CAS in Applied Data Science
Muerren 2024

Géraldine Schaller-Conti

Program of the week

17:00 - 18:30 Tutorial (Geraldine) **Convolutions**
19:00 - 20:00 Dinner at hotel Regina
20:30 - 21:00 Neural Style Transformations (an example on how to use deep ML to create art - Sigve)

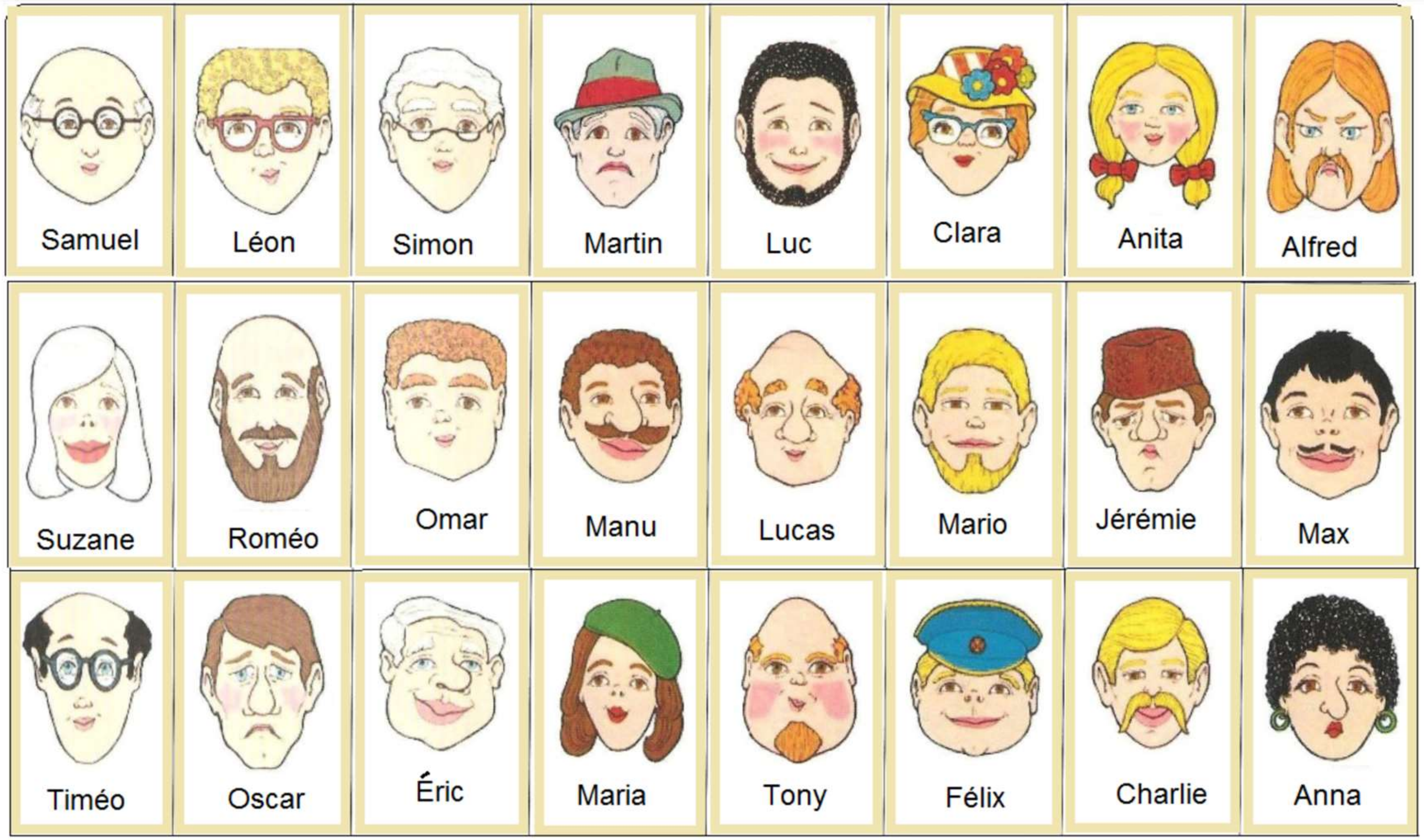
Thursday (Geraldine)
08:15 - 09:00 Lecture **Deep Forward Networks – optimization**
09:15 - 10:00 Tutorial **Transfer Learning**
10:00 - 10:30 Coffee break
10:30 - 12:30 Tutorial **Transfer Learning**
12:30 - 17:00 Skiing, work or whatever
17:00 - 18:15 Tutorial and project discussions **Projects**
19:00 - 22:00 Fondue and sledge ride back down to Regina in the dark
Everyone has to be READY OUTSIDE THE HOTEL at 18:45.
The train up departs at 19:00.

Friday (Geraldine)
08:15 - 09:00 Lecture **Convolutional models and generative models**
09:15 - 10:00 Tutorial **Recurrent Neural Networks**
10:00 - 10:30 Coffee break and Check Out
10:30 - 12:00 Tutorial / Discussion Session **Recurrent Neural Networks**
12:00 - 12:30 Wrap up (Sigve)
12:30 End

About me ☺


EPFL
ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE
CERN
VE RI TAS
HARVARD
Disney Research
heig-vd
HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch
PAG
ne.ch
RÉPUBLIQUE ET CANTON DE NEUCHÂTEL

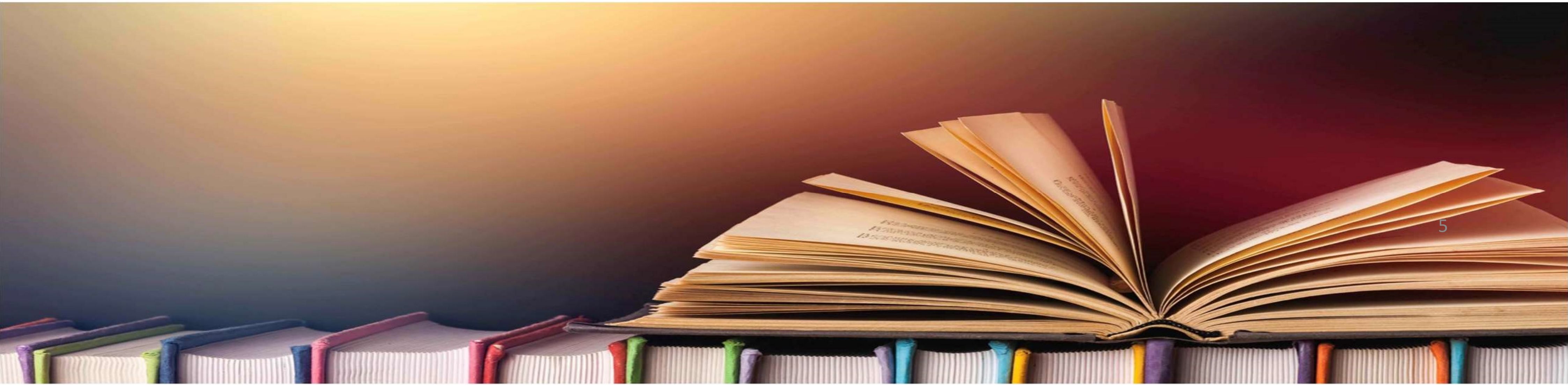
About you ☺



Samuel
Léon
Simon
Martin
Luc
Clara
Anita
Alfred
Suzane
Roméo
Omar
Manu
Lucas
Mario
Jérémie
Max
Timéo
Oscar
Éric
Maria
Tony
Félix
Charlie
Anna

Bibliography

- Deep Learning book (Goodfellow, Bengio, Courville)
- Machine Learning @ Stanford (Prof Andrew Ng)
- Hands-On Machine Learning with Scikit-Learn & Tensorflow (Aurélien Géron)



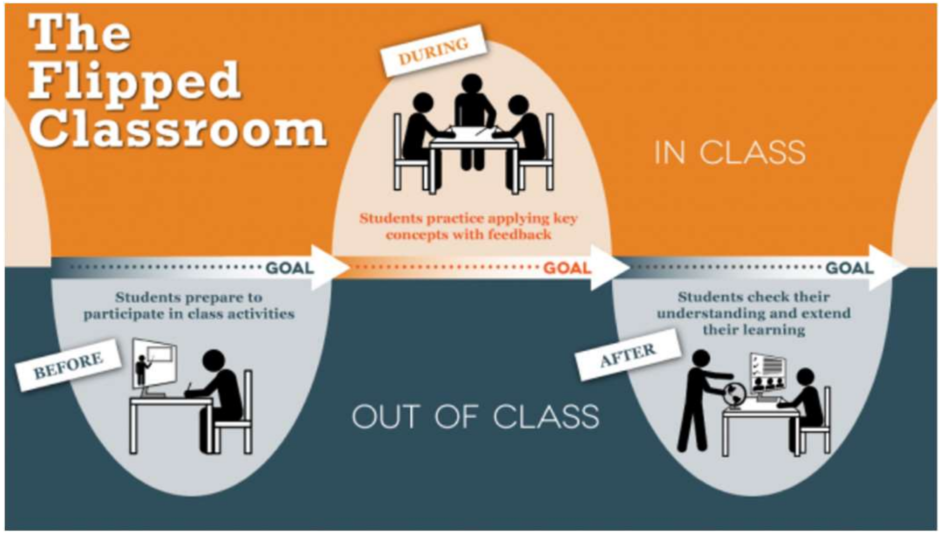
Teaching method

Inverted classroom based

- Introduction lectures
- Real content you learn yourself with the notebooks. *Either to put in practice your knowledge or to learn ahead of another lecture*

Why

- Supposed to be better
- More fun
- Learning by doing



To give back sense to being present (Marcel Lebrun)

Tutorial IV : Convolutions

Introduction

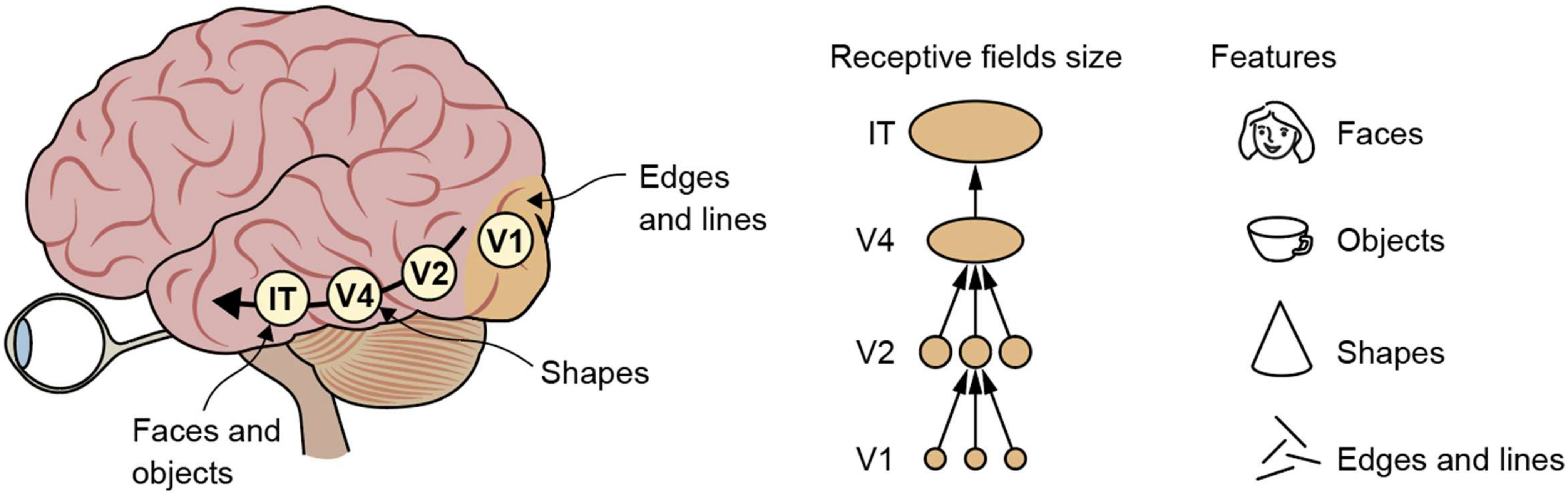
- Goal : basics to perform image recognition

- Program : inverted classroom style
  - Theory
  - Overview to get the big picture of the Notebook
  - Work alone (30 minutes)
  - Work in groups of 4 (30 minutes)

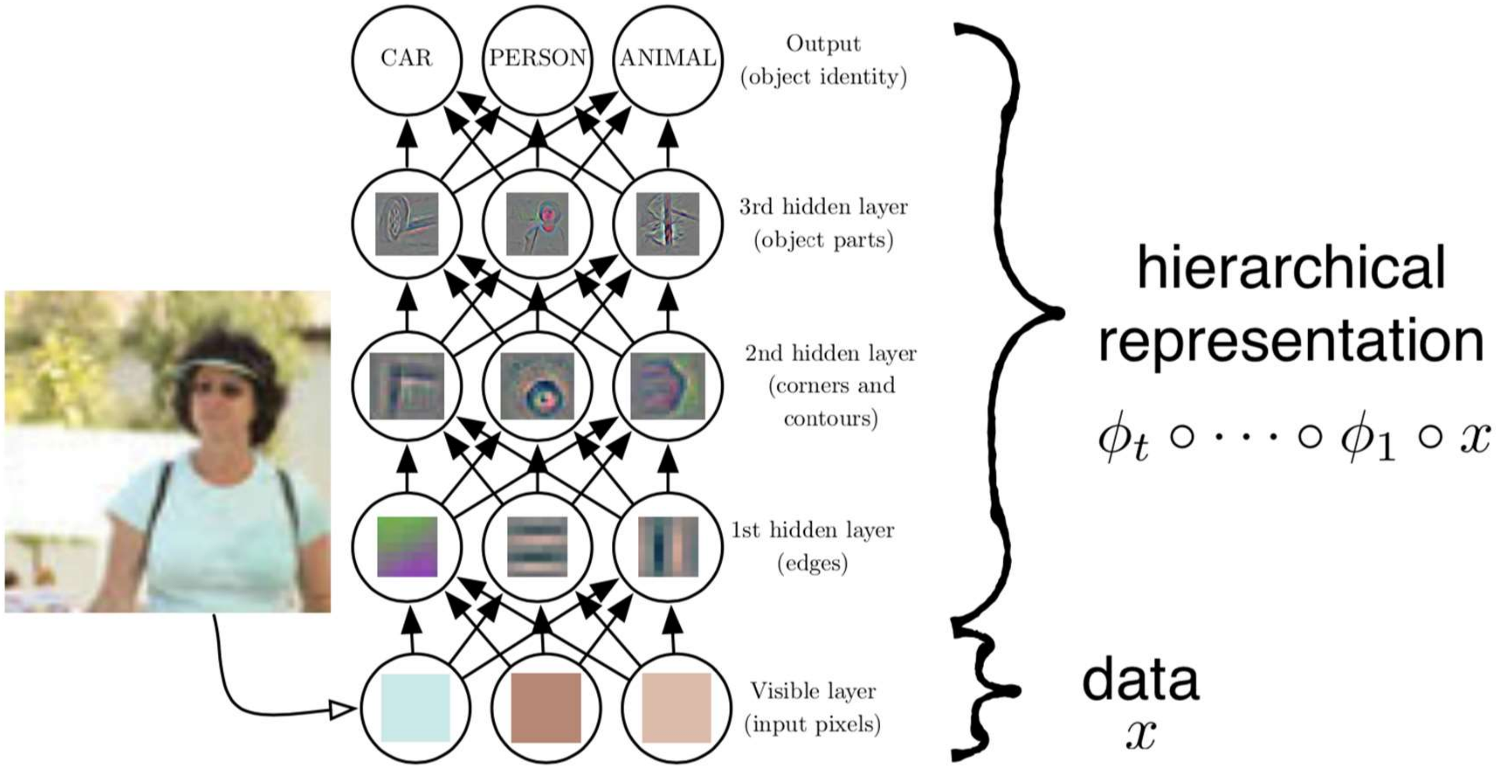
- Technical : Google Colab, Pytorch

Theory

Human vision



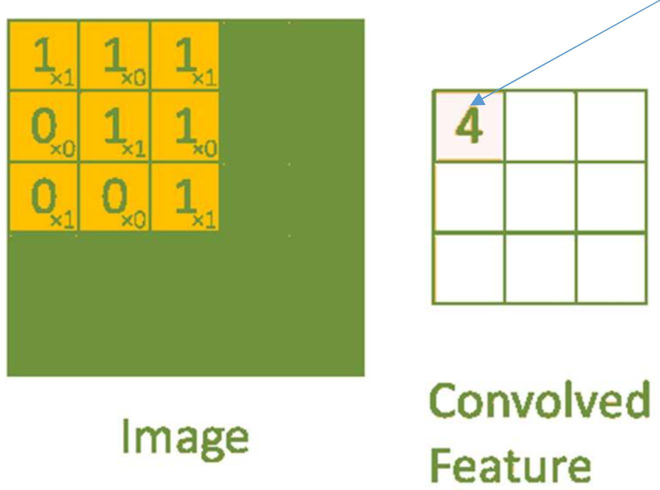
Computer vision



Kernel (filter)

- Used to detect features (vertical/horizontal filter,...)
- Different kernels to create different feature maps → learn to see various patterns and details in images

1	0	1
0	1	0
1	0	1



how well the pattern in the kernel matches the content in that part of the image

= Feature map

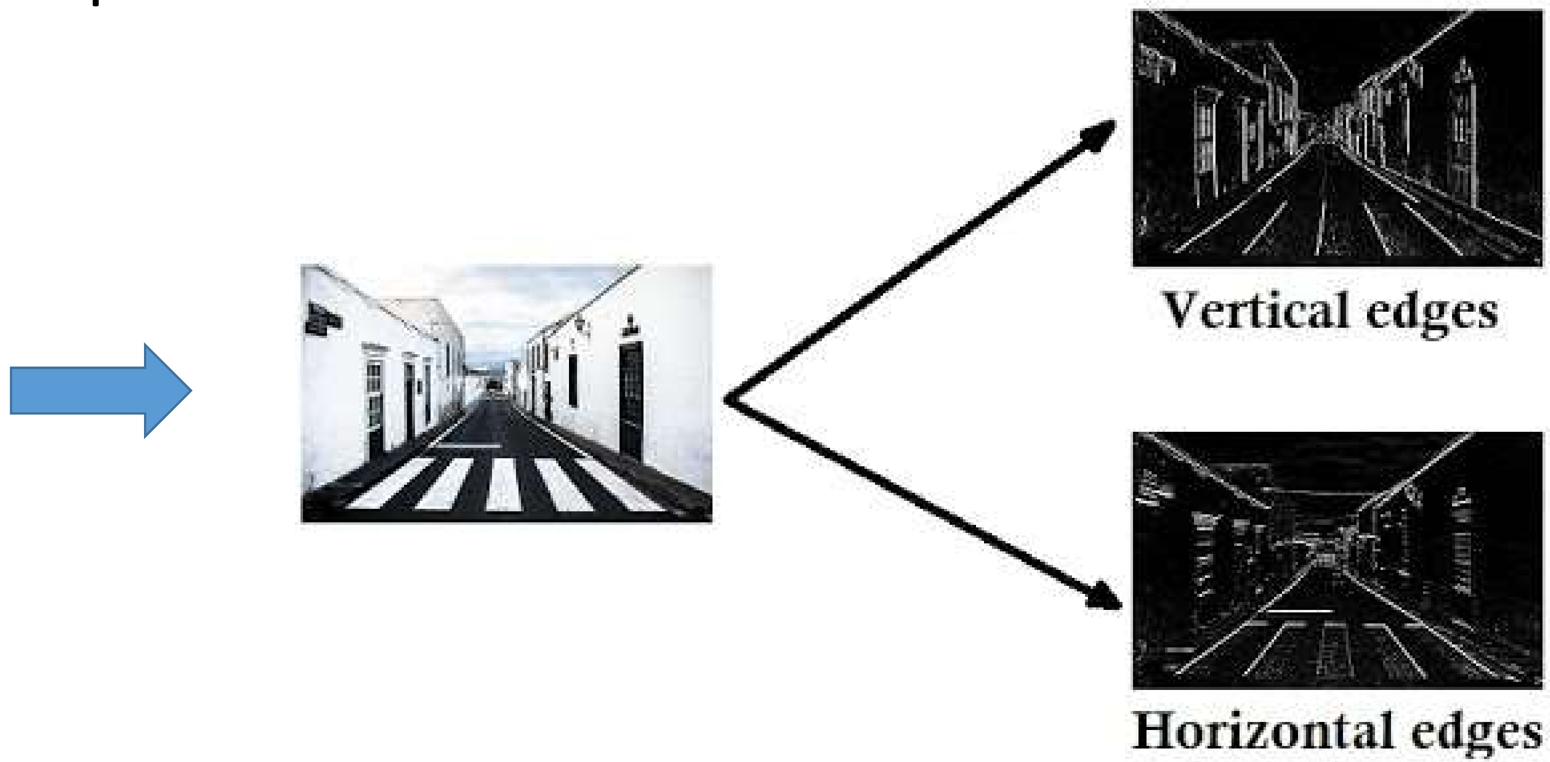
Kernel example

1	0	-1
1	0	-1
1	0	-1

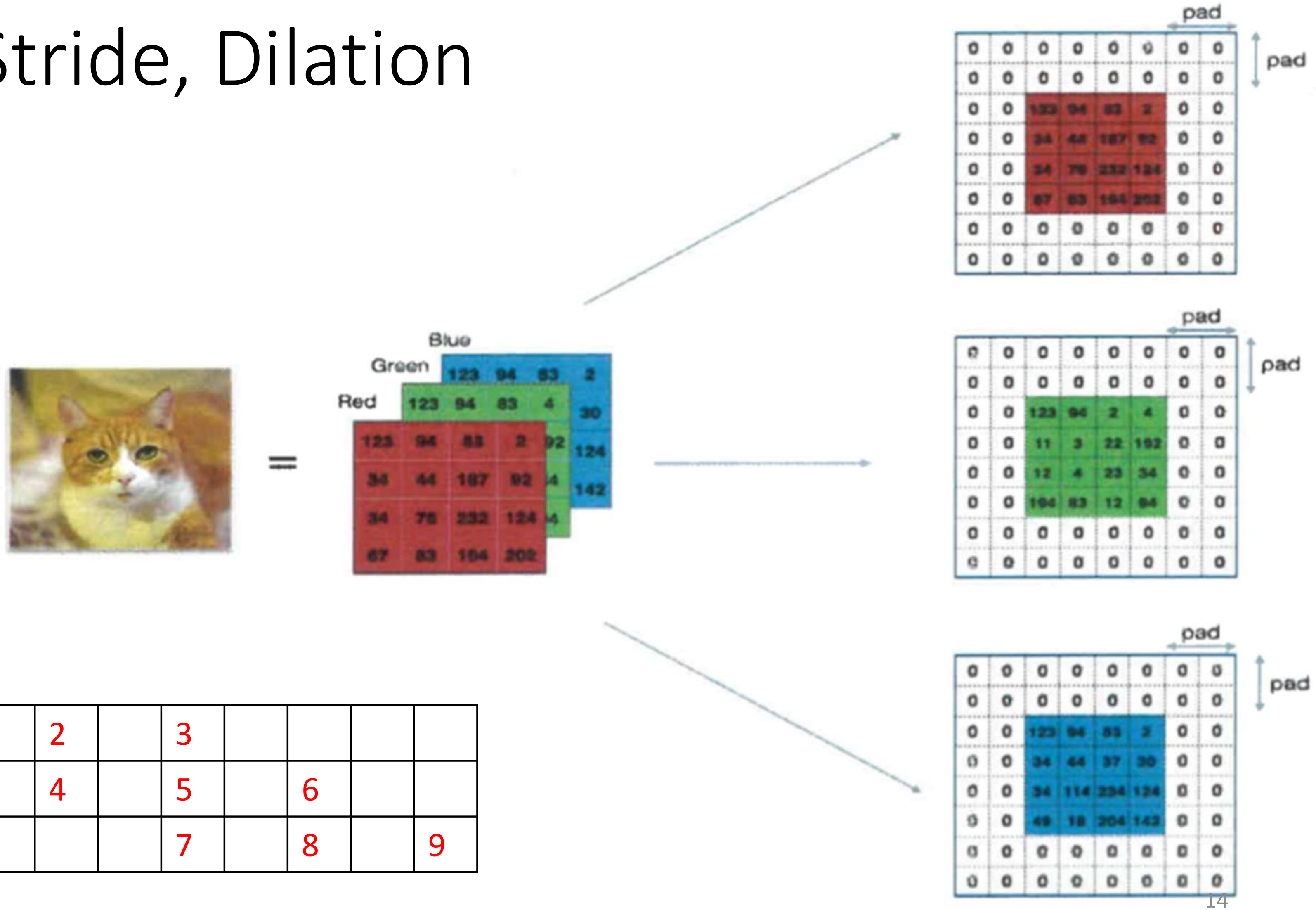
Vertical

1	1	1
0	0	0
-1	-1	-1

Horizontal



Padding, Stride, Dilation

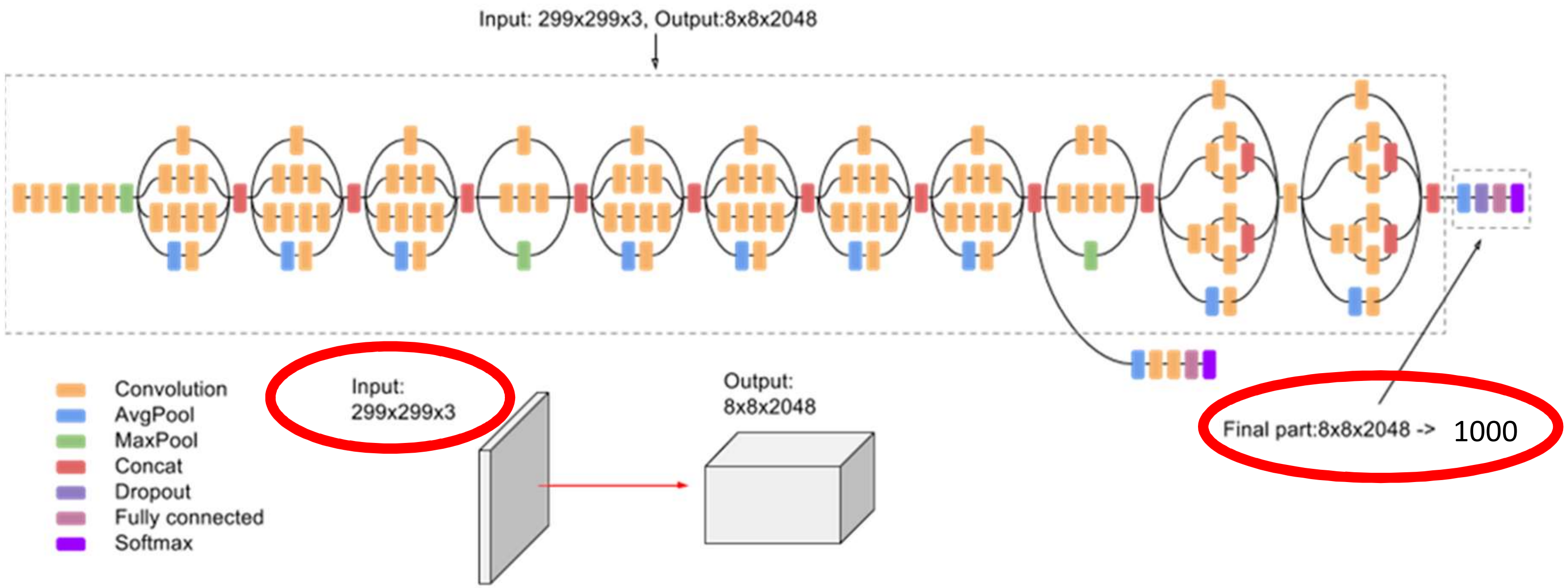


1	2	3
4	5	6
7	8	9

1		2		3				
		4		5		6		
				7		8		9

Inception V3 model

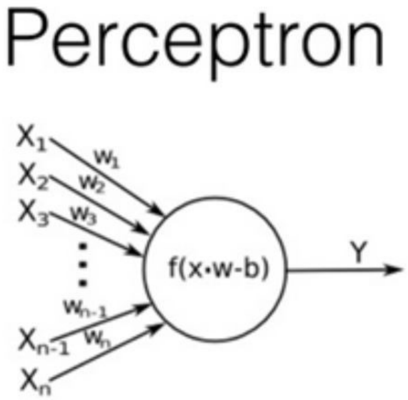
- Deep learning model based on Convolutional Neural Networks
- Used for image classification
- Released in year 2015
- It has 42 layers

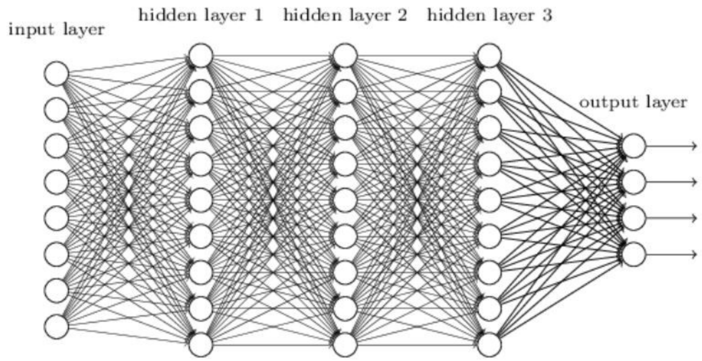


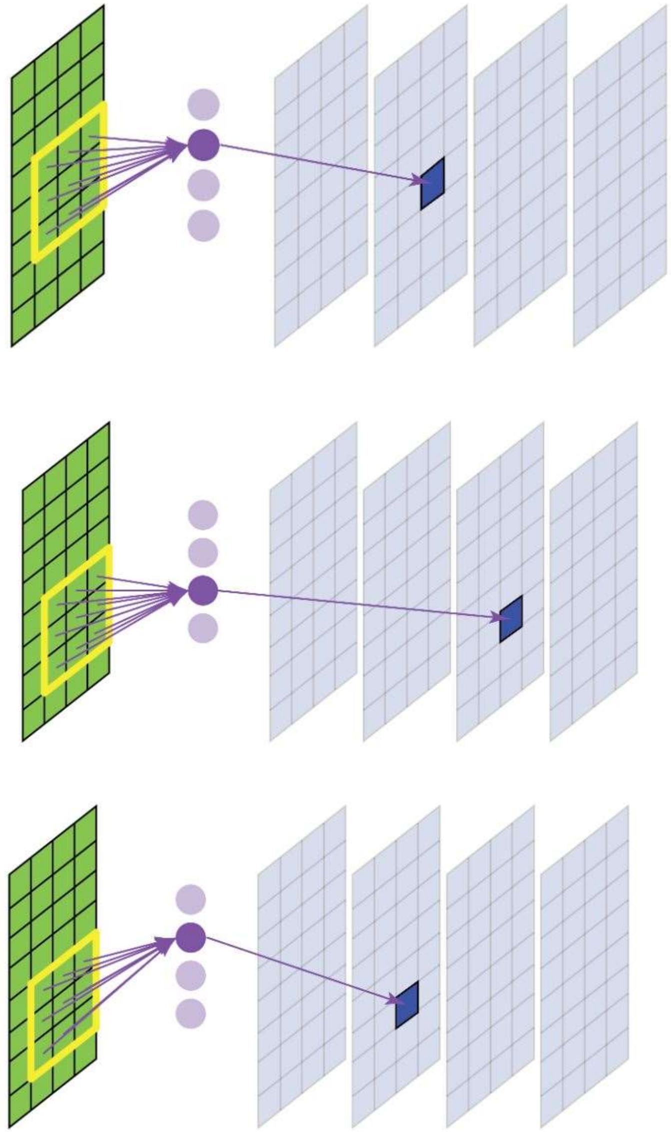
Overview of the notebook

Tutorial IV (1)

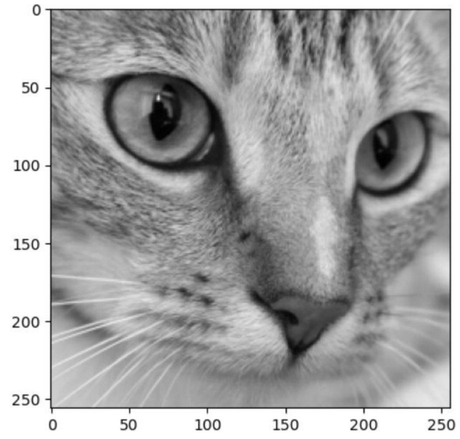
1) Load necessary libraries (common libraries and personal modules)
2) Images
3) Convolutions







Tutorial IV (2)



- Pre-processing of the image
  - Gray-scale, cropping, float conversion, normalization
  - Add dimensions (batch, channel, height, width) (in get_convolved)
  - Convert Numpy to pytorch (in get_convolved)

- Define the convolution
  - Forward model (class Model)
  - Apply 4 convolutions one after each other (inside class Model, call conv_2d function)

- Define the filter
  - Identity filter
  - Convert to np.array (in get_convolved)
  - Add dimensions
  - Convert Numpy to pytorch

```
flt_mtx = [
    [ 0, 0, 0, 0, 0,],
    [ 0, 0, 0, 0, 0,],
    [ 0, 0, 1, 0, 0,],
    [ 0, 0, 0, 0, 0,],
    [ 0, 0, 0, 0, 0,],
] # identity transformation
```

Tutorial IV (3)

- Use it ! (ims_convolved = get_convolved(img_raw, flt_mtx)
  - You get back 5 figures

0	0	0	0	0
0	0	0	0	0
0	0	30	0	0
0	0	0	0	0
0	0	0	0	0

0	0	0	0	0
0	0	0	0	0
0	0	-1	0	0
0	0	0	0	0
0	0	0	0	0

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	1	0	0
0	0	0	0	0

1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

0	-1	1	0	0
0	-1	1	0	0
0	-1	1	0	0
0	-1	1	0	0
0	-1	1	0	0

- Exercise :

1. experiment with different filters and understand what they do, e.g.:

- identity transformation
- identity transformation with positive non-unit values
- identity transformation with negative unit value
- identity transformation off center
- blurring with box filter
- edge detection with + and - bands
- try whatever you like

2. experiment with convolution parameters:

- padding = 1, 2, 3
- stride = 2
- dilation = 2

Tutorial IV (4)

filter type	effect
gaussian	bluring
first derivative of gaussian	detection of edges
second derivative of gaussian	detection of peaks

- Most common filters
  - Define 1D functions
  - Create 2D filters by repeating the 1D filters along axis 0 (np.tile)
  - Multiply by transpose() to get the horizontal dimension (filter size does not change)
  - Use them ! (ims_convolved = get_convolved(img_raw, flt_mtx)

4) Homework (leave it for now)

Tutorial IV (5)

5) Load a pretrained model (inception V3) from torchhub

6) Test the model

- Preprocessing of the image (cropping, shuffle sizes, totensor, normalize adds batch size) → [1,3,299,299]
- Desactivate the gradient (eval mode)
- Get the logits (1000 values), then the probabilities (applying softmax)
- Print out the 5 most probable classes
- Do the same with 100 classes

Tutorial V : Transfer Learning

Introduction

- Goal : use the Inception model to classify images of different nature, learn how to save a model

- Program : inverted classroom style
  - Theory
  - Overview to get the big picture of the notebook
  - Work alone ()
  - Work in groups of 4 (45 minutes, then 2 minute presentation)

- Technical : Google Colab, Pytorch

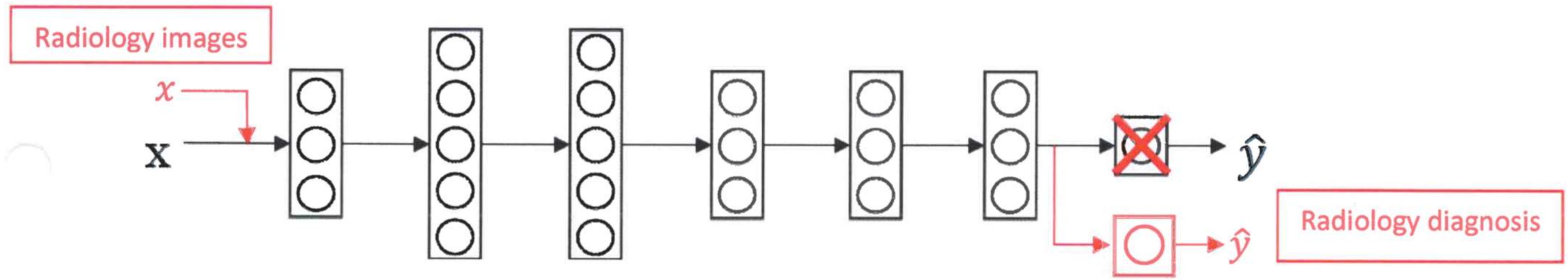
Theory

Transfer Learning

- Try to find an existing neural network that accomplishes a similar task to the one you are trying to tackle

- reuse the lower layers of this network
  - Output layer should usually be replaced

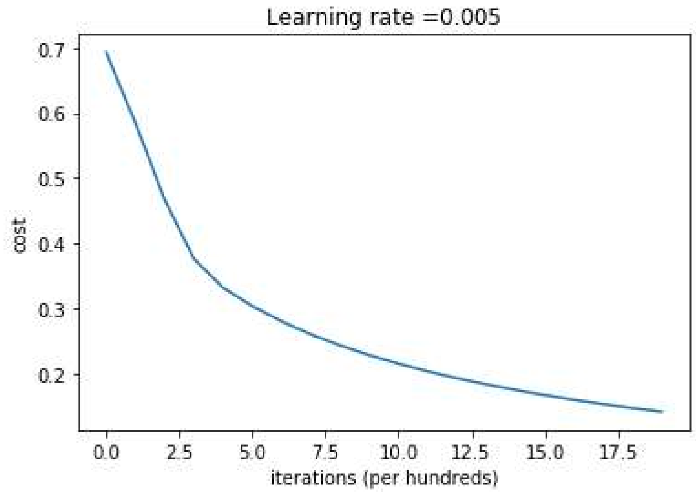
- Speeds up training and requires much fewer training data



Training Loop

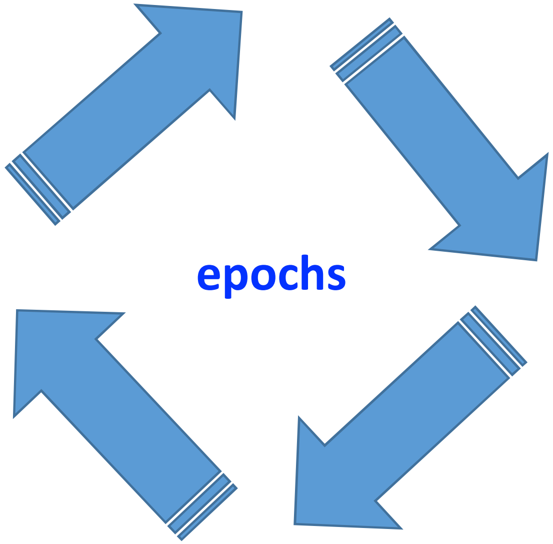
- *Iterative* process

Learning curve



Forward propagation

Make a prediction



epochs

Cost function

Compute the error function

Backward propagation

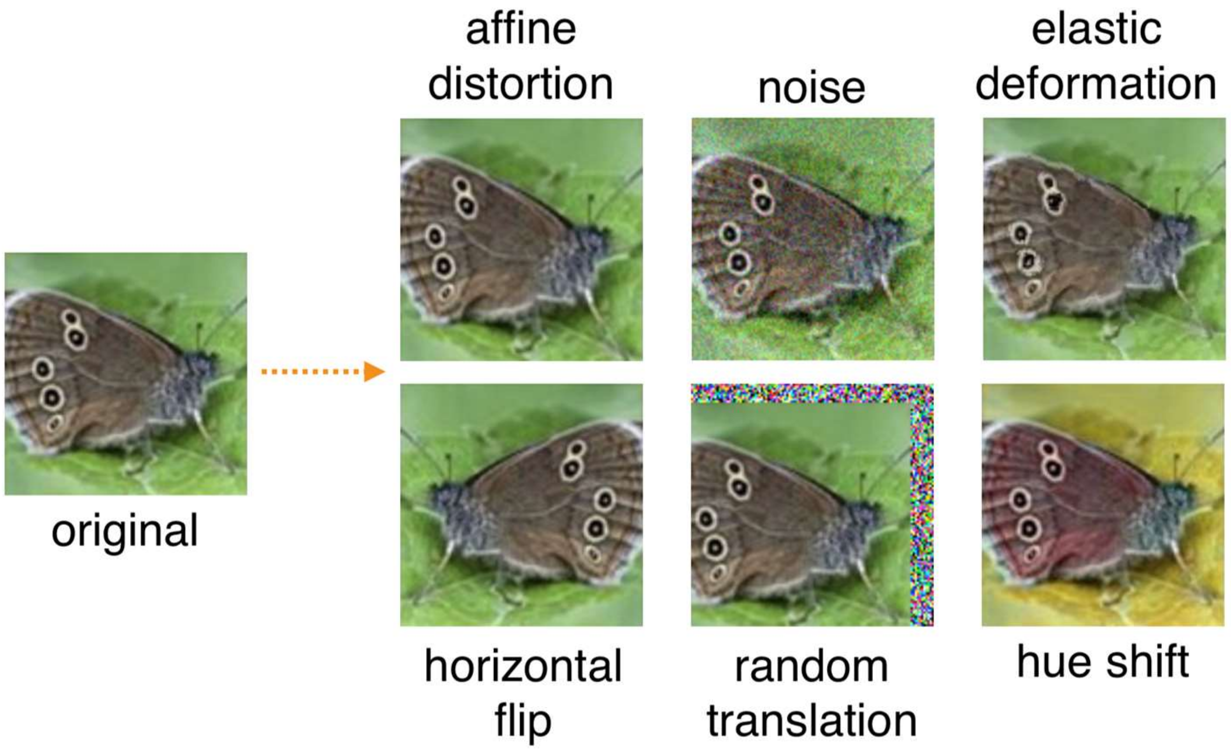
Measure the error contribution from each connection

Gradient descent

Parameter update

Dataset Augmentation

- Apply realistic transformations to data to create new synthetic samples, with same label



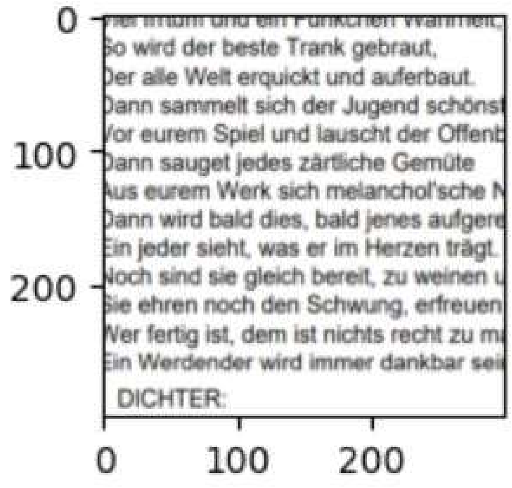
Overview of the notebook

Tutorial V (1)

1) Load necessary libraries (common libraries and personal modules)
2) Transfer Learning

- Load the inception model (base_model)
- Build new model using the base_model (model)
  - Define a head function (in_features, n_classes=2) : use of sigmoid
- Optimization : define the loss function (criterion) and the optimizer
- Helper functions to get the prediction (get_predictions) and to compute the batch accuracy (calculate_accuracy_batch)

Tutorial V (2)



3) Dataset

- Images :
  - Get them from ML3 folder
  - Preprocessing using transforms.Compose (resize, tensor, normalize)
  - Shape [3,299,299]
  - Transform to numpy array for display (im_numpy)
- Labels

- Split dataset into train/test samples
  - Torch.utils.data.random_split
  - Train_test_split with stratify enabled from scikit-learn
- Create data loaders for training/val (beware shuffle param)
- Example : batch of 10 images, plot them, print logits and output classes (res)

Tutorial V (3)

4) Training

- Train_model function
  - Contains the loop on epochs
  - Calls the train and validate functions (beware the params)
  - Save history (loss, accuracy) and model for a given epoch

  - Train function : reset gradients, compute logits and loss, compute gradients (backward prop), update parameters, calculate accuracy of the batch (helper functions), returns train loss and train accuracy

  - Validate function: structure BUT (no optimizer, no backward, no update of the params), returns test loss and test accuracy

- Plot_history function
- Run it ! history = train_model(...) using 70 epochs

Tutorial V (4)

5) Load trained variables from checkpoint

- Choose epochs values
- Load corresponding models
- Call validate function to get the validation loss and validation accuracy (see how it evolves)

6) Save final model for inference

7) Inference :

- Load the model, eval() mode
- Get an image, preprocess (convert to tensor, add batch dimension
- Get the logits and associated class

Tutorial V (4)

8) Improve the results : data augmentation

- Load images from ML3 folder
- Preprocess (resize, Randomcrop, tensor, normalize)
- Convert to Numpy for plotting purposes (im_numpy)
- The rest of the code is similar to previous code

9) Exercise (groups of 4) (35 min), 2min presentation

Tutorial VI : Transfer Learning

Introduction

- Goal : use Recurrent Neural Networks to predict and generate text sequence

- Program : inverted classroom style
  - Theory
  - Overview to get the big picture of the notebook
  - Work alone
  - Work in groups of 4
- Technical : Google Colab, Pytorch

Theory

Overview of the notebook

Tutorial VI (1)

1) Load necessary libraries (common libraries and personal modules)

2) Text data

- Read the data (rnn.txt), print the first 100 words

3) Build the dataset

- 2 dictionaries : word → id (dictionary) and id → word (reverse_dictionary)
  - Build_dictionaries function
- Vocabulary size = 493 (0=most common word)
- Helper functions to get sequences of int or words (text_to_ints, ints_to_text)
- Print example : first 100 words (or int), length of input data=2118 (words_as_int)

Tutorial VI (2)

3) Data streaming

- Create dataset using WordDataSet class to create blocks of text
  - Block length = n_input+1 = 3+1 = 4
- Create DataLoader with batch size=50, preprocess data (separate input and target sequences, stack data and convert Numpy→tensors, put them to GPU)
  - Length of dataset = Number of blocks in sequence = Total length/Block length = 2118/4=529
- Example : print the 50 samples that are in the 1st batch

Tutorial VI (3)

4) Construct model

- Create class RNN
  - Embedding layer (vocab_size, embedding_dim=128)
  - Loop to add the 3 LSTM layers
  - FC layer with vocab_size
- Define sequence of 3 words (n_input), define dimension of 3 LSTM, call the RNN class
- Investigate the model using Tensorboard
  - Create an input of size=5 (x), transform to tensor, add batch
  - SummaryWriter to save the model and be able to open it with tensorboard
  - Print the output size of y : [5,1,493]
- Test the NOT trained model and see that it is bad
  - Get the first batch (break), apply the model, get the predictions, compare to true

Tutorial VI (4)

5) Train the model

- Params : n_input = 3, batch_size=50 , one LSTM layer (128),n_epochs=200
- Create the data loader (preprocess data)
- Optimization part : criterion, optimizer (RMSprop, why?)
- Training loop on epochs, then on batches
  - Initialize gradients
  - Get the output (seq_len, batch_size, vocab_size), reshape it to (seq_len*batch_size, vocab_size)
  - Reshape labels (seq_len, batch_size) to (seq_len*batch_size)
  - Compute the loss between output and true labels
  - Backward prop, param update
  - Compute loss and accuracy
- Plot loss and accuracy

Tutorial VI (5)

6) Generate text with RNN

- Function to generate text (gen_long)
  - Input parameters : model, input sequence, number of words to generate (128)
  - No_grad() because we are in an evaluation mode
  - Loop on number of words, convert to tensor, predict, …)

Back-up slides

- https://www.tensorflow.org/api_docs/python/tf/keras/applications