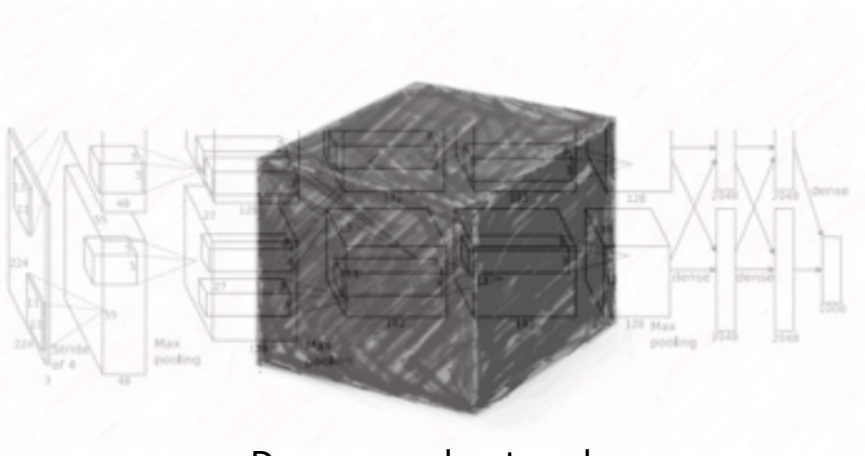
Last lesson

recap, final thought and final project



Dog

Input

Deep neural network

Output

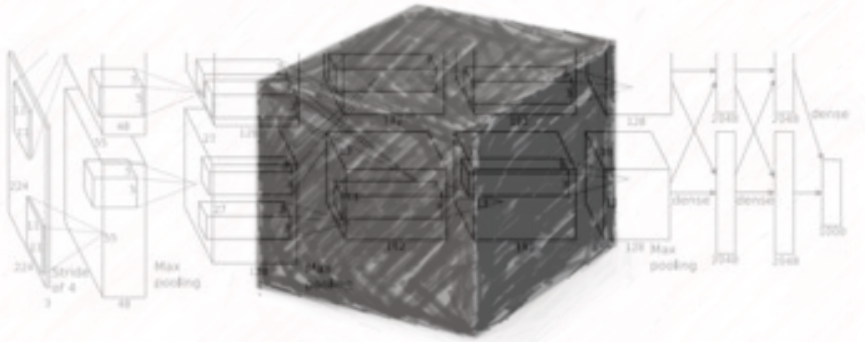
$$x \longrightarrow f(x; W) \longrightarrow \hat{y}$$

Optimization problem:

$$\min_{W} \left[\sum_{i} loss(\hat{y}_i, y_i) \right]$$

Solved using gradient descent:

$$W_{t+1} = W_t - \alpha \cdot \frac{\partial \sum_i loss(\hat{y}_i, y_i)}{\partial W_t}$$



Dog

Input

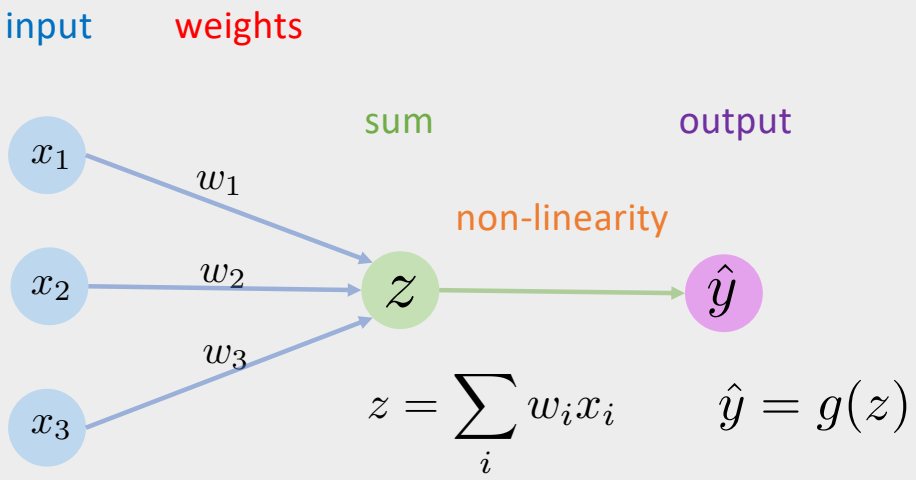
Deep neural network

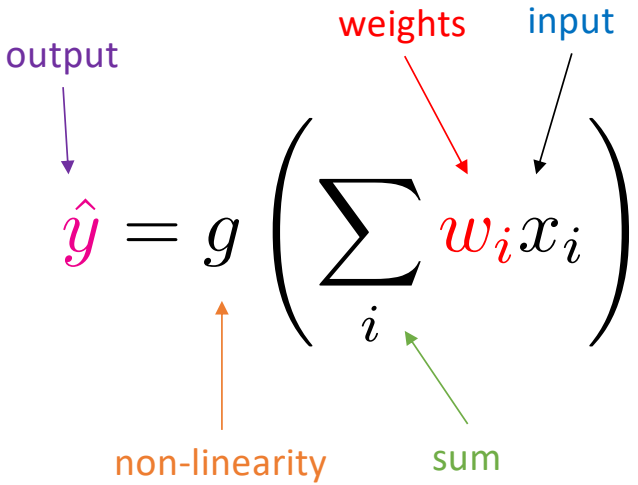
Output

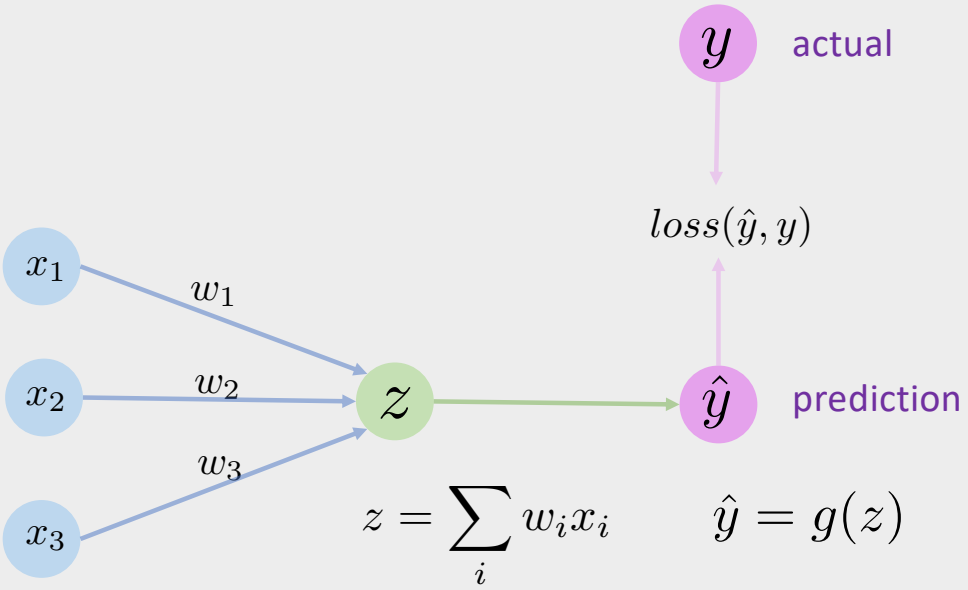
desired output

$$x \longrightarrow f(x; W) \longrightarrow \hat{y} \longleftarrow \text{loss} \longrightarrow y$$

Reminder: **The Perceptron**







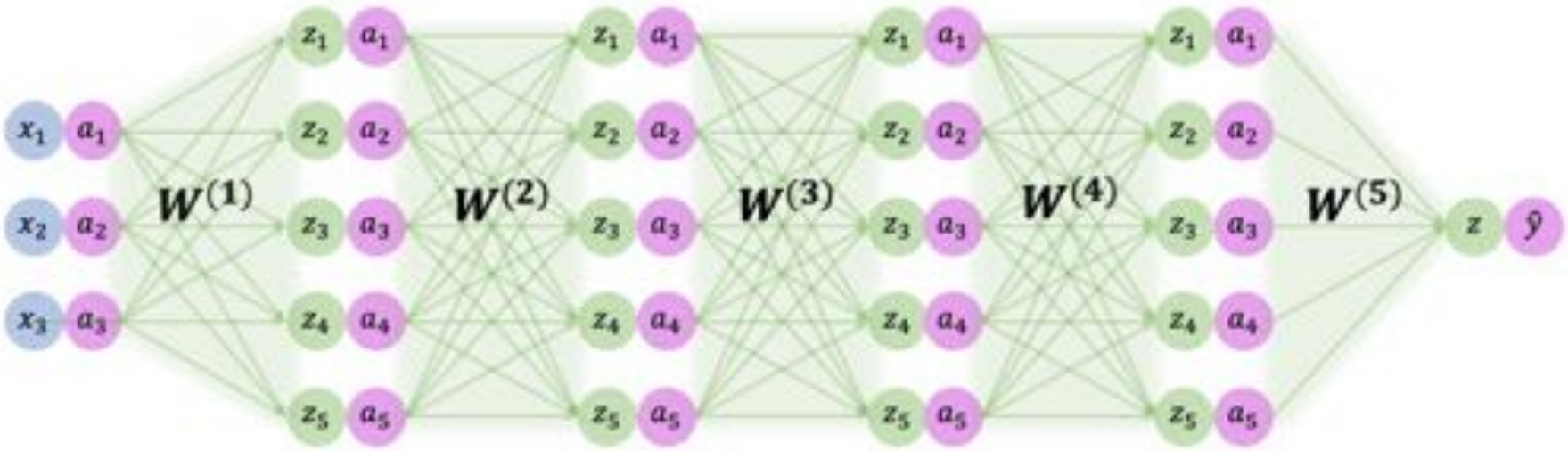
$$g(z_i) = \frac{1}{1 + e^{-z_i}}$$

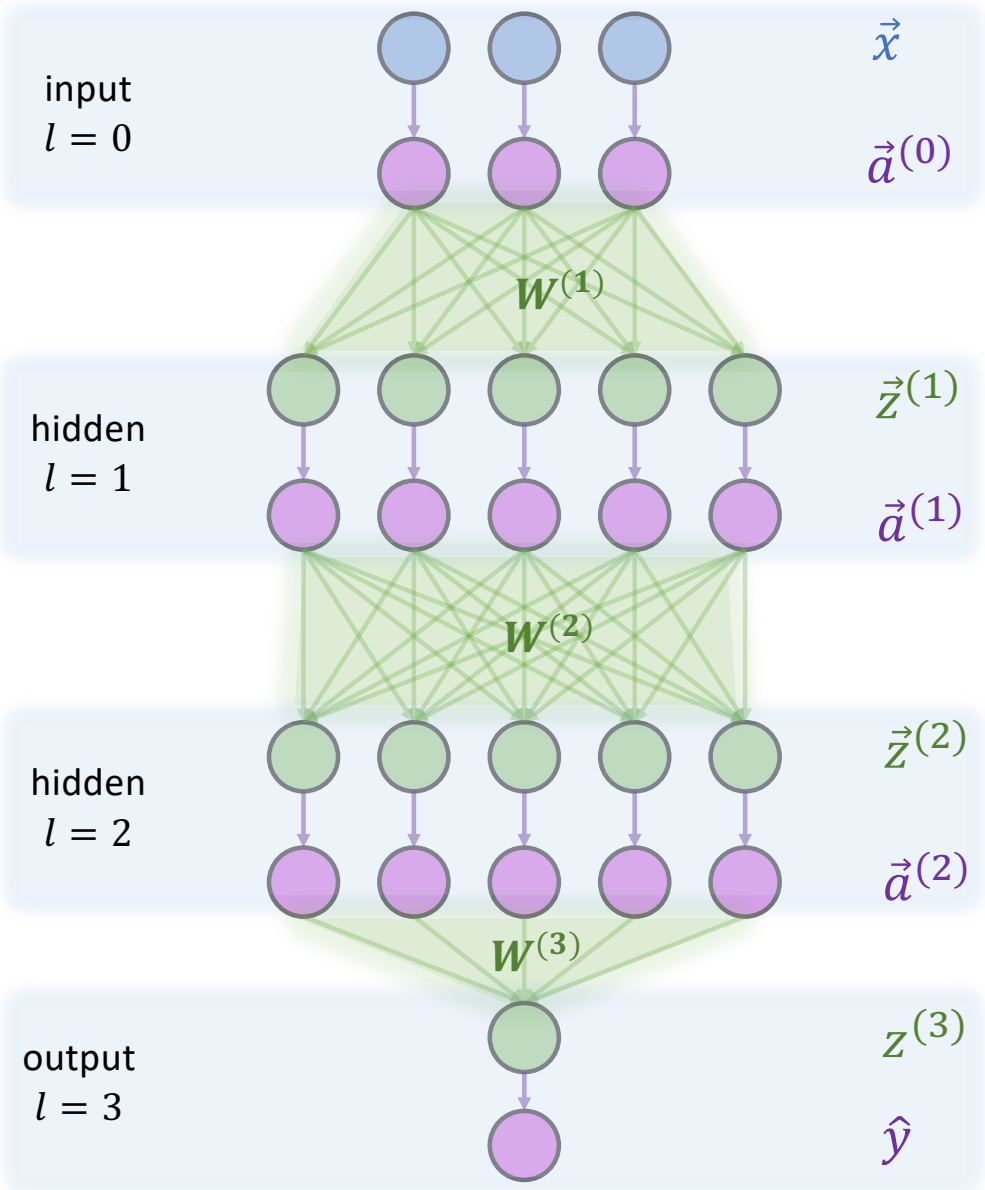
$$l(y, \hat{y}) = \begin{cases} -\log(\hat{y}) & y = 1 \\ -\log(1 - \hat{y}) & y = 0 \end{cases}$$

$$J(\vec{w}) = \sum_i l(y_i, \hat{y}_i)$$

$$\min_w \left[J(\vec{w}) \right] \xrightarrow{\text{Gradient Descent}} \vec{w} := \vec{w} - \alpha \frac{\partial J}{\partial \vec{w}}$$

Multilayer Perceptron
(fully connected / dense layers)





Forward Propagation

$$\vec{a}^{(0)} = \vec{x}$$

$$\vec{z}^{(1)} = \vec{a}^{(0)} \cdot W^{(1)}$$

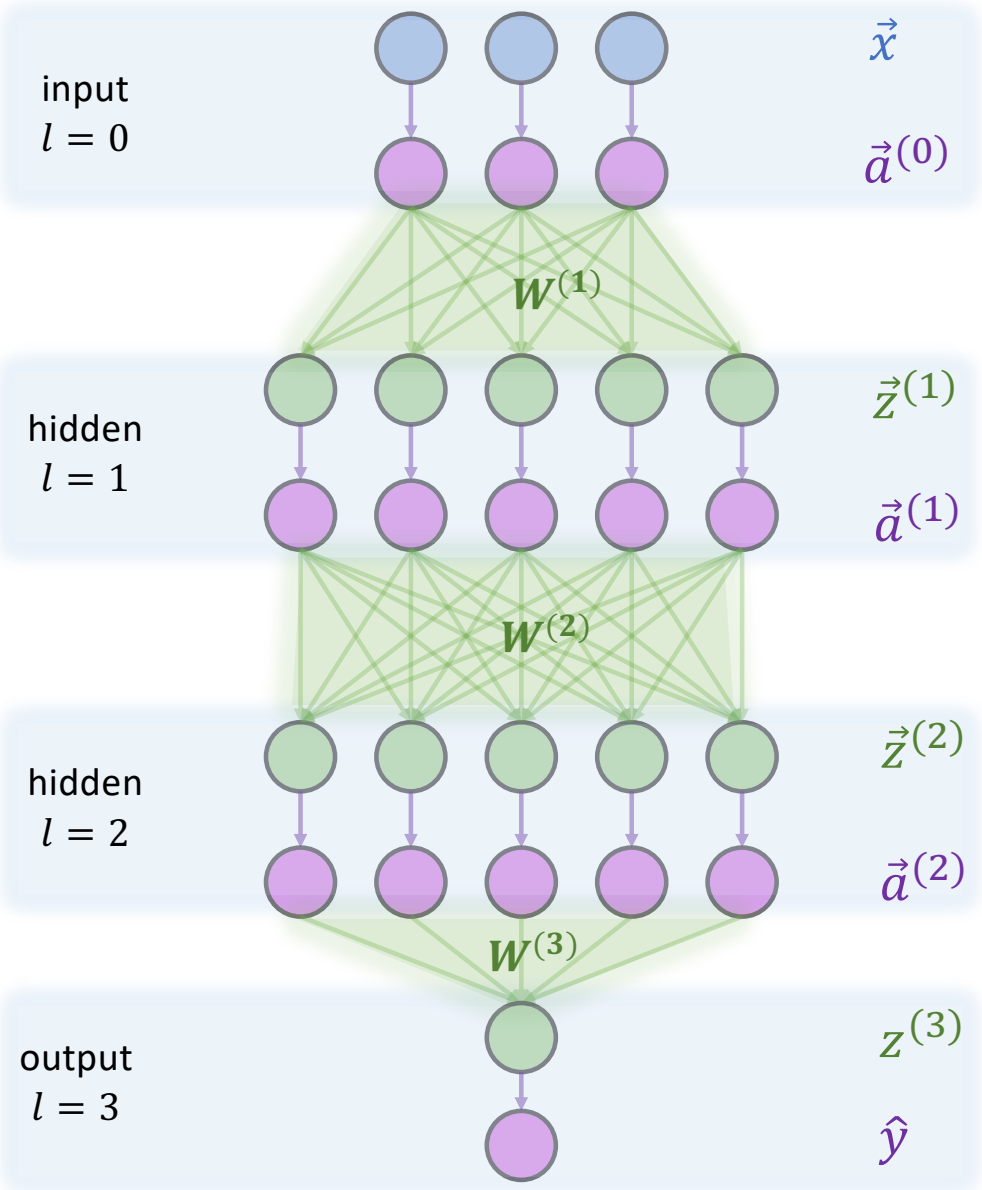
$$a_i^{(1)} = g\left(z_i^{(1)}\right)$$

$$\vec{z}^{(2)} = \vec{a}^{(1)} \cdot W^{(2)}$$

$$a_i^{(2)} = g\left(z_i^{(2)}\right)$$

$$z^{(3)} = \vec{a}^{(2)} \cdot W^{(2)}$$

$$\hat{y} = g\left(z^{(3)}\right)$$



Back Propagation

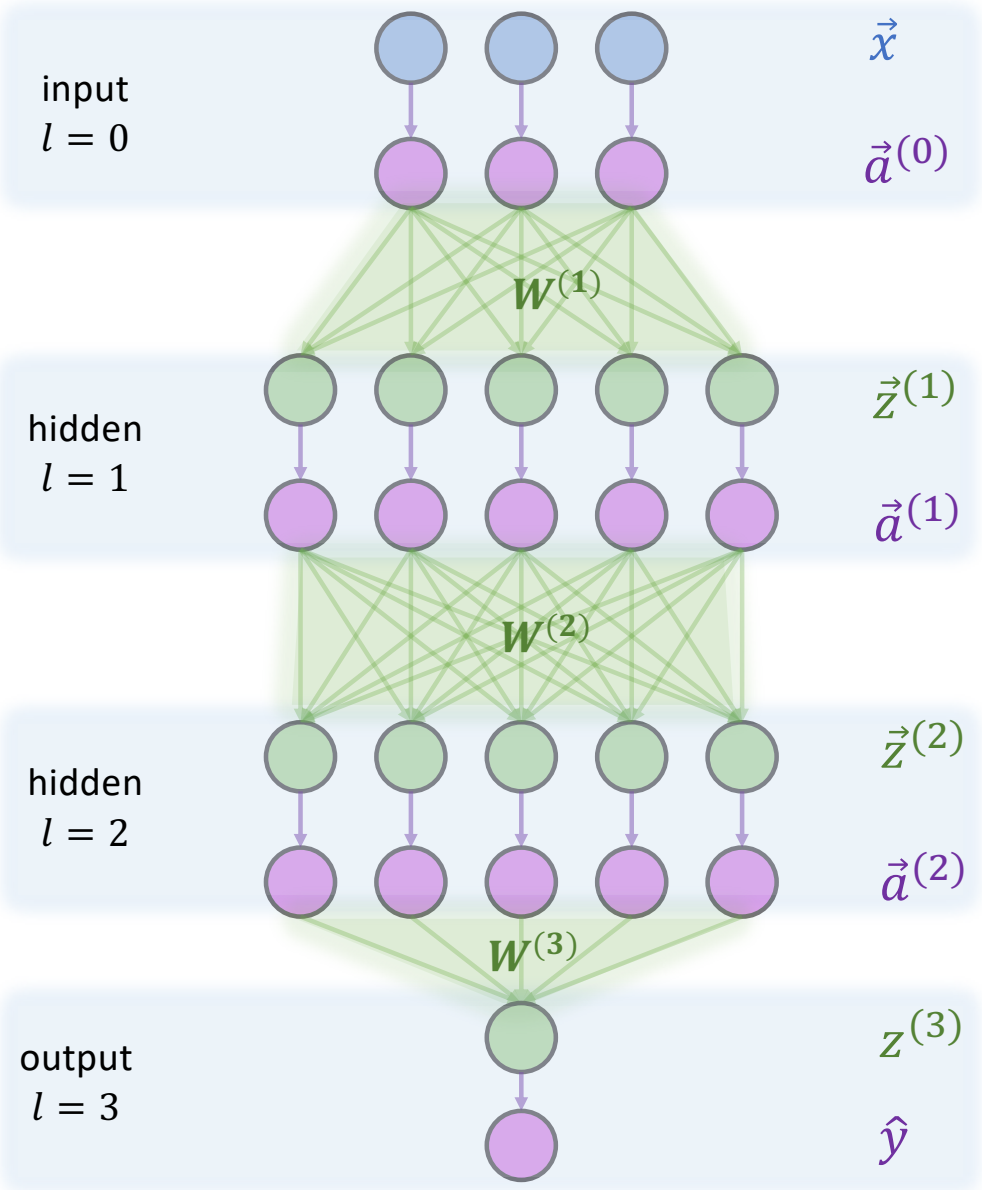
input: y, $\hat{y}$, $\{W^{(l)}\}$, $\{\vec{z}^{(l)}\}$

$\vec{\delta}^{(l_{max})} = \hat{y} - y$

for $l = l_{max} - 1$ to 1:

$$\vec{\delta}^{(l-1)} = \vec{\delta}^{(l)} \cdot W^{T(l)} \odot g'\left(\vec{z}^{(l-1)}\right)$$

return: $\{\vec{\delta}^{(l)}\}$



Gradient Descent (1-step)

input: $\{W^{(l)}\}$, $\{\vec{a}^{(l)}\}$, $\{\vec{\delta}^{(l)}\}$

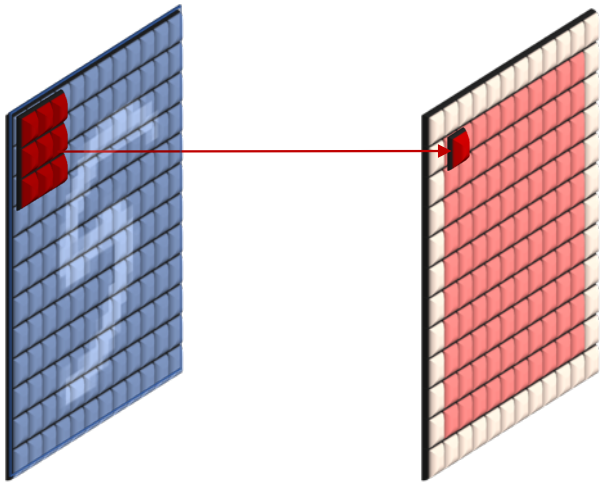
for $l = 1$ to l_{max}:

$$\frac{\partial J}{\partial W^{(l)}} = \vec{a}^{T(l-1)} \cdot \vec{\delta}^{(l)}$$

$$W^{(l)} = W^{(l)} - \alpha \cdot \frac{\partial J}{\partial W^{(l)}}$$

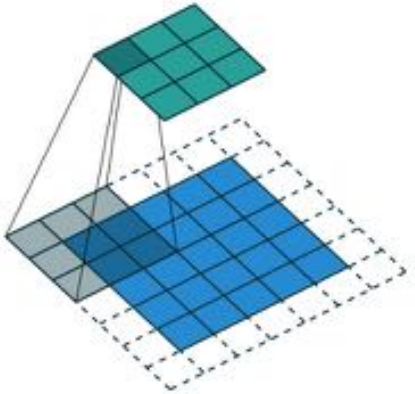
3x3 kernel

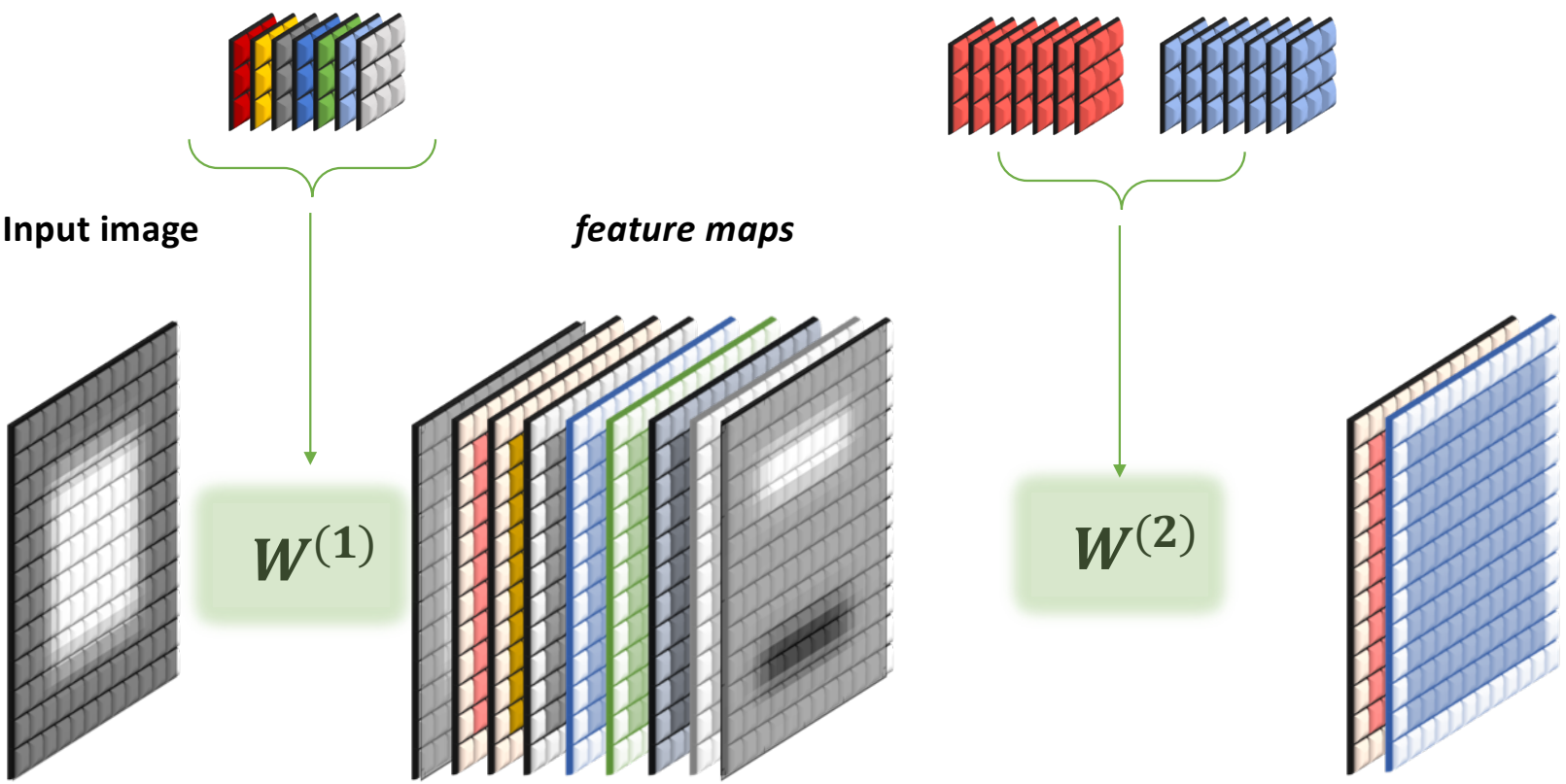
Let’s connect each neuron to only a local region (3x3) of the input volume:



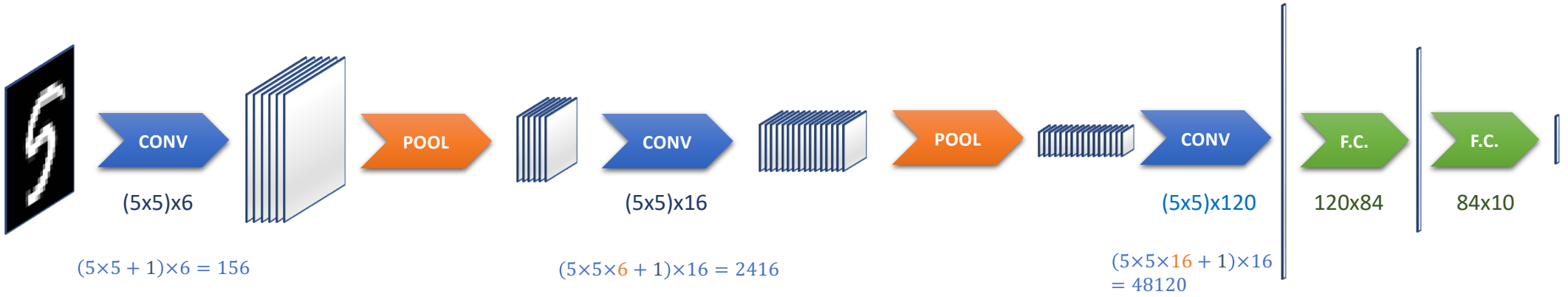
12 x 12

10 x 10




Input image
feature maps
$W^{(1)}$
$W^{(2)}$
Input layer
1st conv layer
2nd conv layer
(depth of 7)
(depth of 2)
12 x 12 x **1**
10 x 10 x **7**
8 x 8 x **2**

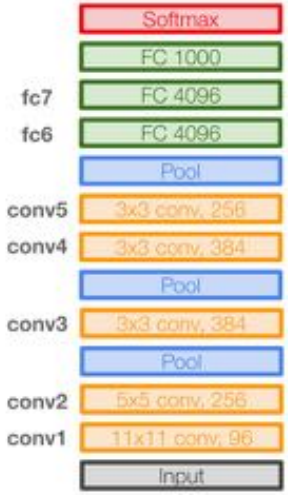
LeNet-5 (LeCun et al., 1998)




```
# LeNet-5 Model
model = Sequential()
model.add(Conv2D(6, kernel_size=(5, 5),
                 activation='relu',
                 input_shape=(32,32,1)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(16, kernel_size=(5, 5),
                 activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(120, kernel_size=(5, 5),
                 activation='relu'))
model.add(Flatten())
model.add(Dense(84, activation='relu'))
model.add(Dense(10, activation='softmax'))
model.summary()
```

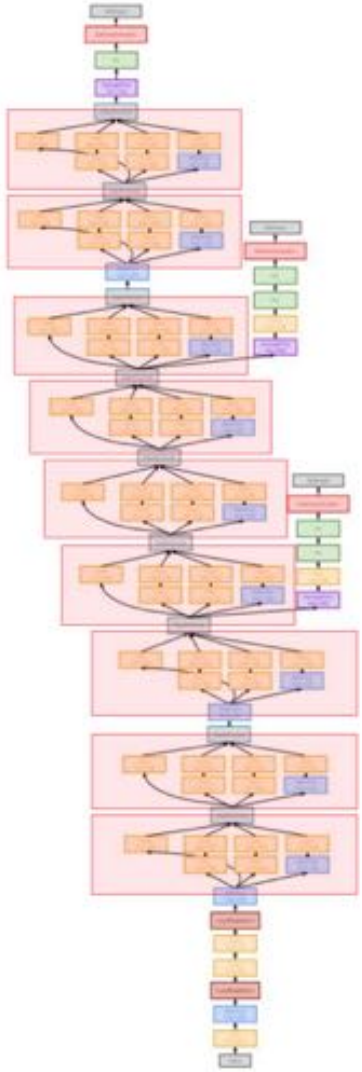
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_19 (Conv2D)           (None, 28, 28, 6)         156
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 14, 14, 6)         0
_________________________________________________________________
conv2d_20 (Conv2D)           (None, 10, 10, 16)        2416
_________________________________________________________________
max_pooling2d_12 (MaxPooling (None, 5, 5, 16)          0
_________________________________________________________________
conv2d_21 (Conv2D)           (None, 1, 1, 120)         48120
_________________________________________________________________
flatten_7 (Flatten)          (None, 120)               0
_________________________________________________________________
dense_13 (Dense)             (None, 84)                10164
_________________________________________________________________
dense_14 (Dense)             (None, 10)                850
=================================================================
Total params: 61,706
Trainable params: 61,706
Non-trainable params: 0
```

AlexNet

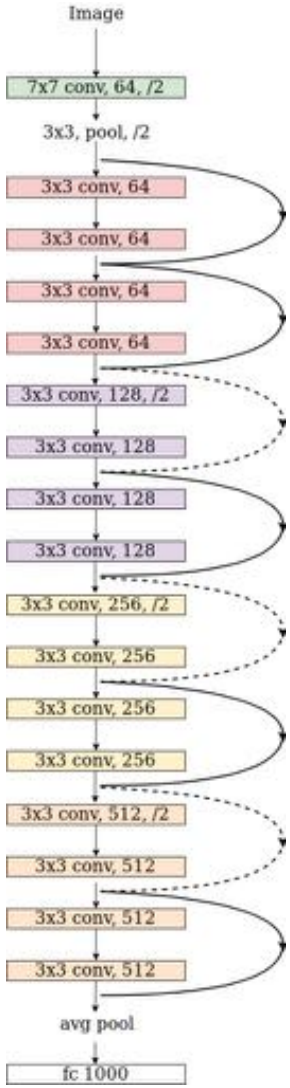


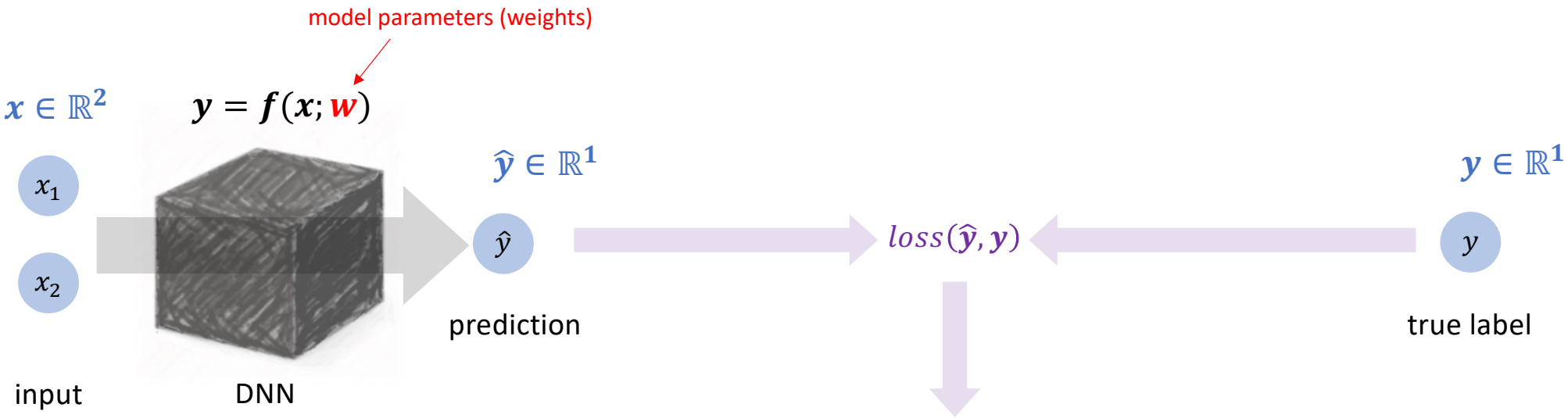
VGG

Inception



ResNet

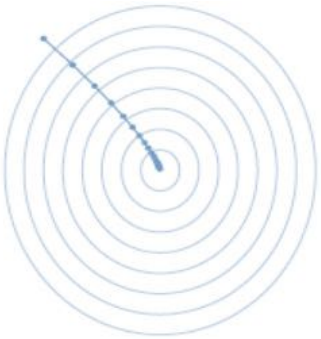




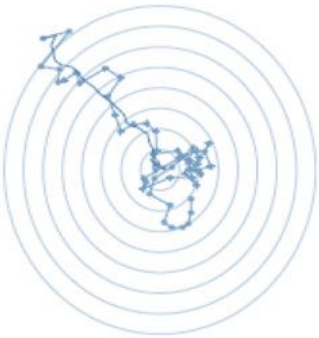
Optimization problem: $w^* = \arg\min_w \left[\sum_i loss(\hat{y}_i, y_i) \right]$

$$w_{t+1} = w_t - \alpha \cdot \frac{\partial J(w_t)}{\partial w_t}$$

Gradient descent algorithm

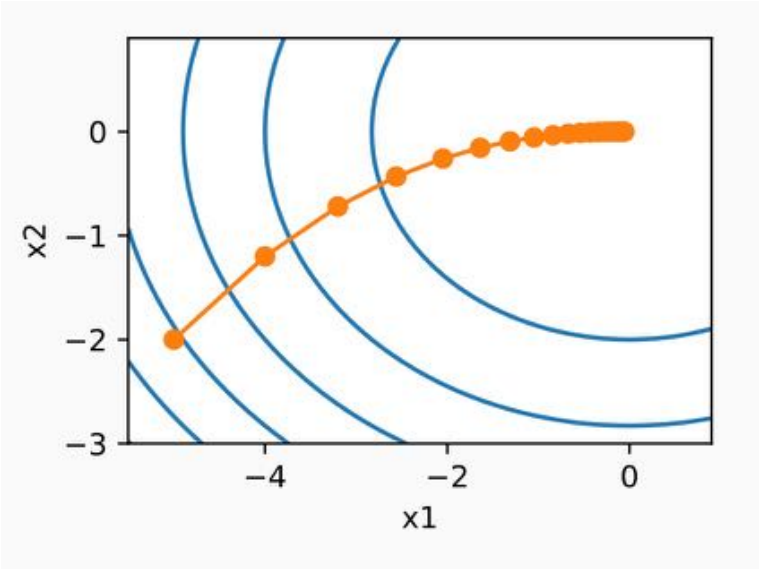


gradient-descent

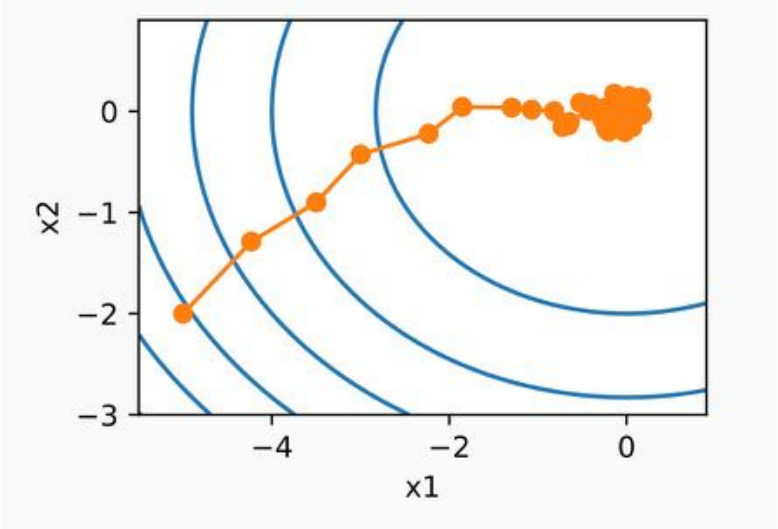


Stochastic gradient-descent

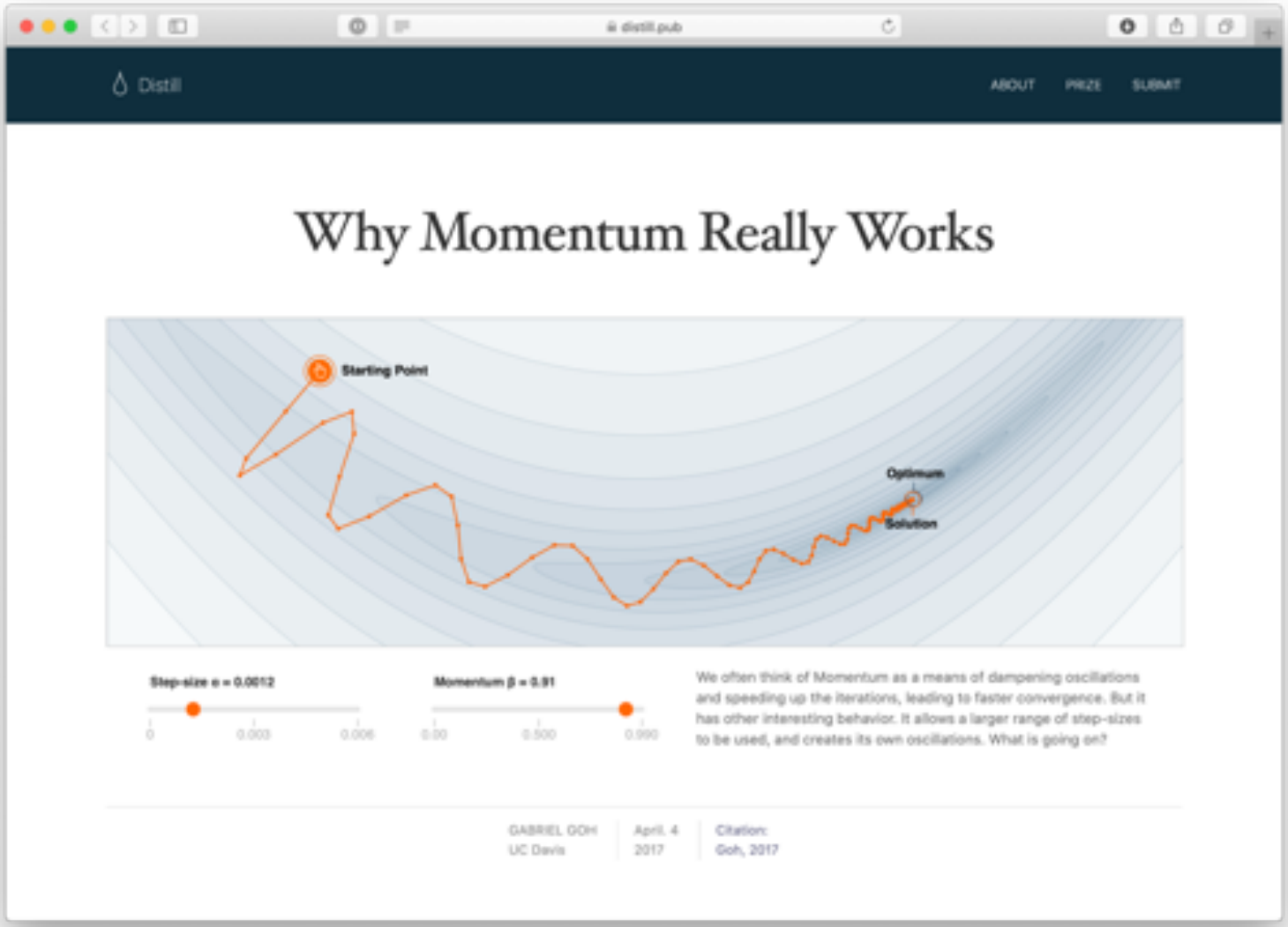
Reminder: 2-D Gradient Descent



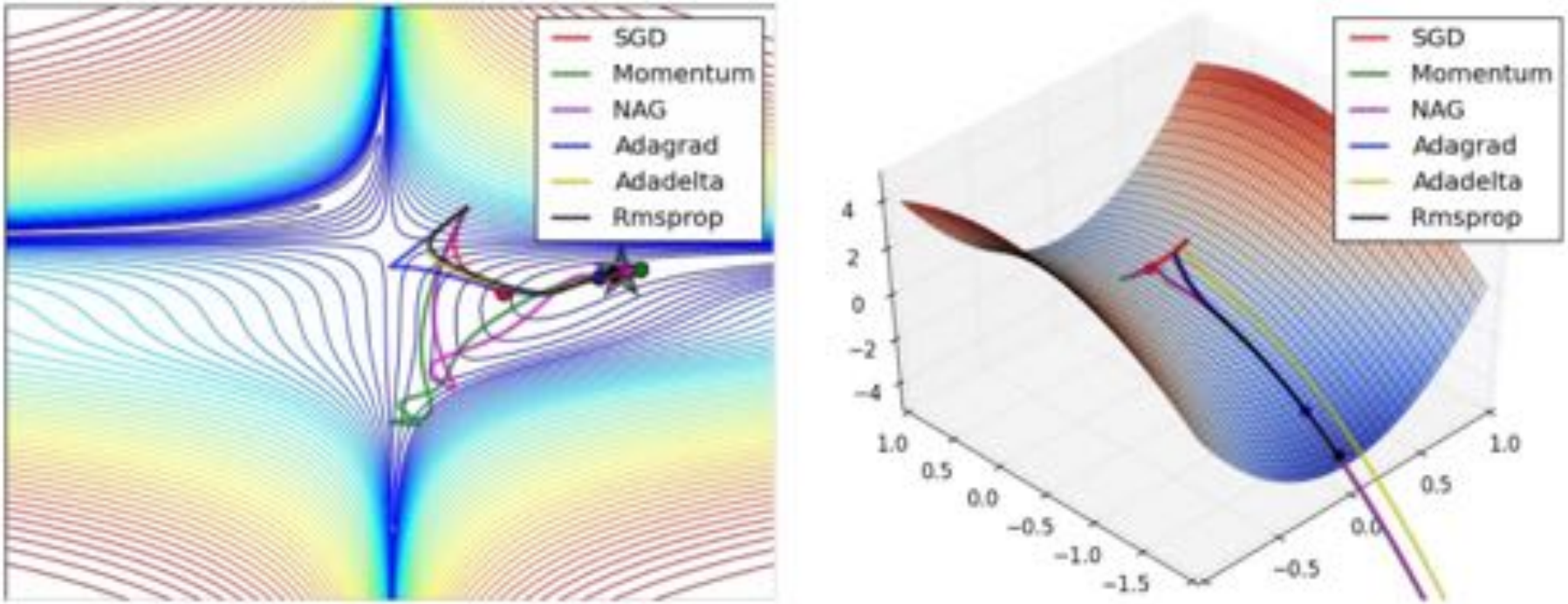
Gradient Descent



Stochastic Gradient Descent (SGD)



https://distill.pub/2017/momentum/



Animations that may help your intuitions about the learning process dynamics. **Left:** Contours of a loss surface and time evolution of different optimization algorithms. Notice the "overshooting" behavior of momentum-based methods, which make the optimization look like a ball rolling down the hill. **Right:** A visualization of a saddle point in the optimization landscape, where the curvature along different dimension has different signs (one dimension curves up and another down). Notice that SGD has a very hard time breaking symmetry and gets stuck on the top. Conversely, algorithms such as RMSprop will see very low gradients in the saddle direction. Due to the denominator term in the RMSprop update, this will increase the effective learning rate along this direction, helping RMSProp proceed. Images credit: Alec Radford.

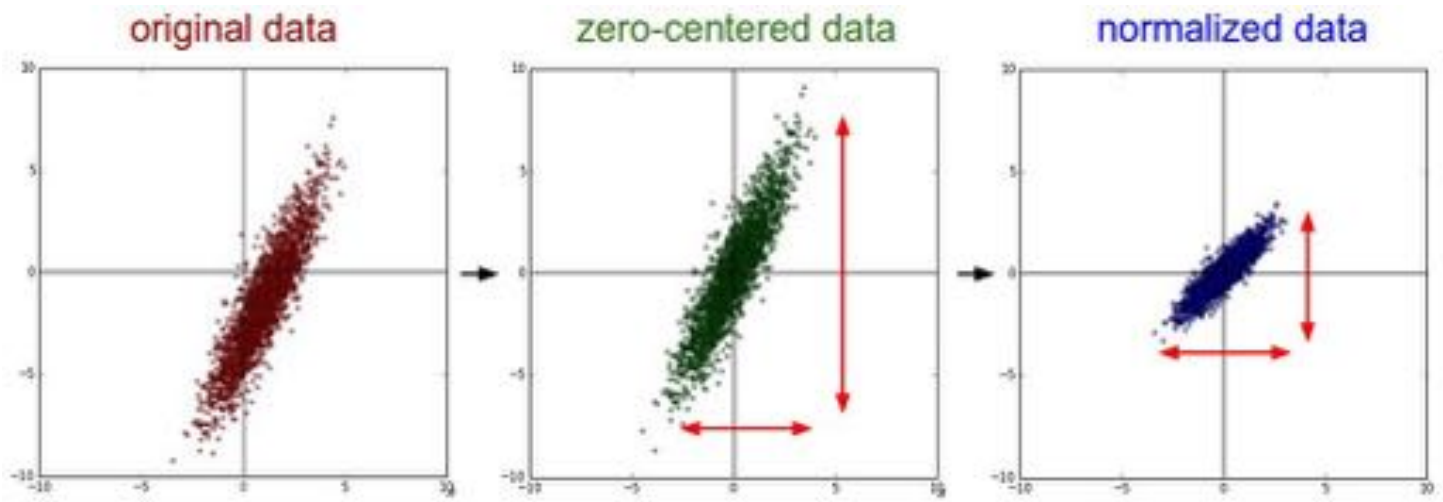
Data preprocessing

- **Mean subtraction** (most common): `X -= np.mean(X, axis = 0)`

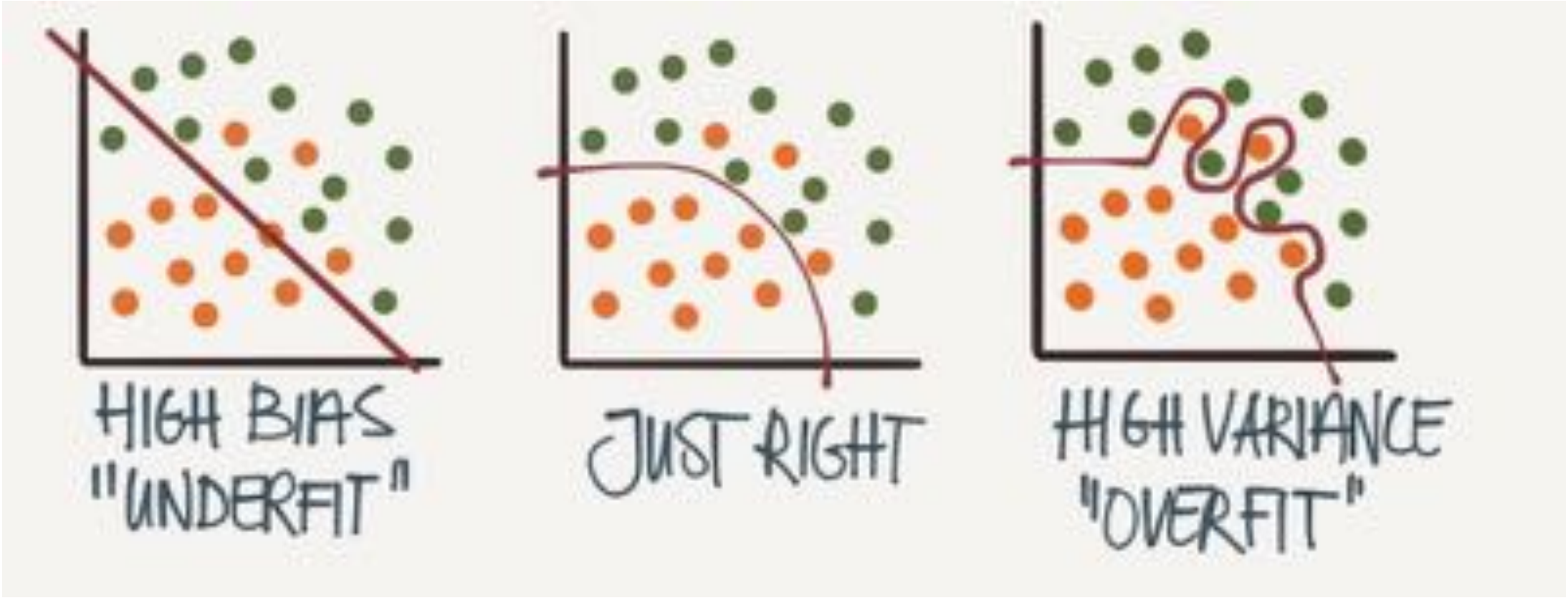
  (subtracting the mean across every individual feature in the data)

- **Normalization**: `X /= np.std(X, axis = 0)`

  (normalizing the data so that all features are approximately the same scale)



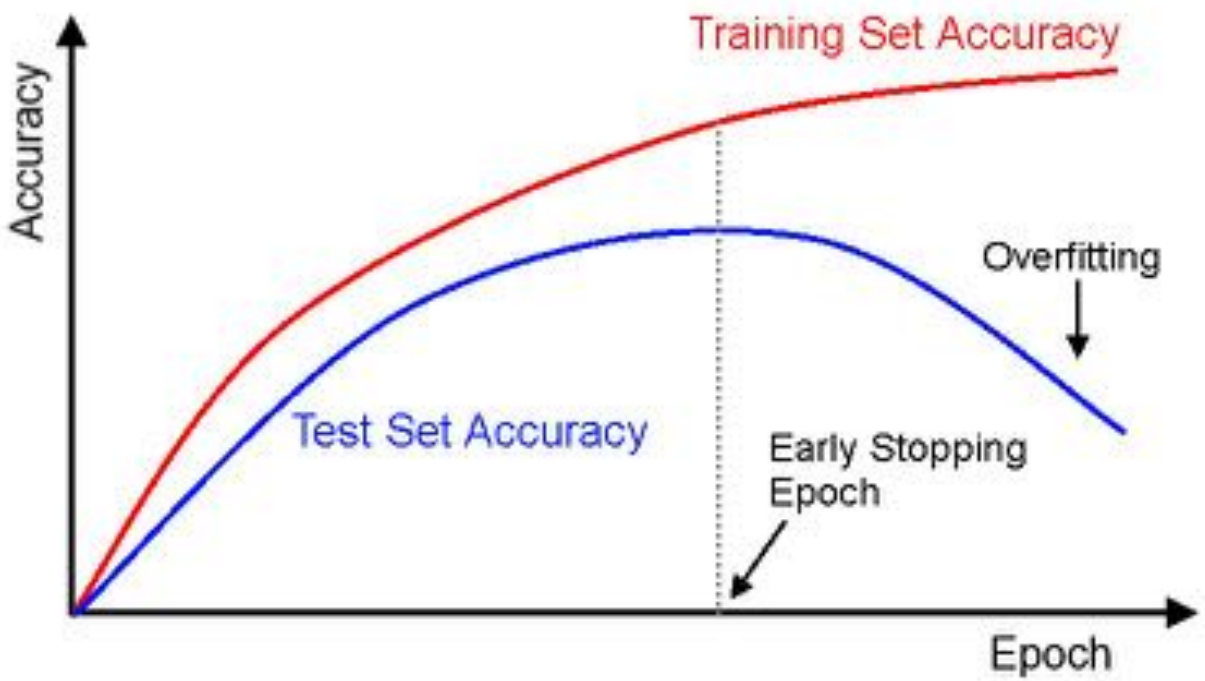
Over-fitting

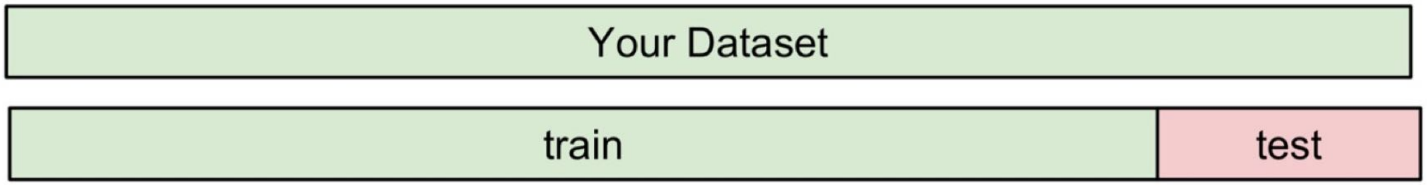


Even when we have far more examples than features, deep neural networks are capable of over- fitting.

Overfitting

- Early stopping
- Regularization
- Dropout





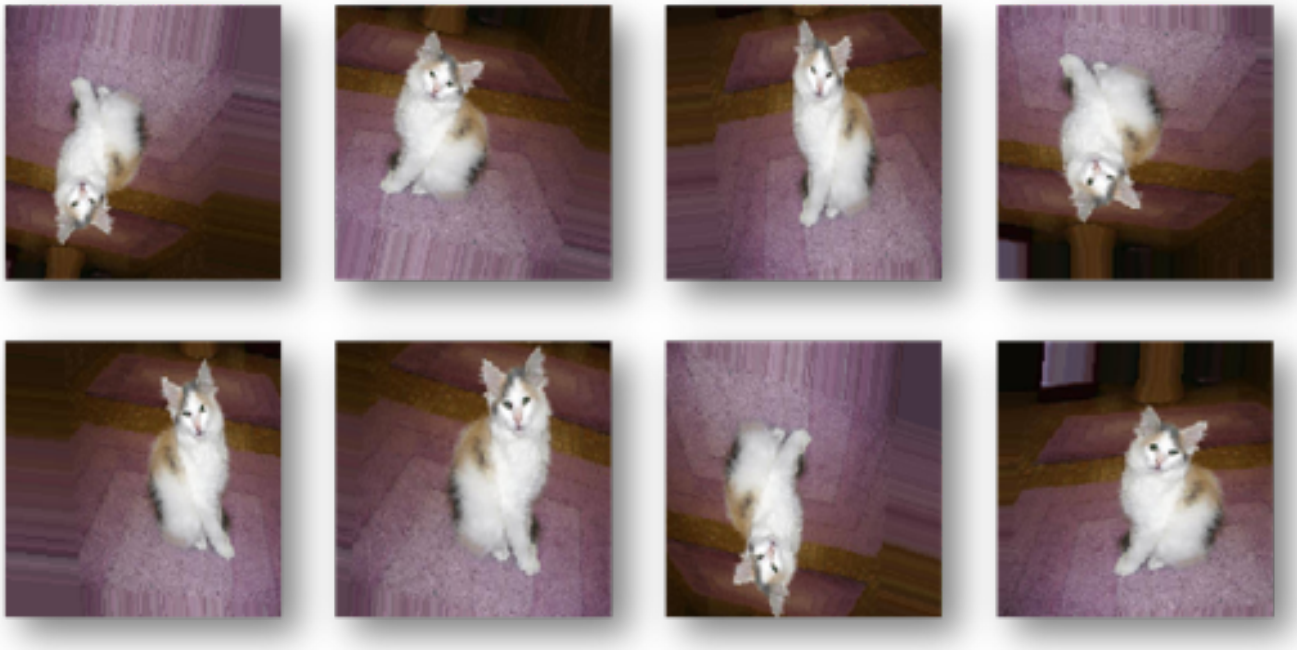
$$\text{Accuracy} = \frac{\text{\# correct predictions}}{\text{\# total predictions}}$$

Data Augmentation

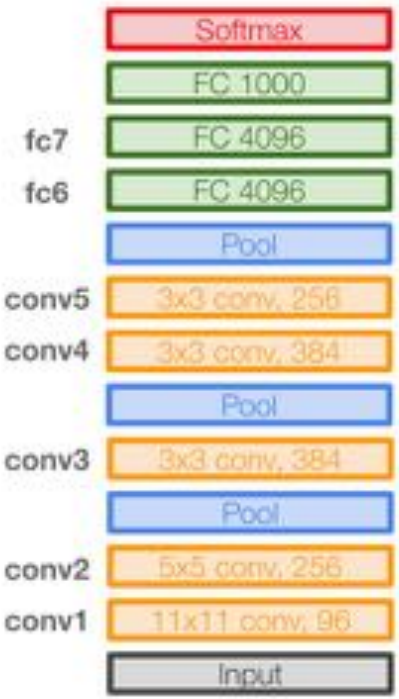
- Artificially creating more images from images you already have by changing the size, orientation etc. of the image

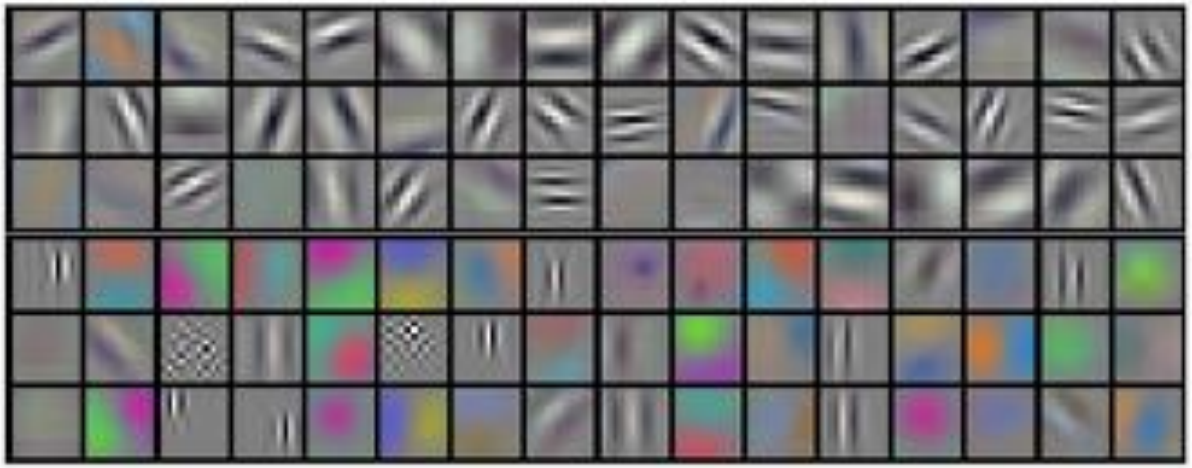
```
from keras.preprocessing.image import ImageDataGenerator

ImageDataGenerator(
        rotation_range=10.,
        width_shift_range=0.1,
        height_shift_range=0.1,
        shear_range=0.,
        zoom_range=.1.,
        horizontal_flip=True,
        vertical_flip=True)
```

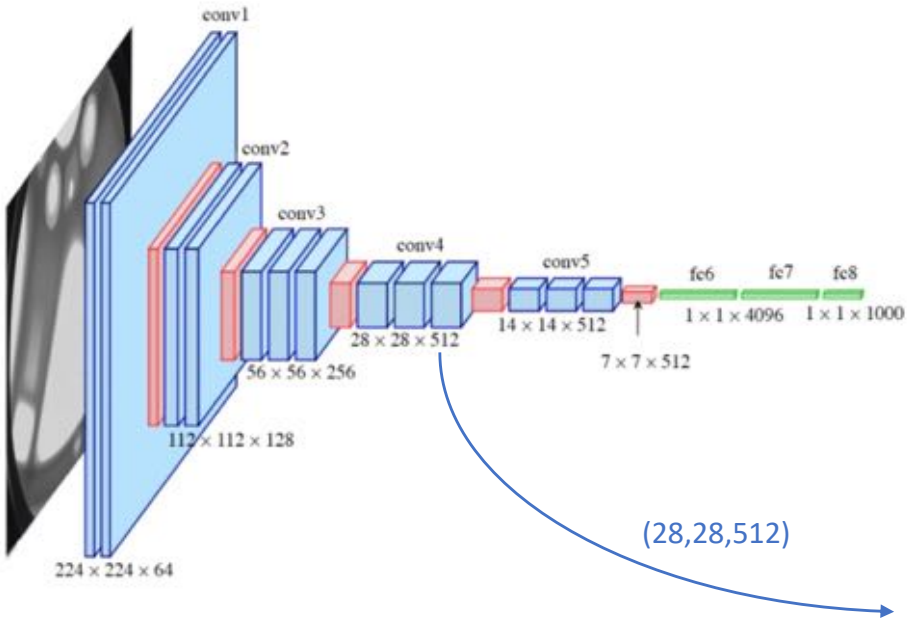


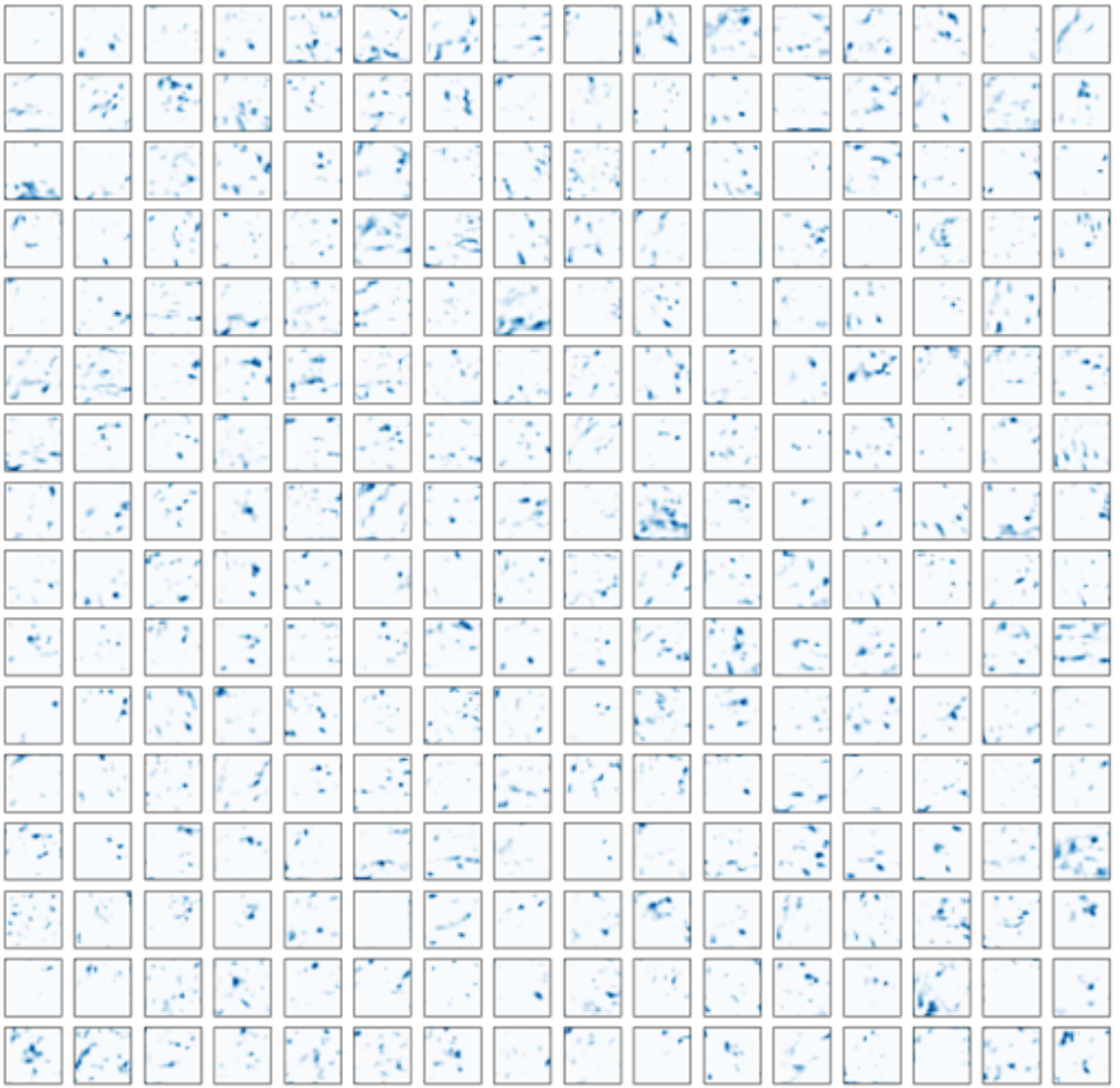
Features visualization





96 convolutional kernels of size 11×11×3 learned by the first convolutional layer on the 224×224×3 input images.

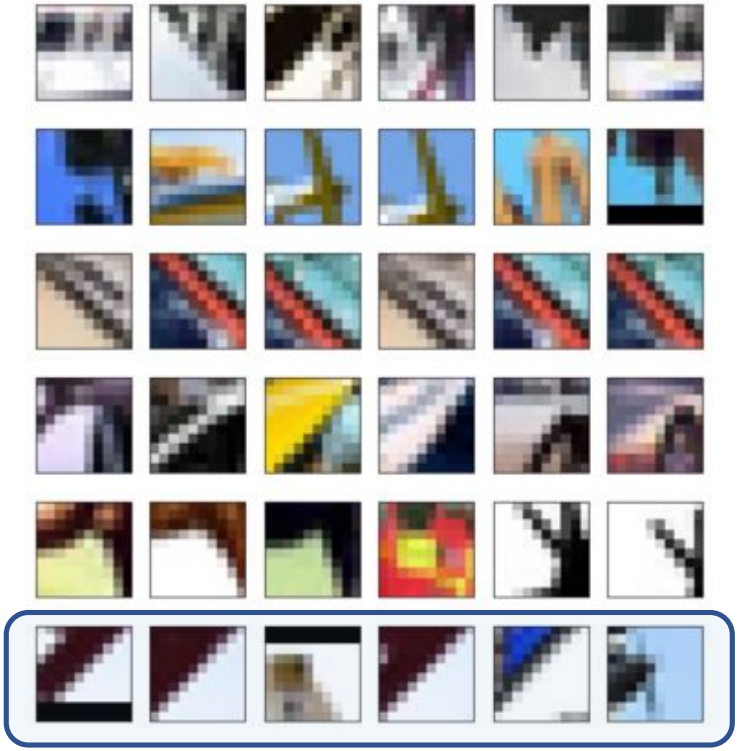




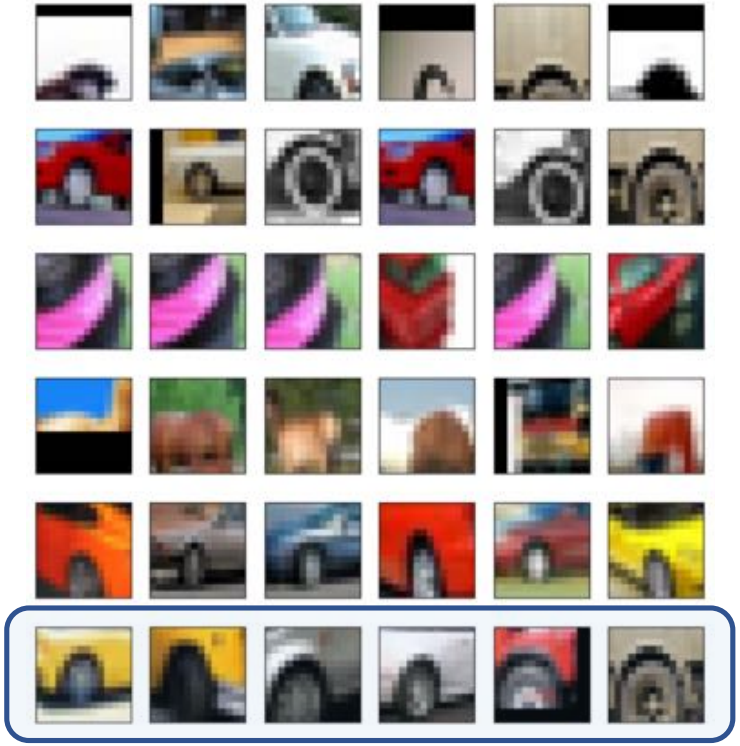
Activation maps

Maximally activating patches

a few patches that excite one specific filter in the 1st conv layer of the network



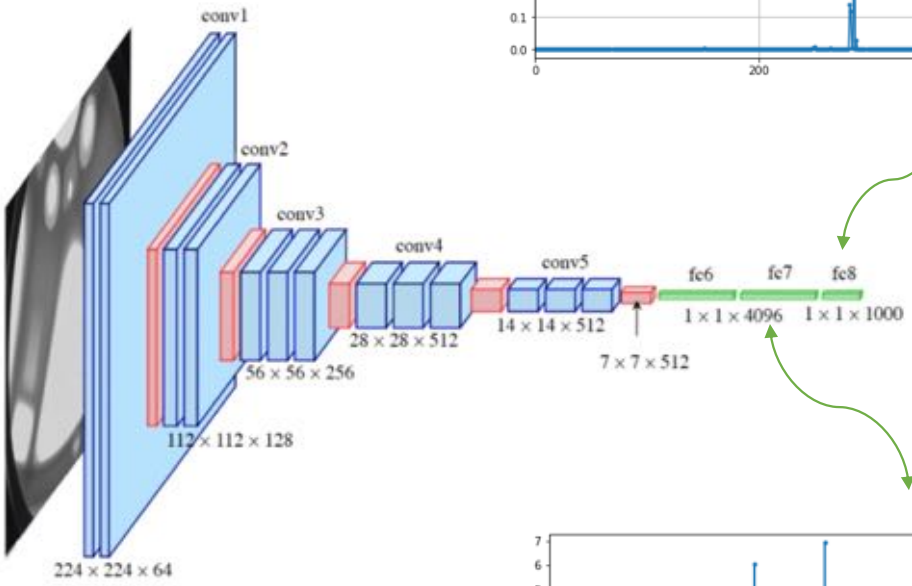
a few patches that excite one specific filter in the 2nd conv layer of the network



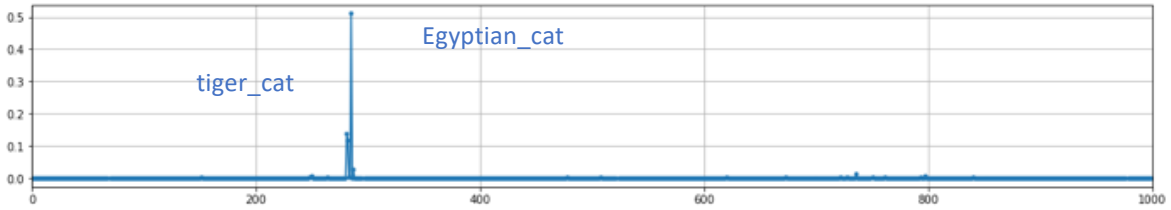
a few patches that excite one specific filter in the 3rd conv layer of the network

(224,224,3)

150528-D **pixels space**

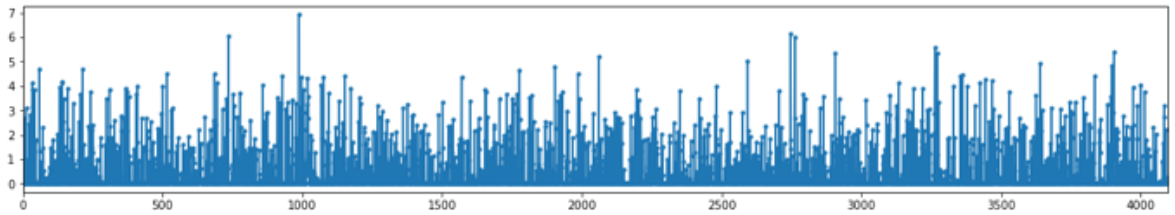


VGG-16

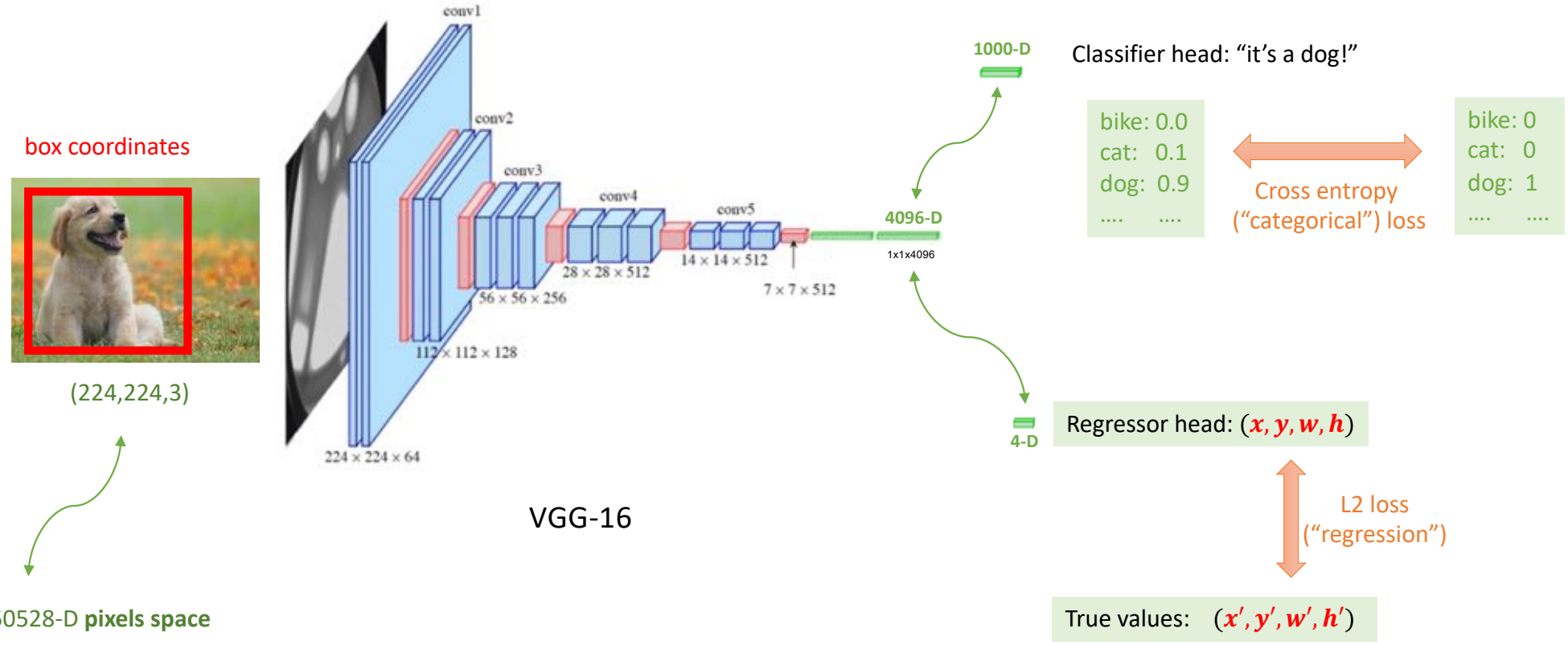


1000-D

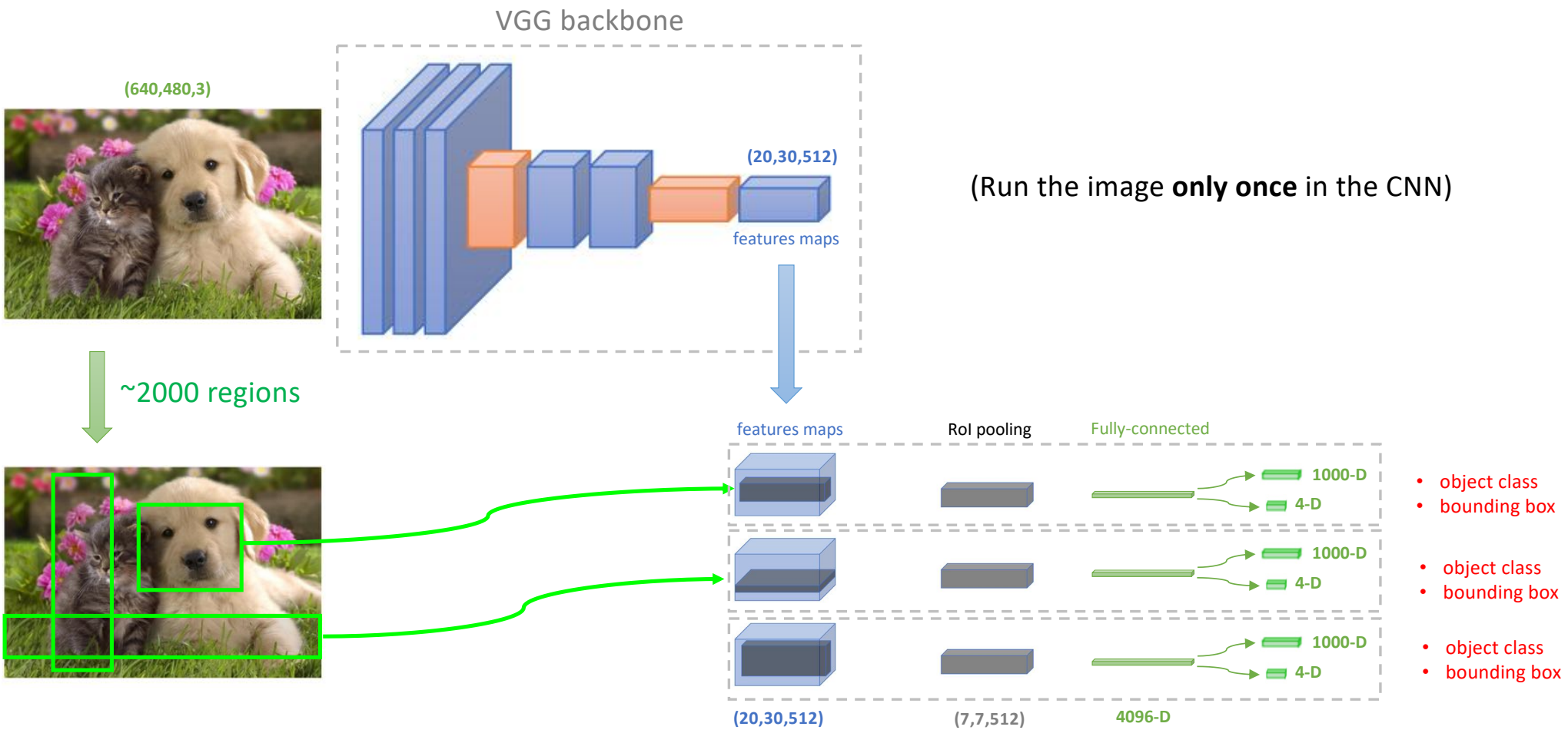
4096-D **features space**



Single object: Classification + Localization



Fast R-CNN

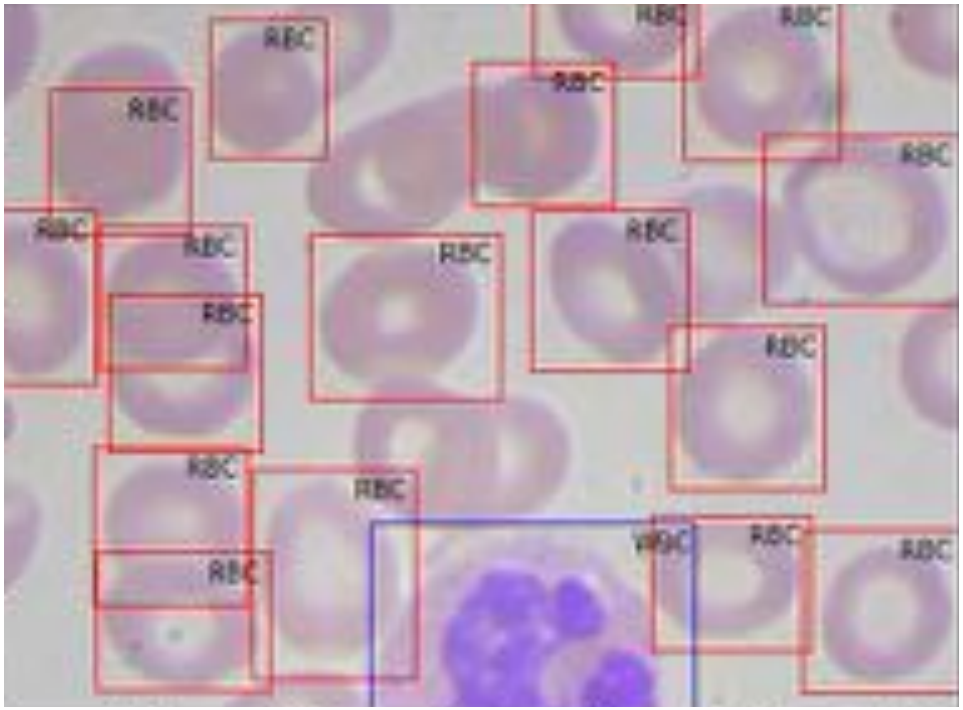


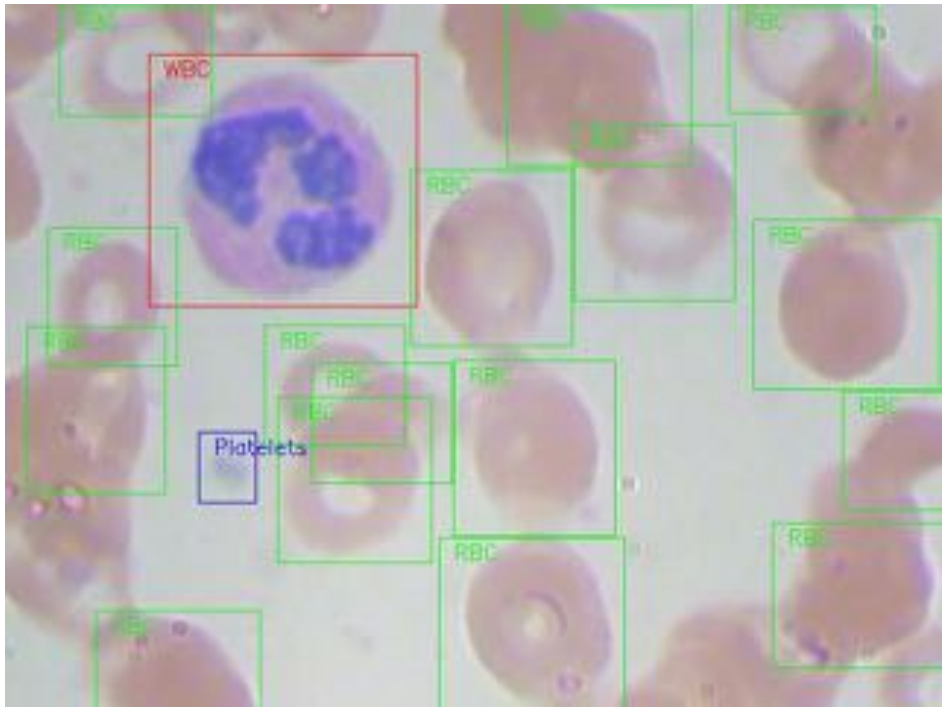
(Run the image **only once** in the CNN)

- object class
- bounding box

- object class
- bounding box

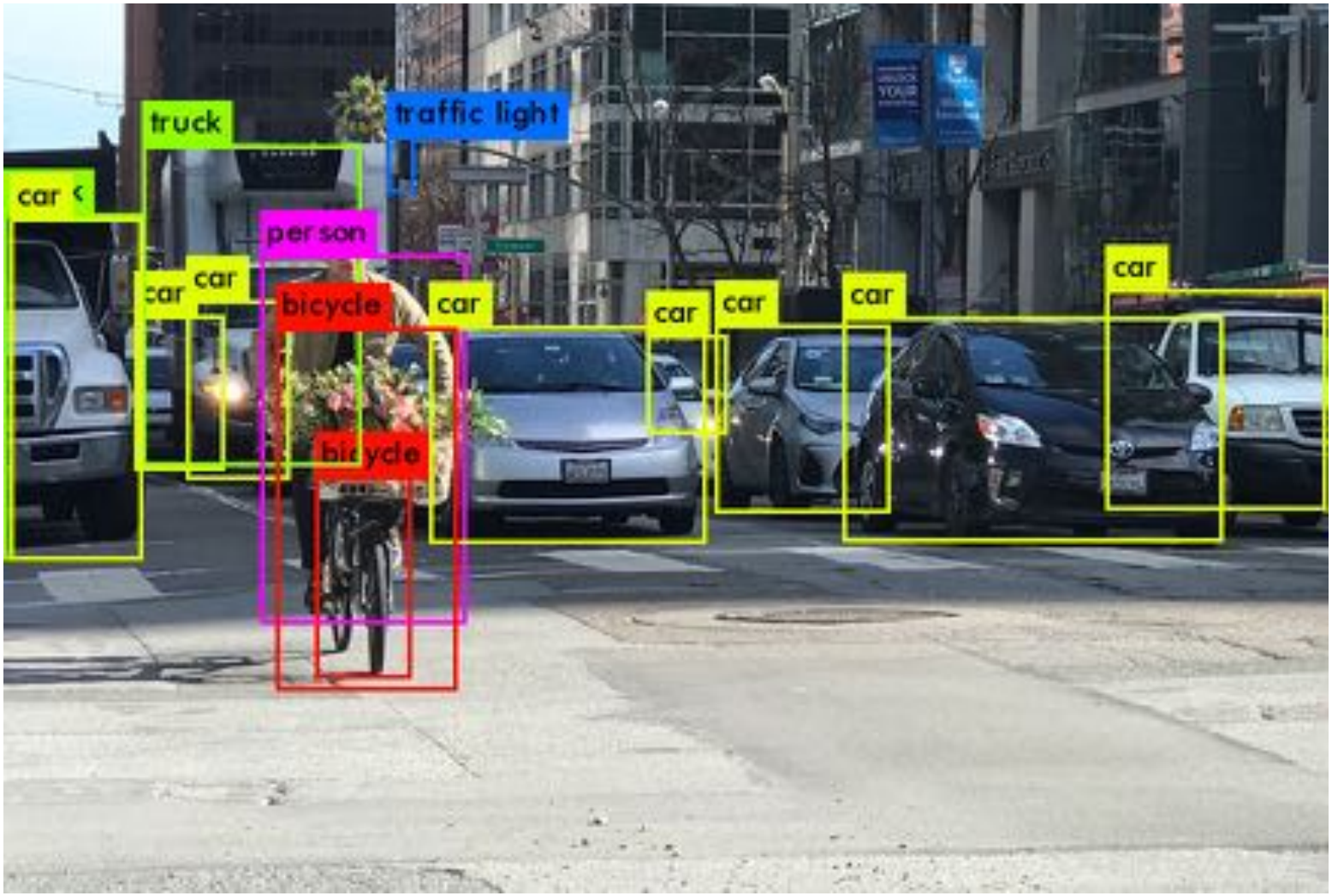
- object class
- bounding box



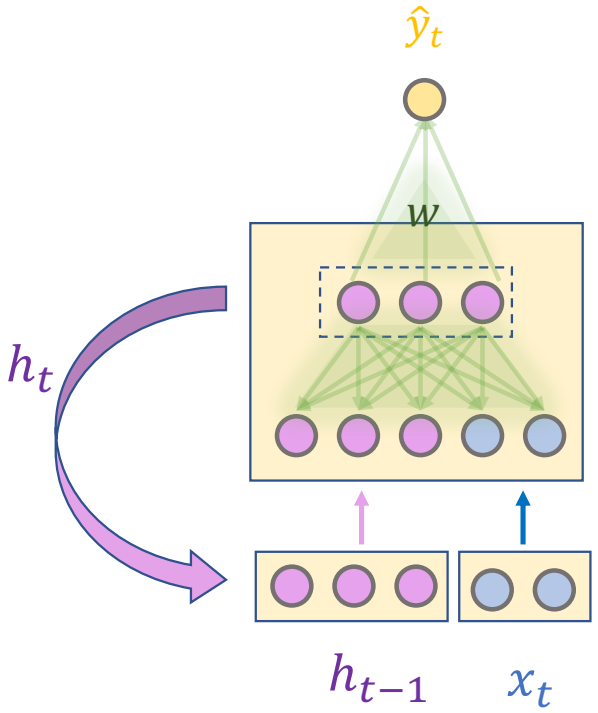


https://www.analyticsvidhya.com/blog/2018/11/implementation-faster-r-cnn-python-object-detection/

YOLO

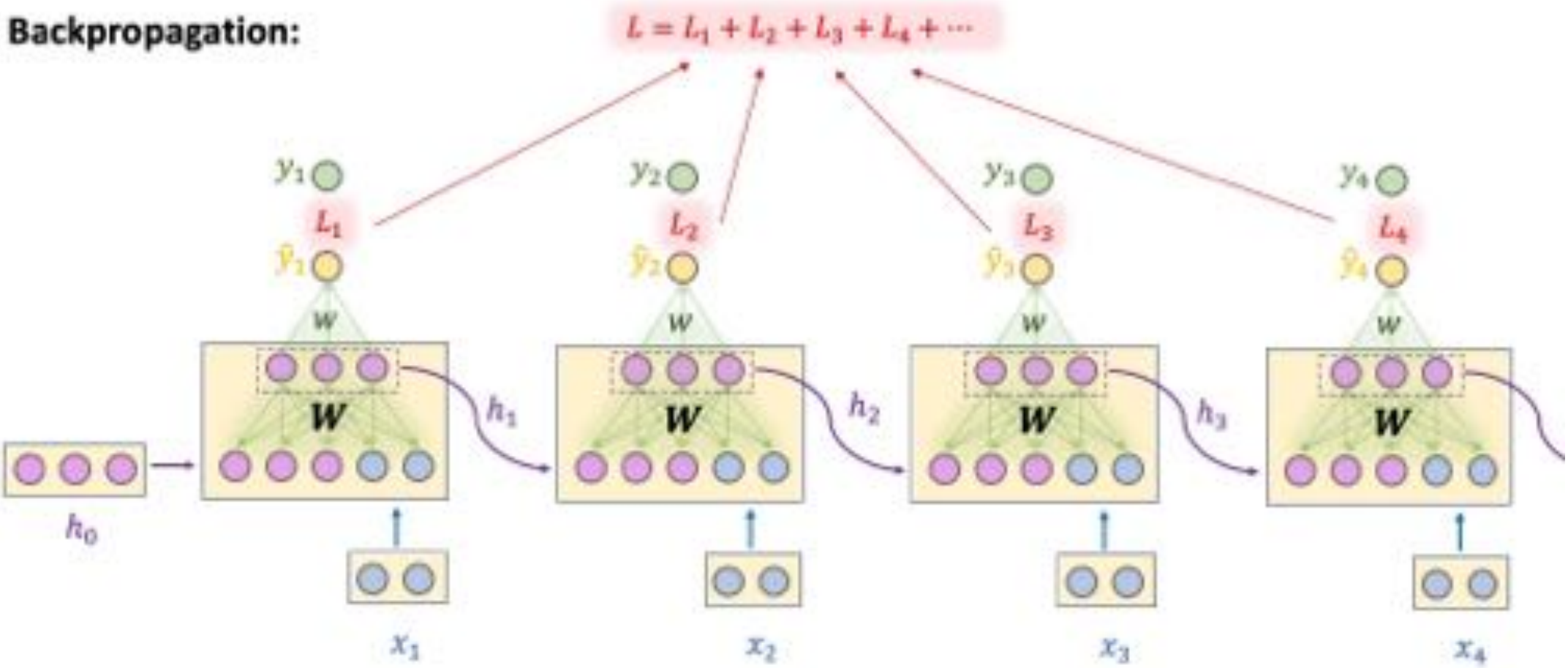

truck
traffic light
car
person
car
car
bicycle
car
car
car
car
car
bicycle

Recurrent neural network (RNN)



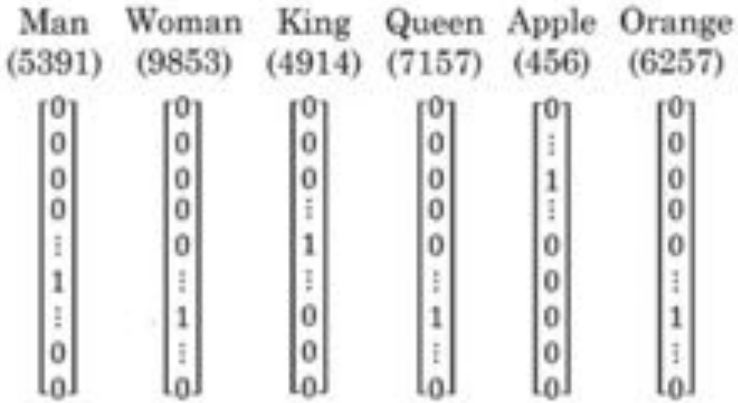
https://www.tensorflow.org/guide/keras/rnn

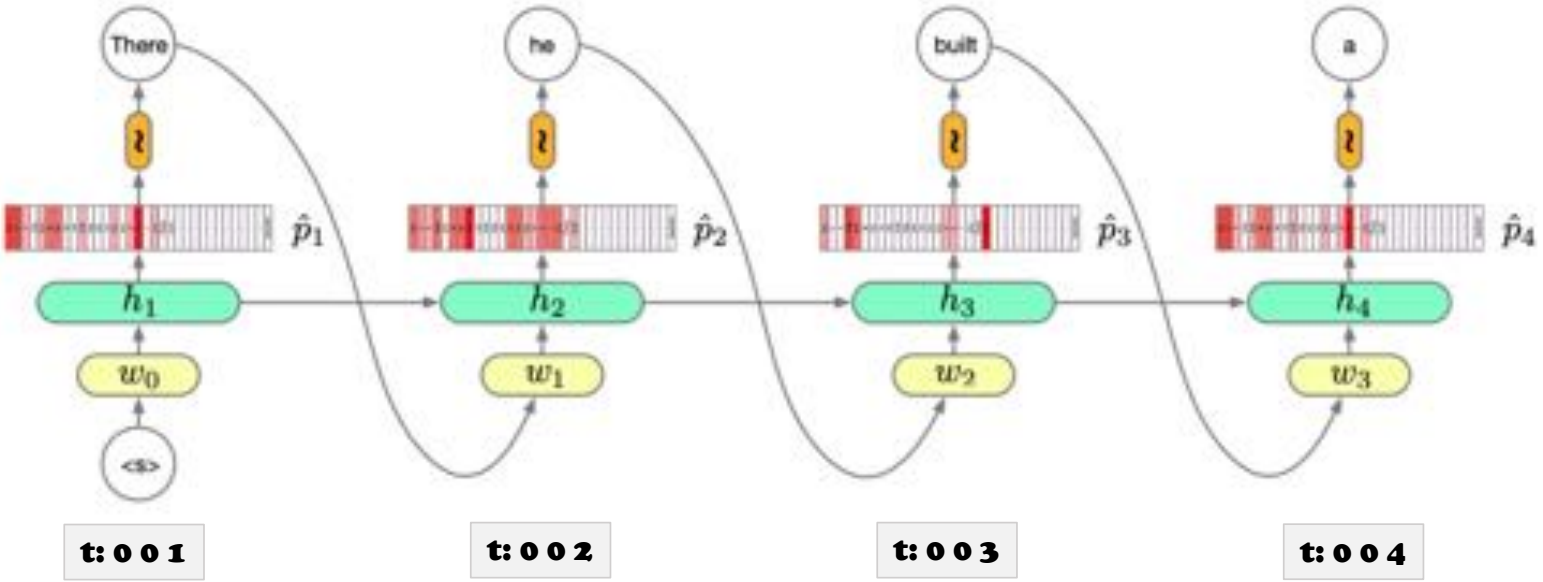
Backpropagation:



Predicting next word

Words representation (1-hot):





at first:

```
tyntd-iafhatawiaoihrdemot  lytdws  e ,tfti, astai f ogoh eoase rrranbyne 'nhthnee e
plia tklrgd t o idoe ns,smtt   h ne etie h,hregtrs nigtike,aoaenns lng
```

train more

```
"Tmont thithey" fomesscerliund
Keushey. Thom here
sheulke, anmerenith ol sivh I lalterthend Bleipile shuwy fil on aseterlome
coaniogennc Phe lism thond hon at. MeiDimorotion in ther thize."
```

train more

```
Aftair fall unsuch that the hall for Prince Velzonski's that me of
her hearly, and behs to so arwage fiving were to it beloge, pavu say falling misfort
how, and Gogition is so overelical and ofter.
```

train more

```
"Why do what that day," replied Natasha, and wishing to himself the fact the
princess, Princess Mary was easier, fed in had oftened him.
Pierre aking his soul came to the packs and drove up his father-in-law women.
```

- **Sentiment Analysis:**
  (many-to-one)

"There is nothing to like in this movie." → ★☆☆☆☆

- **Translation:**

Voulez-vous chanter avec moi? → Do you want to sing with me?

- **Name entity recognition:**
  (many-to-many)

Yesterday, Harry Potter met Hermione Granger. → Yesterday, Harry Potter met Hermione Granger.

- **Video activity recognition:**
  (many-to-one)

Running

- **Image captioning:**
  (one-to-many)

A dog is running in the grass with a frisbee

2014

The OG GAN

2015

Deep Convolutional GAN

2016

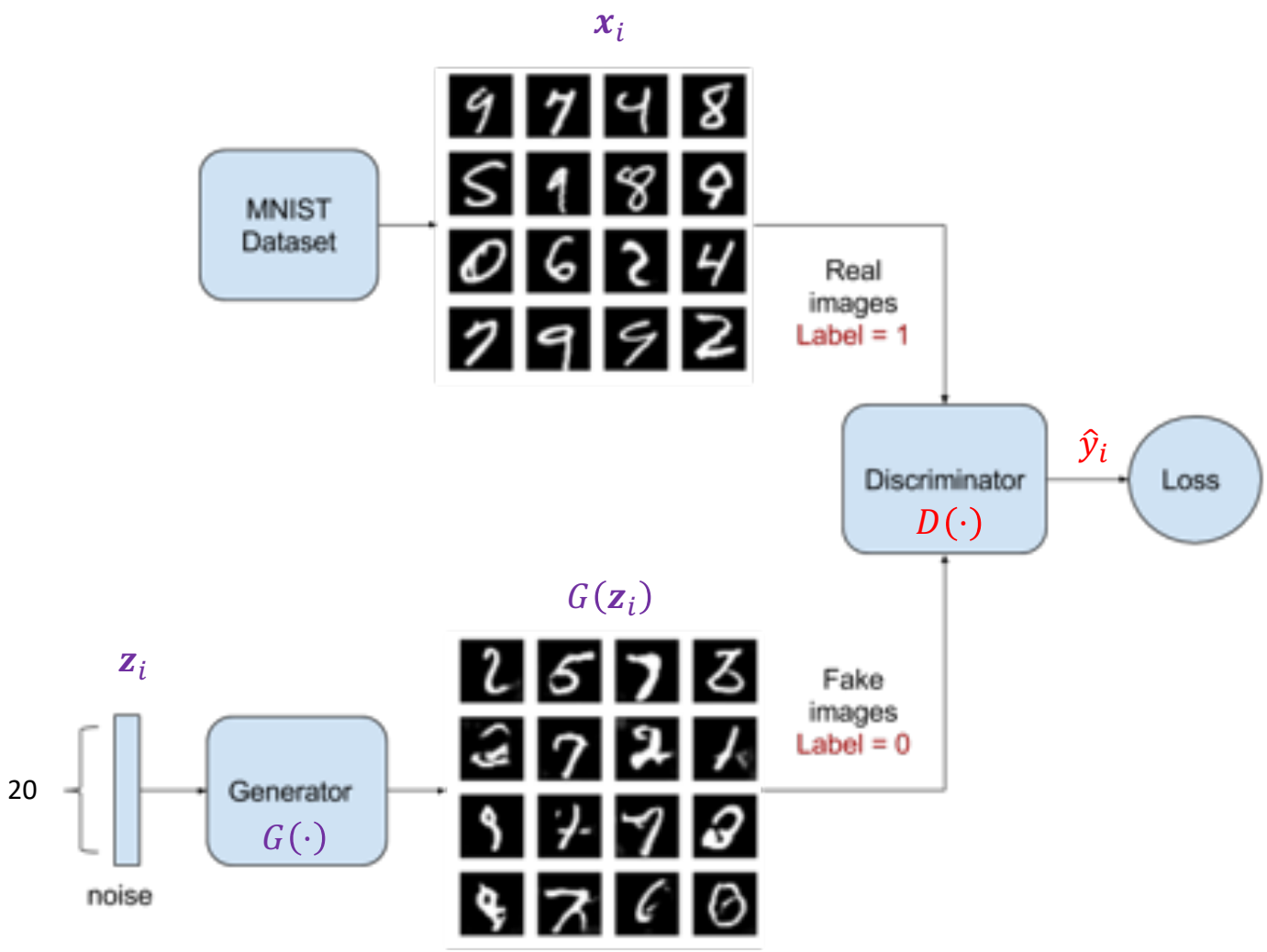
Coupled GAN

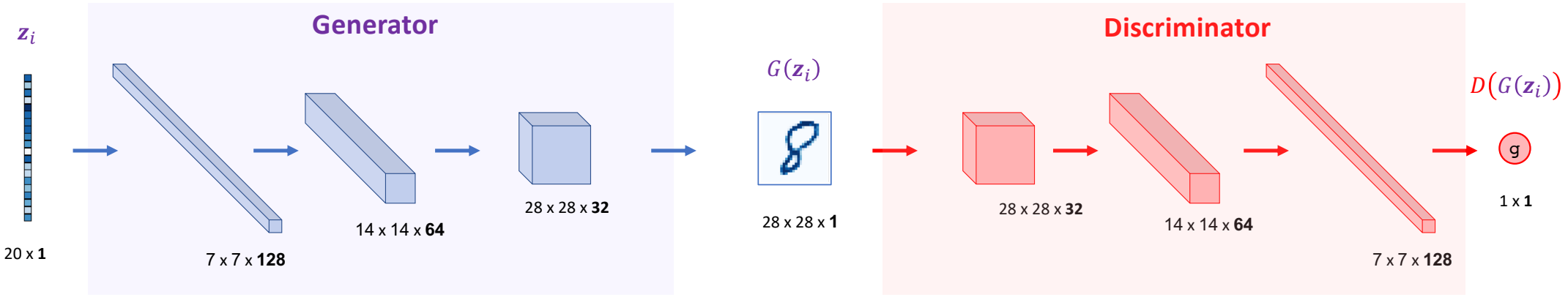
2017

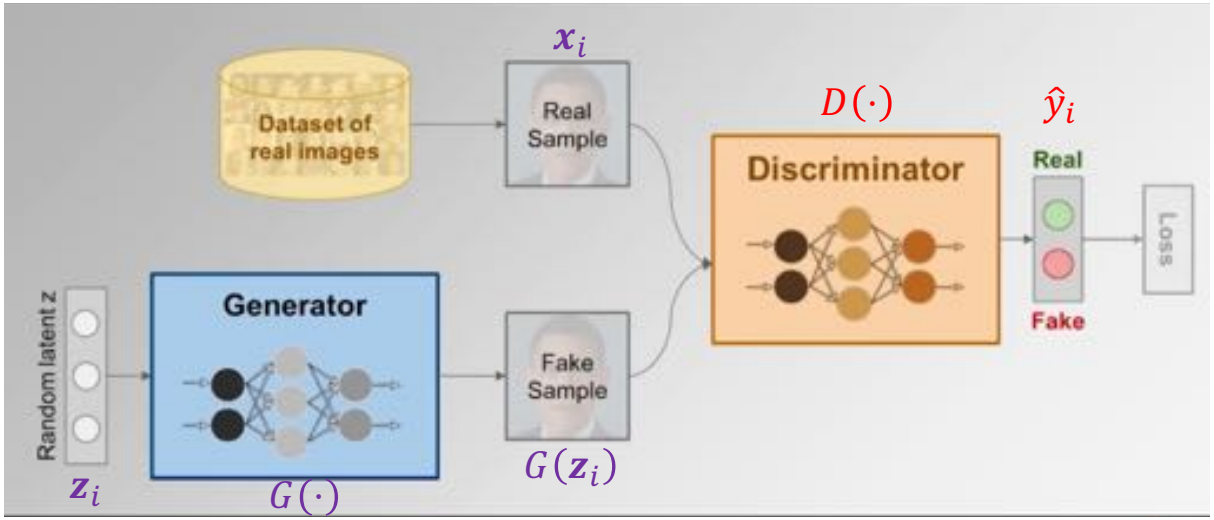
Progressively Growing GAN

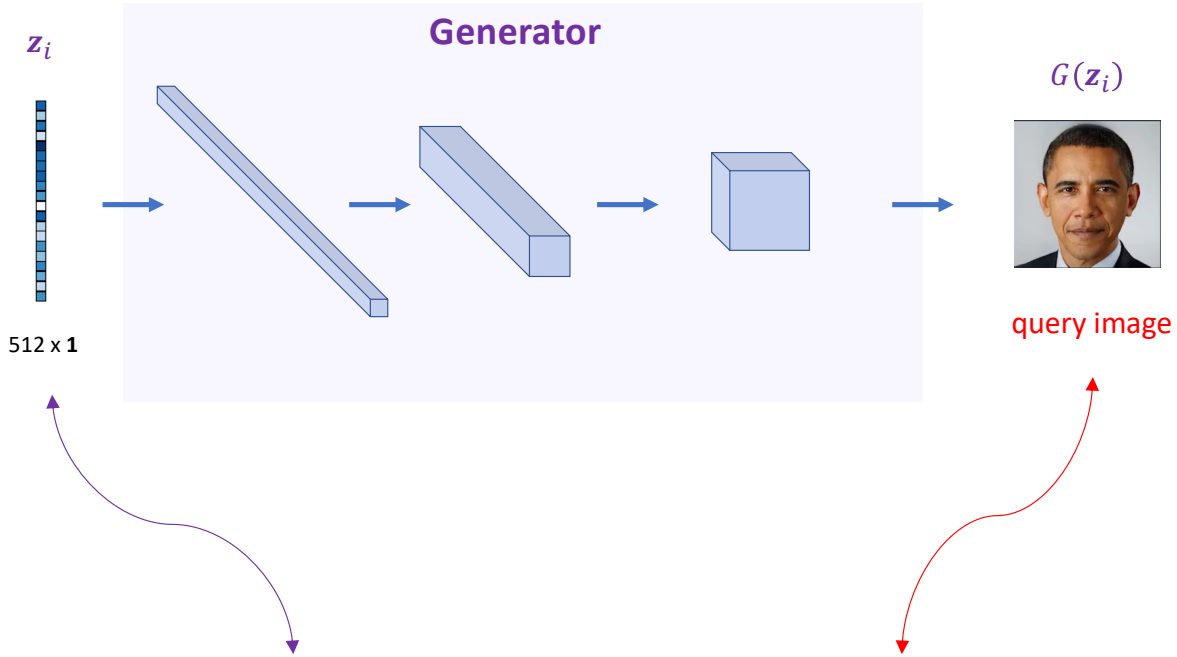
2018

Style-based GAN


x_i
MNIST Dataset
Real images
Label = 1
Discriminator
$D(\cdot)$
$\hat{y}_i$
Loss
$G(z_i)$
z_i
20
Generator
$G(\cdot)$
noise
Fake images
Label = 0

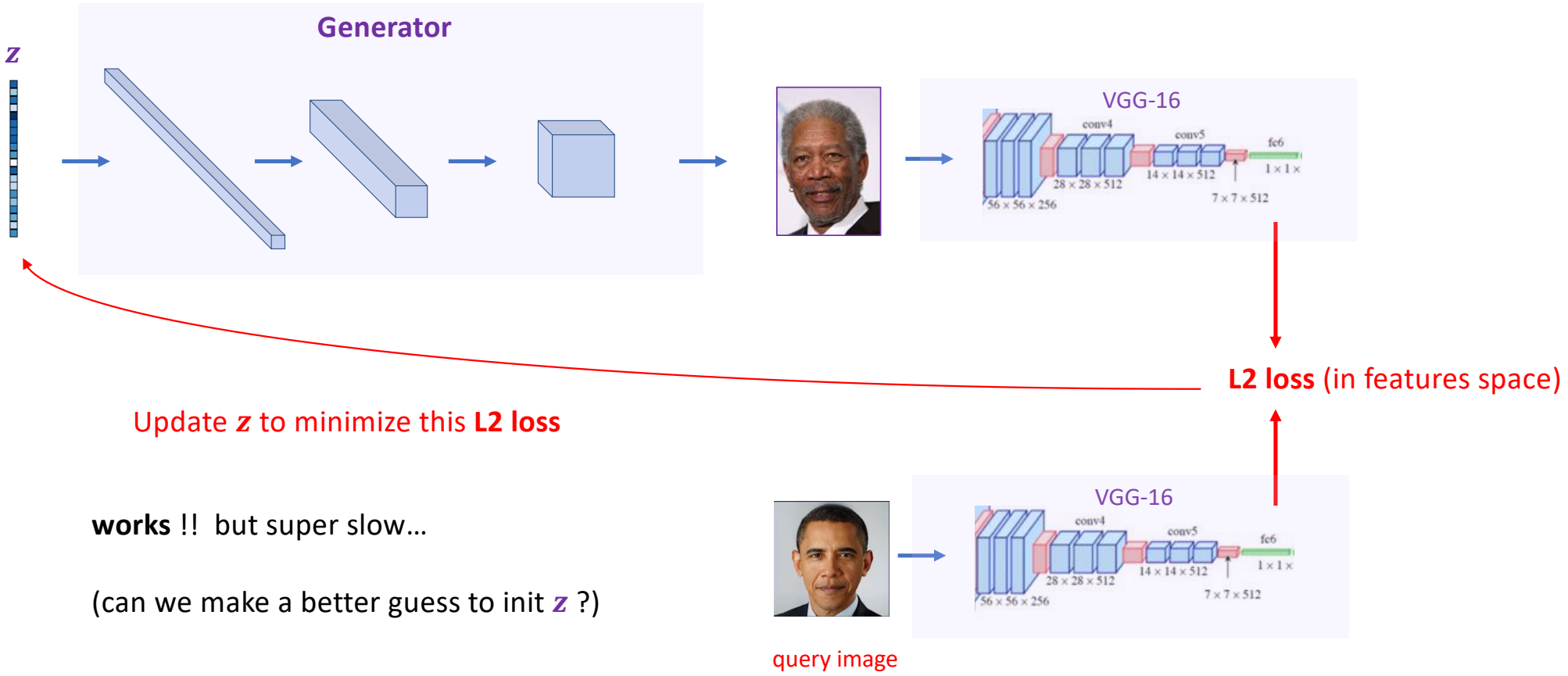

z_i
20 x 1
Generator
7 x 7 x 128
14 x 14 x 64
28 x 28 x 32
$G(z_i)$
28 x 28 x 1
Discriminator
28 x 28 x 32
14 x 14 x 64
7 x 7 x 128
$D(G(z_i))$
g
1 x 1


x_i
Dataset of real images
Real Sample
$D(\cdot)$
$\hat{y}_i$
Discriminator
Real
Fake
Loss
Random latent z
Generator
Fake Sample
z_i
$G(\cdot)$
$G(z_i)$



- How can we find z_i that will produce a specific query image? (random search..?)

- How can we find z_i that will produce a specific query image?



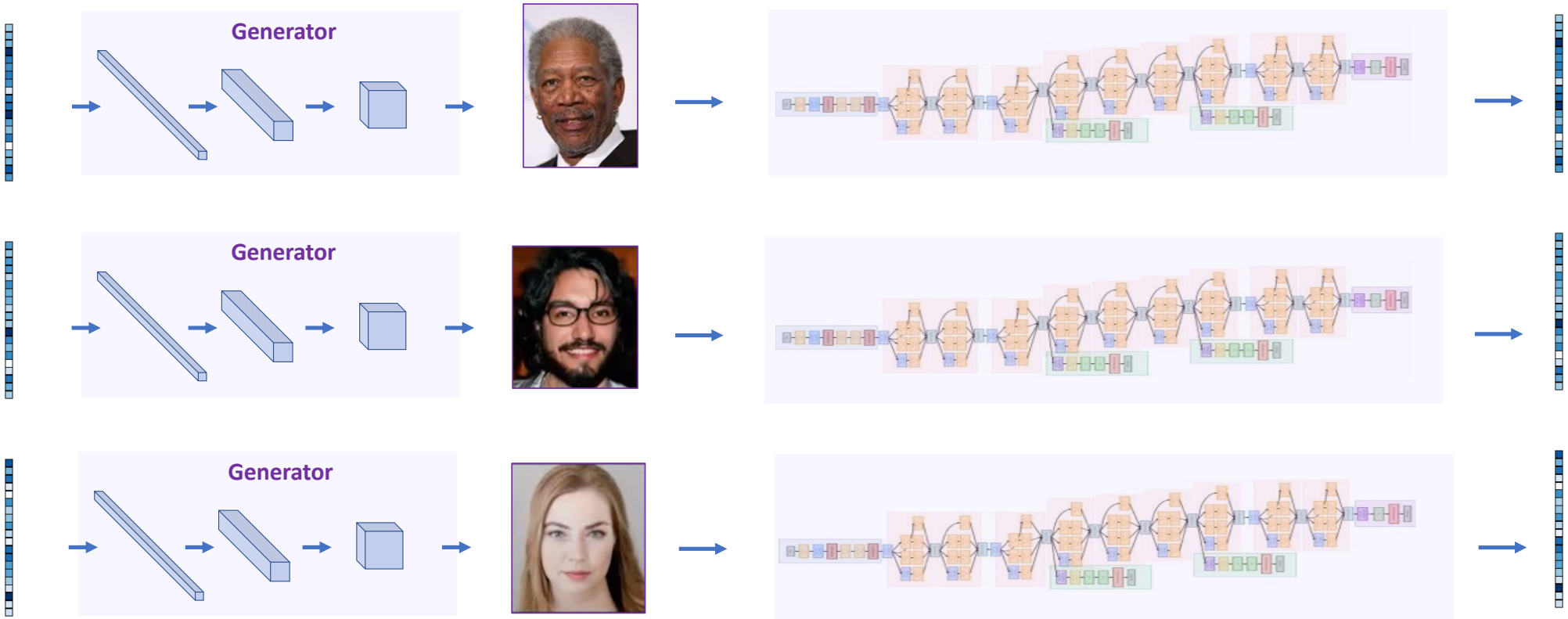
works !! but super slow…

(can we make a better guess to init z ?)

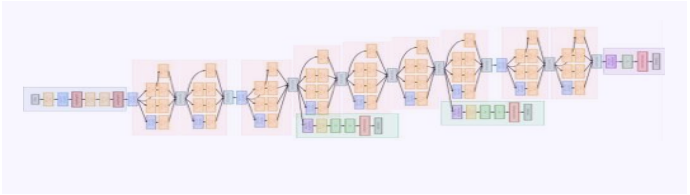
Can we make a better guess to init z ?

Let's generate some training set:

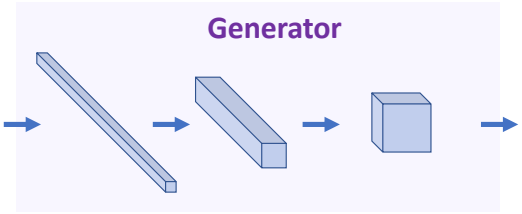
Then, **train a new model** to go from image to z:

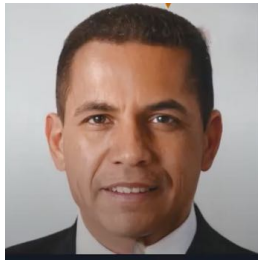


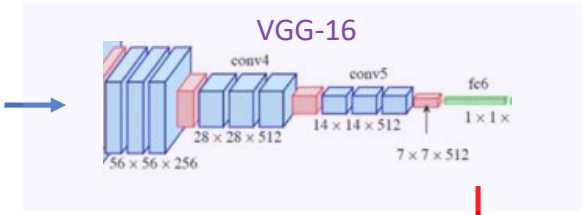
query image



Initial z



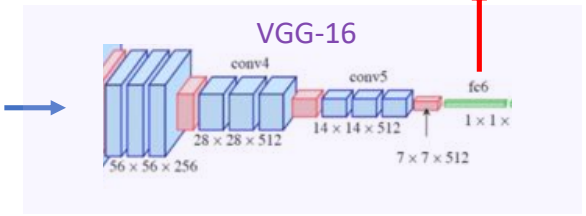




z

L2 loss (in features space)

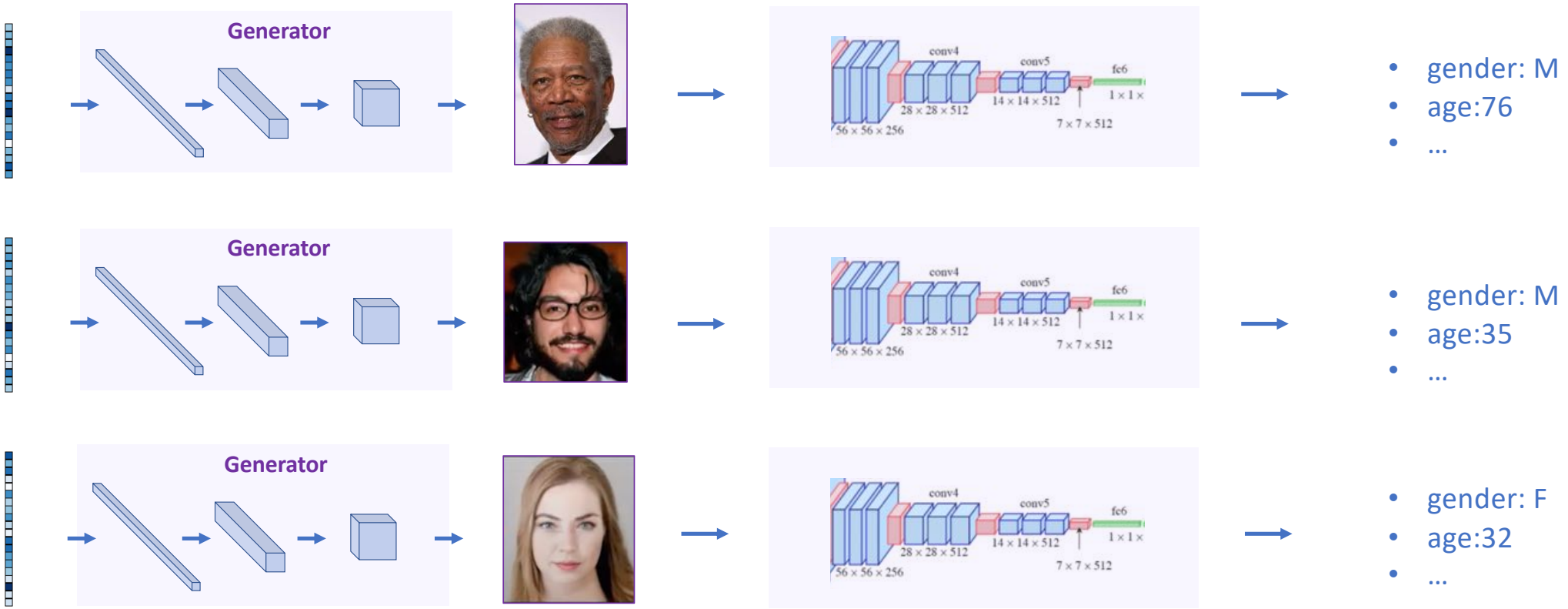
Update z to minimize this **L2 loss**



(Ok, so we found z_i, now what?)

Generate some random vectors…

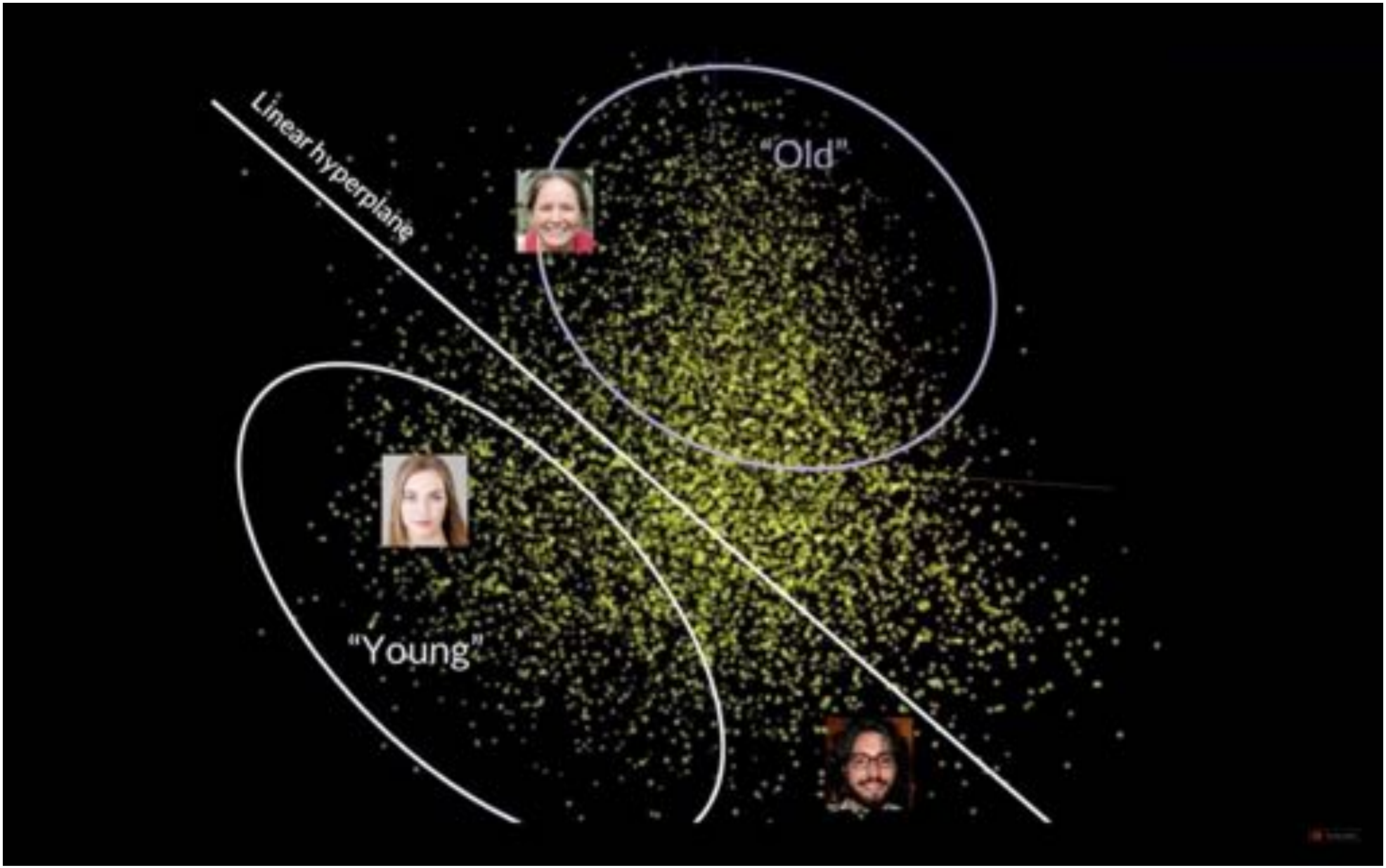
Use pre-trained classifier to extract some attributes:

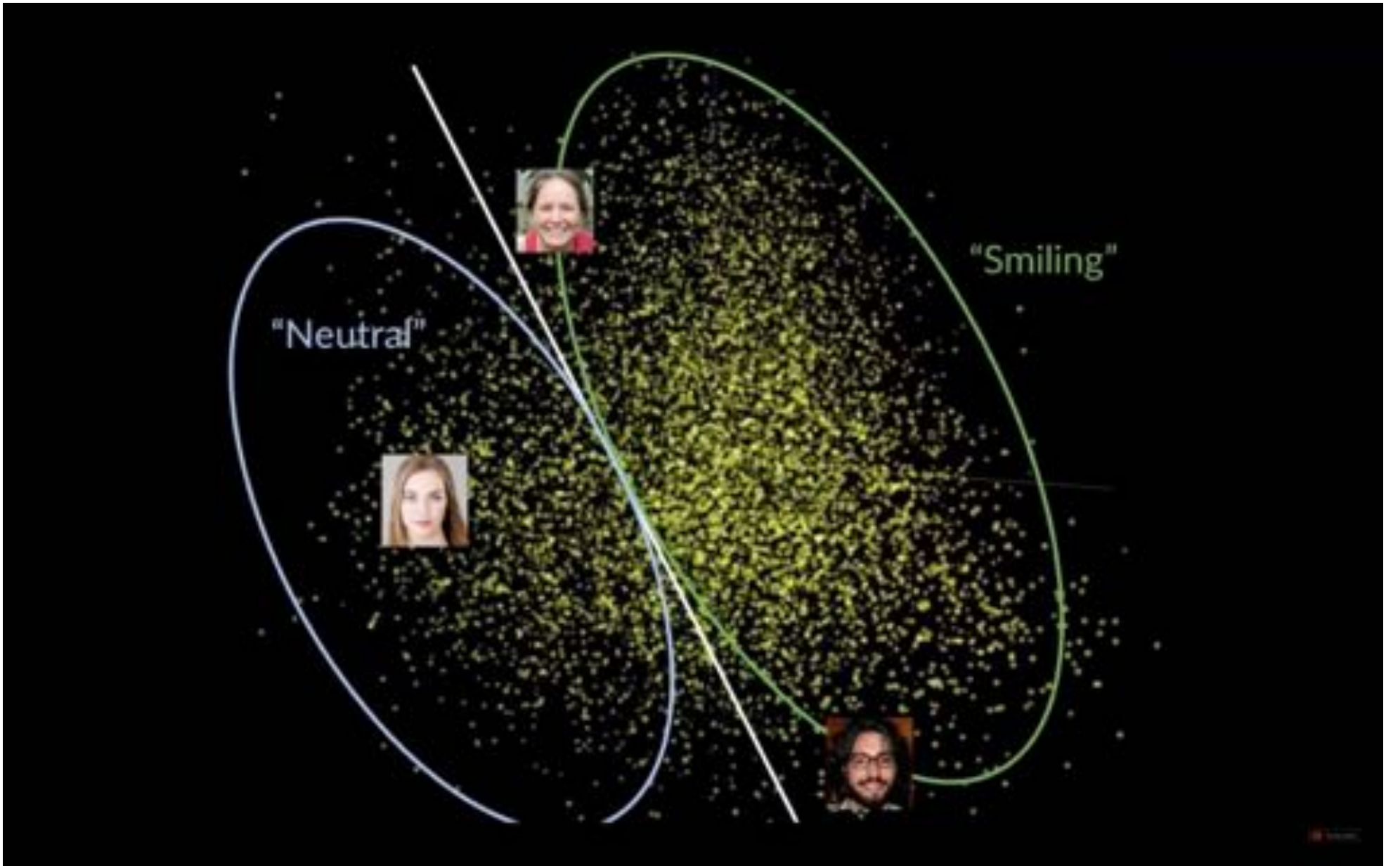


- gender: M
- age:76
- …

- gender: M
- age:35
- …

- gender: F
- age:32
- …


Linear hyperplane
"Old"
"Young"


"Smiling"
"Neutral"


Original
Pose
Age
Expression
Eyeglasses

https://youtu.be/0VH1Lim8gL8

History of Deep Learning Ideas and Milestones*

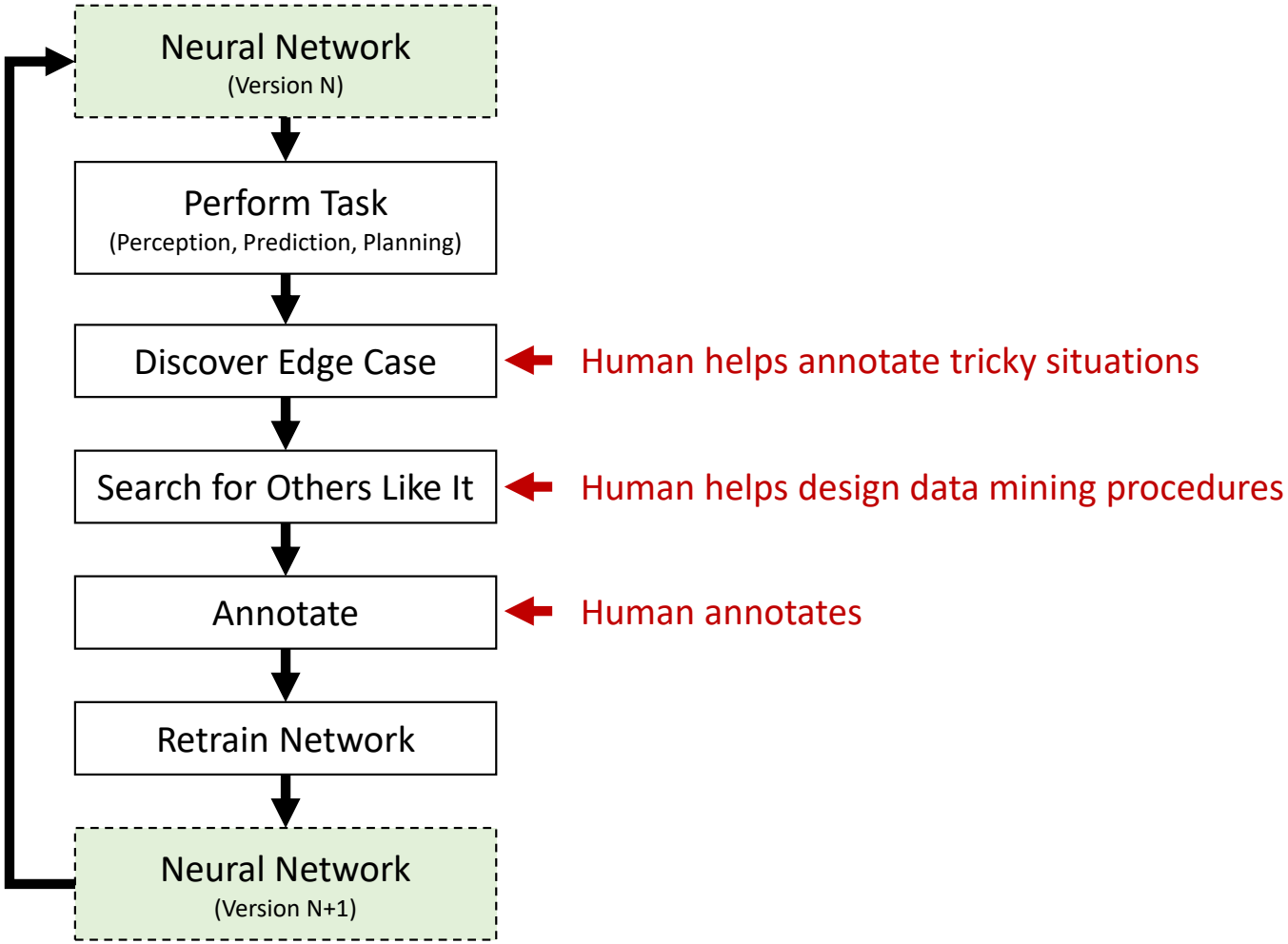
Perspective:

- Universe created
  13.8 billion years ago
- Earth created
  4.54 billion years ago
- Modern humans
  300,000 years ago
- Civilization
  12,000 years ago
- Written record
  5,000 years ago

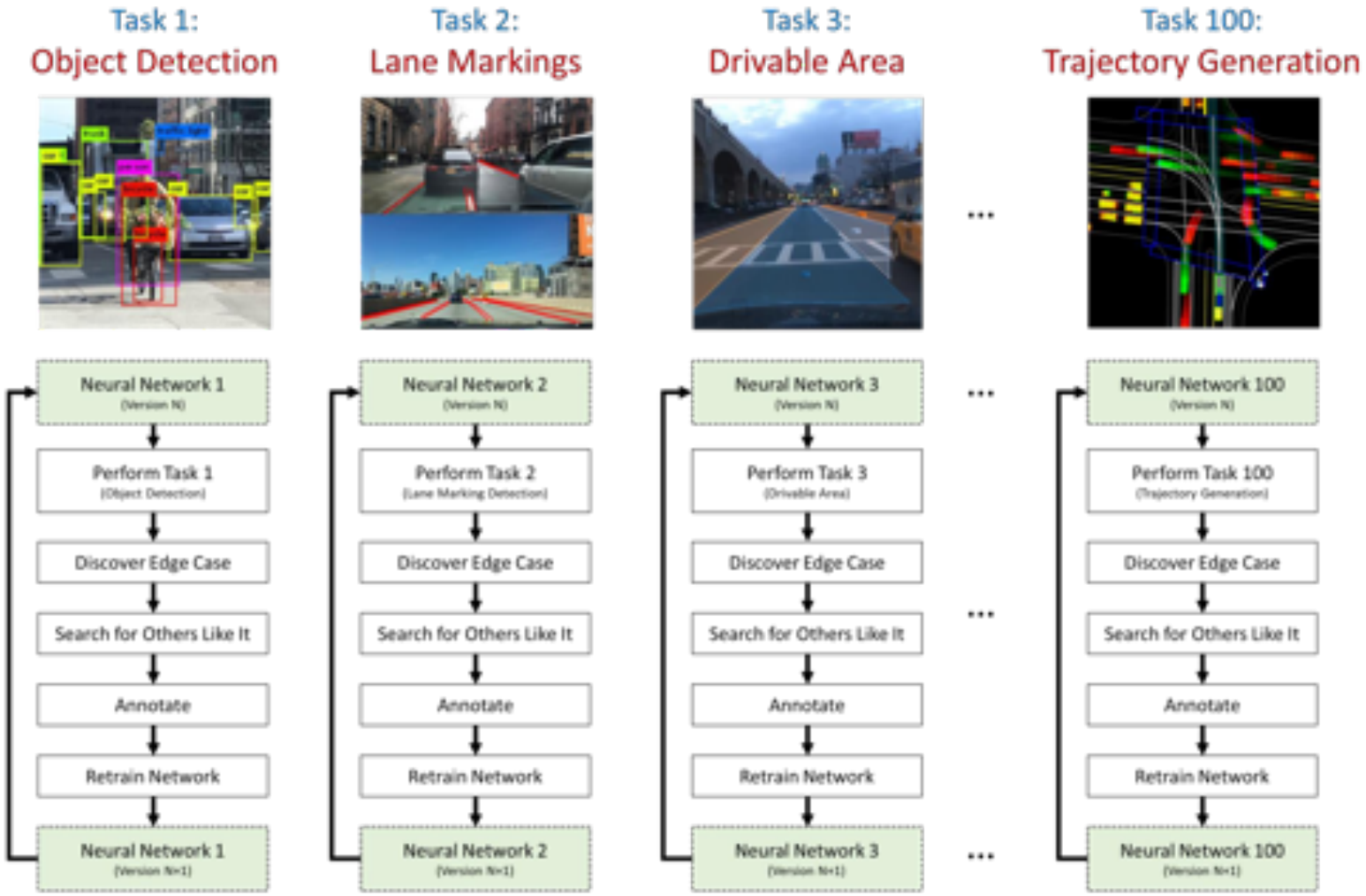
- 1943: Neural networks
- 1957-62: Perceptron
- 1970-86: Backpropagation, RBM, RNN
- 1979-98: CNN, MNIST, LSTM, Bidirectional RNN
- 2006: “Deep Learning”, DBN
- 2009: ImageNet + AlexNet
- 2014: GANs
- 2016-17: AlphaGo, AlphaZero
- 2017: 2017-19: Transformers

* Dates are for perspective and not as definitive historical record of invention or credit

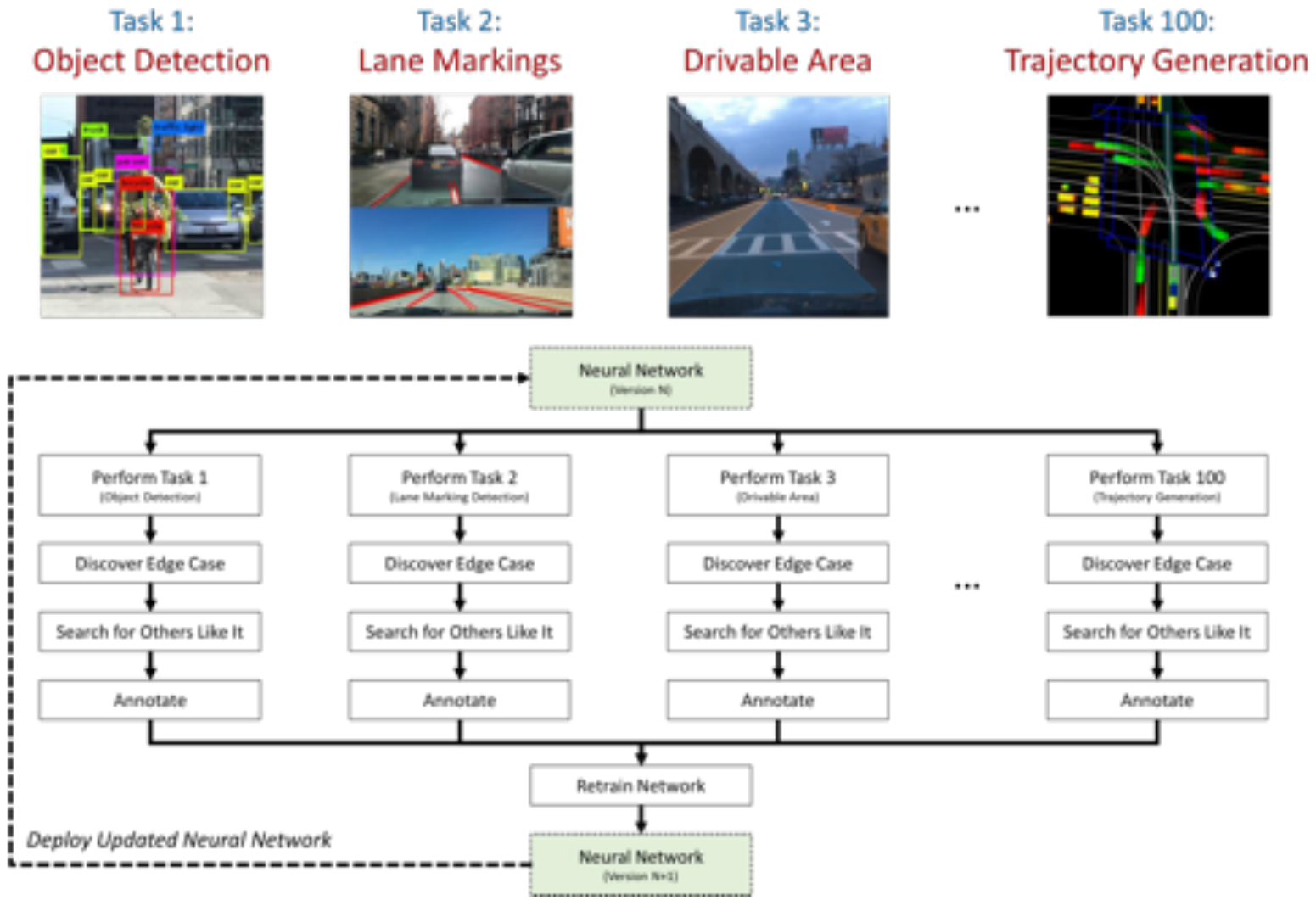
Active Learning Pipeline
(aka Data Engine)



Single-Task Learning

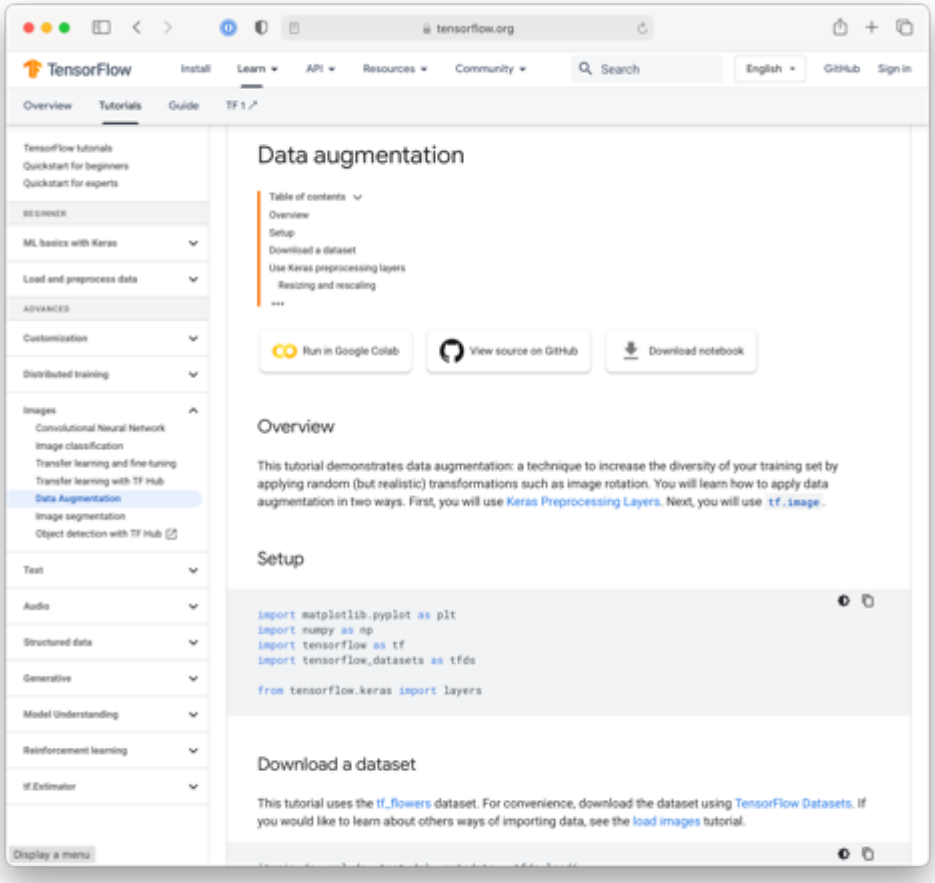


Multi-Task Learning



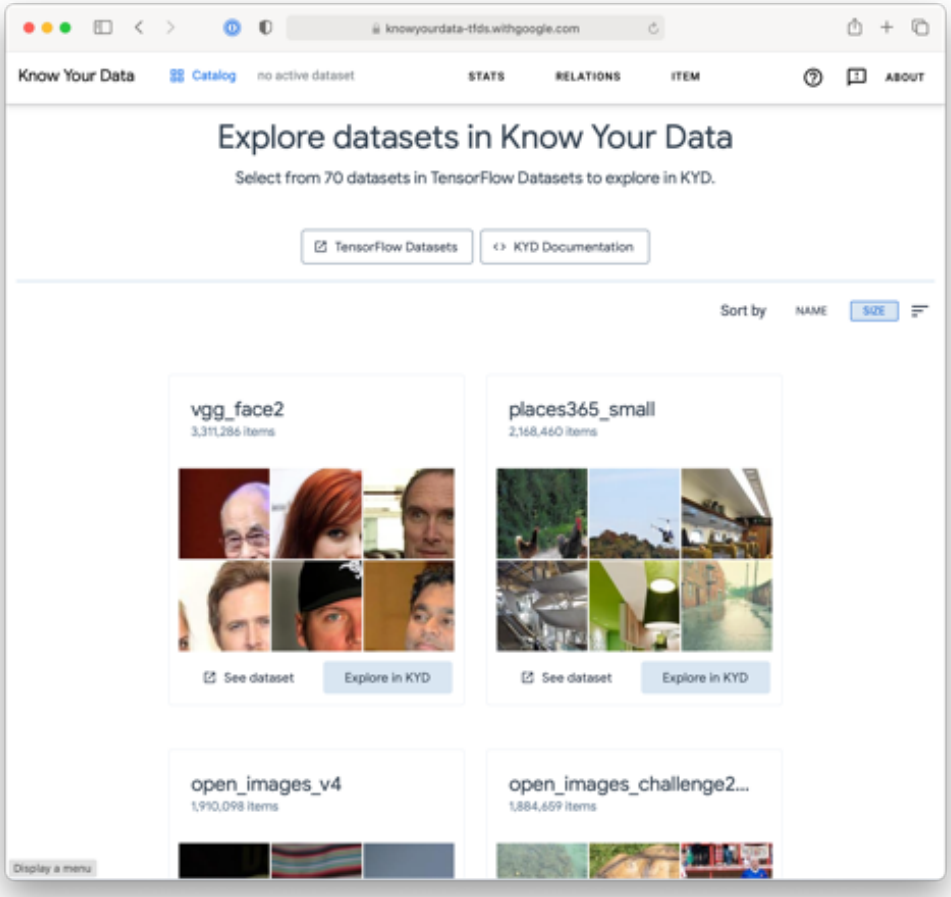
Useful resources

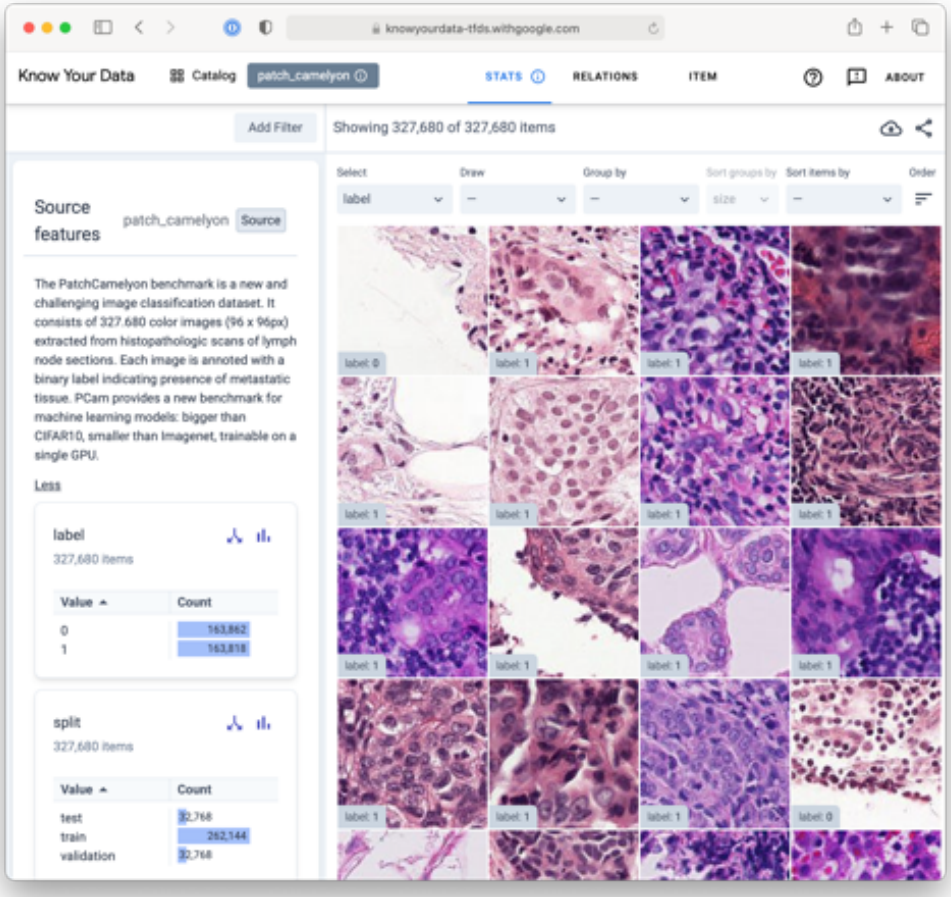
Tensorflow tutorials: https://www.tensorflow.org/tutorials



https://www.tensorflow.org/tutorials/images/data_augmentation

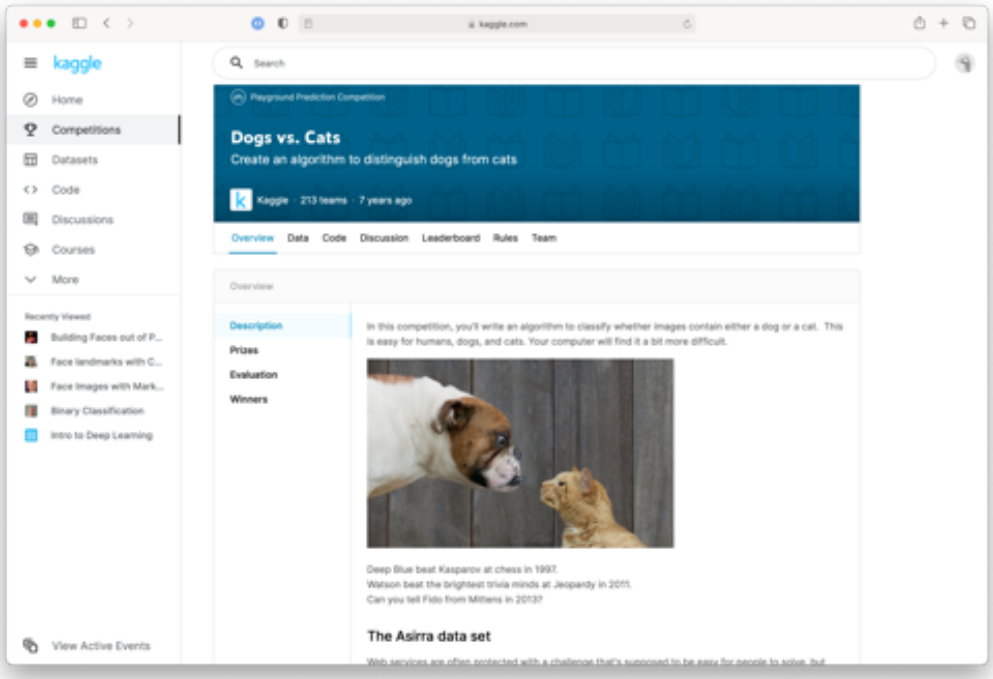
Tensorflow datasets: https://www.tensorflow.org/datasets/overview



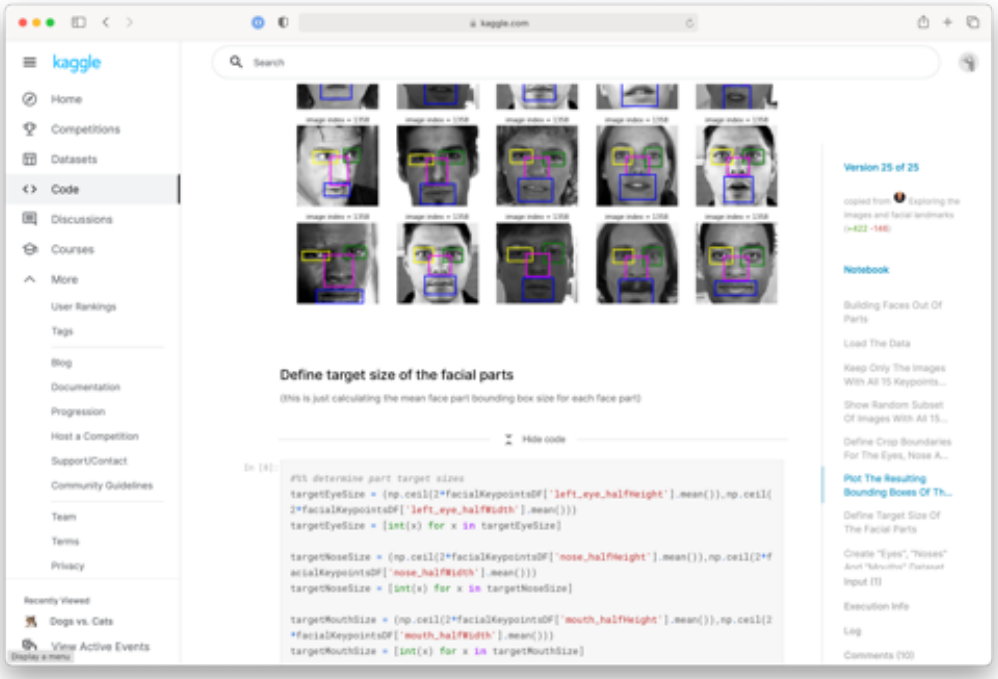


https://knowyourdata-tfds.withgoogle.com

Kaggle:



https://www.kaggle.com/c/dogs-vs-cats/overview



https://www.kaggle.com/selfishgene/building-faces-out-of-parts

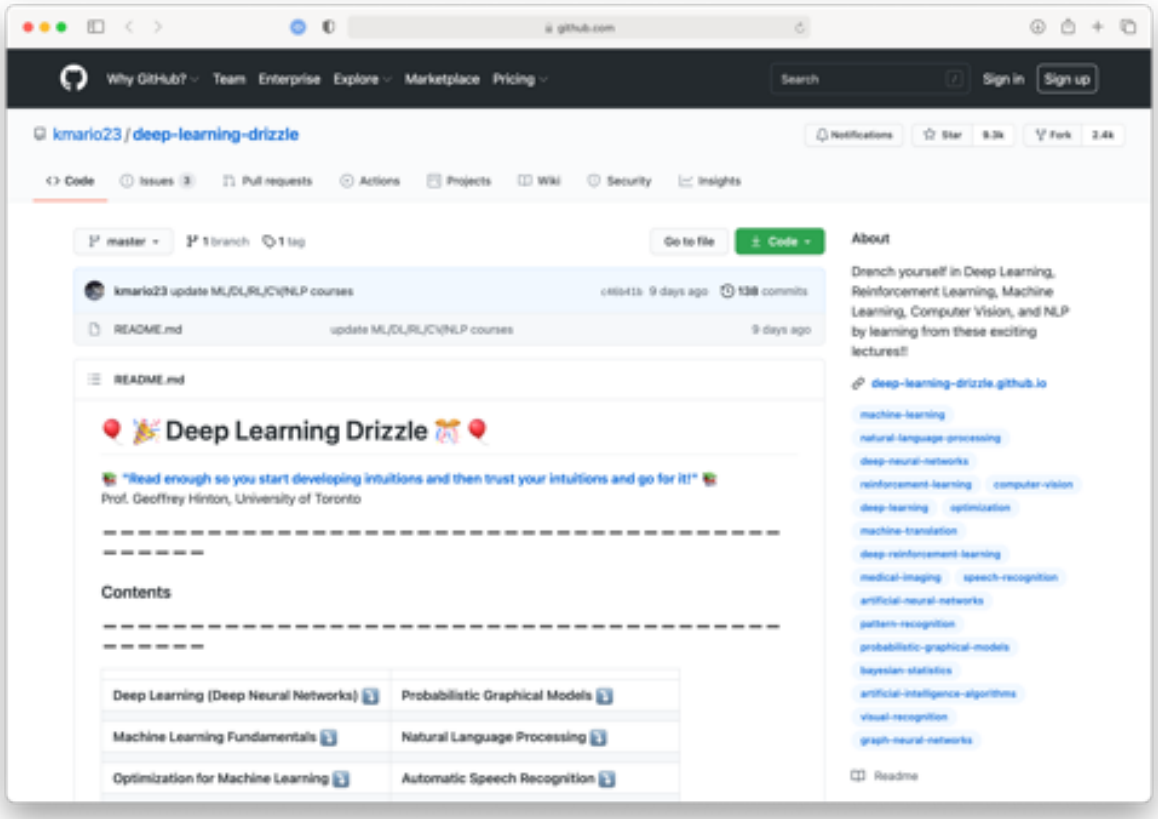
Over 2000 tutorials

Link: http://bit.ly/36skFE7

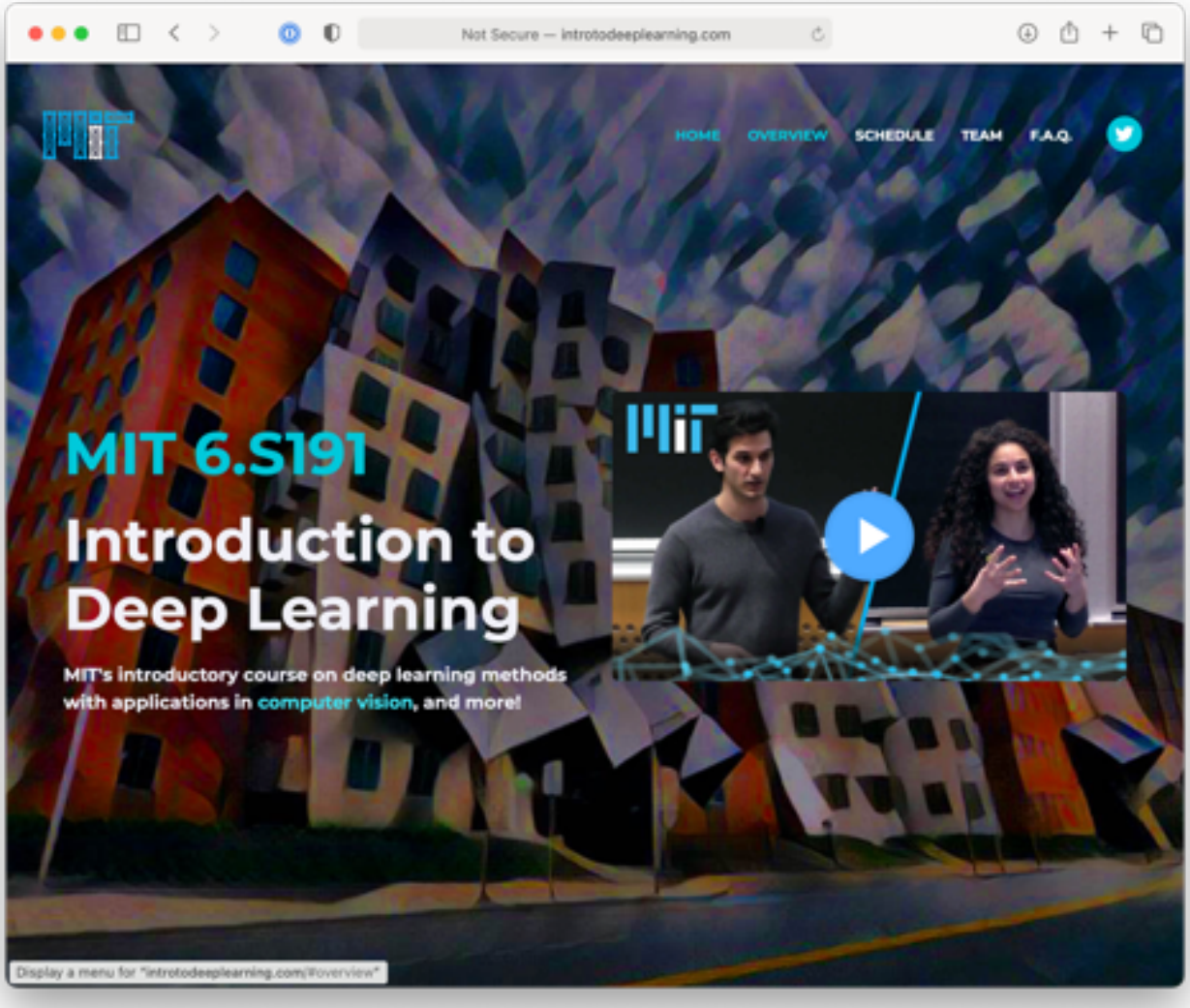
- Topics
  - Machine learning
  - Activation and Loss Functions
  - Bias
  - Perceptron
  - Regression
  - Gradient descent
  - Generative learning
  - Support vector machines
  - Backpropagation
  - Deep Learning
  - Optimization
  - Long Short Term Memory
  - Convolutional Neural Networks
  - Recurrent Neural Nets (RNNs)
  - Reinforcement Learning
  - Generative Adversarial Networks
  - Multi-task Learning
  - NLP
  - Word Vectors
  - Encoder-Decoder
  - TensorFlow
  - PyTorch

https://github.com/kmario23/deep-learning-drizzle

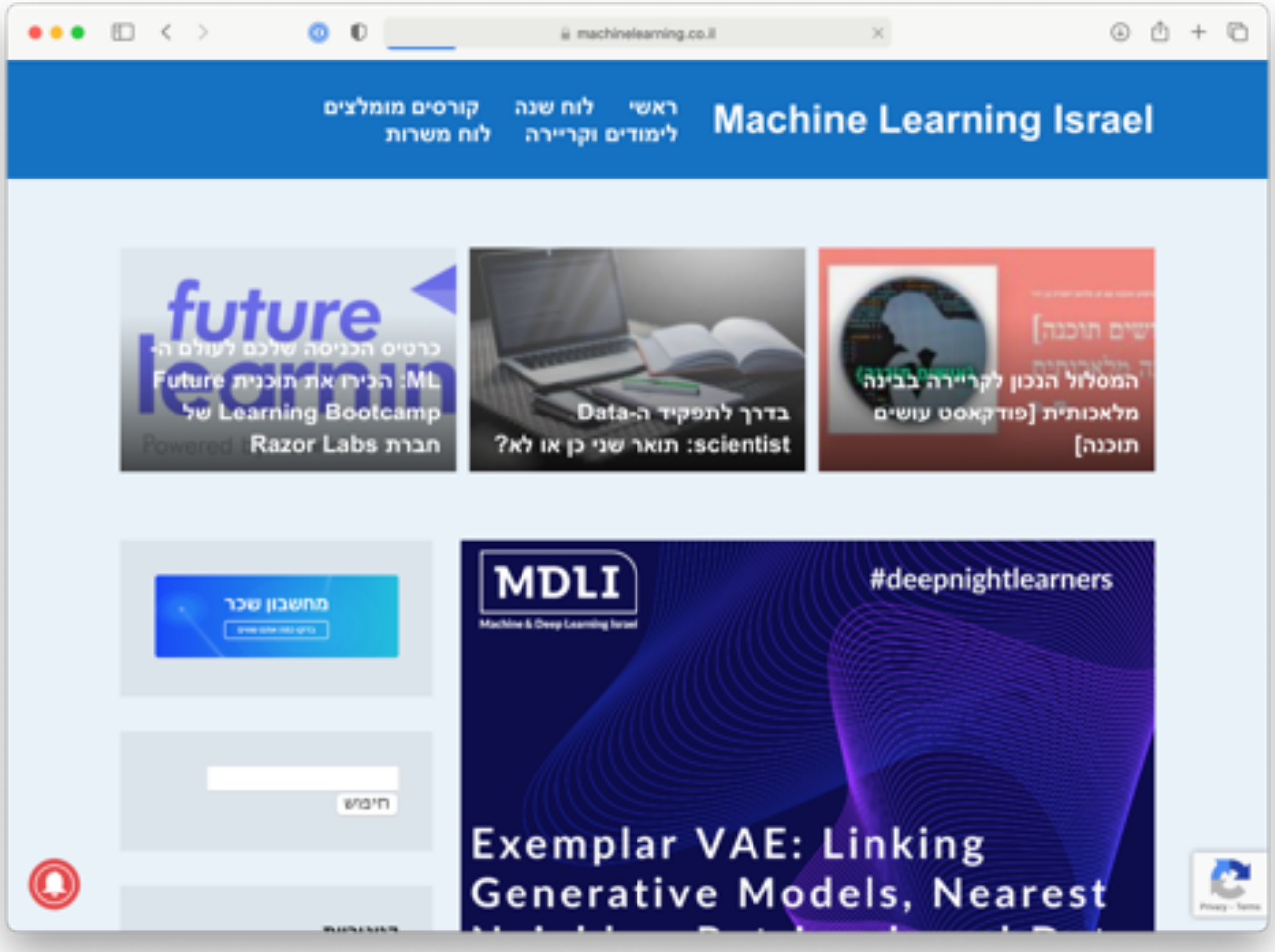
Links to great youtube lectures



9.	**CS231n: CNNs for Visual Recognition**	Andrej Karpathy, Stanford University	CS231n	YouTube-Lectures (Academic Torrent)	2016
10.	**Neural Networks**	Hugo Larochelle, Université de Sherbrooke	Neural-Networks	YouTube-Lectures (Academic Torrent)	2016
11.	**CS224d: Deep Learning for NLP**	Richard Socher, Stanford University	CS224d	YouTube-Lectures (Academic Torrent)	2016
12.	**CS224n: NLP with Deep Learning**	Richard Socher, Stanford University	CS224n	YouTube-Lectures	2017
13.	**CS231n: CNNs for Visual Recognition**	Justin Johnson, Stanford University	CS231n	YouTube-Lectures (Academic Torrent)	2017
14.	**Topics in Deep Learning**	Ruslan Salakhutdinov, CMU	10707	YouTube-Lectures	F2017
15.	**Deep Learning Crash Course**	Leo Isikdogan, UT Austin	None	YouTube-Lectures	2017
16.	**Deep Learning and its Applications**	François Pitié, Trinity College Dublin	EE4C16	YouTube-Lectures	2017
17.	**Deep Learning**	Andrew Ng, Stanford University	CS230	YouTube-Lectures	2018
18.	**UvA Deep Learning**	Efstratios Gavves, University of Amsterdam	UvA-DLC	Lecture-Videos	2018
19.	**Advanced Deep Learning and Reinforcement Learning**	Many legends, DeepMind	None	YouTube-Lectures	2018
20.	**Machine Learning**	Peter Bloem, Vrije Universiteit Amsterdam	MLVU	YouTube-Lectures	2018
21.	**Deep Learning**	Francois Fleuret, EPFL	EE-59	Video-Lectures	2018



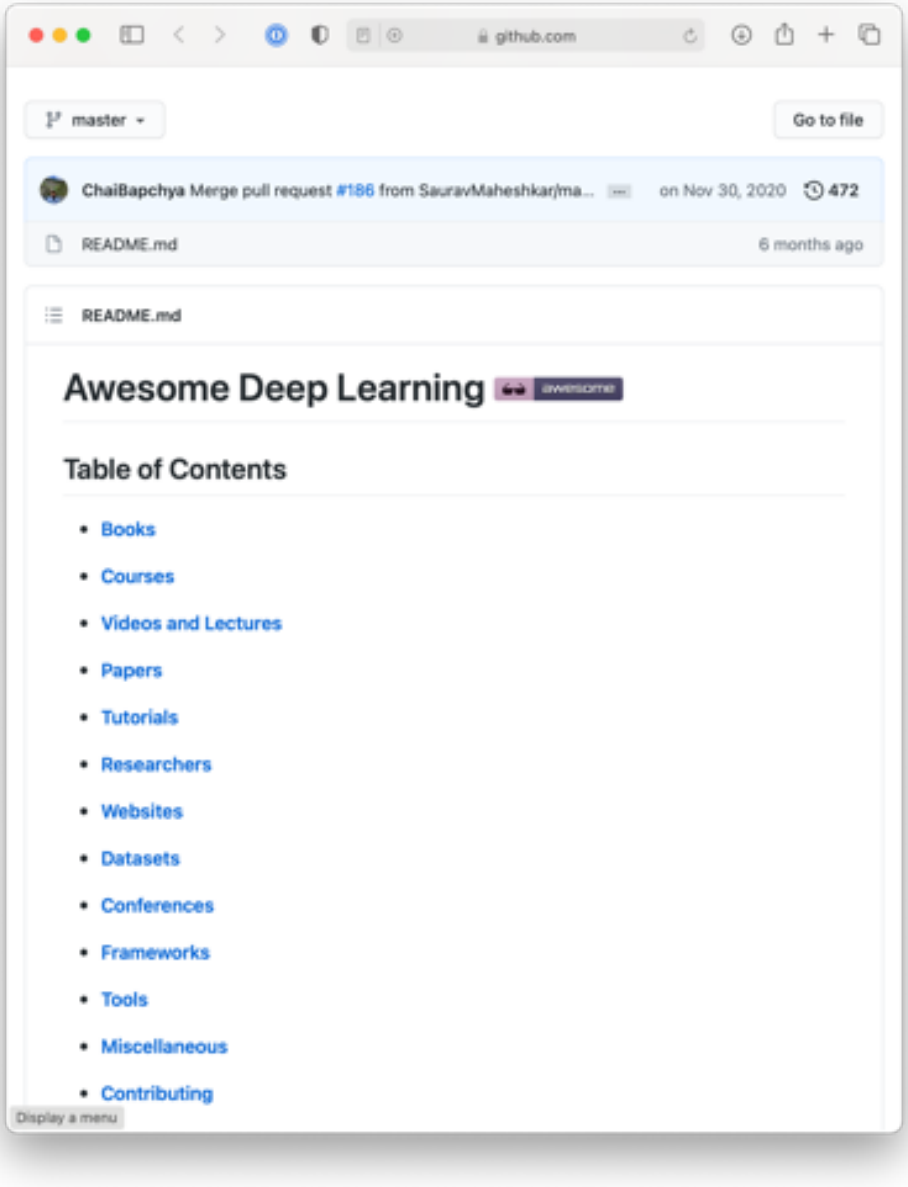
http://introtodeeplearning.com

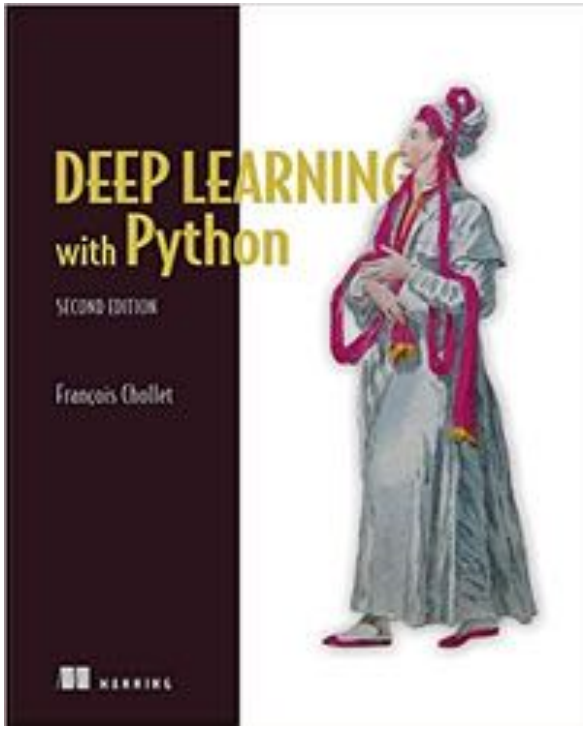


https://machinelearning.co.il

https://github.com/ChristosChristofidis/awesome-deep-learning

A great list of resources..





Highly recommended book
(second edition expected later this year..)

https://www.manning.com/books/deep-learning-with-python-second-edition

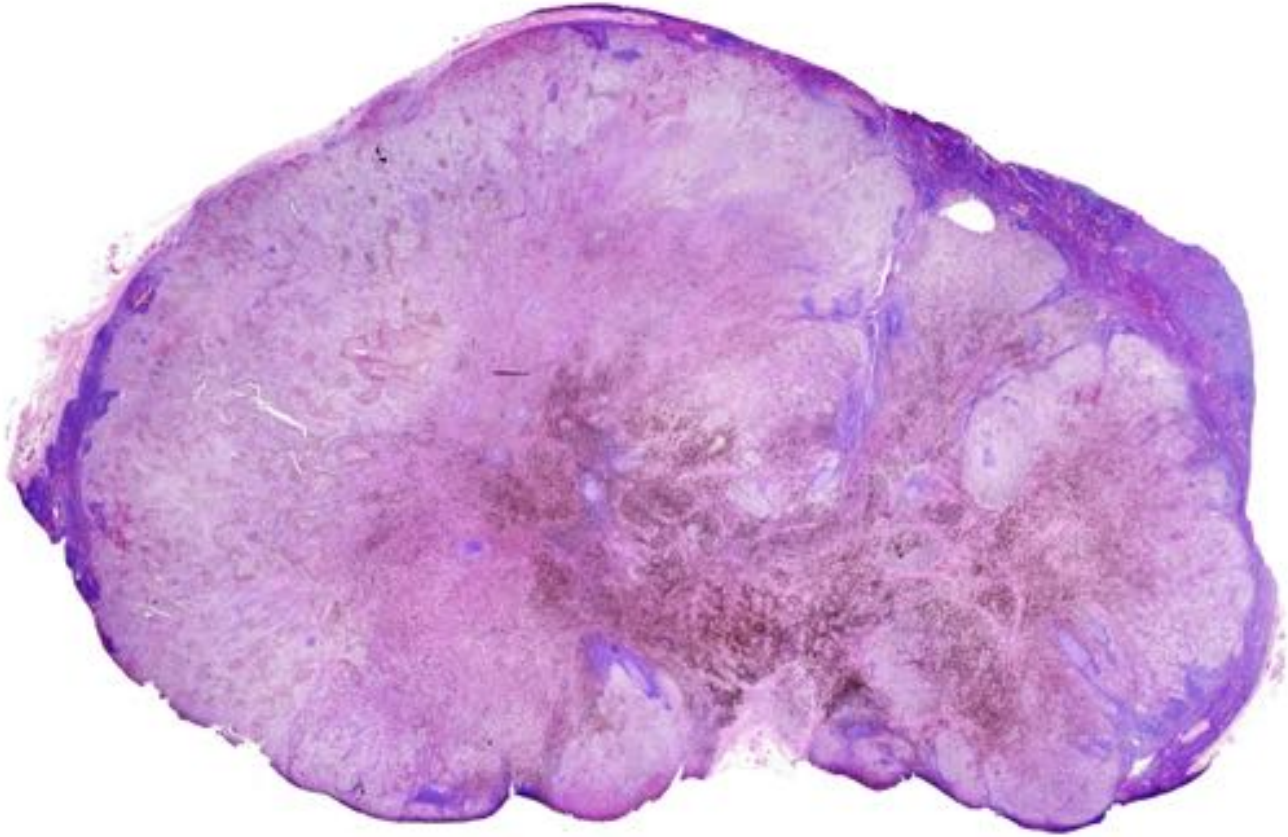
https://github.com/fchollet/deep-learning-with-python-notebooks

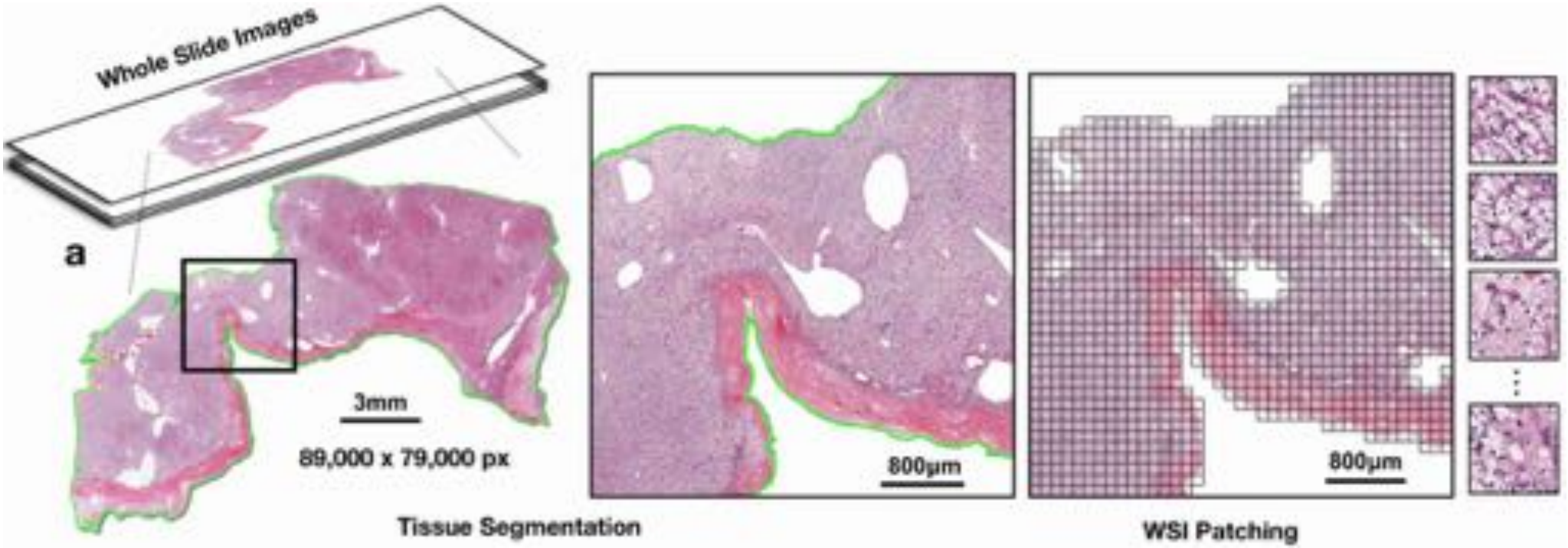
(Great **ipynb** examples code available online)

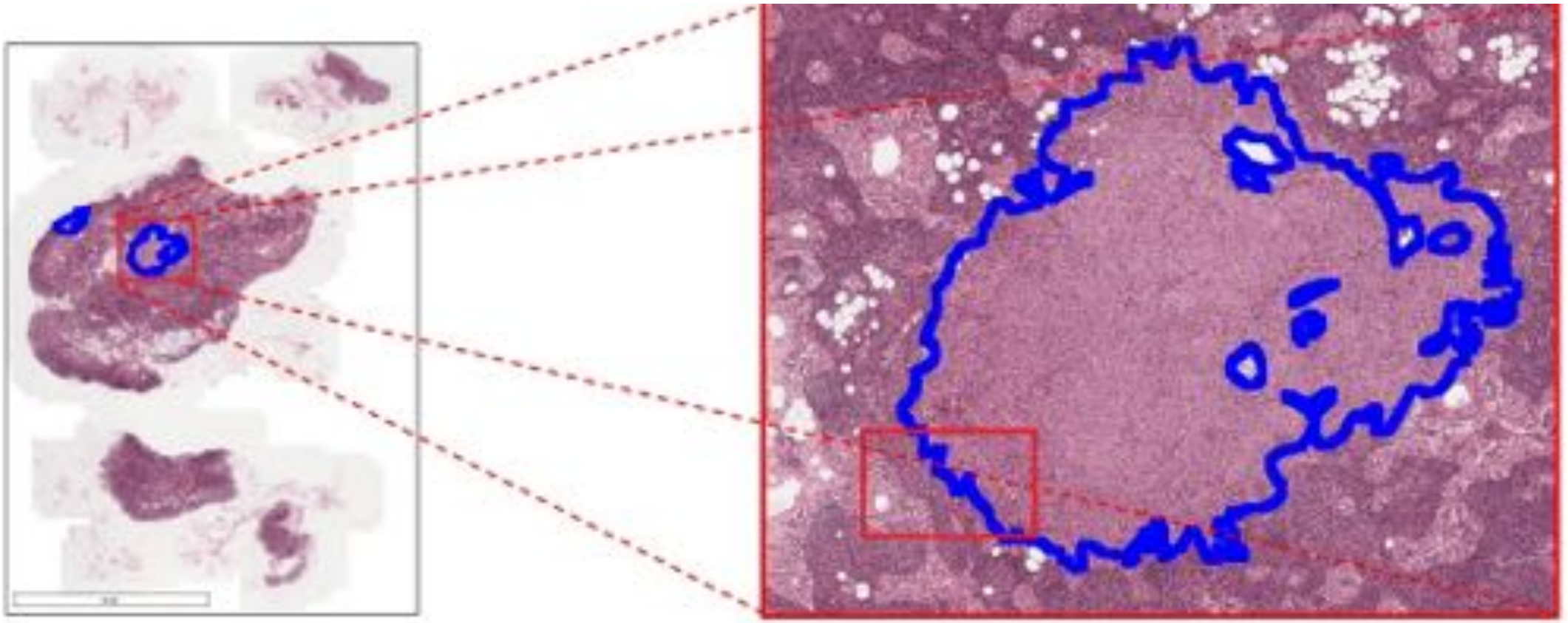
Table of contents

Final Project

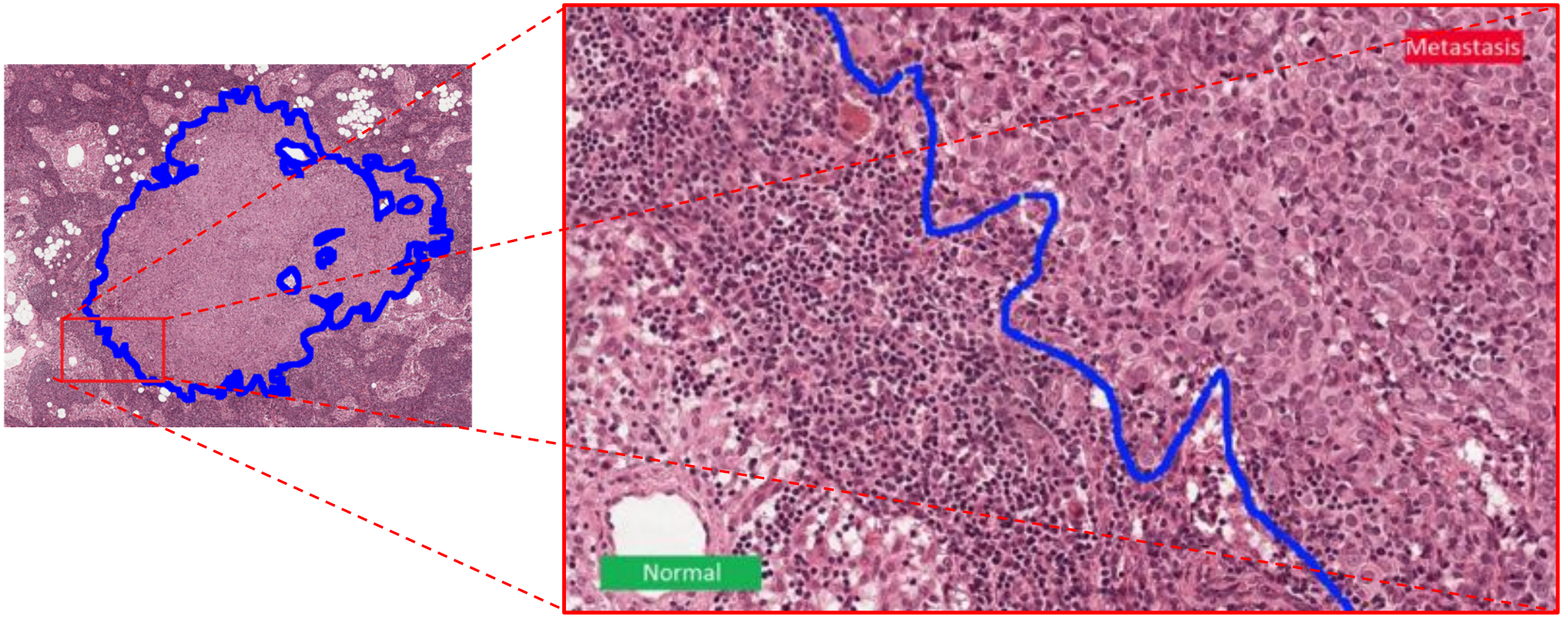
Classification of histopathology images

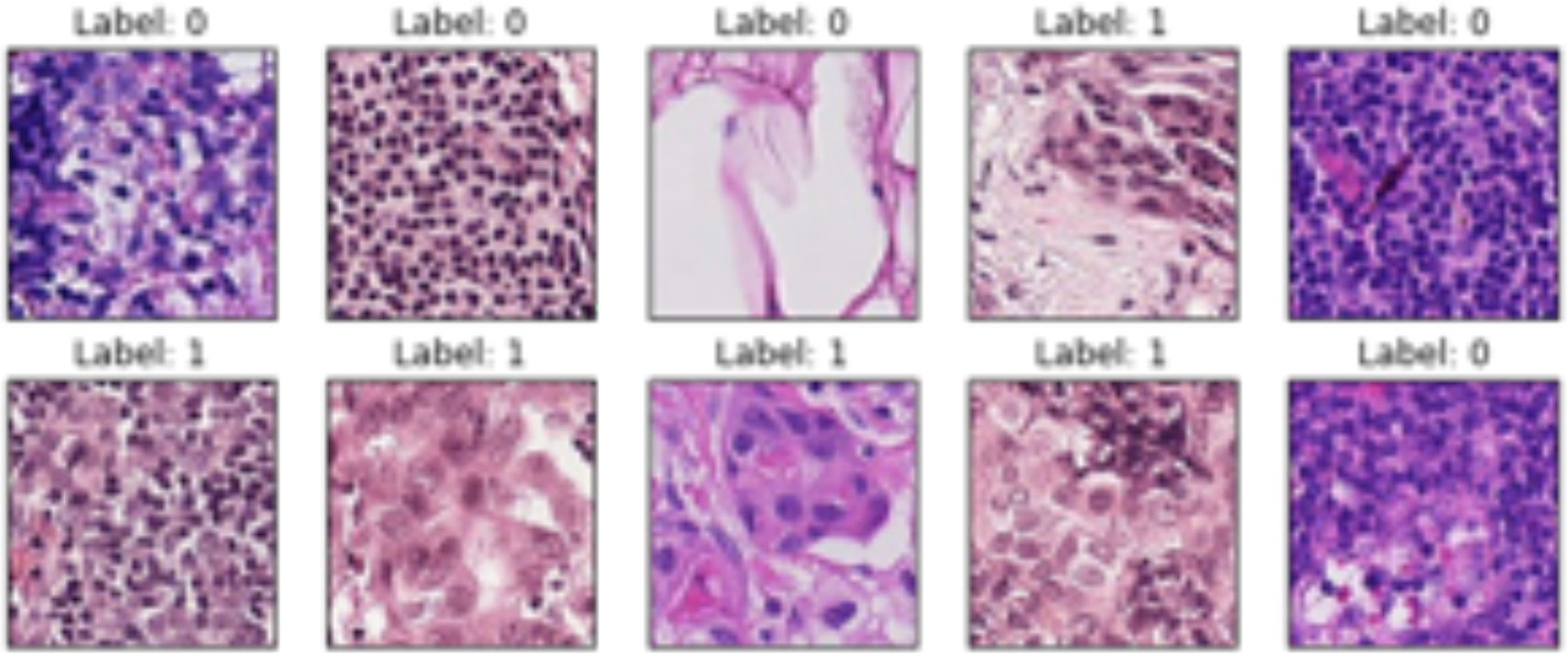



Whole Slide Images
a
3mm
89,000 x 79,000 px
800µm
800µm
Tissue Segmentation
WSI Patching



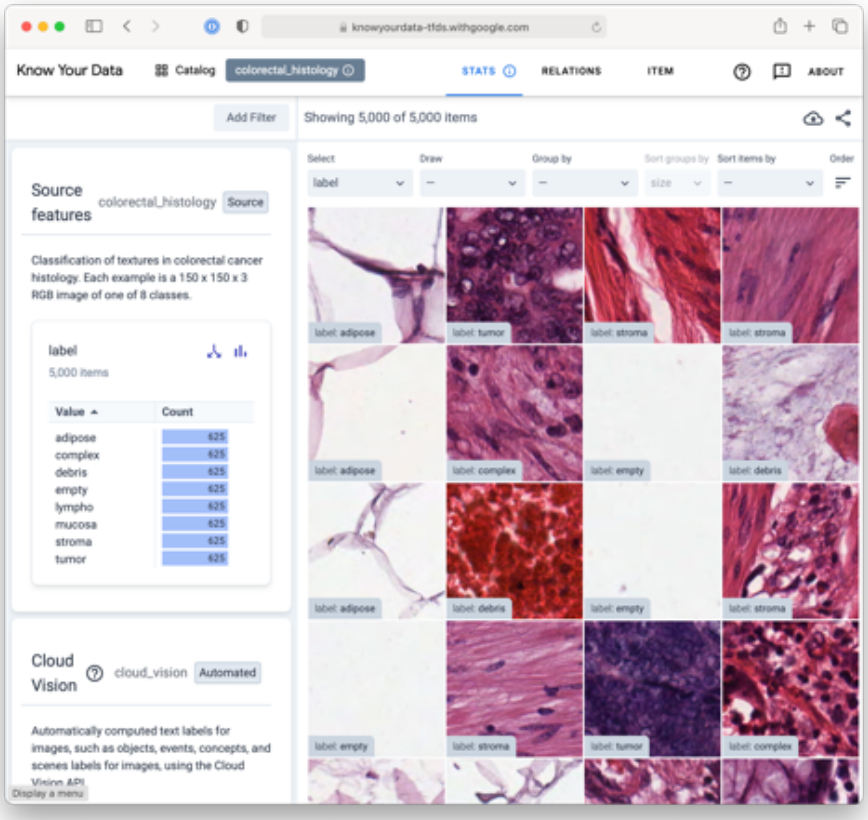
The data in this challenge contains a total of 400 whole-slide images (WSIs)




Label: 0
Label: 0
Label: 0
Label: 1
Label: 0
Label: 1
Label: 1
Label: 1
Label: 1
Label: 0

Colorectal histology dataset:

Classification of textures in colorectal cancer histology.
Each example is a 150 x 150 x 3 RGB image of one of 8 classes



https://knowyourdata-tfds.withgoogle.com/#tab=STATS&dataset=colorectal_histology

(The dataset for the final project)

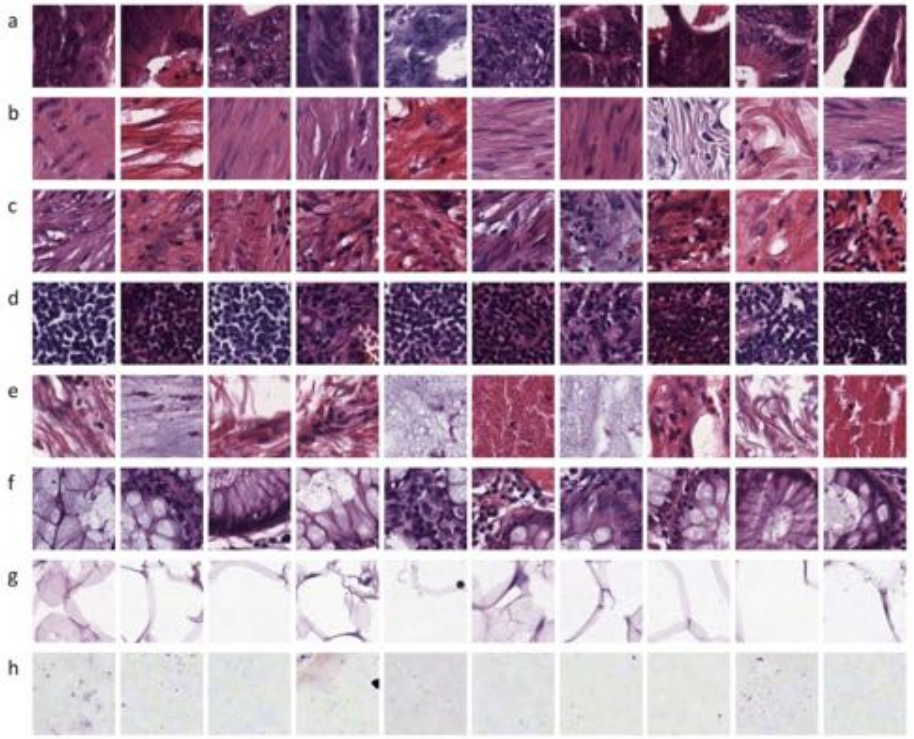


Figure 1. Representative images from our dataset. Here, the first 10 images of every tissue class in our dataset are shown. They represent the wide variation of illumination, stain intensity and tissue textures present in routine histopathological images. Images were extracted from 10 independent samples of colorectal cancer (CRC) primary tumours. (**a**) tumour epithelium, (**b**) simple stroma, (**c**) complex stroma (stroma that contains single tumour cells and/or single immune cells), (**d**) immune cell conglomerates, (**e**) debris and mucus, (**f**) mucosal glands, (**g**) adipose tissue, (**h**) background.

SCIENTIFIC REPORTS

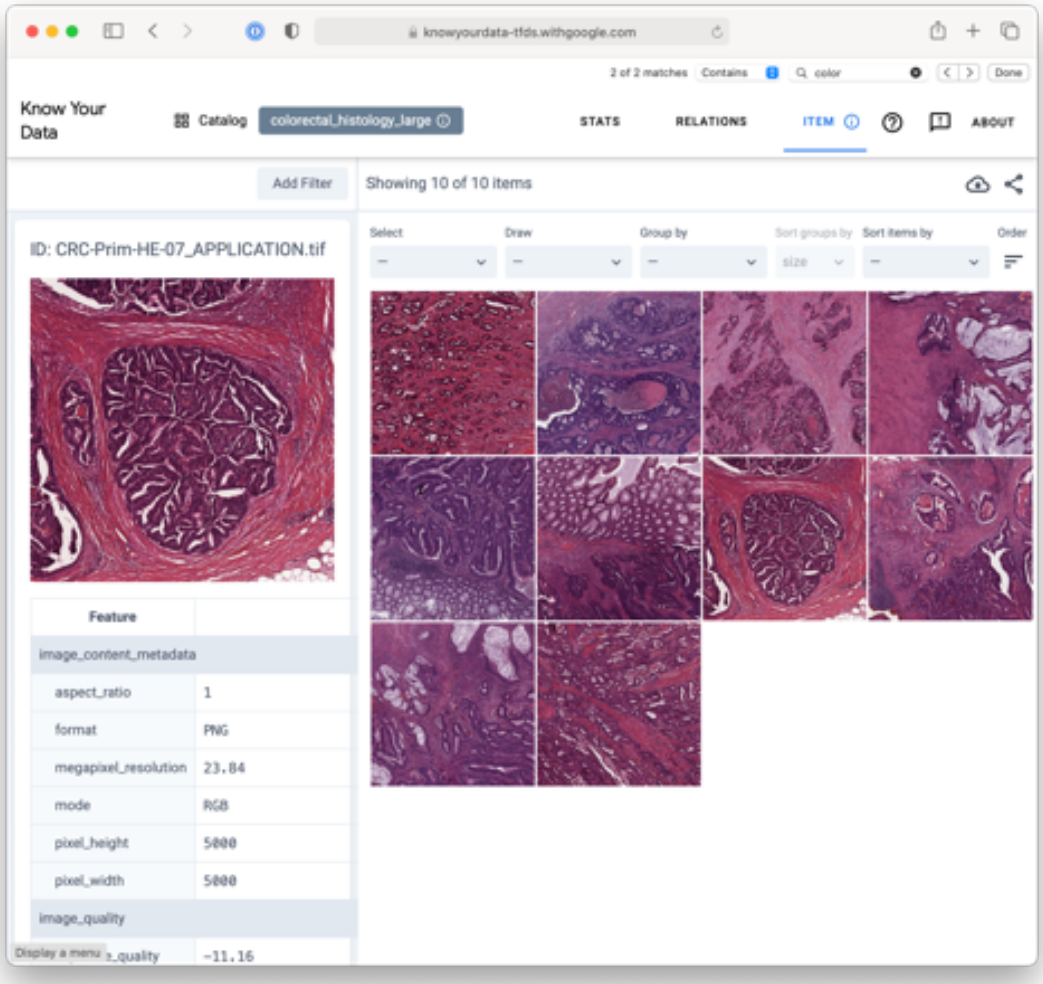
OPEN

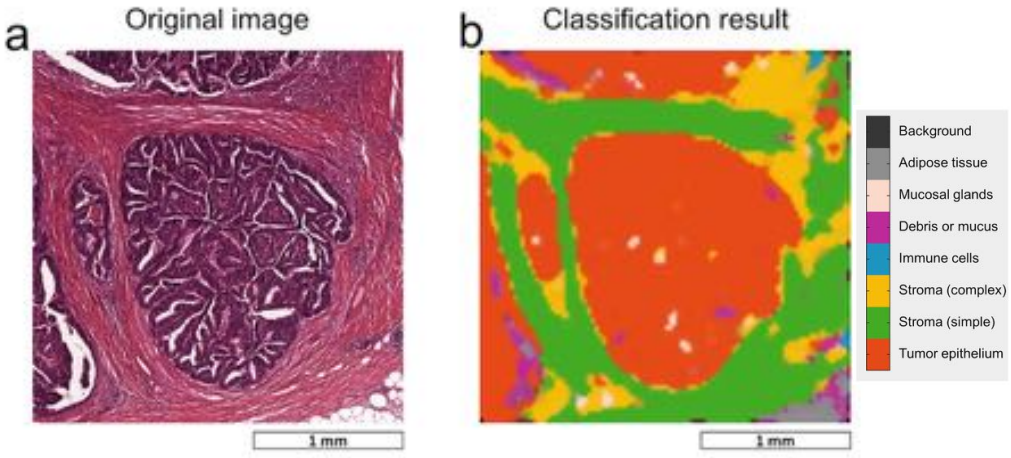
Multi-class texture analysis in colorectal cancer histology

Jakob Nikolas Kather[1,2], Cleo-Aron Weis[1], Francesco Bianconi[3], Susanne M. Melchers[4], Lothar R. Schad[2], Timo Gaiser[1], Alexander Marx[1] & Frank Gerrit Zöllner[2]

Received: 02 March 2016
Accepted: 25 May 2016
Published: 16 June 2016

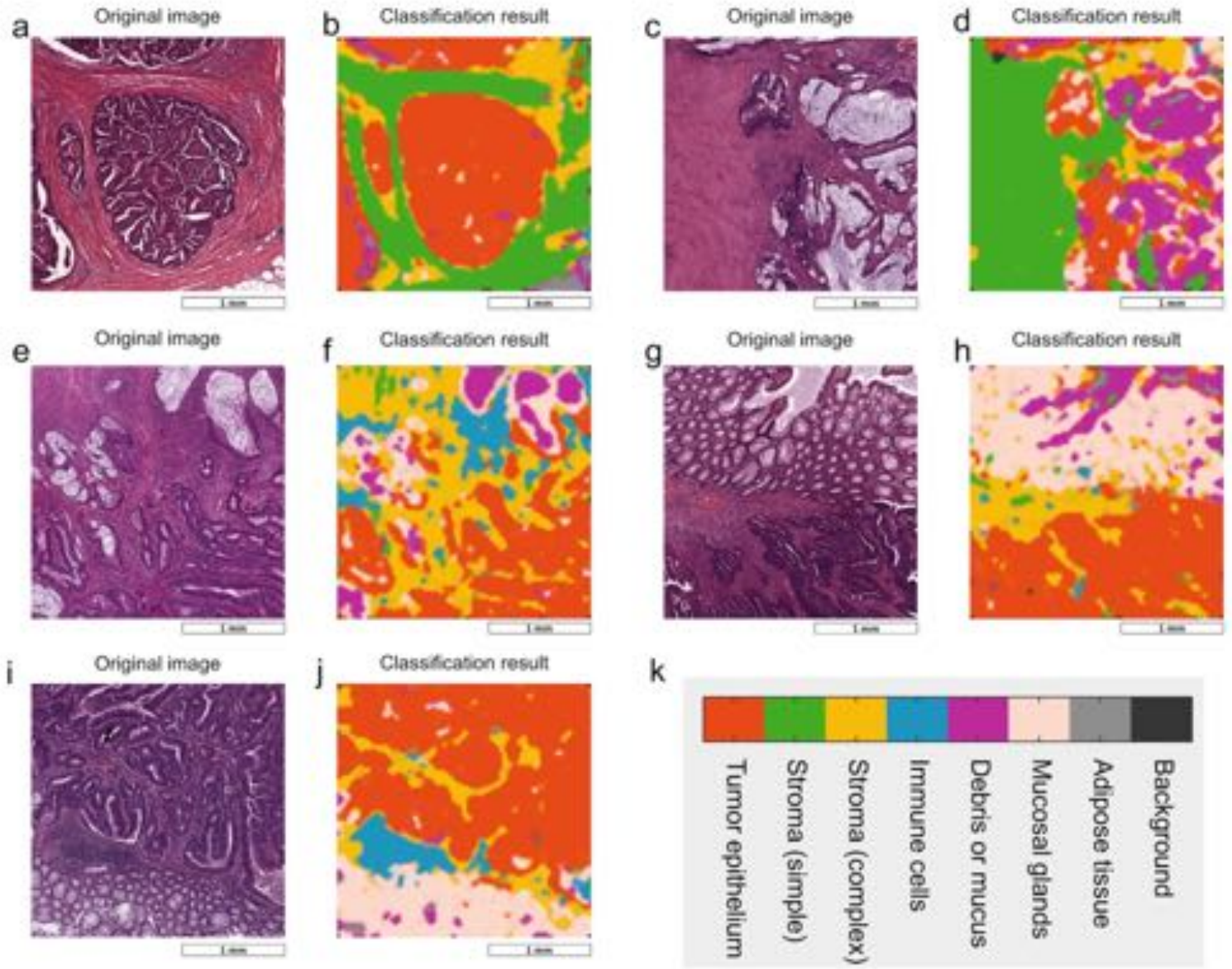
Automatic recognition of different tissue types in histological images is an essential part in the digital pathology toolbox. Texture analysis is commonly used to address this problem; mainly in the context of estimating the tumour/stroma ratio on histological samples. However, although histological images typically contain more than two tissue types, only few studies have addressed the multi-class problem. For colorectal cancer, one of the most prevalent tumour types, there are in fact no published results on multiclass texture separation. In this paper we present a new dataset of 5,000 histological images of human colorectal cancer including eight different types of tissue. We used this set to assess the classification performance of a wide range of texture descriptors and classifiers. As a result, we found an optimal classification strategy that markedly outperformed traditional methods, improving the state of the art for tumour-stroma separation from 96.9% to 98.6% accuracy and setting a new standard for multiclass tissue separation (87.4% accuracy for eight classes). We make our dataset of histological images publicly available under a Creative Commons license and encourage other researchers to use it as a benchmark for their studies.





(5000x5000 pixels image)

https://knowyourdata-tfds.withgoogle.com/#tab=STATS&dataset=colorectal_histology_large


a
Original image
b
Classification result
c
Original image
d
Classification result
e
Original image
f
Classification result
g
Original image
h
Classification result
i
Original image
j
Classification result
k
Tumor epithelium
Stroma (simple)
Stroma (complex)
Immune cells
Debris or mucus
Mucosal glands
Adipose tissue
Background

Project goals

- Train a ConvNet classifier
  (either design your own or use a pretrained network + transfer-learning/fine-tuning)

- Use train/validation protocols
  (e.g., 90% data for training, 10% for validation)

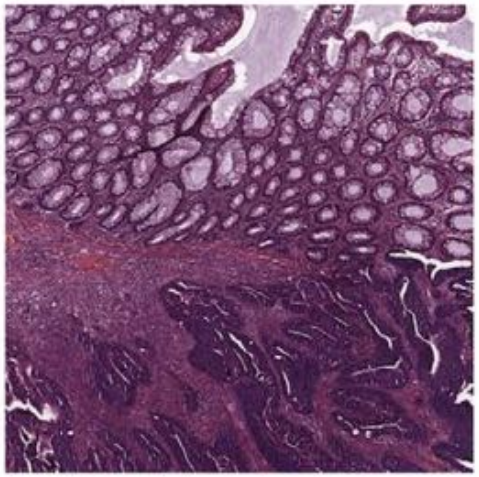
- Try several schemes, and compare your results in a table
  (e.g., different architectures, optimizers, preprocessing & data-augmentation, …)

Project goals

- Using your best model:
  - Plot its confusion matrix.
  - Use its *one-before-last* layer (its “features vector” layer), and **PCA** or ***t*-SNE** to visualize the labelled-data distribution in **2-D**.
  - Pick each of the large 5000x5000 pixels images, slice them into smaller 150x150 patches, and use your model to classify each patch. Visualize the classification results (on-top or side-by-side of the large 5000x5000 images) in two ways:
    - Color each patch (in the large image) using the predicted class label (8 colors for 8 classes).
    - Color each patch using the probability value of the “tumor ep.” class (a ‘heatmap’ result).

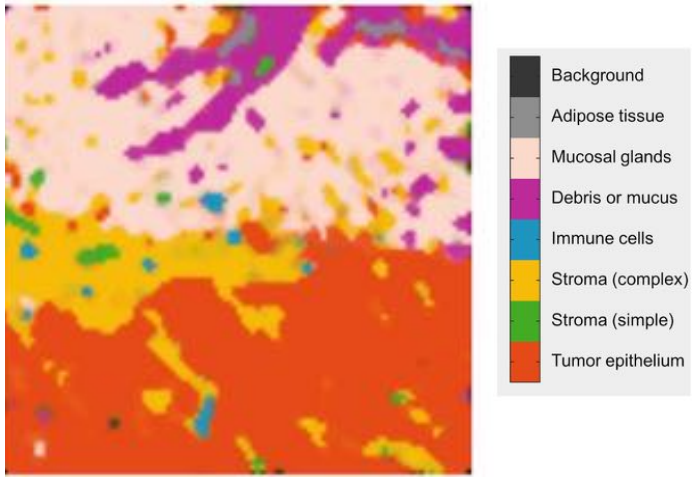
Visualization examples:

Original image

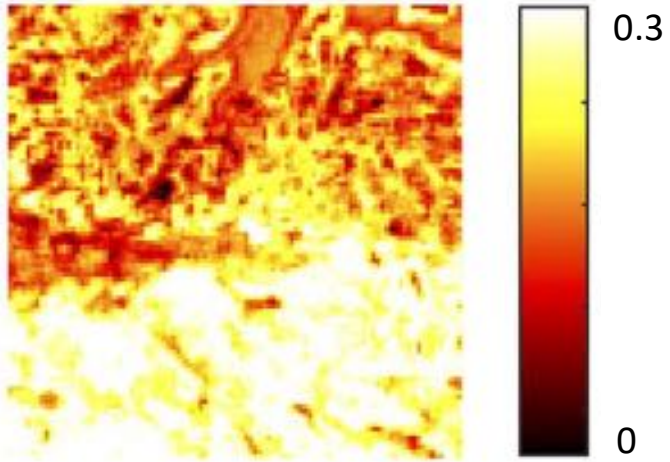


5000x5000 pixels

Classification result



Tumor epithelium



Hint: you can use some overlap (e.g., 50%) when dividing the large image into patches, to get a nicer results.

הנחיות הגשה

- הגשה לבד או בזוגות.
- מועד אחרון להגשת הפרוייקט: **3 יולי 2021**
  - יש להגיש קוד **ipynb מסודר**, כולל הערות, וכל השלבים השונים.
  - יש להוסיף markup-cells כדי להסביר בכל שלב מה ביצעתם (עברית/אנגלית)
  - יש להציג גרפים ותוצאות מסודרות (בגודל סביר, כולל כיתוב לצירים השונים וכו׳) לכל אורך הפרוייקט.
  - לדאוג **לנקות** מהקובץ הסופי את כל הקוד/תוצאות הלא רלוונטים.

- בנוסף, יש להגיש **פוסטר** (בפורמט PPT), לפי ההנחיות באתר:

https://www.hit.ac.il/sciences/projects

קוד אתי

- ניתן להעזר בחברי הקורס, ובחומרים ברשת כמובן, **כל עוד העבודה המוגשת בסוף מבוססת על עבודה עצמאית ומקורית של חברי הקבוצה.**
- יש לצרף קישורים לחומרים ברשת שנעזרתם בהם במהלך העבודה.

Add markup sections to your final *ipynb*:

Part1

In this part, we are going to modify the original video by adding the at the end of half a minute of original video after being rotated.

We rotated the video to test the function of saving the path.

```
def translate(dx,dy):
    M = np.eye(3)
    M[:,2] = [dx,dy,1]
    return M


def rotate(angle):
    M = np.eye(3)
    sin, cos = np.sin(angle), np.cos(angle)
    M[:2,:2] = [[cos,-sin],[sin,cos]]
    return M
```

The function for rotate the image

```
def RotateImage (frame, angle):
    h,w,_ = frame.shape

    T1 = translate(w/2, h/2)
    T2 = translate(-w/2, -h/2)

    angle =angle/100*2*np.pi
    M = rotate(angle)
    P = (T1.dot(M)).dot(T2)
    frame_warp = cv2.warpPerspective(frame.copy(), P, (w,h),borderValue=(255,255,255))

    return frame_warp
```

Describe your functions:

handle_canny_and_refresh_good_features(apply_canny, threshold, F0, feature_params)

Used to apply Canny algo (optional) and find corners using Shi-Tomasi.

apply_canny - Whether to apply Canny alg.
threshold - Canny's threshold parameter.
F0 - The frame to modify.
feature_params - Parameters for the Shi-Tomasi alg.

```
def handle_canny_and_refresh_good_features(apply_canny, threshold, F0, feature_params):
    # Use Canny to find edges.
    if apply_canny:
       F0_gray = cv2.Canny(F0,threshold,0.1)

    # Make sure to return the frame gray.
    else:
        F0_gray = cv2.cvtColor(F0, cv2.COLOR_BGR2GRAY)
    pts0 = cv2.goodFeaturesToTrack(F0_gray, mask = None, **feature_params)

    # Squeeze to pairs of coords.
    if pts0 is not None and len(pts0) > 0:
        pts0 = pts0.squeeze() # (n,1,2) -> (n,2)

    return F0_gray, pts0
```

overlay_image_alpha(img, img_overlay, pos, alpha_mask)

An aux function to display image on top of an image.
Taken from here: https://stackoverflow.com/a/45118011/3450476
And modified it a bit to fit the project needs (removed alpha-channel manipulation so the images won't be displayed semi-transparent)

Add comments:

```
while True:
    ret, frame = cap.read()
    if ret==True:

        # check if we are currently underwater using hue values (underwater mainly blue)
        img_hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
        hue,saturation,values = cv2.split(img_hsv)
        (values,counts) = np.unique(hue,return_counts=True)
        ind=np.argmax(counts)
        output_frame = frame.copy()

        if 90 <= values[ind] <= 110:
            # we are underwater

            # mask lane bounds - clip to / swimming range \
            bounds_mask = np.zeros((h, w), dtype=np.uint8)
            cv2.fillPoly(bounds_mask, points, (255))
            clipped = cv2.bitwise_and(frame,frame,mask = bounds_mask)

            # mask lane by hsl - dark blue
            hsv_cropped = cv2.cvtColor(clipped, cv2.COLOR_BGR2HSV)
            color_mask = cv2.inRange(hsv_cropped, lower, upper)
            imask = color_mask>0
            masked_frame = np.zeros_like(frame, np.uint8)
            masked_frame[imask] = frame[imask]

            # convert to grayscale
            gray = cv2.cvtColor(masked_frame, cv2.COLOR_BGR2GRAY)

            # convert to black and white using thershold
            _,binary = cv2.threshold(gray,30,255,cv2.THRESH_BINARY)

            # erode noise
            eroded = cv2.erode(binary,kernel,iterations = 1)
```

Efficient Convolutional Neural Networks for Cloud Detection in Satellite Images

John Merriman Sholar

CS 231N: Convolutional Neural Networks for Visual Recognition

Problem

Overview

- Goal: identify clouds in satellite images
- Motivation: removing ("masking out") clouds important for downstream analysis for satellite imagery
- Task: image segmentation with classes including dense clouds, cirrus clouds, shadows, water, snow, and land.
- Evaluation: per-pixel classification accuracy, CE loss, qualitative analysis of output, classification speed.

Subtasks

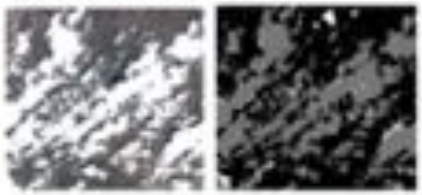
We consider several permutations of the problem definition.

- Given information: image segmentation using all 13 Sentinel-2 bands, or only using RGB.
- Output classes: we consider 2-class, 3-class, and 4-class classification problems, defined as follows.

4 Classes	LAND, CLOUD
3 Classes	LAND, CIRRUS, CLOUD
4 Classes	LAND, CIRRUS, SHADOW, WATER, SNOW, CLOUD

Data

- 60 Sentinel-2 satellite images = 10,980 by 10,980 px. ea.
- Segmented into = 129,000 tiles of size 224 by 224 px.
- Sentinel-2 images have 13 spectral bands, rather than 3 (RGB). RGB frequencies range from .46 μm to .490 μm, while 12 bands range from 2.190 μm to .443 μm, with more information (e.g. infrared, short-wave infrared).
- Satellite imagery is very different from "traditional" imagery (top -down, spatial invariance), lack of central focus, scale), making transfer learning or pretrained classifiers less useful.
- Sentinel-2 ships with a proprietary cloud mask (below), which provides noisy but useful training data.



Baseline Methods

Previous Approaches

- Previous approaches use decision trees and linear classifiers, emphasizing speed.

Pixel-Level Decision Tree

- A recreation of previous research by Hollstein et al.
- Extremely fast, but structures for problems more advanced than the binary problem.
- Useful for a form of *transfer learning*: no ground truth training data for CNNs by applying decision trees, using the Sentinel-2 cloud mask as an extremely noisy label.

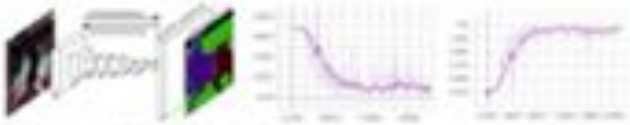
Pixel-Level MLP

- An expansion on the pixel level classification using decision trees. Ultimately no more accurate than decision trees, likely due to the same inability to take advantage of spatial covariance, with significantly increased inference time.

Methods

Fully Convolutional Networks Networks (Long et al.)

- A modification of the Alexnet (Krizhevsky et al) architecture which removes the fully connected layers in favor of fully convolutional layers, and adds a singular deconvolution. (Below are architecture, loss, and accuracy)



Deconvolutional Neural Networks (Noh et al.)

- Parallel convolutional and deconvolutional structure.
- Convolutional first half initialized using ILSVRC pretrained VGG-16

Results

Pixel-Level Decision Trees

Fully Convolutional Networks Networks

Deconvolutional Neural Networks (Noh et al.)

- We are unsuccessful in training deconvolutional neural networks. This is likely a result of several factors. First, for the RGB-only task, the VGG-16 ConvNet is pretrained on ILSVRC data, which does not parallel Sentinel-2 data. For the 13-band task, training may simply require more computational resources.

Inference Speed and Accuracy

	Inference Accuracy (Pi Score)	Inference Speed
Decision Trees	0.655	34 million pixels/second
Fully Conv. Networks	0.822	1 million pixels/second
Deconv. Networks	0.930	40,000 pixels/second

Discussion and Future Work

- Pixel level decision trees achieve visually satisfactory results and high numerical accuracy for the binary problem, but fail on more difficult classes, such as shadows.
- Fully convolutional neural networks produce coarse output, but could be refined by using a series of deconvolutional layers – likely the most viable next step.
- Deconvolutional networks have strong potential for this task, but can't be pretrained on ILSVRC data, which is significantly different from satellite imagery. Future work might include training a DNN end-to-end on Sentinel-2 data.
- Satellite images are extremely large, processing time is key. Future work might include application of SqueezeNet (Iandola et al.) architectures to Sentinel-2.

Photoshop 2.0: Generative Adversarial Networks for Photo Editing

Homero Roman, Michelle Zhang, Brandon Yang
CS231N Spring 2017

Background

- Generative Adversarial Networks (GANs) generate images from a min-max game between generator and discriminator
- Recent research has focused on improving GAN architectures for training and image quality, such as WGAN
- We focus on building an architecture to directly train autoencoders for GANs to better understand the learnt latent space representation of these networks
- Previous research into conditional GANs conditions GAN training on labeled data for more specific image generation.
- Preliminary results in the DCGAN paper illustrate the potential of latent space traversal and selective dropout for tuning image generation and semantic understanding

Dataset

- The **celebA** dataset contains over 200,000 face images of various celebrities
- Labeled with 40 different binary attributes for facial features, including hair color, gender, relative age, and whether or not the person is smiling

Models

- To train a GAN auto-encoder we first trained the DCGAN model for 25 epochs and used the trained generator weights as our encoder
- We experimented with three approaches to encoder networks with the L2 similarity metric
  - Fully connected encoder model baseline
  - Deep convolutional encoder
  - Transfer learning encoder with weights from the trained discriminator from the DCGAN model
- We then experimented with different loss functions to produce more realistic encodings: L1, L2, and SSIM

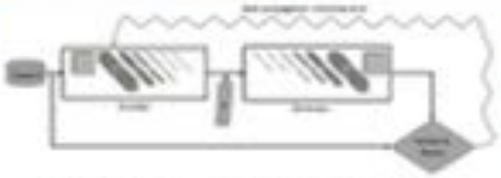


Figure 1: GAN Autoencoder and Generator network overview

Figure 2: Encoder network architectures

Qualitative Results

Training the extended transfer learning encoder :

Using GAN autoencoder to create smiling image:

Evaluation

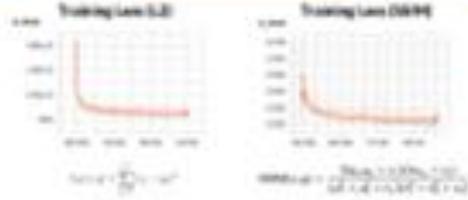


Figure 3: Training loss curves for transfer learning encoder

	L2 Loss	L1 Loss	SSIM Loss
FC Encoder	468.219	-	-
DeepConv Encoder	525.660	-	-
Transfer Learning Encoder	441.906	1990.321	0.293
Extended Transfer Learning Encoder	701.271	-	-

Figure 4: Loss results

Conclusions and Future Work

- GAN autoencoders can be effectively created by transfer learning from the discriminator network
- Manipulating encoded images in the latent vector space with simple vector operations to generate images
- Our model seems limited by inherent biases in the generator and dataset (female faces), which can perhaps be addressed with different generator approaches
- In the future, we want to explore better quantitative metrics for generated image quality and possible end to end architectures for image processing
- Explore different operations in the latent space and selective dropout and effects on generated images

References

1. A. Larsen, S. Sønderby, and O. Winther. Autoencoding beyond pixels using a learned similarity metric. CoRR, abs/1512.09300, 2015.
2. Z. Liu, P. Luo, X. Wang, and X. Tang. Deep learning face attributes in the wild. In Proceedings of International Conference on Computer Vision (ICCV), Dec. 2015.
3. A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. CoRR, abs/1511.06434, 2015.
4. T. White. Sampling generative networks.

Recurrent Neural Network for Politician's Text Generation

Tomer Volinstein
21319 Deep Learning, Spring 2020

Abstract

- Politicians often write posts (short texts) to express their agenda and promote themselves in modern social media platforms such as Facebook and Twitter.
- By scraping thousands of Facebook posts from several key politicians in Israel, and using their texts as a corpus, I will attempt to train a neural network that takes sequences of texts (e.g. sentences) and predicts the next word after each sequence, resulting in a model which can generate full sentences by learning from a corpus of politicians' posts.
- Previous work showed good results under domain-specific corpus such as Books from Project Gutenberg, Tweets of politicians and News headlines.

Model

- The chosen model for the project was based on LSTM network (variation of RNN).
- Each model I experimented with contained an Embedding layer to convert words into vectors of fixed dimension, and an Activation FC layer (softmax).
- I tried different models containing Single/Dual/Bidirectional LSTM layer of 100/150/200 units.
- After the hidden layers I tried to apply a Dropout of 10-25% to prevent overfitting.

Conclusions

I learned that generating "goofy" texts is a common result of text-generator network implementation using LSTM, and modern solutions such as **Transformers** (GPT-2, BERT) are showing more realistic results.

Evaluation

- The model was compiled to calculate categorical cross-entropy as loss function with Adam optimizer and learning rate of 0.0001.
- The model converged after 250 epochs, with 78% accuracy.

Dataset

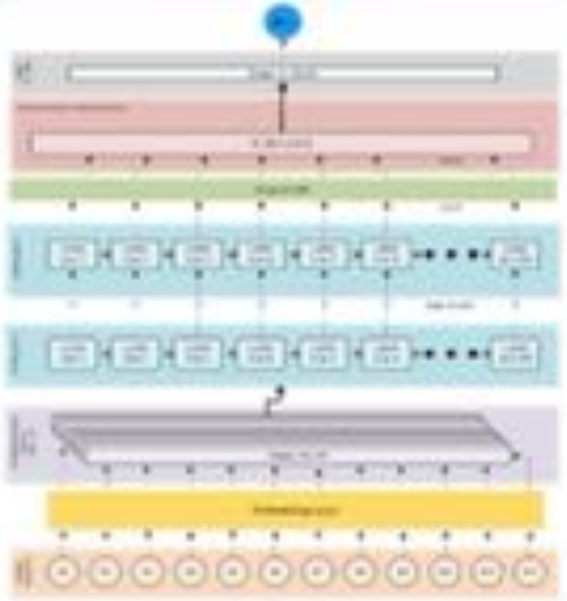
- The Dataset consist of **3,012 Facebook posts** from 7 Israeli politicians which I scraped using a web-crawler I wrote.
- After preprocessing, the training data had **93,357 sequences of texts**, each had a label (predicting next word).
- There are **24,248 unique words/tokens** which indicates the number of classes/labels for classification.

Qualitative Results

Real Facebook posts (example for training data)

Model's result before training

Model's result after training



Chosen model for training (two LSTM layers)

References:
1. Shivam Bansal, Beginners Guide to Text Generation using LSTMs, Kaggle 2018.
2. Jeremy Chow, Simple Text Generation (How to teach a machine to "speak"), Medium article 2019.
3. Zhang Xiv, Neural Text Generation: A Practical Guide, Stanford CS 2018.

Neural Style Transfer Project

סטודנטית שנה"ל תשף סמסטר ב': אנה אבא

למידה עמוקה

ד"ר יונתן רובין

מכון טכנולוגי חולון
Holon Institute of Technology

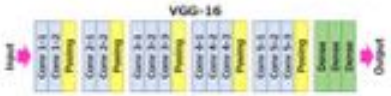
תיאור כללי על הפרויקט:

פרויקט זה נלקח מתחום של רשתות נוירונים עמוקות - **Deep Neural Networks**.

הוא לוקח רשת נוירונים קיימת בשם VGG-16 , שמקבלת קלט של תמונה וכוללה לזהות ולסווג את מה שרואים בתמונה.
רשת זו בעלת שכבות רבות , כאשר כל שכבה מחזיקה נתונים ומידע מסוים שמאפיינים אותה.

האלגוריתם בפרויקט מקבל 2 תמונות , תמונה א' שמייצגת תוכן מסוים ותמונה ב' שמייצגת סטייל מסוים.

ע"י שימוש בנתונים של שכבות מסוימות ב VGG-16, ניתן ליצור תמונה חדשה שמייצגת את התוכן של תמונה א' עם הסטייל של תמונה ב'.



שלבי המיתון:

שלב ראשון – מורידים את רשת ה VGG-16 באמצעות keras. טוענים את התמונות הרצויות ויוצרים תמונה שלישית שהיא ריבוע אפור שמייצג "רעש" - תמונת העדכון. הרשת מצפה לקבל input מסוים, לכן משנים את גודל התמונות ל (1,224,224,3) ושולחים אותן לפונקציית ה - vgg16.preprocess_input(image) שמעבדת אותן וכך מתאימה אותן ל input שהרשת מצפה לקבל.

שלב שני – מגדירים את פונקציית ה – content loss . הרשת VGG-16 מורכבת ממספר שכבות. כל שכבה נותנת פלט בהתאם לתמונה שהוכנסה לרשת. לכן, בוחרים באחת השכבות האחרונות, מחשבים את הפלט שיצרה תמונת התוכן המקורית ואת הפלט שיצרה תמונת העדכון. נחשב את המרחק בין שתי שכבות הפלט ונחזיר את התוצאה. ככל שהמרחק קטן יותר, כך תוכן התמונות דומה יותר, לכן נרצה להקטין את המרחק הזה ככל האפשרי.

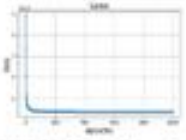
שלב שלישי – מגדירים את פונקציית ה – style loss . גם כאן בוחרים שכבה מסוימת ומחשבים את הפלט של תמונת הסטייל ותמונת העדכון.
כל שכבת פלט – הופכים למטריצת גרם. כל שכבה מורכבת מפילטרים – כל פילטר הוא מטריצה. מטריצת גרם מייצגת את הקורלציה בין כל הפילטרים באותה השכבה.
במילים פשוטות, היא מייצגת את הסטייל של התמונה, ללא קשר לתוכן או למיקום הפיקסלים. מחשבים את המרחק בין שתי מטריצות הגרם ומחזירים את התשובה. ככל שהמרחק קטן יותר, כך התמונות דומות יותר בסטייל שלהן. לכן נרצה להקטין את המרחק. את השלב הזה נבצע על חמש שכבות לאורך הרשת.

שלב רביעי – מחשבים את ה loss הכללי:
loss_total = α * content_loss + β * style_loss
כאשר α = 1 ו β = α / 1e-3.
נגדיר את פונקציית הגרדיאנטים שמקבלת את ה - loss_total , גוזרת אותו לפי תמונת העדכון ומחזירה את הגרדיאנטים שהתקבלו.

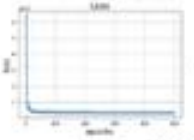
שלב חמישי – מריצים את הפונקציה **fmin_l_bfgs_b** - n פעמים.
פונקציה זו מקבלת את ה- loss והגרדיאנטים (שכבר הגדרנו) בתוך פונקציה , ובאמצעות אלגוריתם שנקרא **L-BFGS** ממזערת את ה - loss ומשנה את תמונת העדכון בהתאם.

תוצאות:

רגפה א' - אייפל



רגפה ב' - כלב



ה- Loss מתחיל להתכנס כבר בהתחלה.
אחרי כמה מאות איטרציות, התוצאה לא משתנה משמעותית.

מסקנות והמלצות להמשך:

ניתן להשתמש ברשתות נוירונים למטרות שונות, בין היתר גם ליצירת אומנויות.

ניתן לקבל תוצאות שונות על ידי בחירה שונה של פרמטרים כמו לדוגמה לבחור בשכבות אחרות ב style loss , רק בשכבות העמוקות ברשת או להיפך רק בנמוכות.

מקורות עזר:

1. A Neural Algorithm of Artistic Style – https://arxiv.org/pdf/1508.06576.pdf
2. Neural Style Transfer on Real Time Video – https://towardsdatascience.com/neural-style-transfer-on-real-time-video-with-full-implementable-code-ac2dbc0e9822

Thank you, and...

GOOD LUCK !