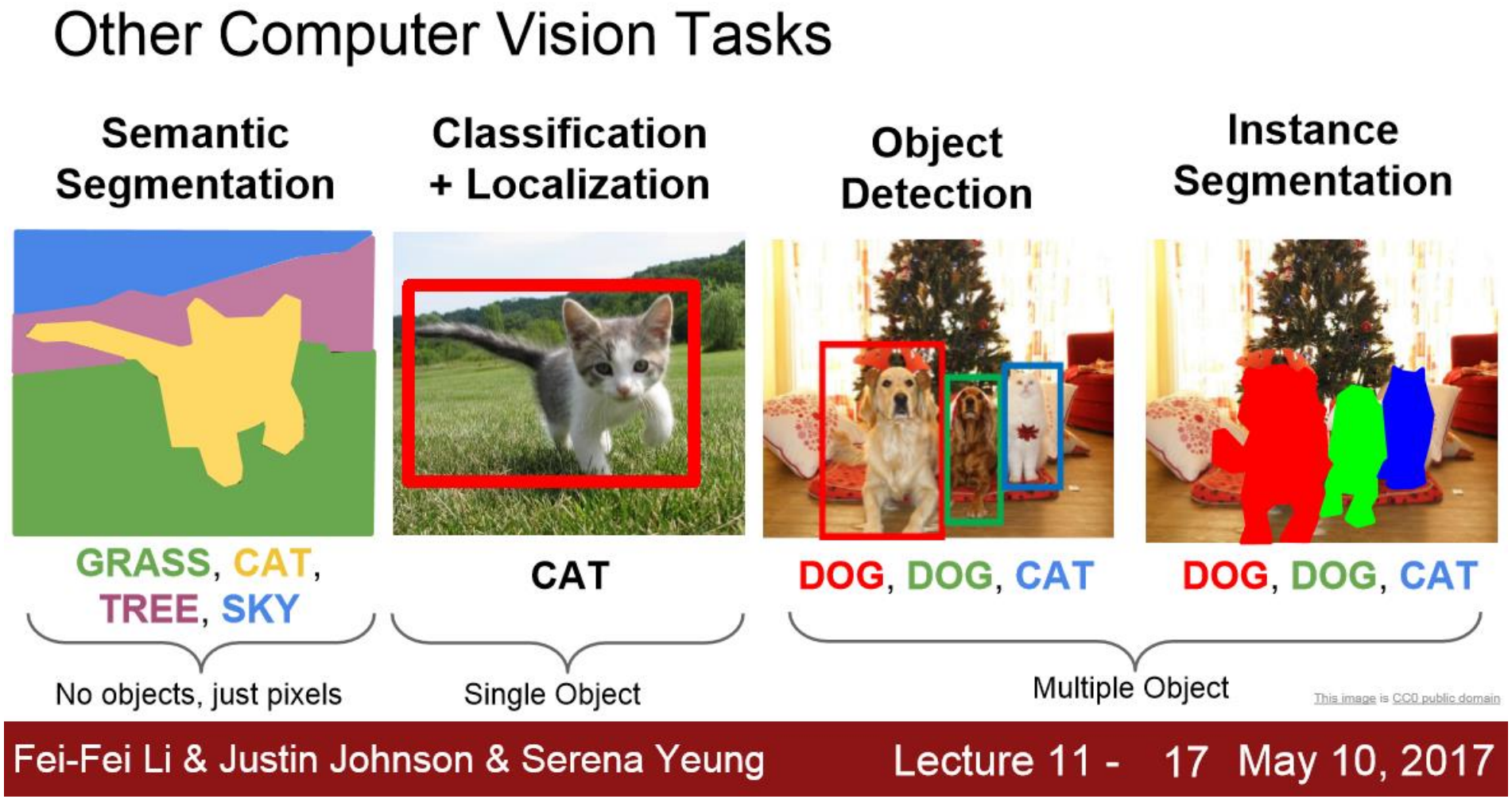
# Computer visie

# Wat zijn de grootste problemen bij computervisie?

# Wat zijn de grootste problemen bij computervisie?

- Algoritmes zien individuele pixels ipv deelfiguren
  - Eigenlijk nog erger want ze zien maar 1 kleur
- Resolutie van huidige figuren is heel groot
  - Heel veel parameters/gewichten die getrained moeten worden
  - 1024 * 1024 figuur heeft per neuron en miljoen gewichten
- Heel veel data nodig om al deze gewichten te trainen

# Toepassingen:



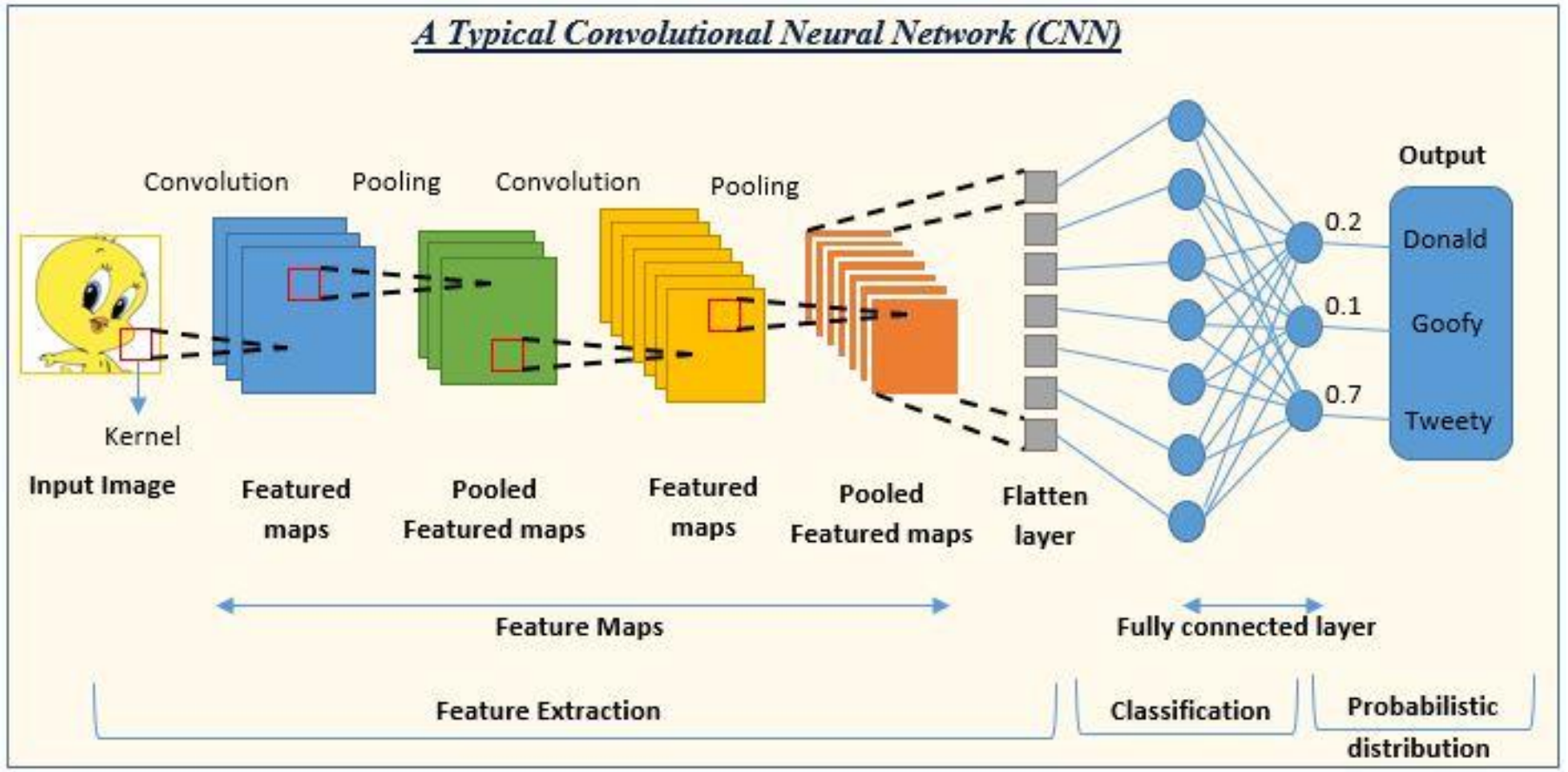
# Convolutioneel neuraal netwerk of CNN

- Poging om computers beelden te laten zien zoals mensen het doen
  - Kijken naar nabijgelegen pixels om kleine objecten te herkennen (ogen, neus, oren, …)
  - Kijken naar deze nabijgelegen objecten om grotere zaken te herkennen (gezicht)
  - Herhaal het vorige
- Het aantal parameters in het neuraal netwerk reduceren
  - Efficiënter om te trainen
  - Minder data nodig voor een goed model op te bouwen

# Convolutioneel neuraal netwerk of CNN

Nieuwe lagen: Convolutionele en Pooling

Standaard Neuraal netwerk



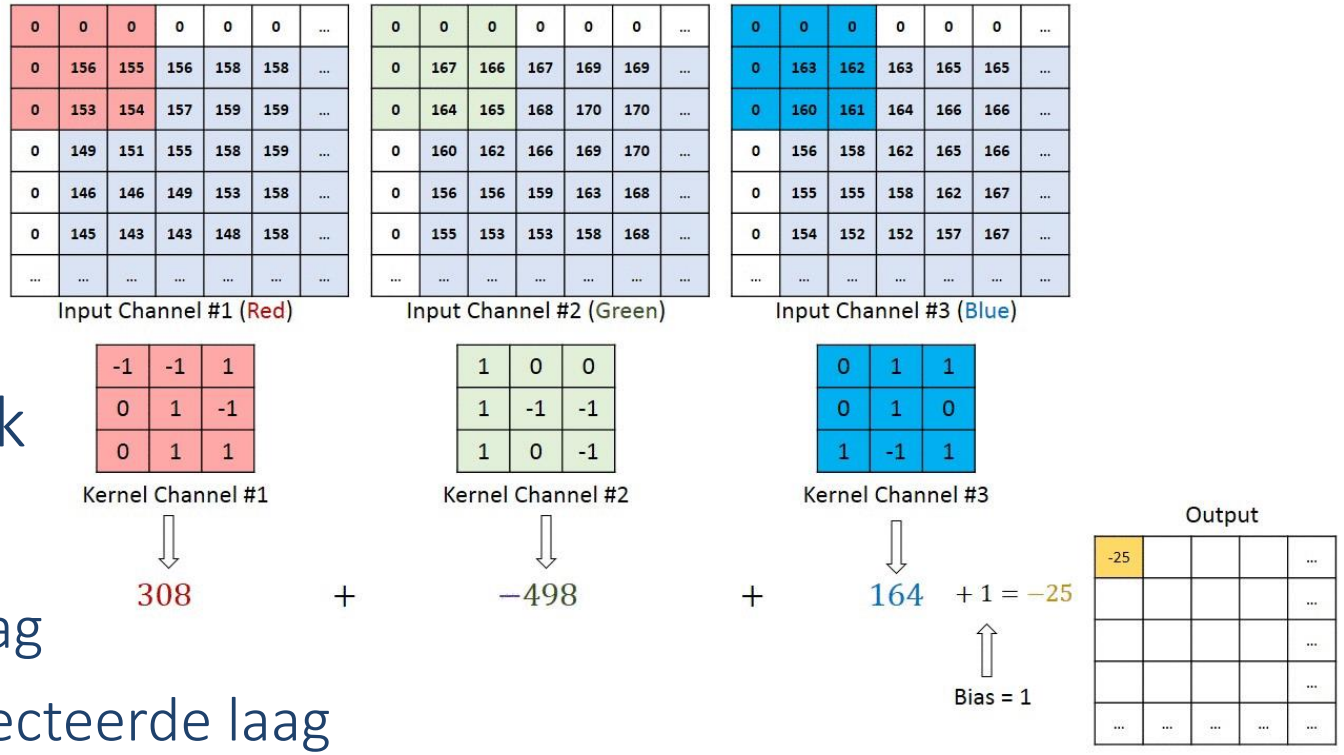
# Convolutioneel neuraal netwerk of CNN

- Convolutionele laag
  - Zoek verband tussen nabijgelegen pixels
- Pooling laag
  - Reduceer de dimensies
- Standaard Neuraal netwerk
  - Kan 1 of meerdere lagen bevatten

- Goede bron met animaties: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

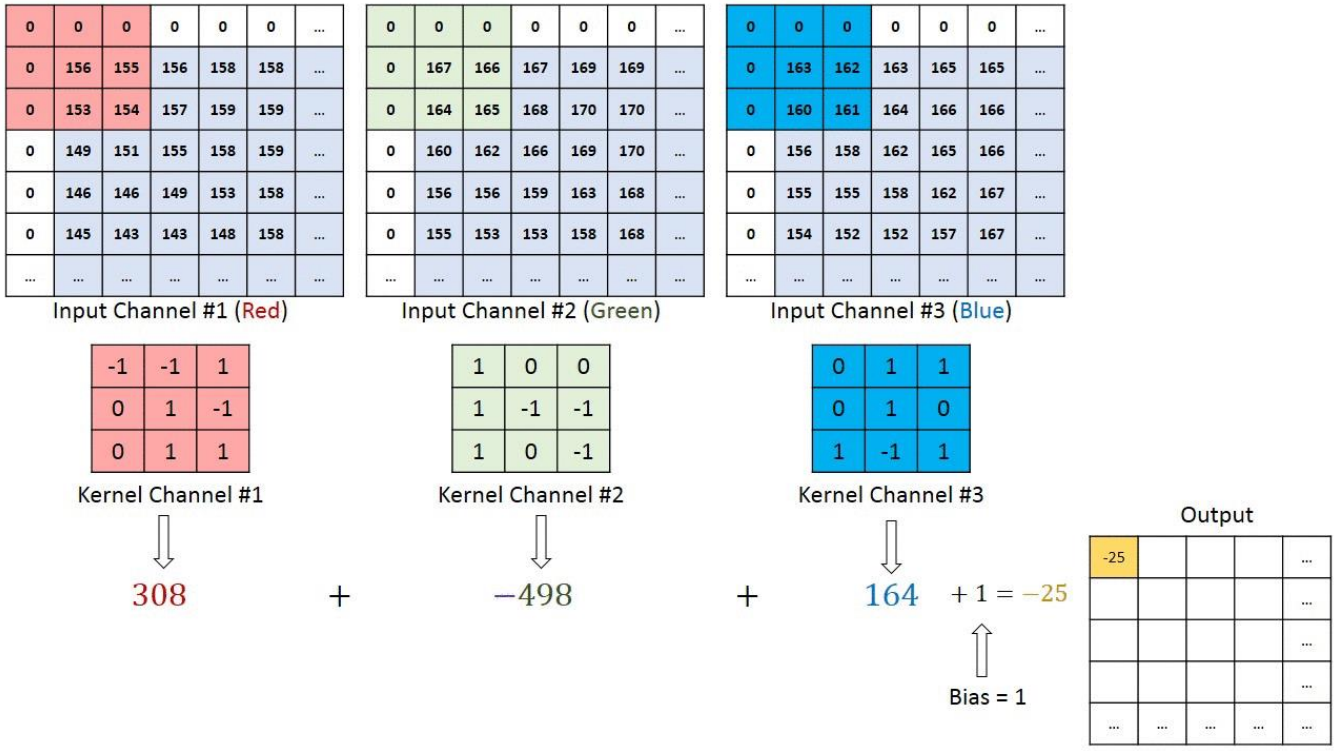
# Convolutionele laag

- Pixel-waarden naar hogere niveau features omzetten
- Zet elke pixel om naar een combinatie van de omliggende pixels
  - 3x3 of 5x5 meestal
- Kernel = de matrix met gewichten
  - Worden getraind
  - Alle pixels met dezelfde gewichten
- Meerdere kernels per laag mogelijk
- In geval van 3x3
  - 27 gewichten * aantal lagen vorige laag
  - Veel minder dan een volledig geconnecteerde laag



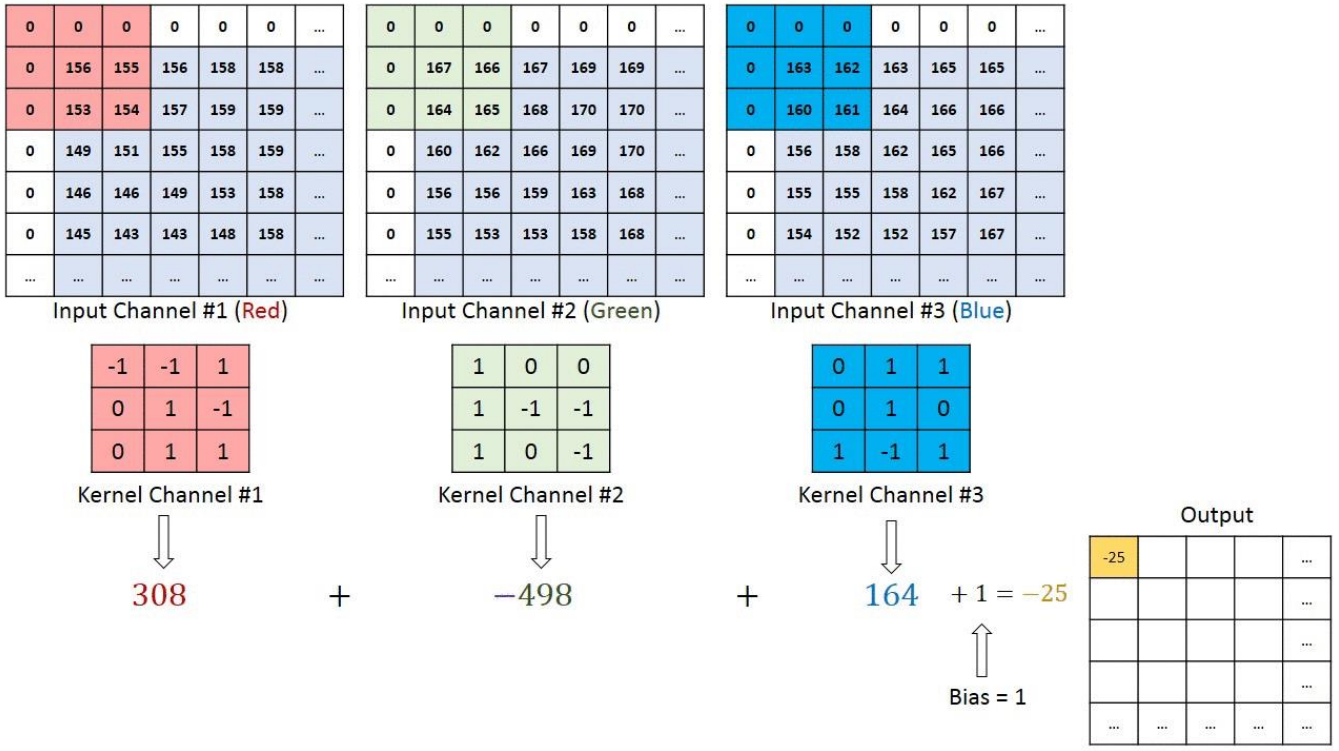
# Convolutionele laag

- Twee mogelijke manieren van padding mogelijk
  - Padding: wat je moet doen aan de randen aangezien de pixels niet bestaan
  - Valid padding
    - Laat de randen weg
    - Resulteert in lagere dimensie
  - Same padding of zero padding
    - Voeg nullen toe (zoals hiernaast)
    - Dimensies blijven hetzelfde (same)



# Convolutionele laag

- Stride
  - Het aantal pixels dat de kernel opschuift elke stap
  - Stapgrootte
    - Typisch 1 maar soms groter
  - Stride groter dan 1 resulteer in
    - Kleinere dimensie van outputs



# Convolutionele laag: hyperparameters

- Dimensies van de kernels
- Type van padding
- Aantal kernels
- Stapgrootte - stride

# Convolutionele laag: eindresultaat

- Convolutionele laag zoekt naar hogere level features
- Meedere convolutionele lagen kunnen gebruikt worden
- Meerdere kernels per laag nodig
- Dimensies van de figuren blijven gelijk
  - Aantal gewichten van het laatste neurale network neemt sterk toe
- Vergeet geen activatiefunctie uit te voeren na elke convolutionele laag
  - Vaak ReLu gebruikt: alle negatieve waarden worden 0

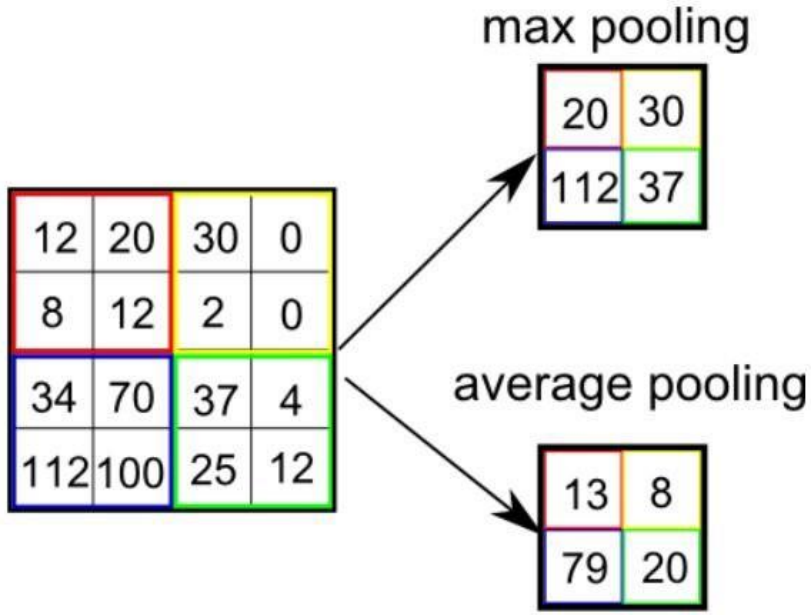
# Oefening convolutionele laag

- Input: Grijsbeelden van 48 x 48

- Wat zijn de outputdimensies indien de kernel
  - 3x3, zero padding, stride 1 ->
  - 3x3, zero padding, stride 2 ->
  - 5x5, valid padding, stride 1 ->
  - 7x7, valid padding, stride 3 ->

# Pooling layer

- Dimensionality reduction
  - Performantie kost van convolutionele laag compenseren
  - Bepaald door de “stride” van de laag
- Dominante features detecteren
- Hierbij wordt gebruik gemaakt van een filter/matrix
  - Geen gewichten om te trainen



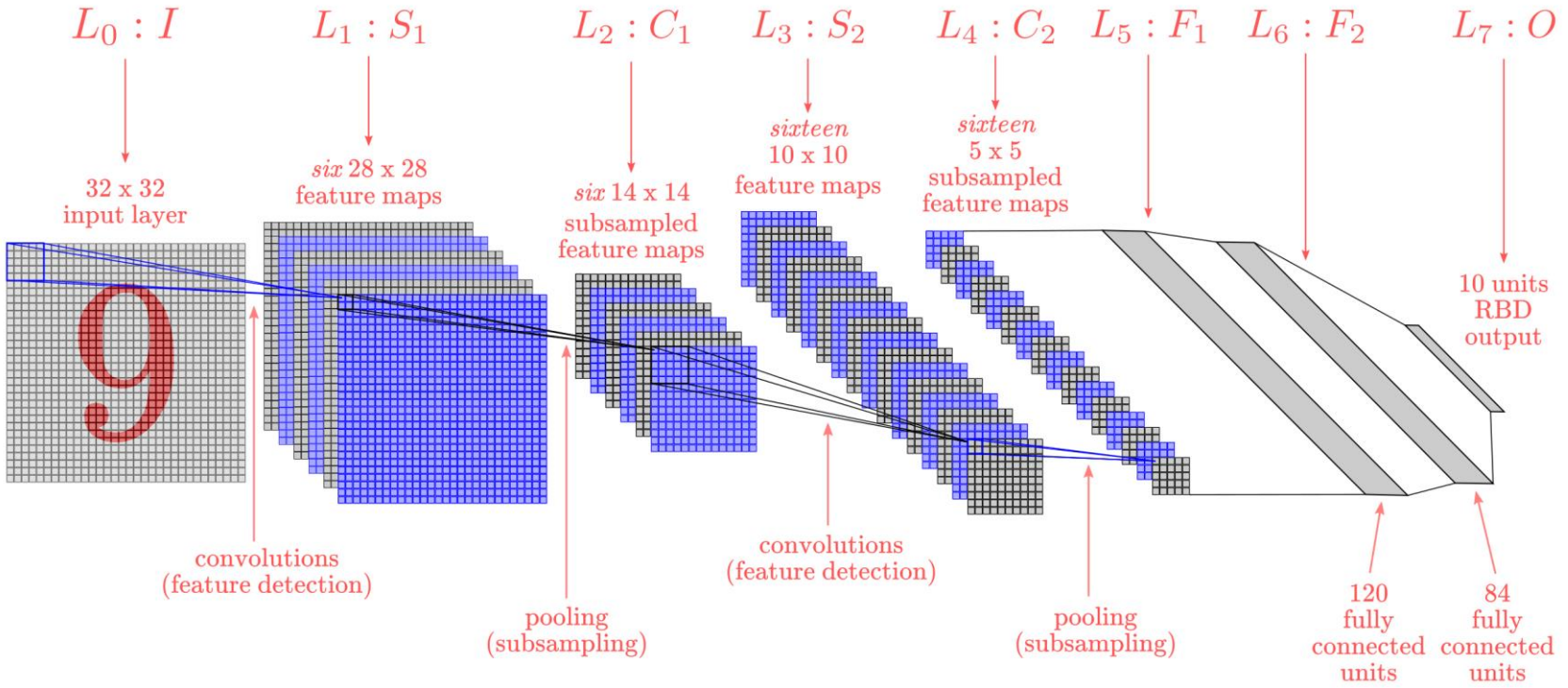
# Pooling layer: Hyperparameters

- Dimensies en stride
  - Hoe groot is de filter waarin we gaan kijken en hoe veel pixels schuiven we de filter op elke stap (stride)
  - Bepaalt de reductie in dimensies
  - Typisch: 2x2 met een stride van 2 wat resulteert in een halvering van de dimensies
- Type pooling
  - Max pooling
    - Onderdrukt ruis en typisch beter
  - Average pooling

| | Layer Type | Kernel Attribute | Number of Filters | Feature Map Size |
|---|---|---|---|---|
| | Image Input Layer | | | 115 × 51 × 3 |
| L1 | Convolutional Layer | 7 × 7 × 3, stride 2, no padding | 96 | 55 × 23 × 96 |
| | ReLU Layer | | | |
| | CCN Layer | | | |
| | Max Pooling | 3 × 3, stride 2, no padding | | 27 × 11 × 96 |
| L2 | Convolutional Layer | 5 × 5 × 96, stride 1, 2 × 2 zero padding | 128 | 27 × 11 × 128 |
| | ReLU Layer | | | |
| | CCN Layer | | | |
| | Max Pooling | 3 × 3, stride 2, no padding | | 13 × 5 × 128 |
| L3 | Convolutional Layer | 3 × 3 × 128, stride 1, 1 × 1 zero padding | 256 | 13 × 5 × 256 |
| | ReLU Layer | | | |
| L4 | Convolutional Layer | 3 × 3 × 256, stride 1, 1 × 1 zero padding | 256 | 13 × 5 × 256 |
| | ReLU Layer | | | |
| L5 | Convolutional Layer | 3 × 3 × 256, stride 1, 1 × 1 zero padding | 128 | 13 × 5 × 128 |
| | ReLU Layer | | | |
| | Max Pooling | 3 × 3, stride 2, no padding | | 6 × 2 × 128 |
| F1 | Fully Connected Layer | | | 4096 |
| | ReLU Layer | | | |
| F2 | Fully Connected Layer | | | 2048 |
| | ReLU Layer | | | |
| | Dropout | | | |
| F3 | Fully Connected Layer | | | 5 |
| | Softmax Layer | | | |

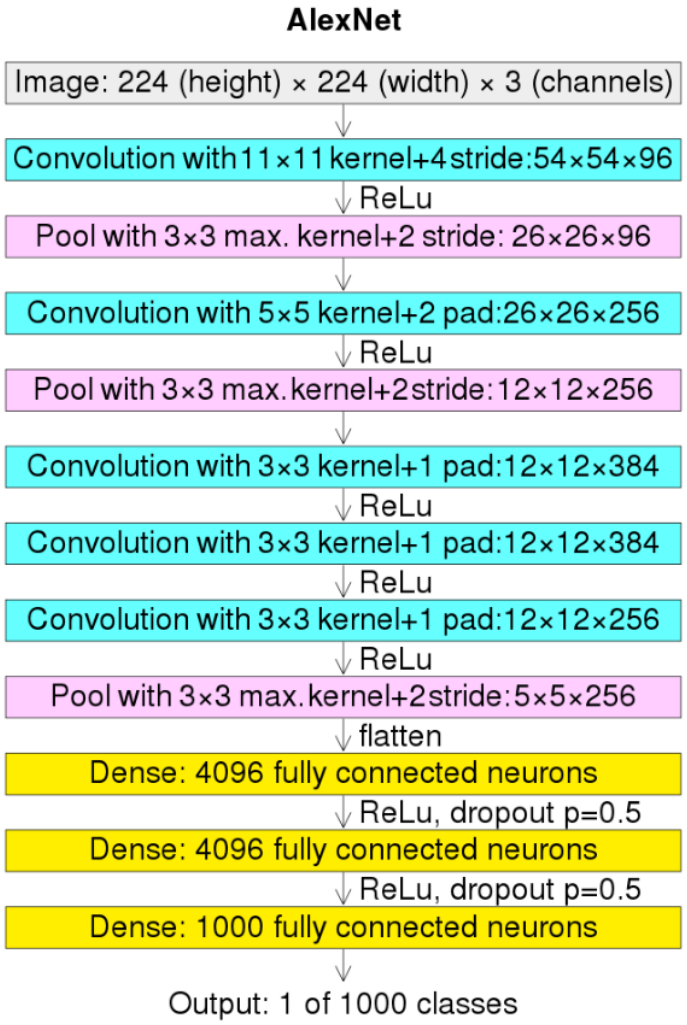
# Veel gebruikte CNN's

- LeNet
  - Voorgesteld in 1989
  - Om handgeschreven cijfers te herkennen
    - Toegepast voor postcodes
  - Kernels van 5x5
  - Hoe van 6 naar 16?
    - Kernel is 6x5x5



# Veel gebruikte CNN's

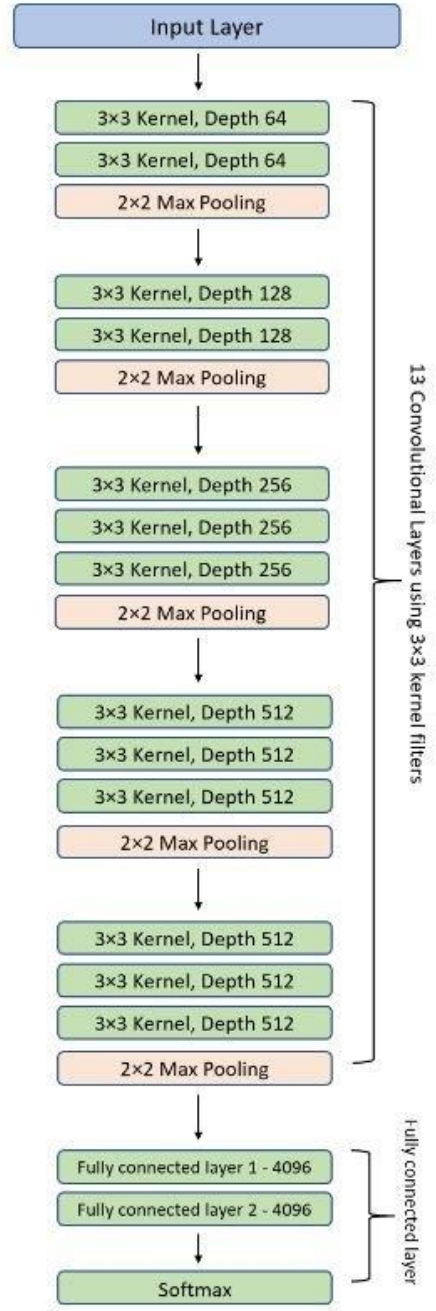
- AlexNet
  - Heel invloedrijk concept geweest binnen deep learning en impact van GPU's



# Veel gebruikte CNN's

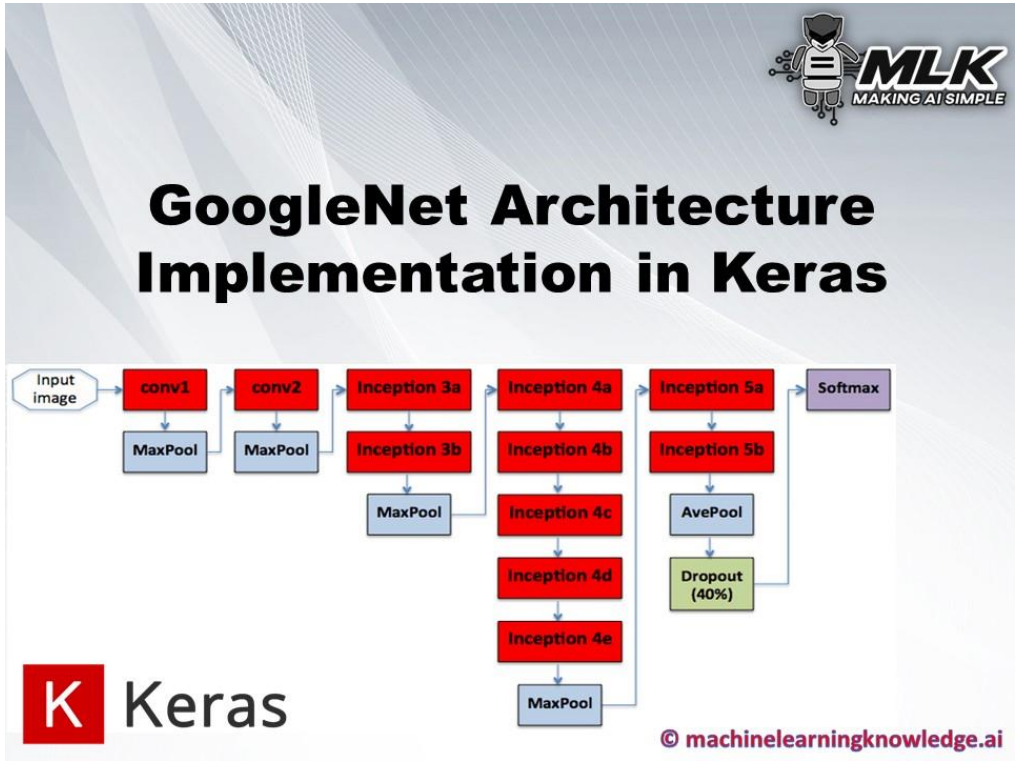
- VGGNet
  - Visual geometry group
  - Deep CNN
    - Deep = veel convolutionele lagen
    - 16 tot 19 lagen
  - Meer lagen resulteert in betere performantie
    - Niet altijd het geval bij standaard neurale netwerken

- Bron: https://viso.ai/deep-learning/vgg-very-deep-convolutional-networks/



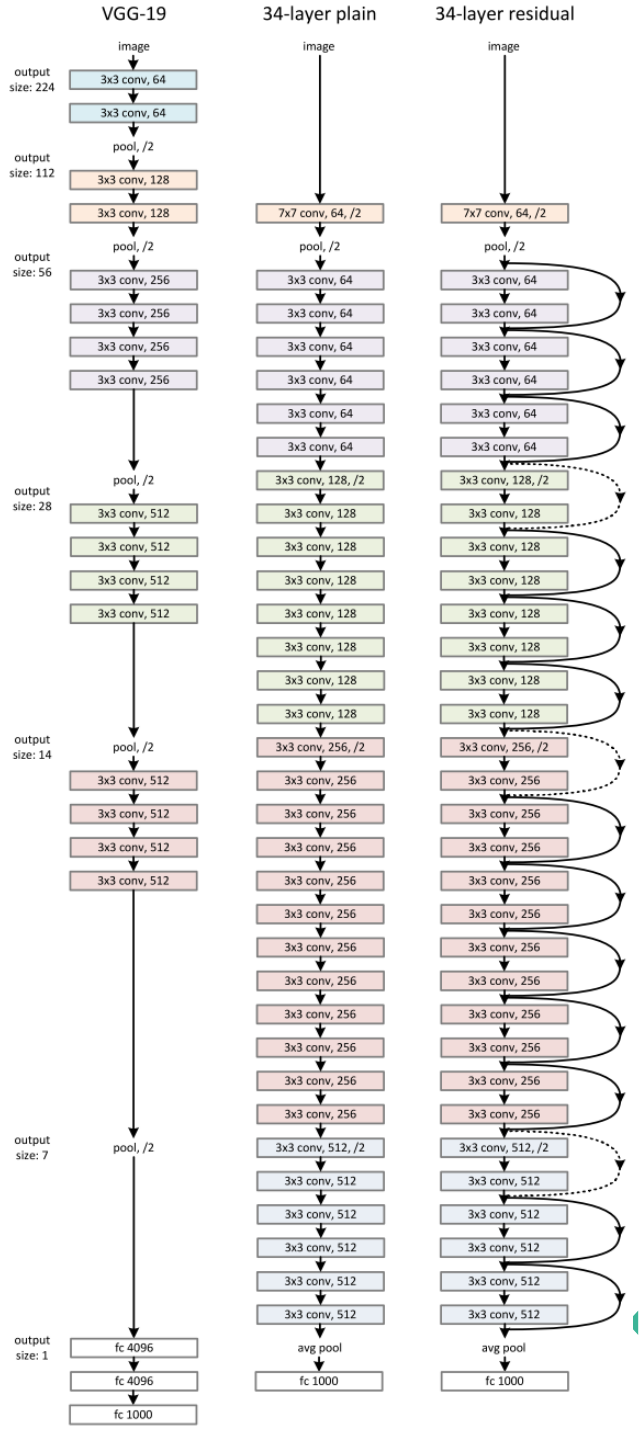
# Veel gebruikte CNN's

- GoogLeNet
  - 27 lagen in het model
  - Heel snel de dimensies reduceren
    - Grote kernel 7x7 om niet te veel informatie te verliezen
  - Inception lagen bevatten ook convolutionele lagen
  - Kan 1000 klassen herkennen in een figuur



# Veel gebruikte CNN's

- ResNet
  - Residual neural netwerk
  - Maakt gebruik van identity shortcut connections
    - Laat de output ook een aantal lagen overslaan en geef het opnieuw als input
    - Lost gedeeltelijk het vanishing gradient probleem op
      - Meerdere lagen zouden steeds beter moeten zijn
- Basis voor een reeks state-of-the-art modellen in computervisie



# Huiswerk