

Deep Learning Opdracht: Schilderijen Classificeren

2^e Bachelor EL-ICT
Specialisatie Artificiële Intelligentie

andy.louwyck@vives.be
stefaan.haspeslagh@vives.be



Doel

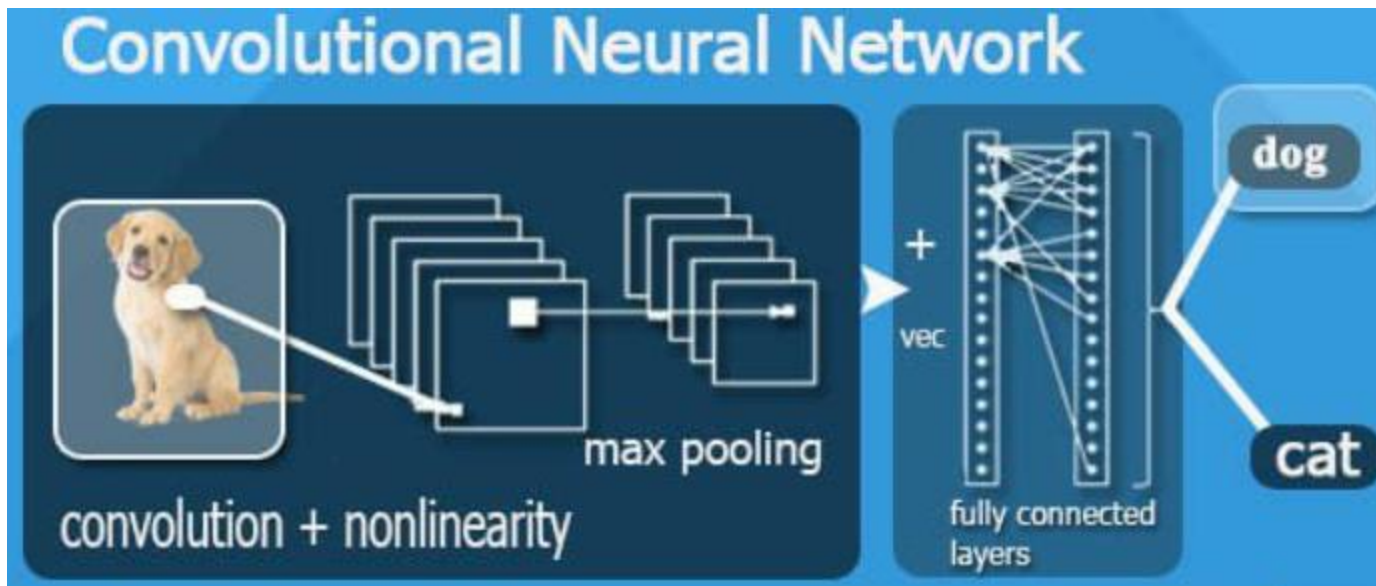
Ontwikkel een applicatie die schilderijen classificeert

Computer says: Picasso



Hoe?


M.b.v. een convolutioneel neuraal netwerk (ConvNet)
(zie hoofdstukken 8 en 9)

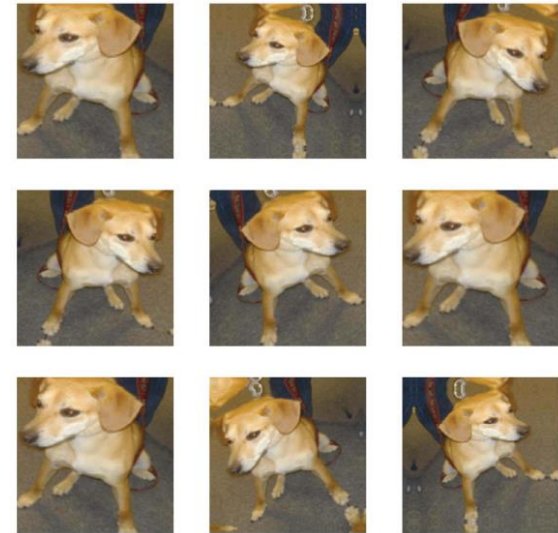


Praktisch

- **Individuele opdracht** – uitwisselen van info mag wel
- **Start:** 31 maart 2025
- **Deadline:** Woensdag 21/5/2025 om 23u59
- **Begeleiding:** tijdens de lessen
- **Examen:**
 - Mondelinge verdediging op examen
- **Omgevingen:**
 - Python modules ontwikkelen: PyCharm + GitHub
 - Modellen trainen: Google Colab
 - Notebooks met documentatie: Google Colab

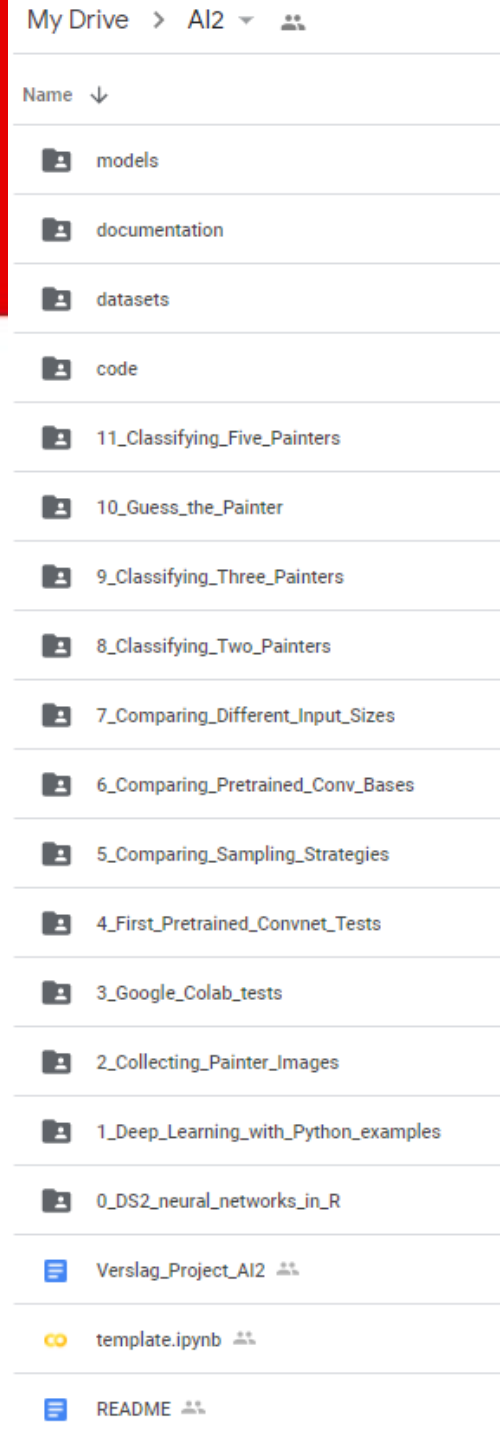
Stappen

- **Dataset:**
 - Images verzamelen
 - Images opkuisen
 - Module ontwikkelen om datasets te creëren
- **Modellen:**
 - Data augmentation? 
 - Sampling strategie?
 - ConvBase?
 - Image input size?
 - Combinaties van schilders uitproberen
- **Applicatie:**
 - Minimaal: 2 schilders met 95% nauwkeurigheid
 - Finale model(len) inbouwen in eenvoudige (web)app



Oplevering

- **GitHub repo** delen met de docent
 - shaspesl-ai
 - ReadMe met korte uitleg over structuur
 - Code om images te manipuleren
 - Code om modellen te trainen
 - Code Applicatie + korte manual
 - Versies vermelden in documentatie!
 - Projectverslag met uitgebreide uitleg stappen, mag in notebook!!!
 - Literatuur
 - Datasets
 - Modellen

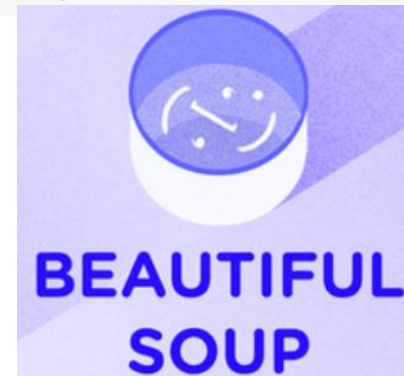


Dataset: images verzamelen

- **Gegeven:** schilderijen van 3 schilders
 - 1529 images met schilderijen van Picasso
 - 628 images met schilderijen van Rubens
 - 330 images met schilderijen van Mondriaan
 - Gedeelde folder op OneDrive
- **Zelf verzamelen:** schilderijen van Rembrandt
 - Manueel images downloaden
 - 1 website waarvan je images gaat scrapen
 - Bijv. <http://www.rembrandtpainting.net/>
 - Scripts om te scrapen → GitHub repo

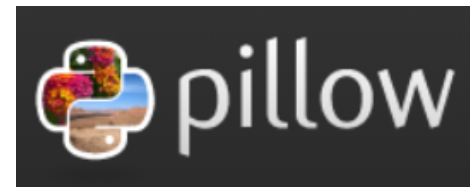
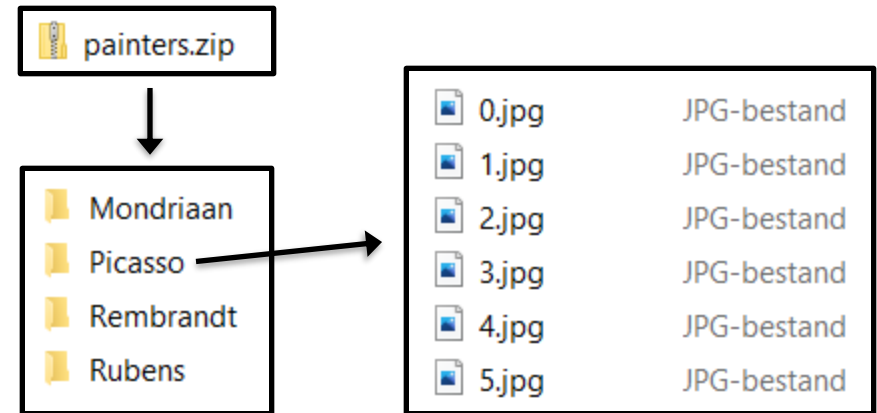
📁 Mondriaan
📁 Picasso
📁 Rubens

```
from bs4 import BeautifulSoup  
import requests
```



Dataset: images opkuisen

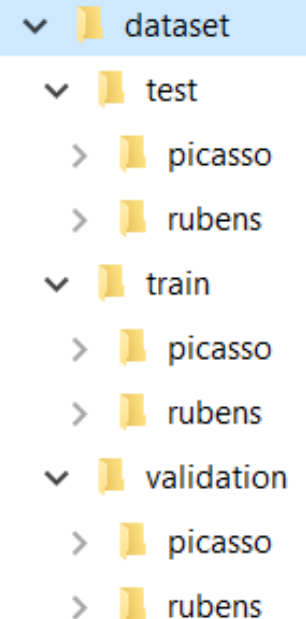
- Images per schilder
- Originele size behouden
- Corrupte files verwijderen
- Enkel JPEG images
 - Andere formaten omzetten
- Images hernoemen
 - Consistente filenamen
 - Bijv. nummeren per schilder
- Comprimeren: zip, HDF5, ... ?
- Python libraries
 - Pillow (zie vak “Programming in Python”)
 - OpenCV
 - ...
- Scripts → GitHub



Module ontwikkelen om datasets te creëren

- **Originele dataset** bevragen:
 - Hoeveel images van bepaalde schilder?
 - Statistieken ivm image size: min, max, mean, median, P25, P75
- **Nieuwe dataset** creëren: train, validation, test
 - Aantal images?
 - Undersampled
 - Oversampled
 - Imbalanced
 - Image size?
 - Originele size
 - Opgegeven size
 - Opgegeven statistiek: min, max, mean, median, ...
 - Welke interpolatie methode? Bilineair, nearest neighbor, bikubisch
- Images **visualiseren**: 1 image, slideshow, ...
- Python module → GitHub

```
import os
import shutil
```



```
dataset
├── test
│   ├── picasso
│   └── rubens
├── train
│   ├── picasso
│   └── rubens
└── validation
    ├── picasso
    └── rubens
```

Dataset creëren: sampling strategie

- **Undersampled:**
 - Aantal samples in elke klasse = aantal samples van kleinste klasse
- **Oversampled:**
 - Aantal samples in elke klasse = aantal samples van kleinste klasse
 - Willekeurig kopiëren van samples in klassen met te weinig samples
 - Dubbels worden weggewerkt met data augmentation technieken
- **Imbalanced:**
 - Behouden van originele aantal samples in elke klasse
 - Werken met gewichten: parameter `class_weight`
- **Voorbeeld: 600 Rubens en 1200 Picasso**
 - Undersampled: 600 images in elke klasse
 - Oversampled: 1200 images in elke klasse
 - Imbalanced: Rubens 2 x groter gewicht dan Picasso

```
class_weight = {0: 1.0 , 1: 2.0}  
class_weight
```

```
{0: 1.0, 1: 2.0}
```

```
m_imbalanced.fit(  
    steps_per_epoch=100,  
    epochs=20,  
    class_weight=class_weight  
)
```

Dataset creëren: images resizen

Bv. Pillow: methodes `resize` en `thumbnail`

`Image.resize` (*size, resample=None, box=None, reducing_gap=None*) [\[source\]](#)

Returns a `resize`d copy of this image.

Parameters

- **size** – The requested size in pixels, as a 2-tuple: (width, height).
- **resample** – An optional resampling filter. This can be one of `PIL.Image.NEAREST`, `PIL.Image.BOX`, `PIL.Image.BILINEAR`, `PIL.Image.HAMMING`, `PIL.Image.BICUBIC` OR `PIL.Image.LANCZOS`. If the image has mode “1” or “P”, it is always set to `PIL.Image.NEAREST`. If the image mode specifies a number of bits, such as “I;16”, then the default filter is `PIL.Image.NEAREST`. Otherwise, the default filter is `PIL.Image.BICUBIC`. See: [Filters](#).

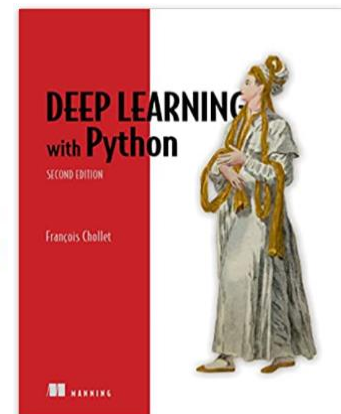
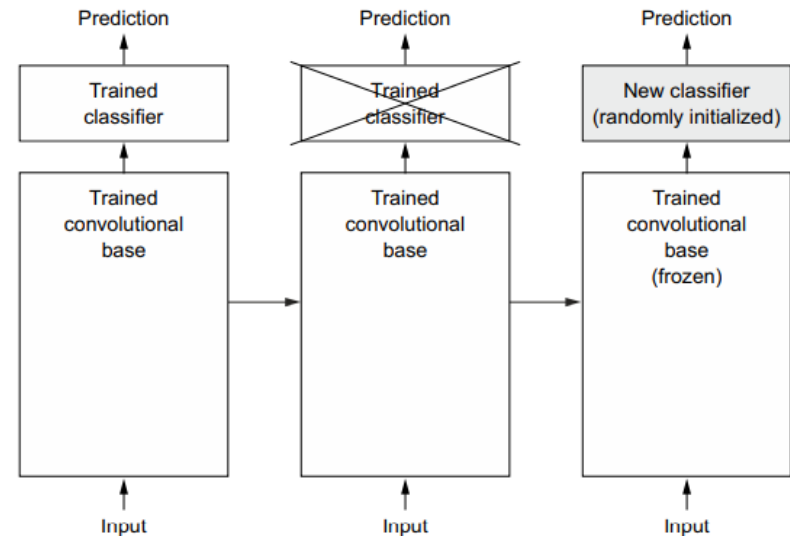
`Image.thumbnail` (*size, resample=3, reducing_gap=2.0*) [\[source\]](#)

Make this image into a `thumbnail`. This method modifies the image to contain a `thumbnail` version of itself, no larger than the given size. This method calculates an appropriate thumbnail size **to preserve the aspect of the image**, calls the `draft()` method to configure the file reader (where applicable), and finally resizes the image.

Aspect ratio!!!

Modellen: belangrijkste onderdeel!!

- Transfer learning
 - Welk pretrained ConvNet?
 - Bovenste laag ConvBase hertrainen?
- Datasets:
 - Beste sampling strategie?
 - Beste data augmentation technieken?
 - Beste image input size?
- Werk stap voor stap:
 - 2 schilders (minimum vereiste!)
 - 3 schilders
 - 4 schilders
- Inspiratie: boek hfst 8
- Valideren en testen!!! Zie ML
- Werk gestructureerd!
- Bespreek alle stappen in je verslag
- Notebooks + datasets → Google Drive

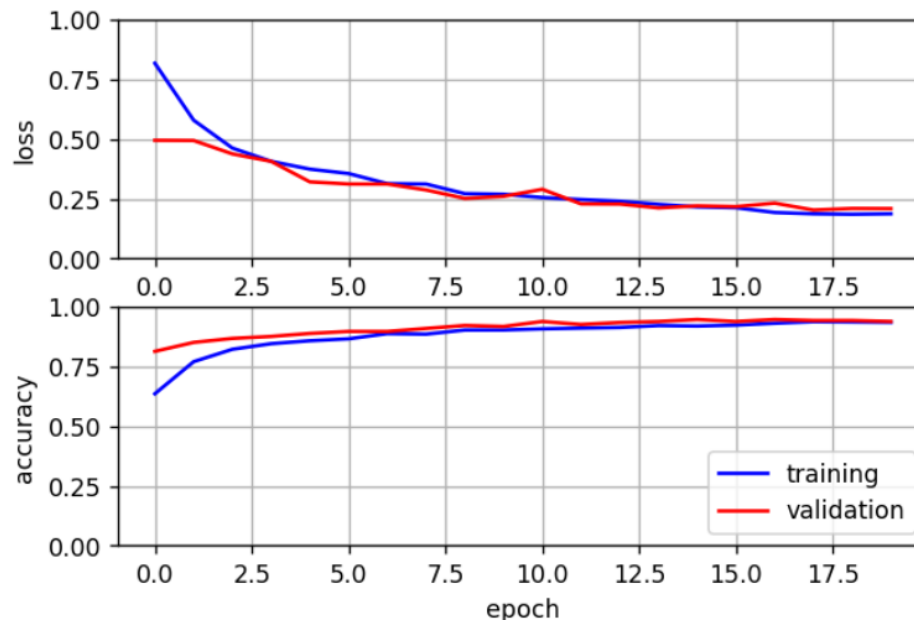


Modellen trainen: callbacks

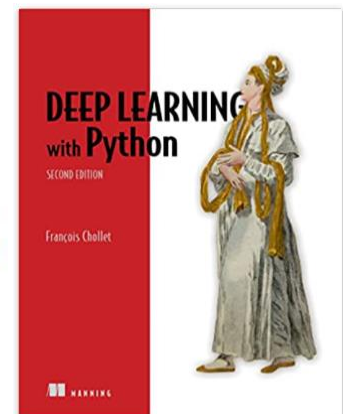
Maak gebruik van callbacks tijdens het trainen:

- Zie hfst 7 uit het boek: EarlyStopping, ModelCheckpoint
- Schrijf eigen callback om loss & accuracy “real-time” te plotten

```
history = m.fit(epochs=20, steps_per_epoch=steps_per_epoch)
```



```
training loss: 0.18774836206187803  
validation loss: 0.2094840109348297  
training accuracy: 0.93395835  
validation accuracy: 0.9375
```



Modellen trainen: start

Zie hoofdstuk 8:

- Neem 600 images van Rubens en 600 images van Picasso:
 - Trainingset: 400
 - Validatieset: 100
 - Testset: 100
- Pas achtereenvolgens de volgende modellen toe:
 - Eenvoudige convnet uit Listing 8.7 (p215)
 - Zelfde model met data augmentation: Listing 8.14 (p221) en 8.16 (p223)
 - Feature extraction met VGG16: Listing 8.19 (p227) en 8.20 en 8.21 (p229)
 - Uitbreiding met data augmentation: Listing 8.23 (p231) en 8.25 (p232)
 - Finetuning van VGG16 conv base: Listing 8.27 en 8.28 (p236)
- Maak plots van training en validatie loss en nauwkeurigheid
- Evalueer je modellen mbv een testset en maak confusion matrices

Modellen trainen: vervolg

- Dataset uitbreiden + volgende zaken onderzoeken:
 - Beste sampling strategie?
 - Beste data augmentation technieken?
 - Beste image input size?
- Transfer learning toepassen + volgende zaken onderzoeken:
 - Welk pretrained ConvNet?
 - Bovenste laag ConvBase hertrainen? (= finetunen)
- Aantal schilders uitbreiden + verschillende combinaties uitproberen:
 - 2 schilders (minimum vereiste!)
 - 3 schilders
 - 4 schilders
 - 5 schilders

Modellen evalueren

- Hoe ga je de dataset opsplitsen?
 - Hold-out validation?
 - (Iterated) K-fold crossvalidation?
- Welke evaluatiemetrieken pas je toe?
 - Confusion matrix
 - Accuracy
 - Recall en precision
 - ROC en AUC
 - ...
- Hoe krijg je inzicht in de werking van je ConvNet? Zie hfst 9
 - Class activations heatmaps (bijv. fout geclassificeerde images)
 - Visualiseren van “intermediate activations”

Demo-applicatie

- Finale model(len) trainen
- Inbouwen in eenvoudige applicatie
 - Notebook
- Werking:
 - Gebruiker laadt willekeurige image op
 - Model zegt welke schilder het is
- Code + korte manual → GitHub

Computer says: Mondriaan



Veel plezier!!