A simple (and limited) introduction

Simone Gennai
(e Francesco Brivio)

Disclaimer

- This is not a lesson on AI
- This is not an exhaustive tutorial on neural network
- This is not the description of the SOA of ML models
- This is just an introduction to basic concepts and information needed to make you start on the project you have been assigned to

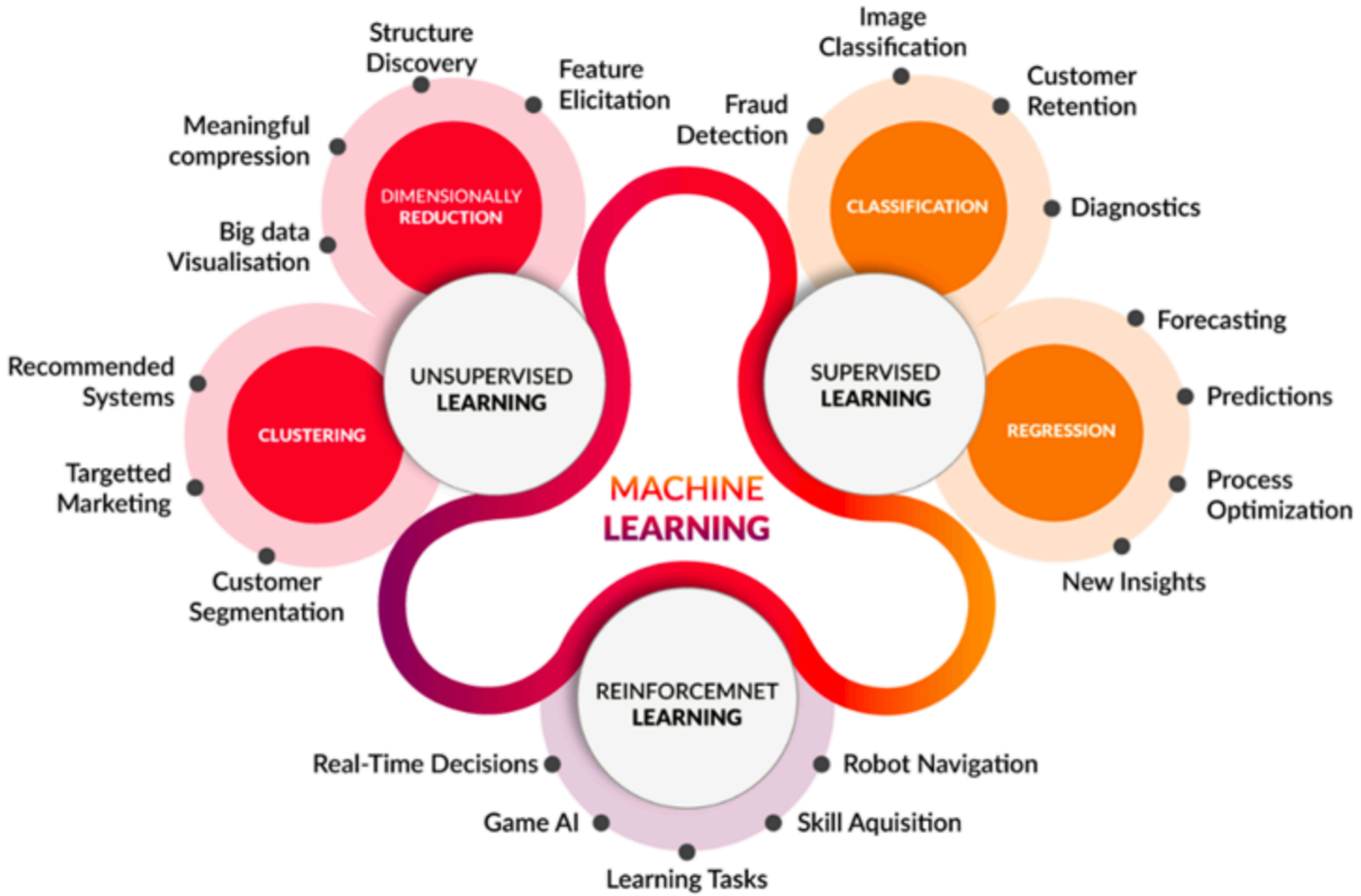
Simone.Gennai@cern.ch

What ML can do for us ?

- Short answer: a lot!
  - Many ML models freely available on GitHub repositories, tutorials available, easy to use python API, etc. etc. can help in solving complicated problems at a reasonably fast inference time.
- Longer answer: it depends!
  - nothing comes for free … most consuming part of the application of ML is a proper training of your models.
    - That would mean availability of tons of good data …
    - … and lots of computing power
    - Lately pre-trained models are becoming more and more popular and can simplify a bit the final user life
  - Pros and cons has to be evaluated carefully for each problem
  - Sometimes an easier approach can give the same results of very complex ML models with much fewer drawbacks!

Simone.Gennai@cern.ch

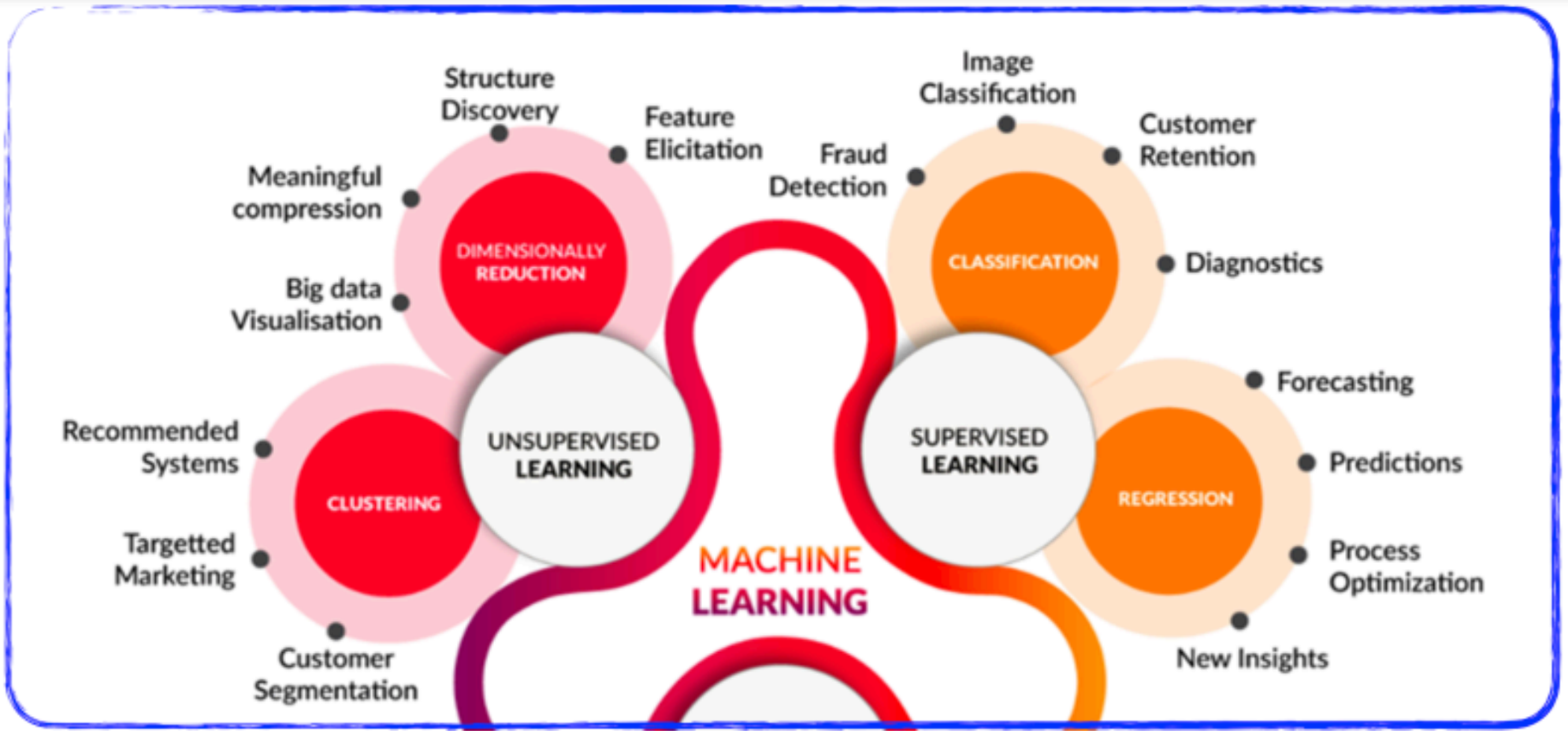
Three most popular ML categories



https://course.productize.ml/machine-learning/why-ml-and-why-now

Simone.Gennai@cern.ch

Three most popular ML categories

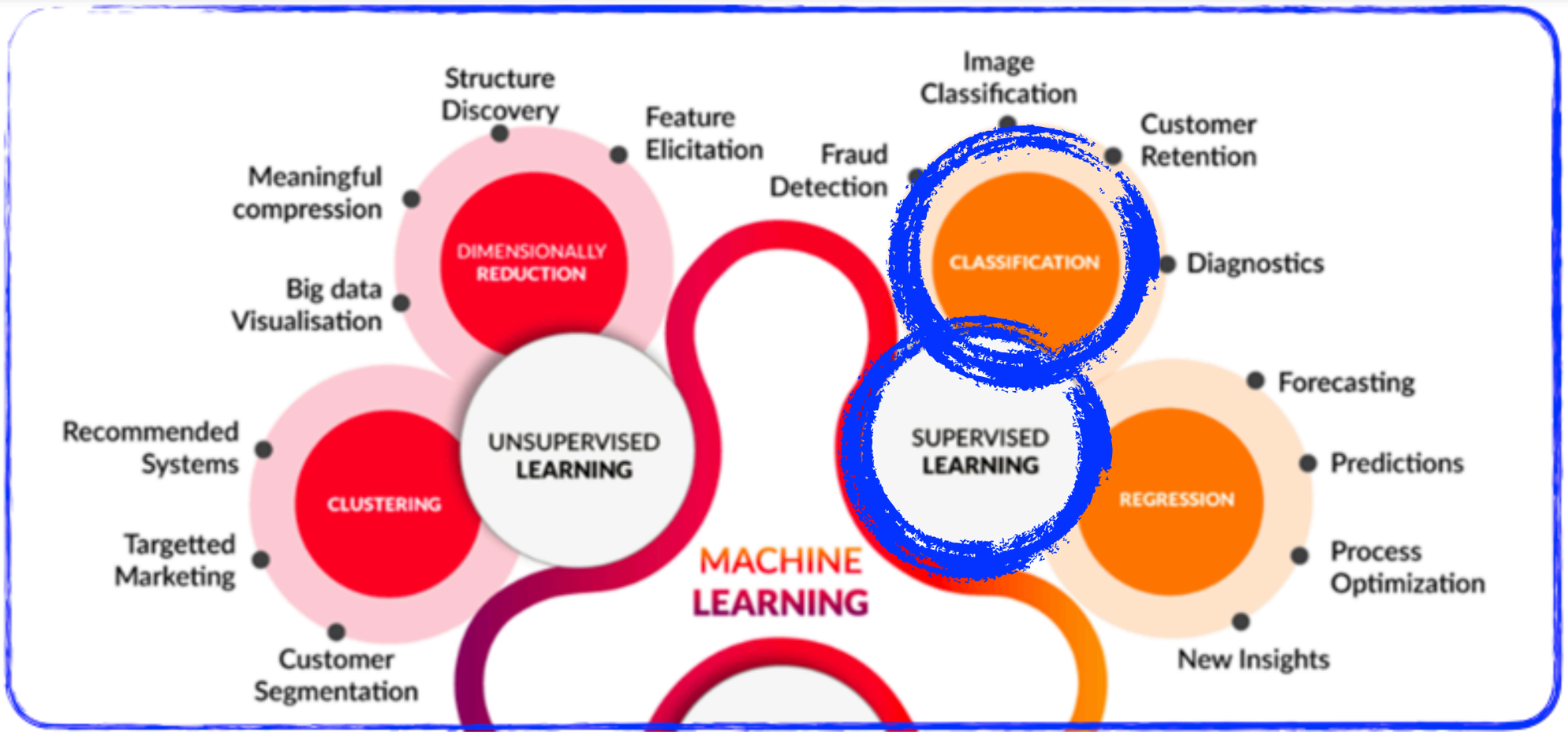


Most used models in particle physics and medicine are either supervised or unsupervised models.

https://course.productize.ml/machine-learning/why-ml-and-why-now

Simone.Gennai@cern.ch

Three most popular ML categories



For the purposes of this hackathon we will concentrate on supervised learning.

https://course.productize.ml/machine-learning/why-ml-and-why-now

Simone.Gennai@cern.ch

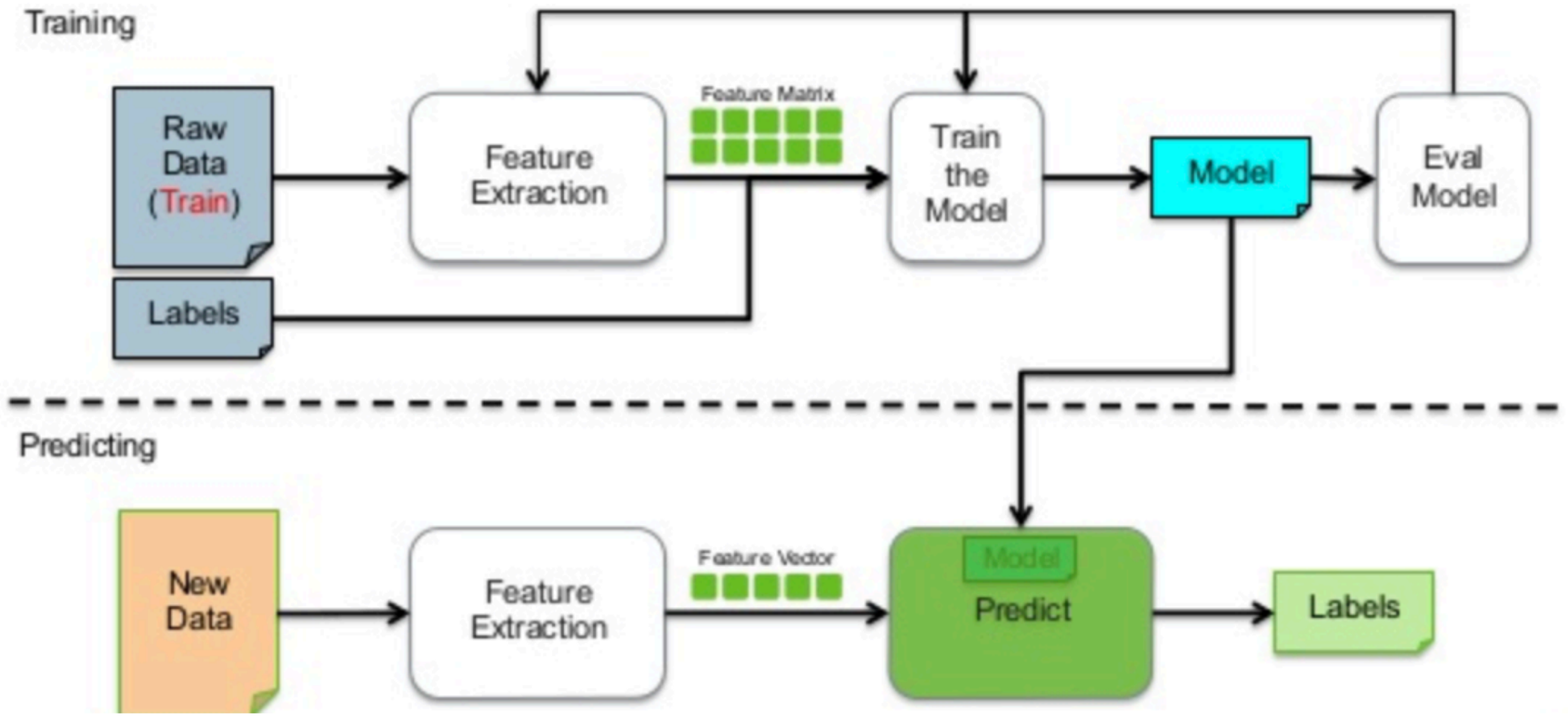
What does supervised learning really mean?

- Basically you use labelled data to “teach” your model towards what you want it to learn.
  - e.g. discriminating between dogs and cat pictures, reconstrucing particle trajectories, identify cancer cells in an TC scan.
- This also means that you need these labelled data for the model to work properly
  - In many cases this comes for free , e.g. in particle physics we can use Monte Carlo simulation to label processes.
  - In other cases it is more difficult, like categorised images on internet
  - In extreme cases it becomes very complicated and only small samples are avialble which makes the training hard

Simone.Gennai@cern.ch

What does the training do?

- A NN Model can be seen as a fitting function with a large number of parameters (weights and biases).
  - The training procedure is the way the best parameters that matches the input of your network to the desired output are found



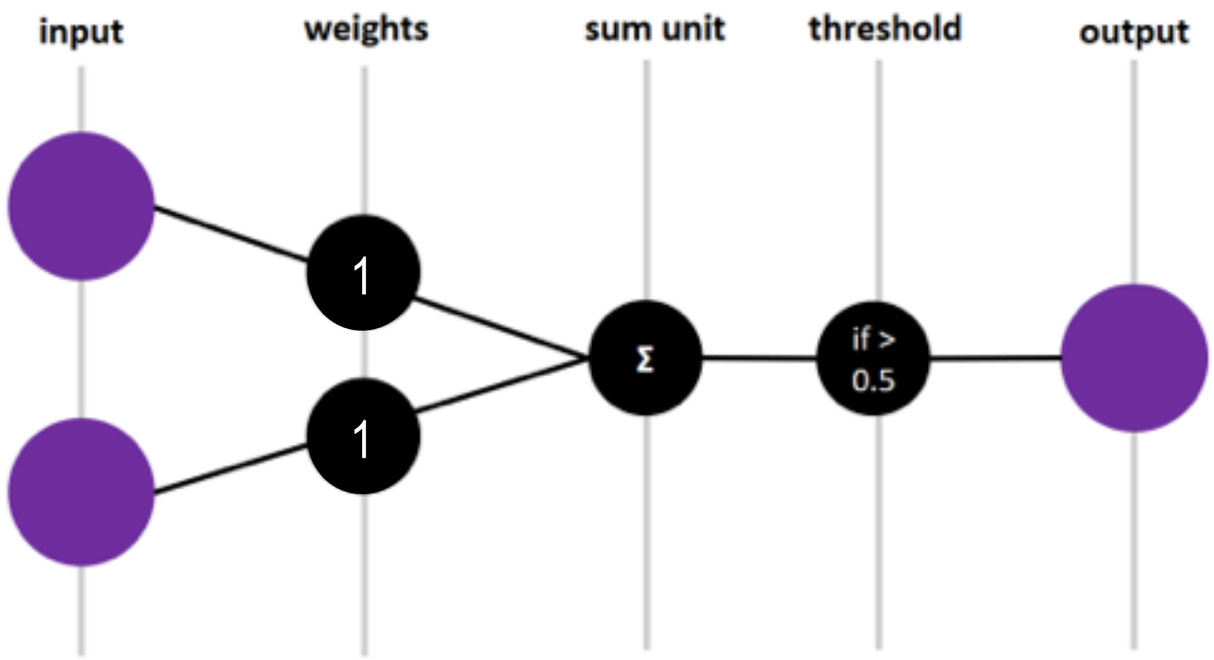
Simone.Gennai@cern.ch

Starting from the basis

Simone.Gennai@cern.ch

Single Layer Perceptron

- The building unit …
  - from https://sefiks.com/2020/01/04/a-step-by-step-perceptron-example/



Perceptron for AND Gate

Logical AND truth table

X_1	X_2	Y
0	0	0
0	1	0
1	0	0
1	1	1

In order to generalize it and make also the thresholds a parameters, the “bias" term was introduced.

Simone.Gennai@cern.ch

Single Layer Perceptron

- The building unit …
  - from https://sefiks.com/2020/01/04/a-step-by-step-perceptron-example/
- Let's simulate the training step adjusting the weights by hand using the table here as input and output
- Let's start with weights = 1
  - 1st row: $\sum w_i * x_i = 1 * 0 + 1 * 0 = 0 (< 0.5)$

X_1	X_2	Y
0	0	0
0	1	0
1	0	0
1	1	1

Simone.Gennai@cern.ch

Single Layer Perceptron

- The building unit …
  - from https://sefiks.com/2020/01/04/a-step-by-step-perceptron-example/
- Let's simulate the training step adjusting the weights by hand using the table here as input and output

X_1	X_2	Y
0	0	0
0	1	0
1	0	0
1	1	1

- Let's start with weights = 1
  - 1st row: $\sum w_i * x_i = 1 * 0 + 1 * 0 = 0 (< 0.5)$
  - 2nd row: $\sum w_i * x_i = 1 * 0 + 1 * 1 = 1 (> 0.5)$
    - Being the sum > 0, the result is "1" which does not match the table => we need to update the weights!
- Let's try weights = 0.4

Simone.Gennai@cern.ch

Single Layer Perceptron

- The building unit …
  - from https://sefiks.com/2020/01/04/a-step-by-step-perceptron-example/
- Let's simulate the training step adjusting the weights by hand using the table here as input and output
- Let's start with weights = 1
  - 1st row: $\sum w_i * x_i = 1 * 0 + 1 * 0 = 0 \, (< 0.5)$
  - 2nd row: $\sum w_i * x_i = 1 * 0 + 1 * 1 = 1 \, (> 0.5)$
    - Being the sum > 0, the result is "1" which does not match the table => we need to update the weights!

X_1	X_2	Y
0	0	0
0	1	0
1	0	0
1	1	1

- Let's try weights = 0.4
  - 2nd row: $\sum w_i * x_i = 0.4 * 0 + 0.4 * 1 = 0.4 \, (< 0.5)$ => it will work also for the other rows

Simone.Gennai@cern.ch

Single Layer Perceptron

- The building unit …
  - from https://sefiks.com/2020/01/04/a-step-by-step-perceptron-example/
- Let's simulate the training step adjusting the weights by hand using the table here as input and output
- Let's start with weights = 1
  - 1st row: $\sum w_i * x_i = 1 * 0 + 1 * 0 = 0 \, (< 0.5)$
  - 2nd row: $\sum w_i * x_i = 1 * 0 + 1 * 1 = 1 \, (> 0.5)$
    - Being the sum > 0, the result is "1" which does not match the table => we need to update the weights!

X_1	X_2	Y
0	0	0
0	1	0
1	0	0
1	1	1

- Let's try weights = 0.4
  - 2nd row: $\sum w_i * x_i = 0.4 * 0 + 0.4 * 1 = 0.4 \, (< 0.5)$ => it will work also for the other rows
    - N.B. also weights = 0.3 would work => there is usually more than one set of weights that are working on a specific sample
    - They may not work well on inputs outside the tested ones during the training
      - that's what is called over/under training!

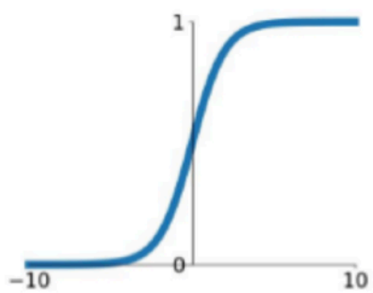
Simone.Gennai@cern.ch

How do we make it non linear?

- With the use of activation functions!
  - For more info look at: https://nnfs.io/mvp/

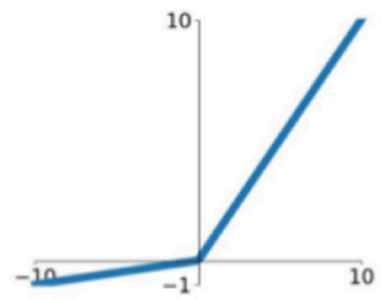
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



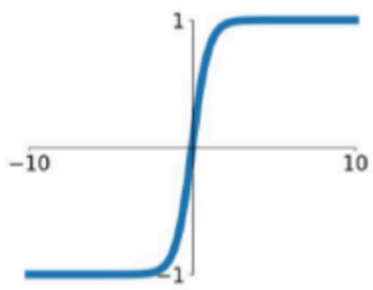
Leaky ReLU

$$\max(0.1x, x)$$



tanh

$$\tanh(x)$$

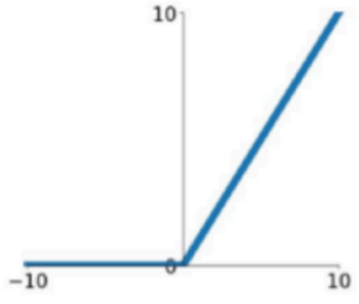


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

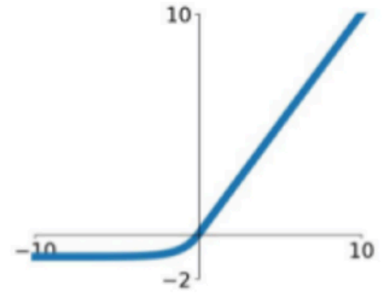
ReLU

$$\max(0, x)$$



ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

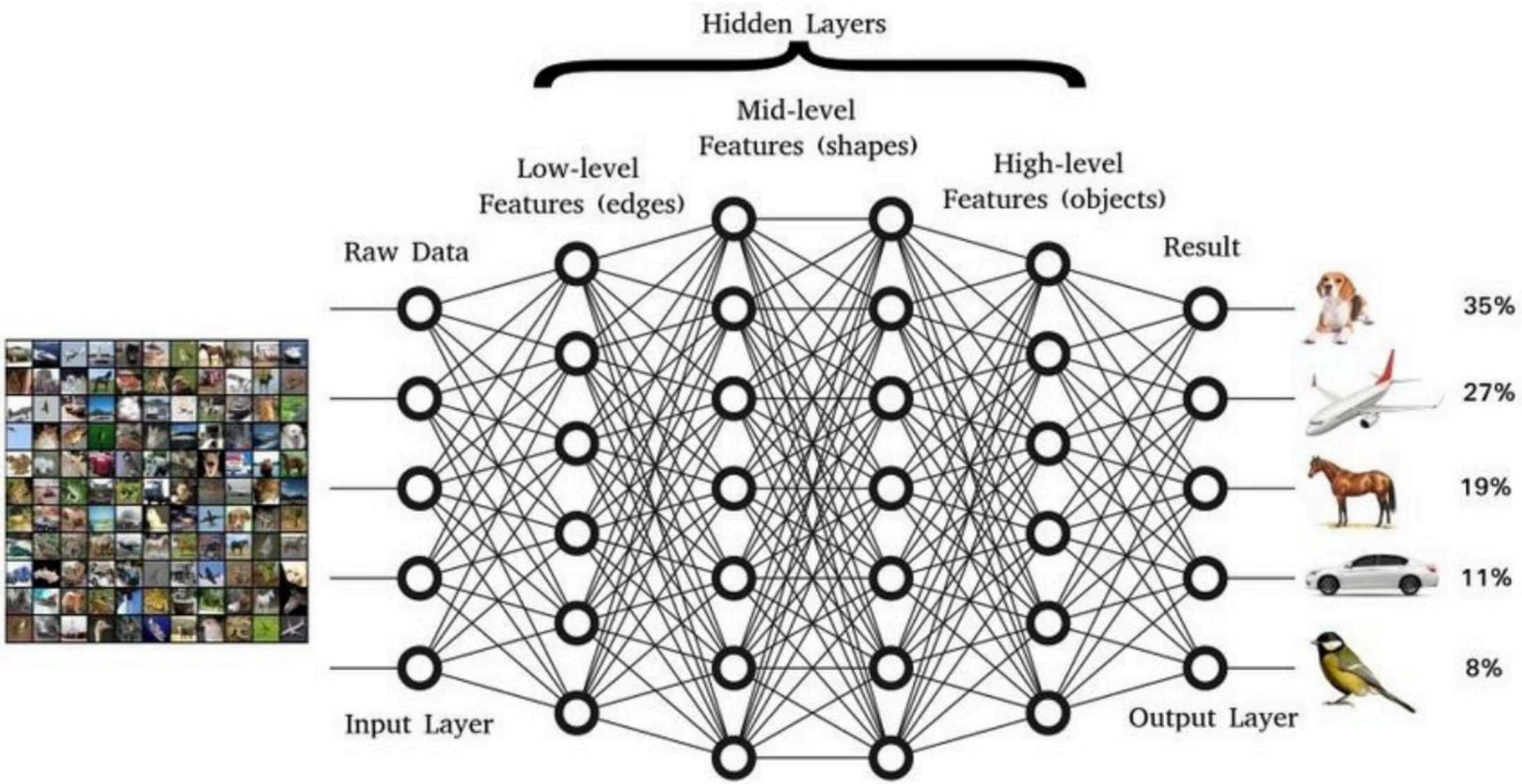


https://miro.medium.com/v2/resize:fit:1200/1*ZafDv3VUm60Eh10OeJu1vw.png

Simone.Gennai@cern.ch

Deep Neural Networks

- DNNs are a combination of perceptrons nodes arranged in layers
  - Sugiarto, Indar & Pasila, Felix. (2018). Understanding a Deep Learning Technique through a Neuromorphic System a Case Study with SpiNNaker Neuromorphic Platform. MATEC Web of Conferences. 164. 01015. 10.1051/matecconf/201816401015.



Simone.Gennai@cern.ch

How do we manage to set the weights for such complex networks?

- As in the finding of the parameters of a linear fit to data, we need a quantity to minimize in order to find the best set of weights that reproduce the data
  - This is easier told than done …
- This function is usually called the loss (or cost) function
  - Depending on which problem we want to solve, we need different loss functions
    - In some cases we also need to combine several of them together
- Most common loss functions are:

Binary cross-entropy

$$L = -\frac{1}{m}\sum_{i=1}^{m}\left(y_i \cdot log(\hat{y}_i) \; + \; (1-y_i)\cdot log(1-\hat{y}_i)\right)$$

y_i = predicted value

$\hat{y}_i$ = target value

Mean Absolute Error

$$L = -\frac{1}{m}\sum_{i=1}^{m}|y_i - \hat{y}_i|$$

Mean Square Error

$$L = -\frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2$$

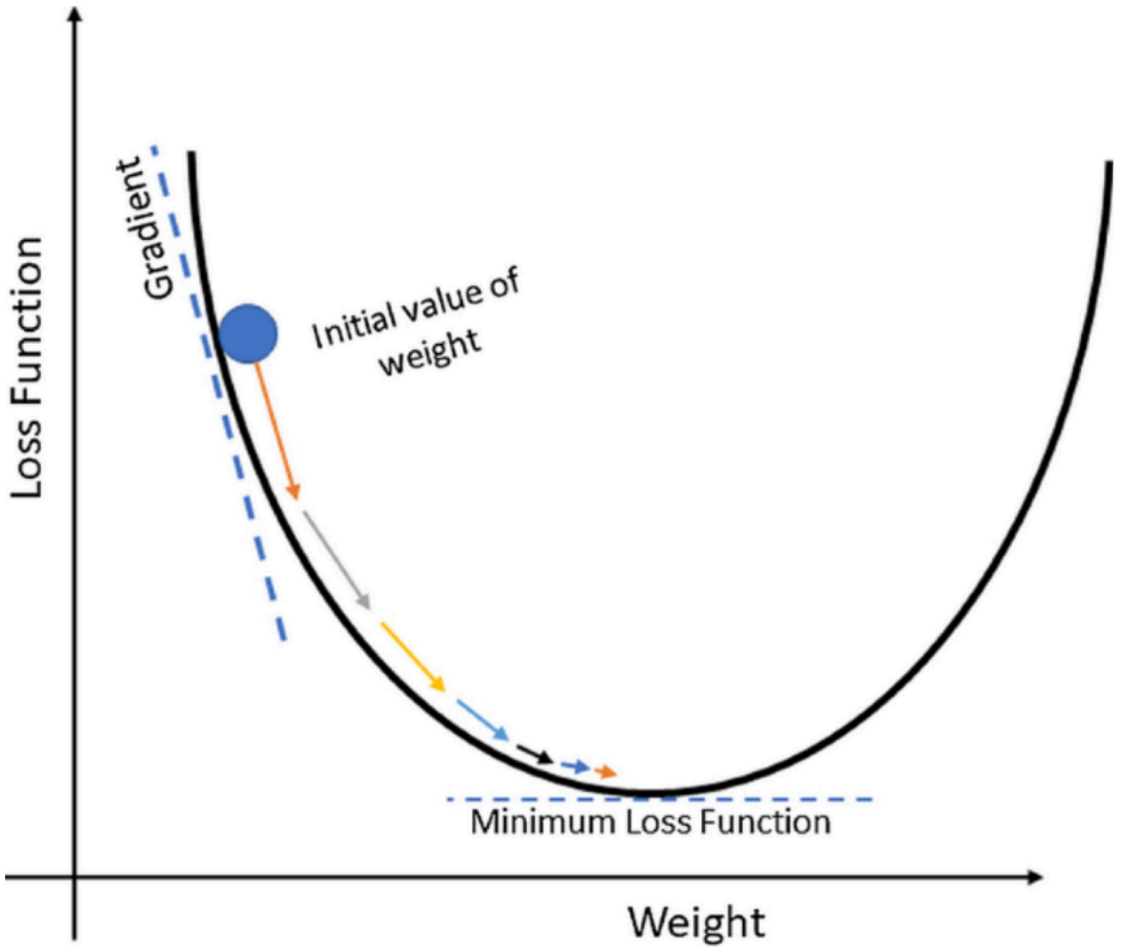
How do you minimize the loss?

- We use the notion of gradient to move along the weights parameter space to find a minimum

Simone.Gennai@cern.ch

How do you minimize the loss?

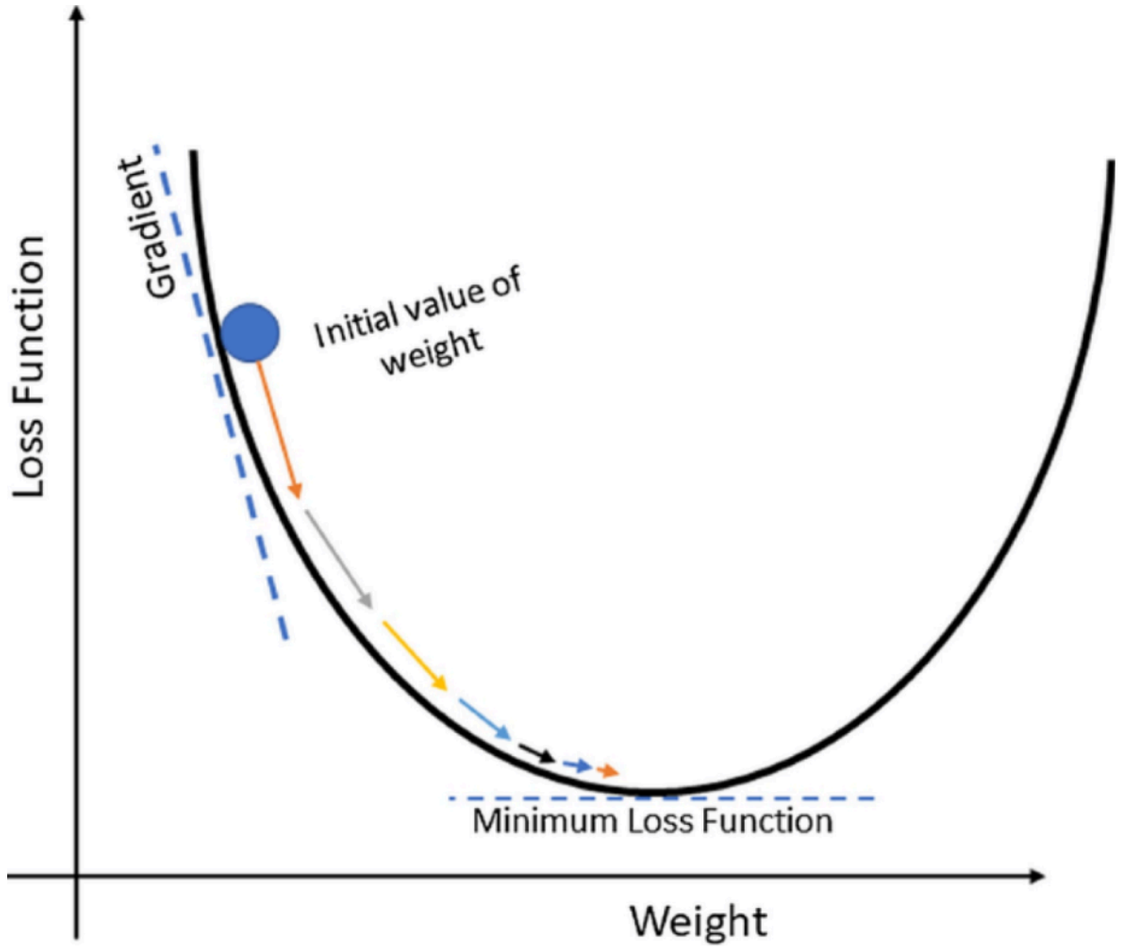
- We use the notion of gradient to move along the weights parameter space to find a minimum

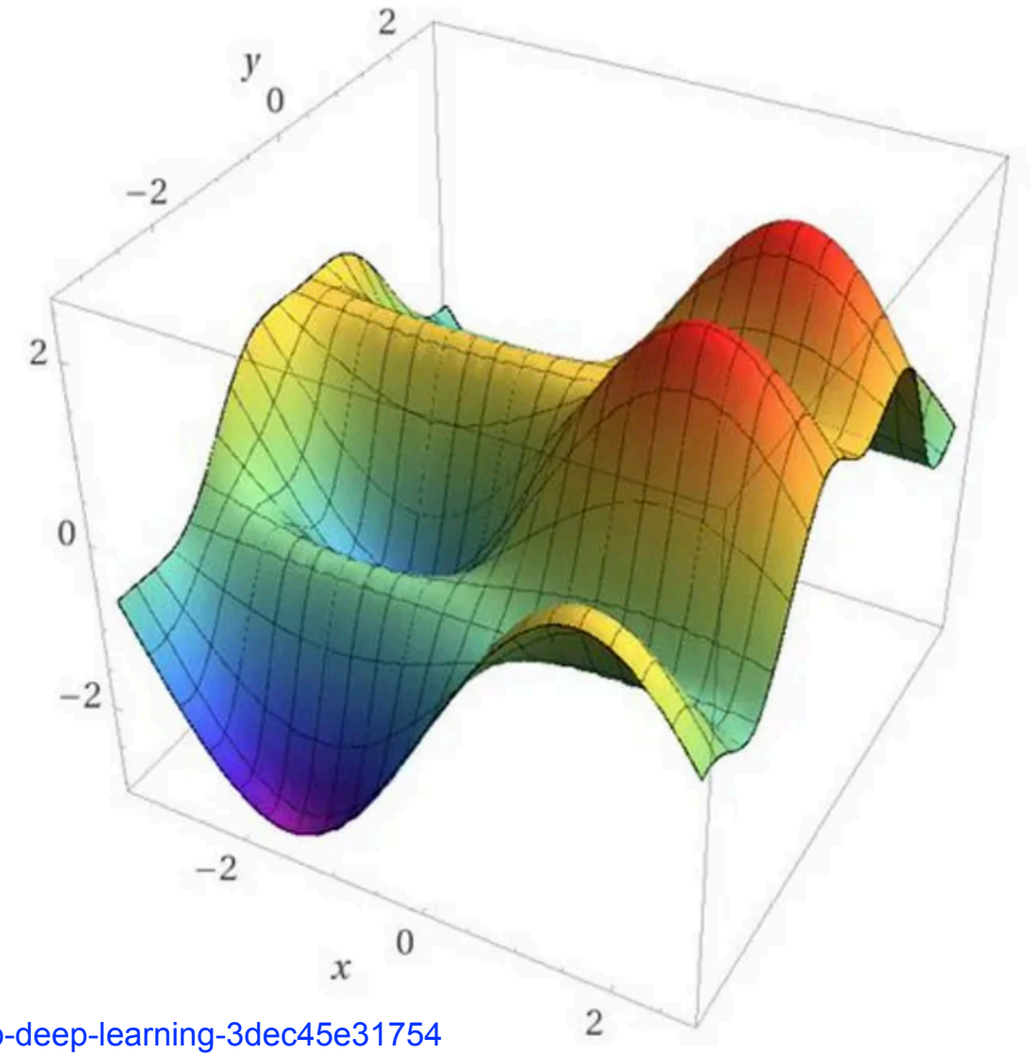


Simone.Gennai@cern.ch

How do you minimize the loss?

- We use the notion of gradient to move along the weights parameter space to find a minimum
  - And let's hope it is not a secondary one too far from the real minimum …





https://medium.com/@justkushhere/propagation-and-optimization-the-key-to-deep-learning-3dec45e31754

Simone.Gennai@cern.ch

The backpropagation theorem

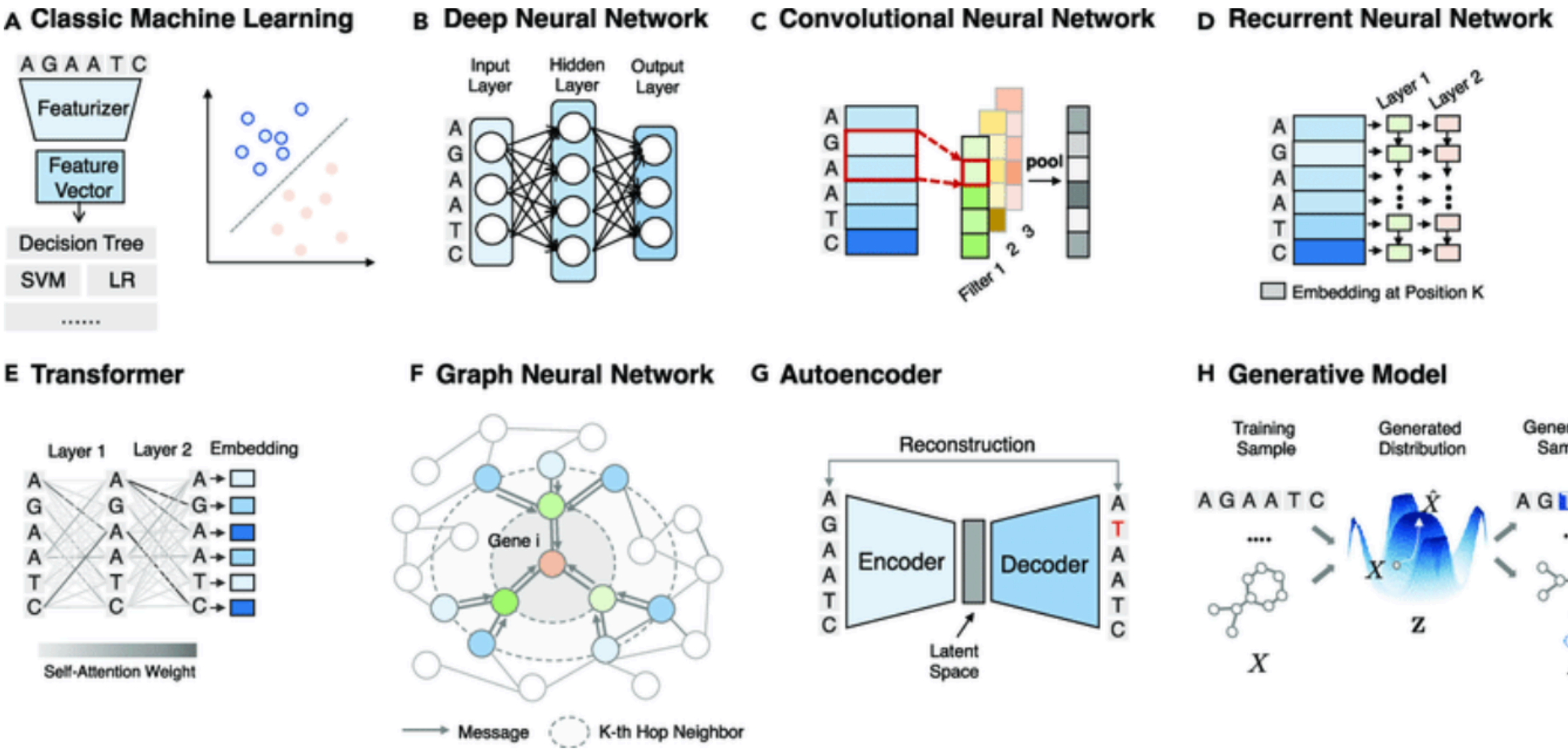
- In oder to speed up the minimization steps and finding the best set of weights there was a major breakthrough in 1962 by Frank Rosenblatt
  - Introducing the backpropagation theorem, which was used in order compute the new gradient after each iteration of the minimization process
- Nowadays we have several ways to compute the gradient
  - Most used ones are: SGD (stochastic gradient descendent) and ADAM (adaptive moment estimation)
- When using packages like Keras/Tensorflow or Pytorch these functions are builtin and they become just parameters of the network to be chosen

Simone.Gennai@cern.ch

More complicated architectures

Simone.Gennai@cern.ch

Choosing the right model for your data



Huang, Kexin & Xiao, Cao & Glass, Lucas & Critchlow, Cathy & Gibson, Greg & Sun, J.. (2021). Machine learning applications for therapeutic tasks with genomics data. Patterns. 2. 100328. 10.1016/j.patter.2021.100328.

Simone.Gennai@cern.ch

Choosing the right model for your data

- What we have discussed so far applies to any classification problem.
- The choice of which algorithm to choose does depend on the nature of the data you are using
- In particle physics we deal with particle distributed in space
  - Particles are well separated from each other, each particle has lot of information associated with them
    - Momentum, ID, isolation, production vertex, etc. etc.
  - Most used networks are graph neural networks which adapts well to the structure of our data
    - Nonetheless other models can still be used, provided input data is processed accordingly

Simone.Gennai@cern.ch

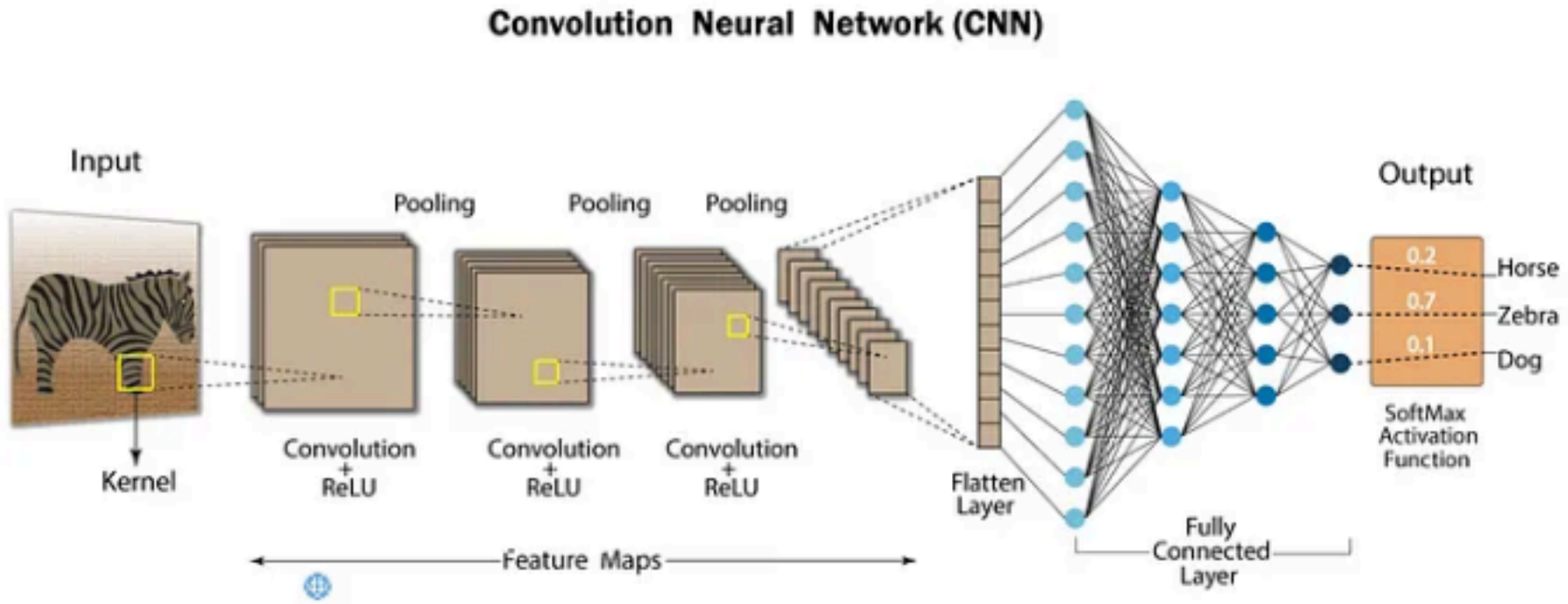
Convolutional Neural Networks

- For image classification Convolutional Neural Networks are mostly used:
  - It breaks the image in small patterns and these patterns are used to identify what is in the image
  - There are two stages in a CNN:
    - A first set of layers analyze the image and extract as many features as possible
    - A second set of layers (normal dense layers in this case) is used to “learn" from these features which kind of image was given in input
- CNN can be used to identify objects/animals/faces in the image but also to define contours and object counting

Simone.Gennai@cern.ch

CNN

- CNN can be made of several blocks which are repeated within the same network a number of times.
  - At the very end there is always a flattening layer that collapses all the information into a large vector that is then passed to a fully connected network that learns through this large vector which kind of image it is



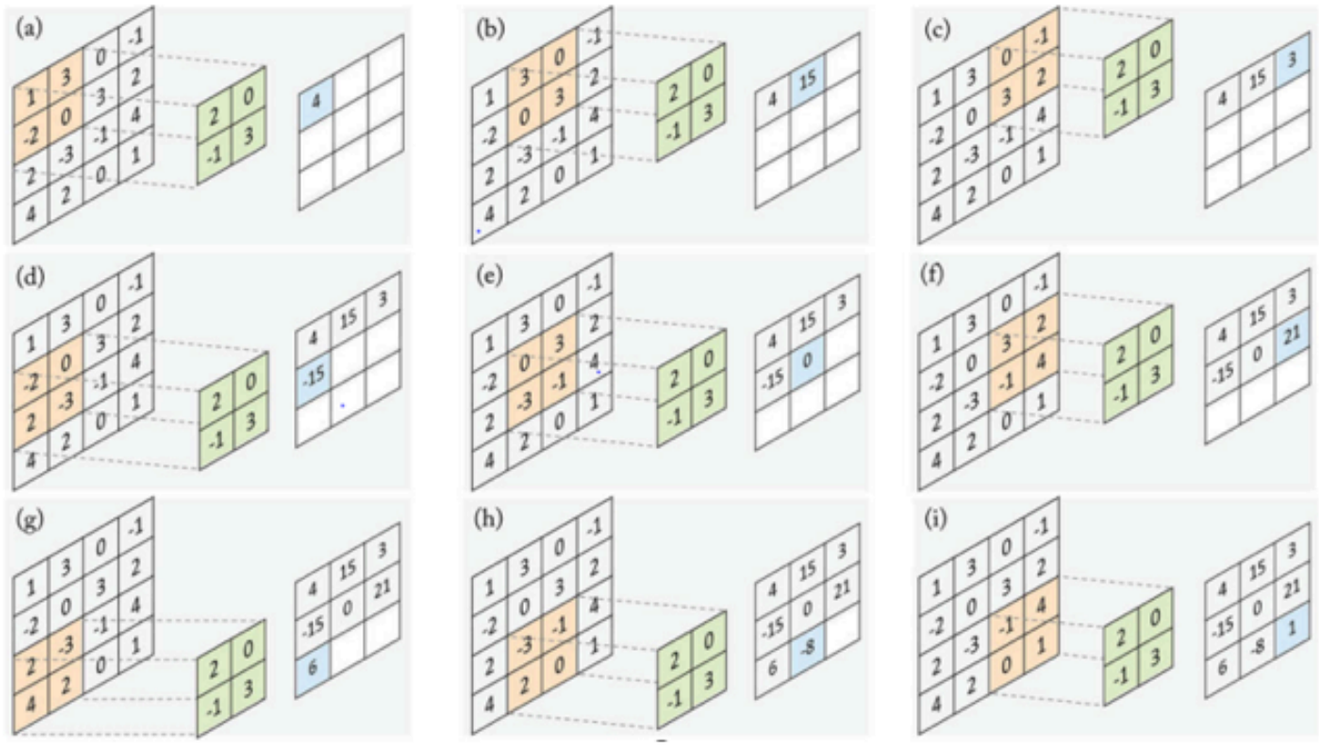
CNN FOR CLASSIFICATION

https://abhishek673.medium.com/image-classification-using-convolutional-neural-network-and-vision-transformer-29758796e08a

Simone.Gennai@cern.ch

How convolution works

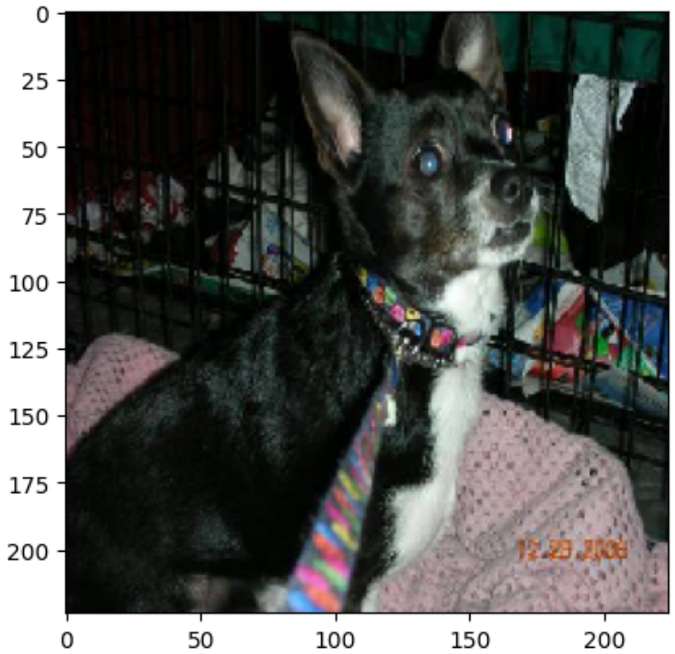
- Each layer consists in a number of filters (kernel) that can detect basic structures (like edges)
- Combining these filters one after the other and going deep into the network the structures that can be detected becomes more and more of high level
- A simple convolutional filter is just a matrix that is applied to the image pixel matrix with a given “stride”
- Each CNN layer has several parameters:
  - # of filters
  - Dimension of each filter
  - Stride
  - Padding
  - …

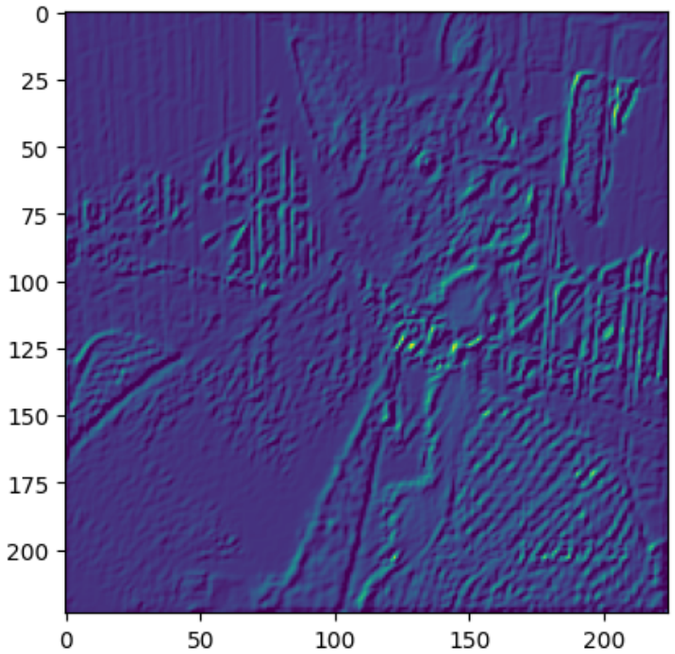


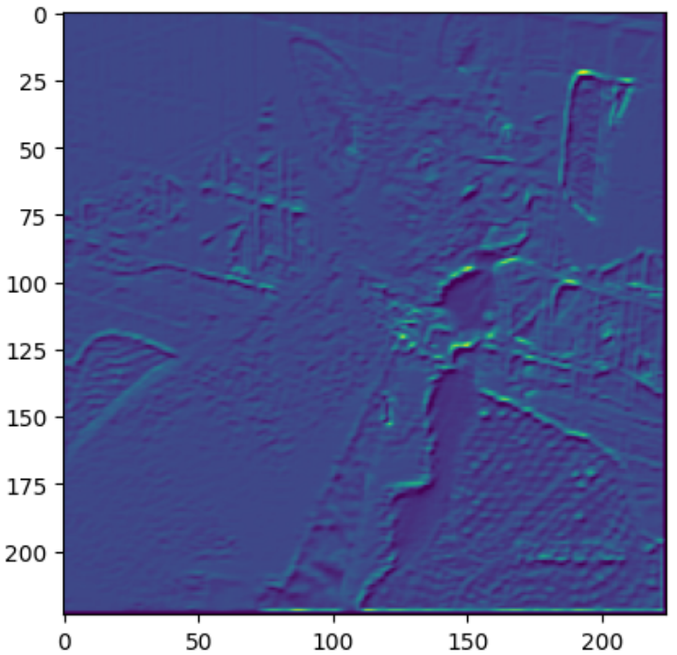
Convolutional Layer Architecture – Source: (Khan et al., 2018)

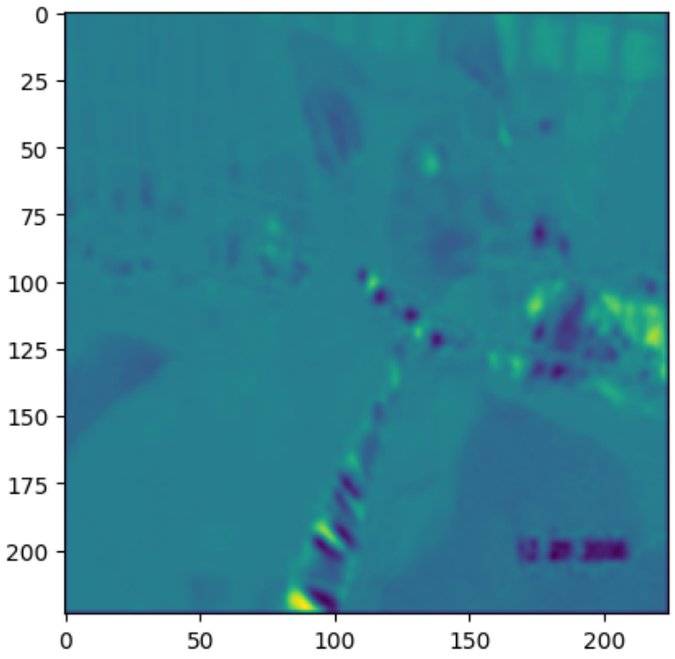
Simone.Gennai@cern.ch

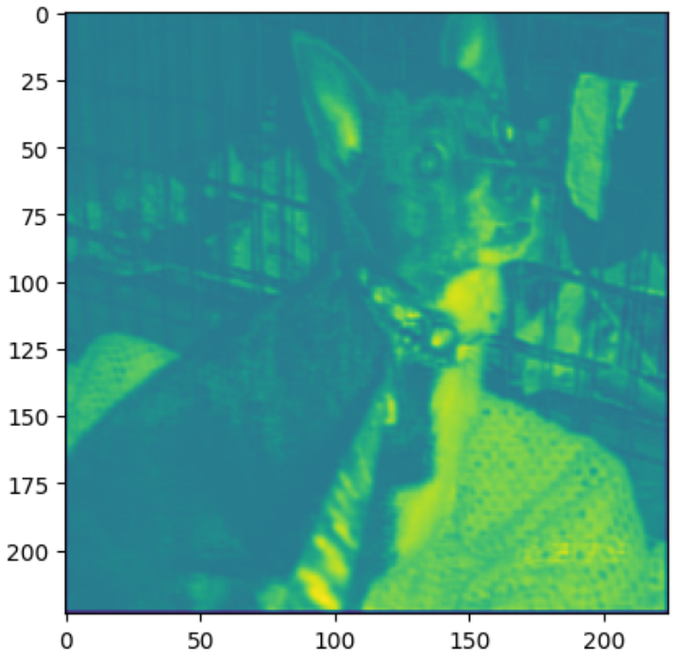
What does a CNN really sees









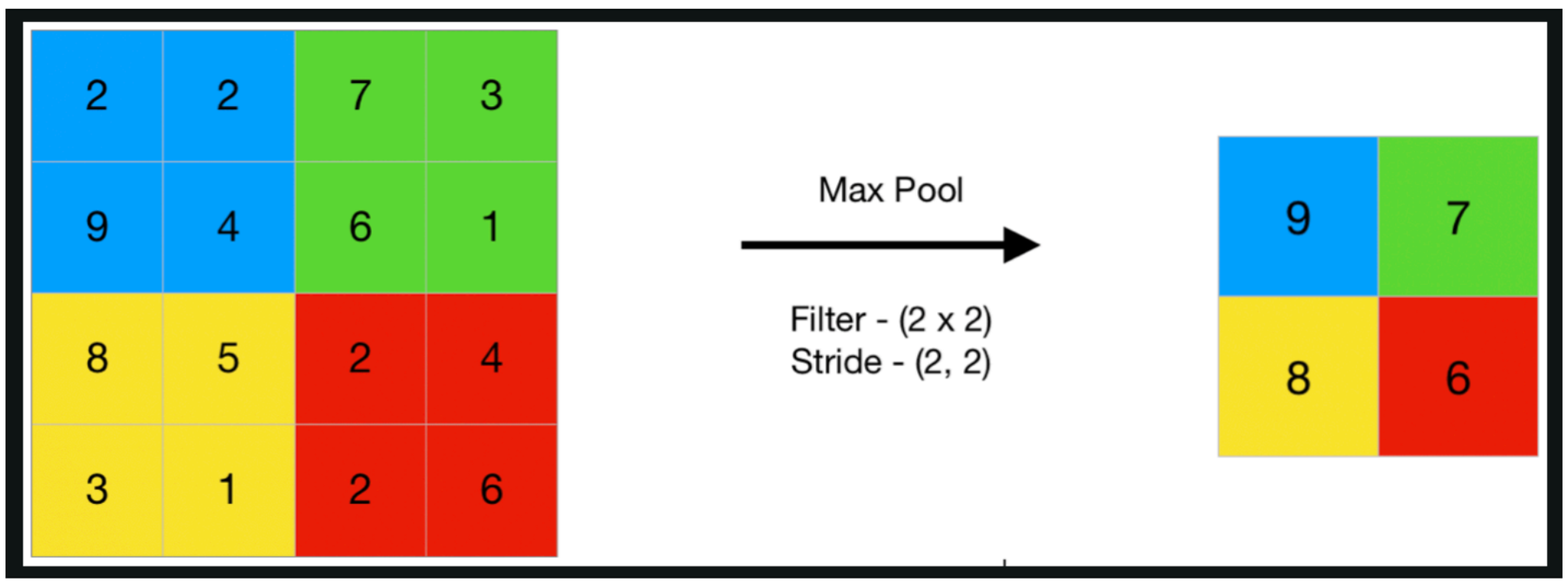


Output of filters in a random order from one of the first CNN layers of a model

Simone.Gennai@cern.ch

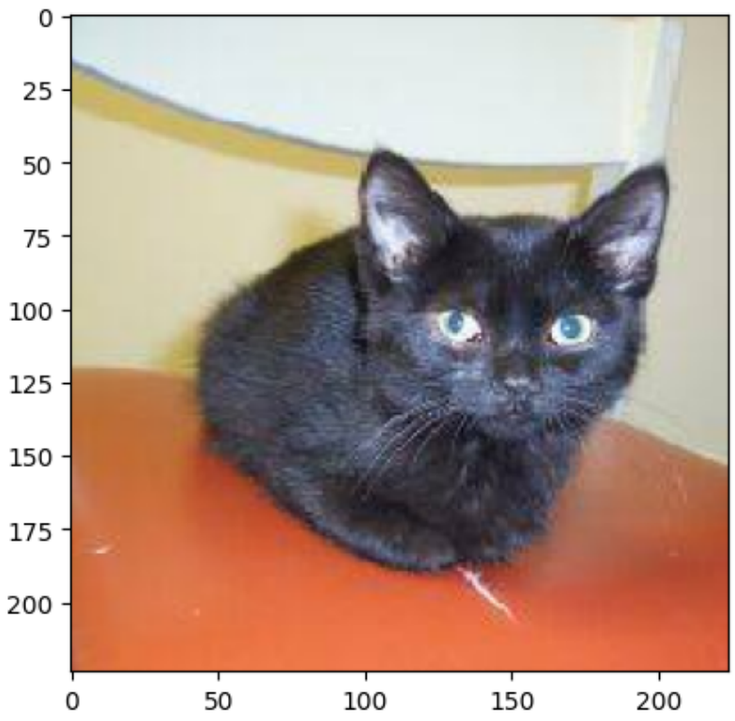
Pooling layer

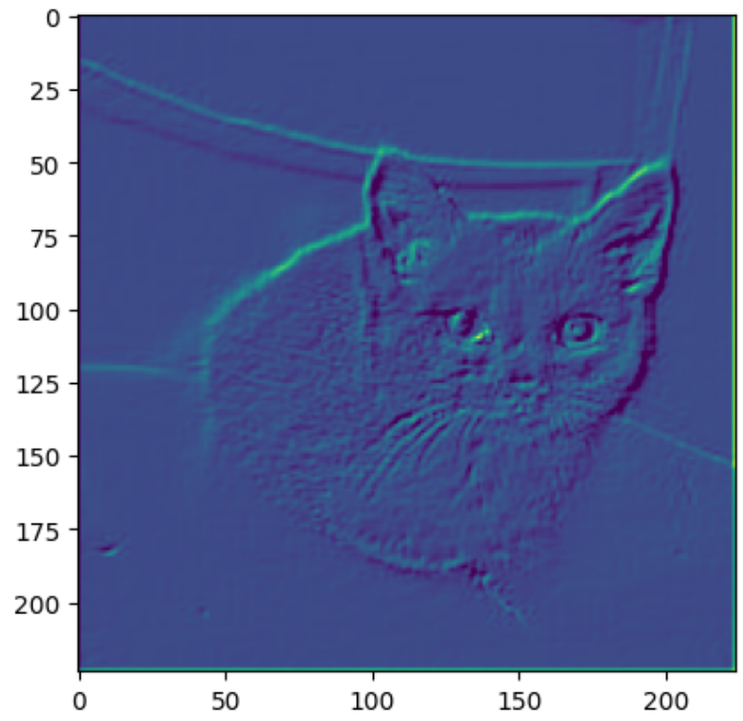
- The pooling layer is a filter used to reduce the image resolution preserving the features of the image
  - It is usually applied after deconvolution layers to avoid blowing up the memory and the computing time
- There are several different choices (max/min pooling, average pooling, adapting pooling)

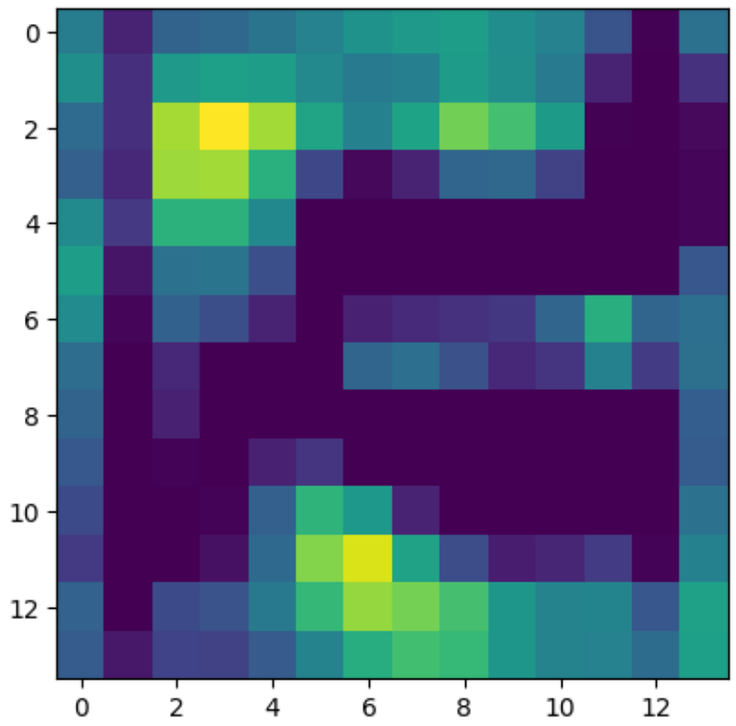


Simone.Gennai@cern.ch

What does a CNN really sees







Output of the first filter in the first layer and one of the last filters in the last layer of the same CNN. N.B. different image resolution

Simone.Gennai@cern.ch

How do we code all this?

- Pretty easily with Tensorflow/Keras

```
# building a linear stack of layers with the sequential model
model = Sequential()

# convolutional layer
model.add(Conv2D(50, kernel_size=(3,3), strides=(1,1), padding='same', activation='relu', input_shape=(32, 32, 3)))

# convolutional layer
model.add(Conv2D(75, kernel_size=(3,3), strides=(1,1), padding='same', activation='relu'))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Dropout(0.25))

model.add(Conv2D(125, kernel_size=(3,3), strides=(1,1), padding='same', activation='relu'))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Dropout(0.25))

# flatten output of conv
model.add(Flatten())

# hidden layer
model.add(Dense(500, activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(250, activation='relu'))
model.add(Dropout(0.3))
# output layer
model.add(Dense(10, activation='softmax'))
```


Simone.Gennai@cern.ch

Possible pitfalls in model training

- Due to the large number of parameters and the many convolutional operations CNN may be difficult to properly train.
  - First you need a lot of properly labelled images to let the network learn the relevant features.
    - That's easy for dogs and cats, much less for medical images
  - Second the use of a GPU is mandatory if we want to finish the training in a reasonable time and being able to choose among different configurations
  - Third you need to pay attention to overtraining and biases
    - Not easy with a limited sample of images
      - Even though you can use standard techniques to make your life easier:
        - Dropout, early stopping, synthetic data generation
  - Fourth, input data has to be properly prepared
    - Particularly true in case of medical imaging
- That's why using pre-trained models can help a lot the final user!

Simone.Gennai@cern.ch

Pre-trained models

Simone.Gennai@cern.ch

How does a pretrained model work?

- Large models (O(10E6) parameters) trained on very large image databases (like imagenet)
- First part of the models (feature extractions) are taken as they are with the weights obtained by their training on e.g. imagenet DB
- Second part (dense layers) are overwritten with new layers adapted to the scope of the single usercase
  - We train only these dense layers, less need of GPU power, faster, no need for a large dataset, we finetune with the usecase dataset

Simone.Gennai@cern.ch

How does a pretrained model work?

ImageNet

Edit

Introduced by Jia Deng et al. in ImageNet: A large-scale hierarchical image database

The **ImageNet** dataset contains 14,197,122 annotated images according to the WordNet hierarchy. Since 2010 the dataset is used in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), a benchmark in image classification and object detection. The publicly released dataset contains a set of manually annotated training images. A set of test images is also released, with the manual annotations withheld. ILSVRC annotations fall into one of two categories: (1) image-level annotation of a binary label for the presence or absence of an object class in the image, e.g., "there are cars in this image" but "there are no tigers," and (2) object-level annotation of a tight bounding box and class label around an object instance in the image, e.g., "there is a screwdriver centered at position (20,25) with width of 50 pixels and height of 30 pixels". The ImageNet project does not own the copyright of the images, therefore only thumbnails and URLs of images are provided.

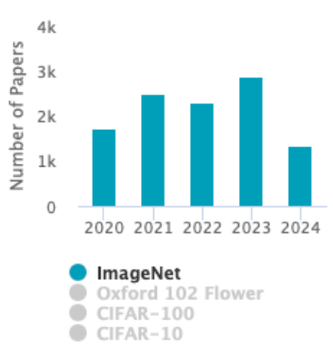
- Total number of non-empty WordNet synsets: 21841
- Total number of images: 14197122
- Number of images with bounding box annotations: 1,034,908
- Number of synsets with SIFT features: 1000
- Number of images with SIFT features: 1.2 million

Source: ImageNet Large Scale Visual Recognition Challenge

Homepage

Source: https://cs.stanford.edu/people/kar...

Usage



Benchmarks

Edit

Trend	Task	Dataset Variant	Best Model	Paper	Code
	Image Classification	ImageNet	OmniVec		
	Neural Architecture Search	ImageNet	DeepMAD-50M		
	Self-Supervised Image Classification	ImageNet	DINOv2		
	Semi-Supervised Image Classification	ImageNet - 10% labeled data	Meta Co-Training		
	Self-Supervised Image Classification	ImageNet (finetuned)	DINOv2		

License

Edit

Custom (research, non-commercial)

Modalities

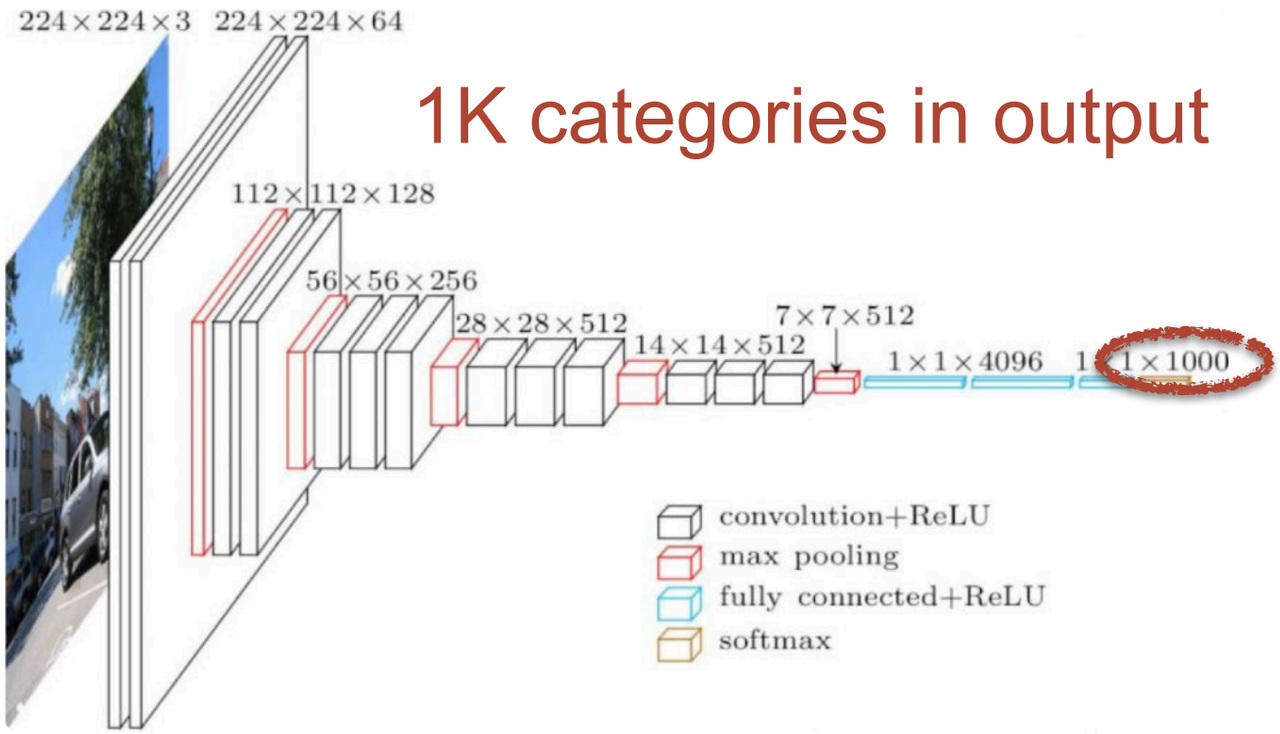
Edit

Images

Plenty of resources also with links to the original model

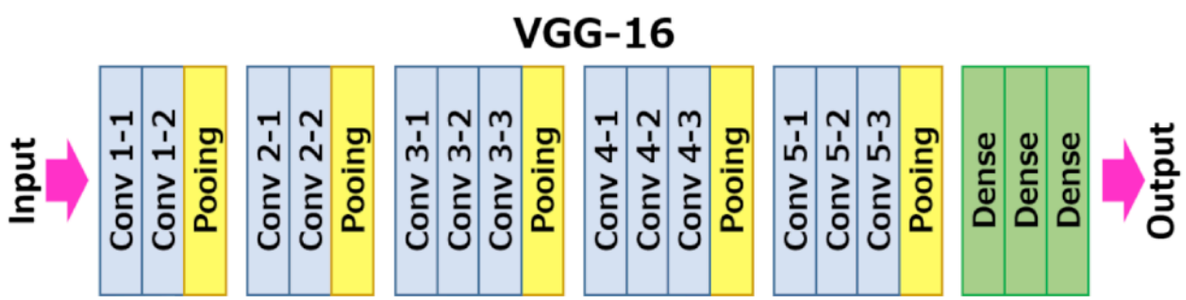
Simone.Gennai@cern.ch

Pretrained models : VGG-16



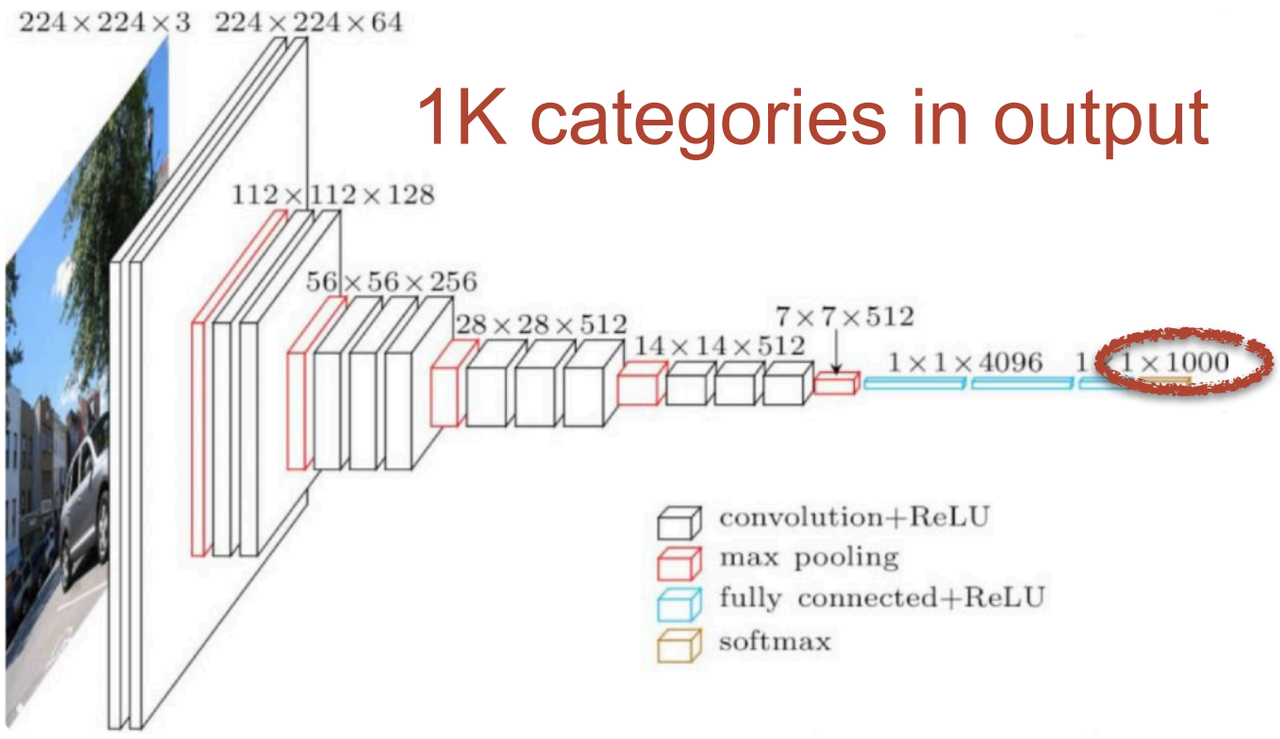
1K categories in output

Here is a more intuitive layout of the VGG-16 Model.



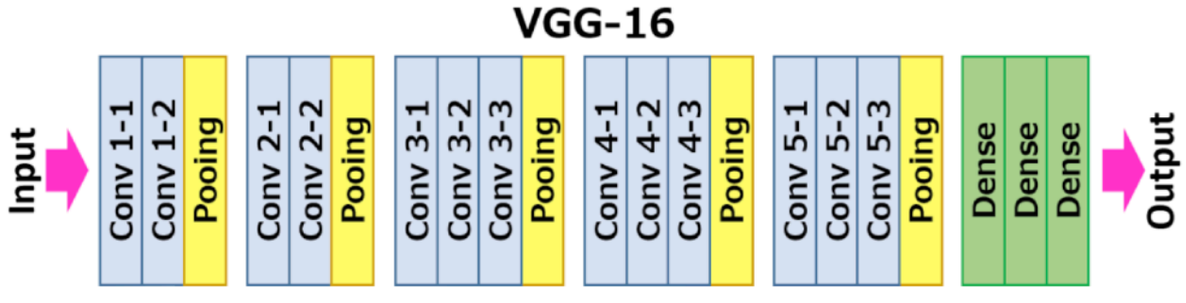
Simone.Gennai@cern.ch

Pretrained models : VGG-16



1K categories in output

Here is a more intuitive layout of the VGG-16 Model.



Layer (type)	Output Shape	Param #
input_layer_1 (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1,792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36,928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73,856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147,584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295,168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590,080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590,080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1,180,160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
fc1 (Dense)	(None, 4096)	102,764,544
fc2 (Dense)	(None, 4096)	16,781,312
predictions (Dense)	(None, 1000)	4,097,000

Total params: 138,357,544 (527.79 MB)
Trainable params: 138,357,544 (527.79 MB)

Simone.Gennai@cern.ch

Pretrained models : VGG-16

Layer (type)	Output Shape	Param #
input_layer_1 (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1,792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36,928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73,856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147,584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295,168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590,080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590,080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1,180,160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
fc1 (Dense)	(None, 4096)	102,764,544
fc2 (Dense)	(None, 4096)	16,781,312
predictions (Dense)	(None, 1000)	4,097,000

Total params: 138,357,544 (527.79 MB)
Trainable params: 138,357,544 (527.79 MB)

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1,792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36,928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73,856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147,584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295,168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590,080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590,080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1,180,160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten_2 (Flatten)	(None, 25088)	0
dense_4 (Dense)	(None, 512)	12,845,568
dropout_2 (Dropout)	(None, 512)	0
dense_5 (Dense)	(None, 1)	513

Total params: 27,560,769 (105.14 MB)
Trainable params: 12,846,081 (49.00 MB)
Non-trainable params: 14,714,688 (56.13 MB)

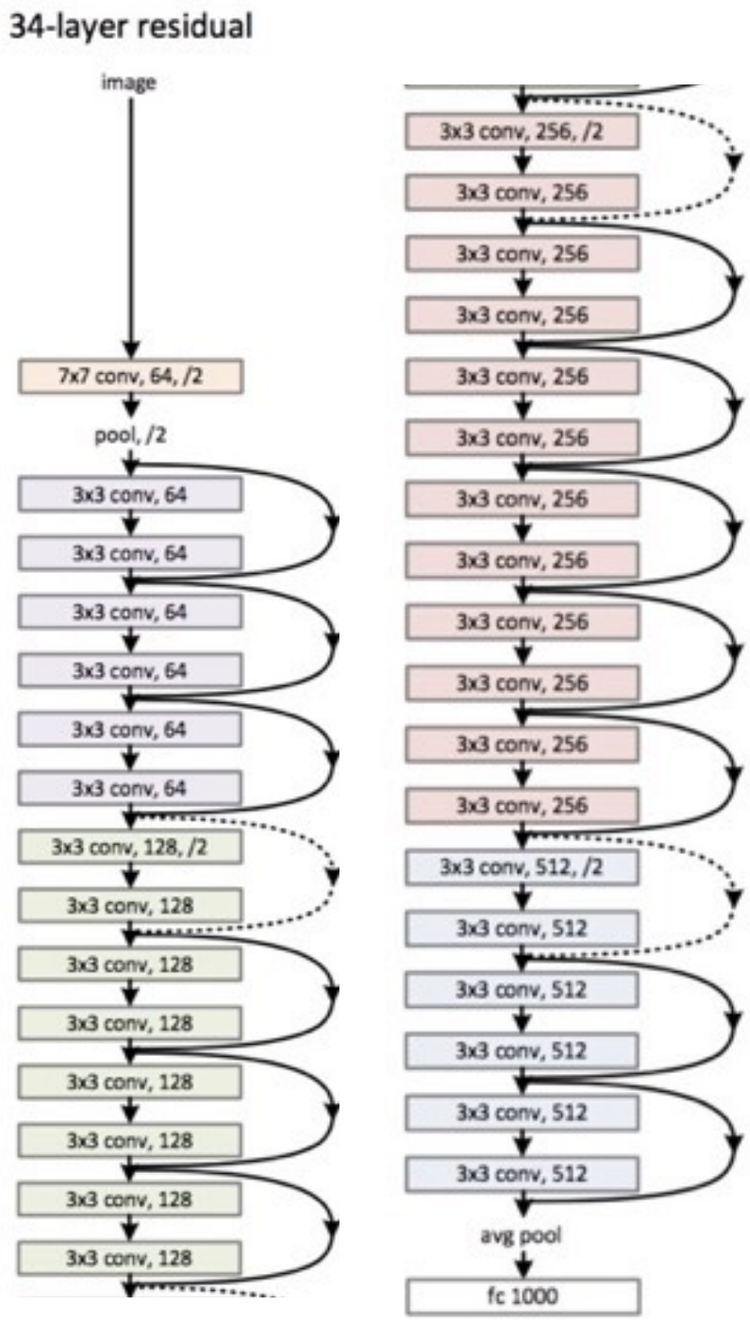
Simone.Gennai@cern.ch

Pretrained models : VGG-16

```
Epoch 1/10
/usr/local/lib/python3.10/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: You
  self._warn_if_super_not_called()
100/100 ━━━━━━━━━━━━━━━━━━━━ 28s 154ms/step - acc: 0.6809 - loss: 0.7950 - val_acc: 0.8730 - val_loss: 0.2921
Epoch 2/10
/usr/lib/python3.10/contextlib.py:153: UserWarning: Your input ran out of data; interrupting training. Make sure tha
  self.gen.throw(typ, value, traceback)
100/100 ━━━━━━━━━━━━━━━━━━━━ 5s 49ms/step - acc: 0.0000e+00 - loss: 0.0000e+00 - val_acc: 0.8730 - val_loss: 0.2921
Epoch 3/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 15s 153ms/step - acc: 0.8797 - loss: 0.2632 - val_acc: 0.8710 - val_loss: 0.3092
Epoch 4/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 5s 51ms/step - acc: 0.0000e+00 - loss: 0.0000e+00 - val_acc: 0.8710 - val_loss: 0.3092
Epoch 5/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 36s 153ms/step - acc: 0.8950 - loss: 0.2362 - val_acc: 0.8690 - val_loss: 0.3127
Epoch 6/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 5s 51ms/step - acc: 0.0000e+00 - loss: 0.0000e+00 - val_acc: 0.8690 - val_loss: 0.3127
Epoch 7/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 36s 155ms/step - acc: 0.9257 - loss: 0.1717 - val_acc: 0.9070 - val_loss: 0.2273
Epoch 8/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 5s 52ms/step - acc: 0.0000e+00 - loss: 0.0000e+00 - val_acc: 0.9070 - val_loss: 0.2273
Epoch 9/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 16s 157ms/step - acc: 0.9396 - loss: 0.1325 - val_acc: 0.9050 - val_loss: 0.2319
Epoch 10/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 5s 51ms/step - acc: 0.0000e+00 - loss: 0.0000e+00 - val_acc: 0.9050 - val_loss: 0.2319
```


Simone.Gennai@cern.ch

Pretrained models: ResNet34



Material from https://arxiv.org/pdf/1512.03385

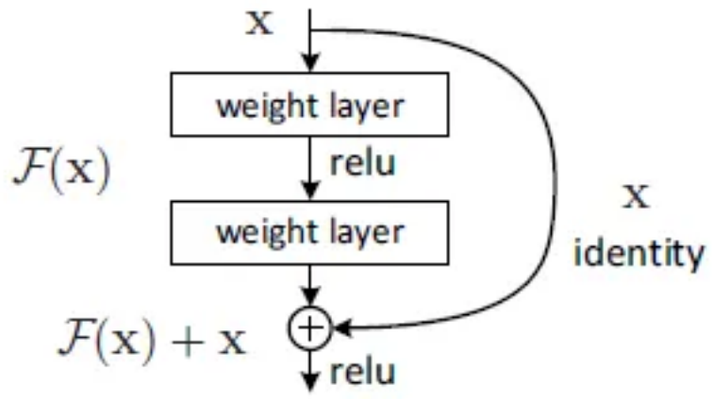
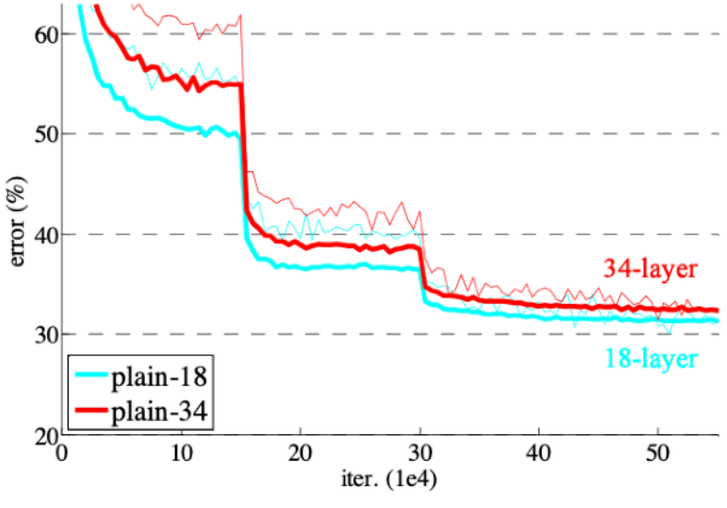
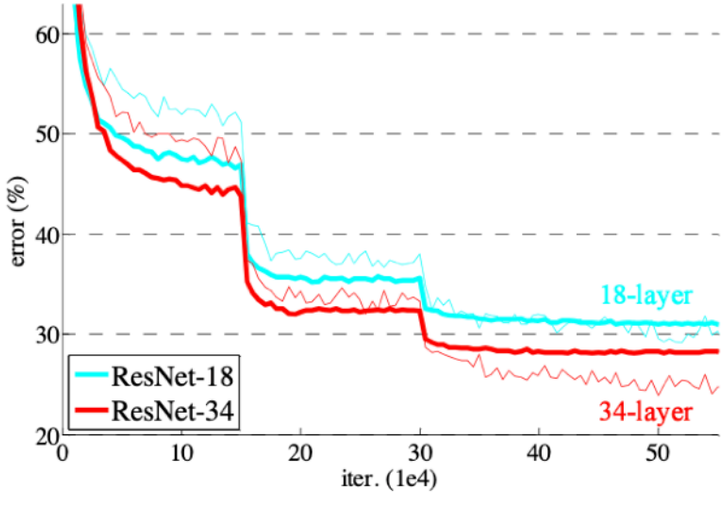


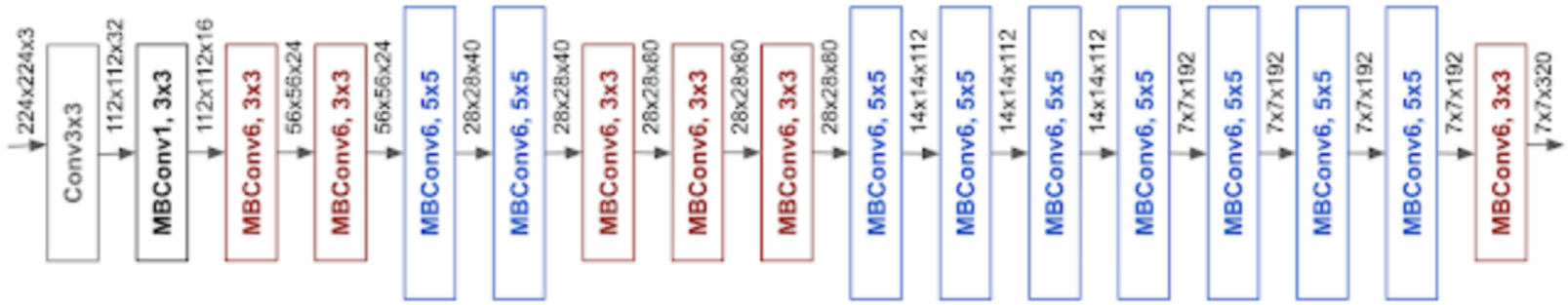
Figure 3. Residual building block



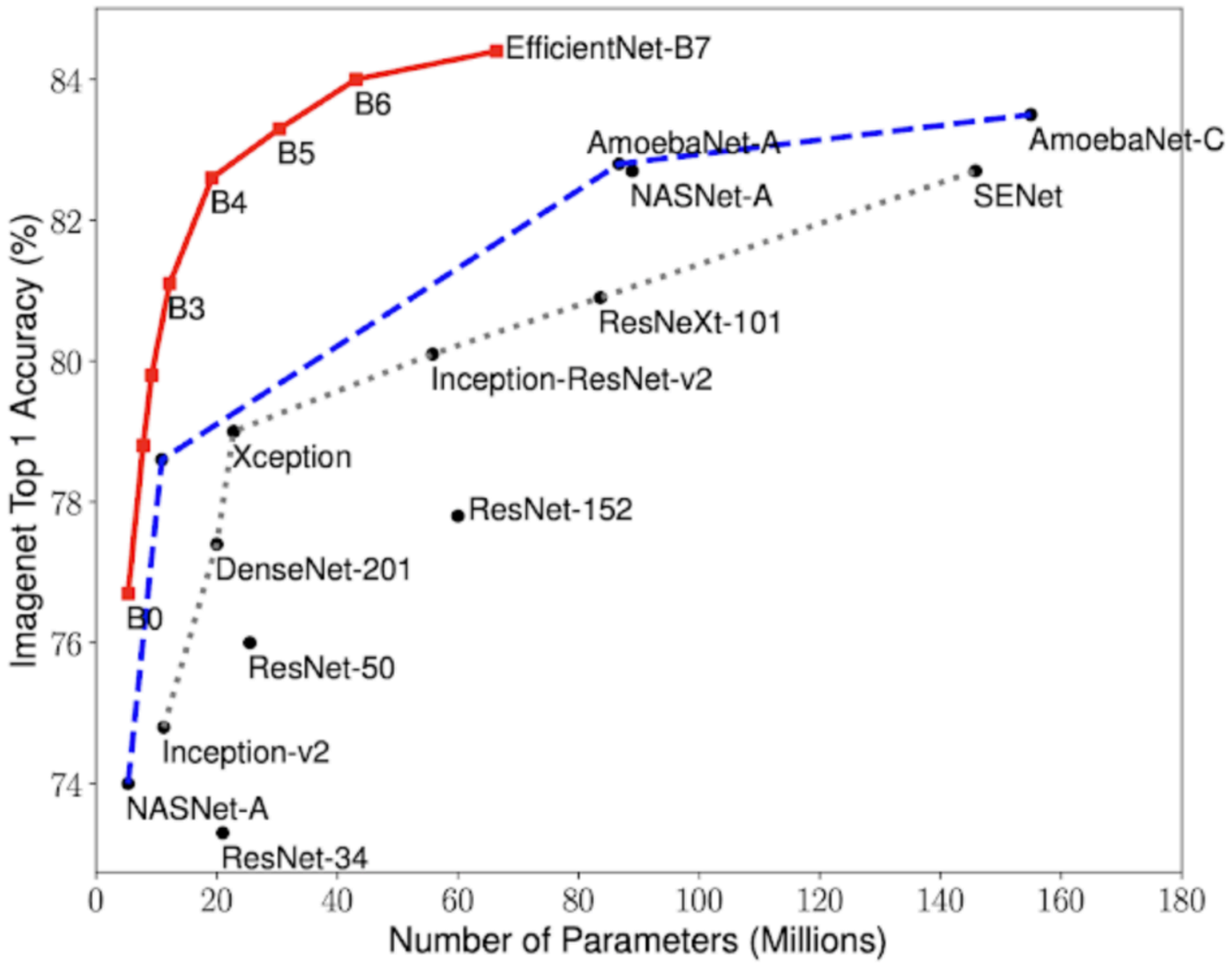


Simone.Gennai@cern.ch

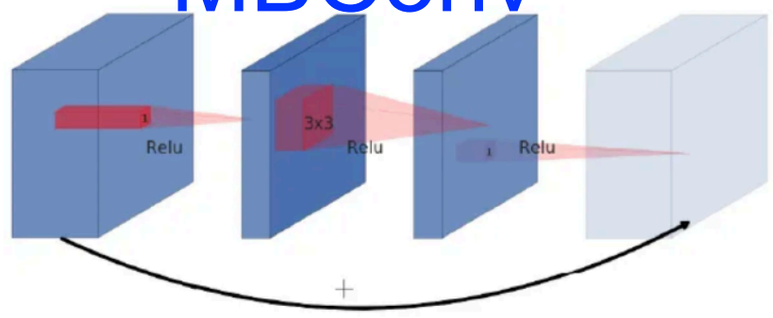
Pretrained models: EfficientNet



EfficientNet



MBConv



A residual block connects wide layers with a skip connection while layers in between are narrow

Simone.Gennai@cern.ch

Pretrained models

Model	Year	Number of Parameters	Top-1 Accuracy
VGG-16	2014	138 Million	74.5%
ResNet-50	2015	25 Million	77.15%
Inception V3	2015	24 Million	78.8%
EfficientNetB0	2019	5.3 Million	76.3%
EfficientNetB7	2019	66 Million	84.4%

Simone.Gennai@cern.ch

What is this hackaton about?

Simone.Gennai@cern.ch

Classifying medical images

- One of the most useful (to me) use of ML is its application to medical images
  - It can help a doctor to identify a lesion in the brain, or to analyze impact of COVID on the lungs or even study the probability of recidive in cancer affected patients
- The problem with medical images is that there are only limited size samples with incomplete annotations
  - Very often the size is of O(100) images, not enough to properly train and tune complex models that are usually needed for image analysis
    - Consider the number of parameters that models like VGG has …
- That's the idea of using pre-trained samples with a fine tuning on small image sample

Simone.Gennai@cern.ch

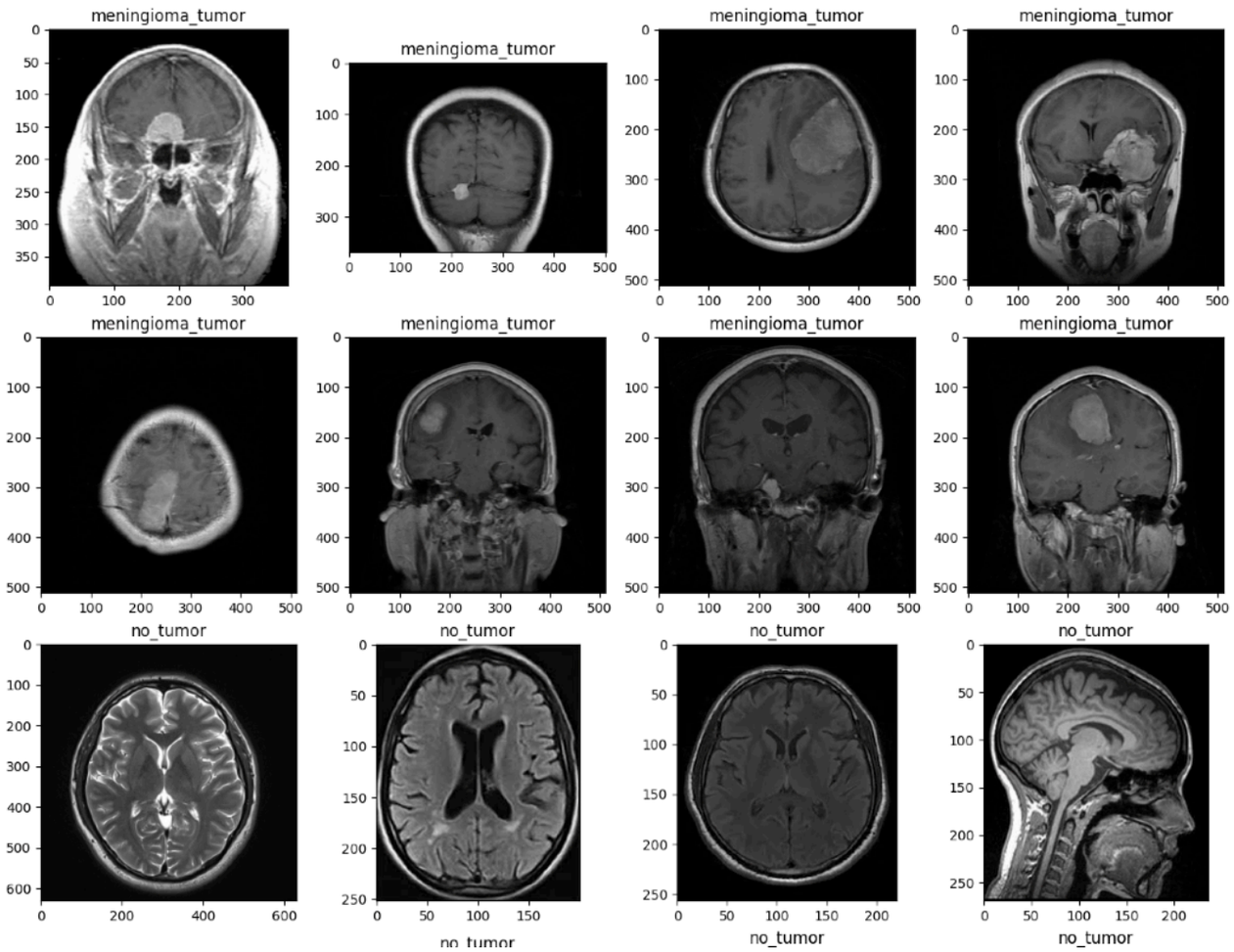
Brain tumor classification

- Thanks to the power of large models trained on (1E6) images we can try to classify (through MRI images) brain tumors
  - We will also need to define the metrics we want to use to see if our multiclassification task is succesful
- The dataset has been prepared by researchers at MIT and made available in the kaggle database
  - It includes images of 3 types of tumors and one set without any tumor.
  - Images are already divided between training and testing dataset and already arranged in the 4 classes

Data Explorer

Version 2 (93.08 MB)

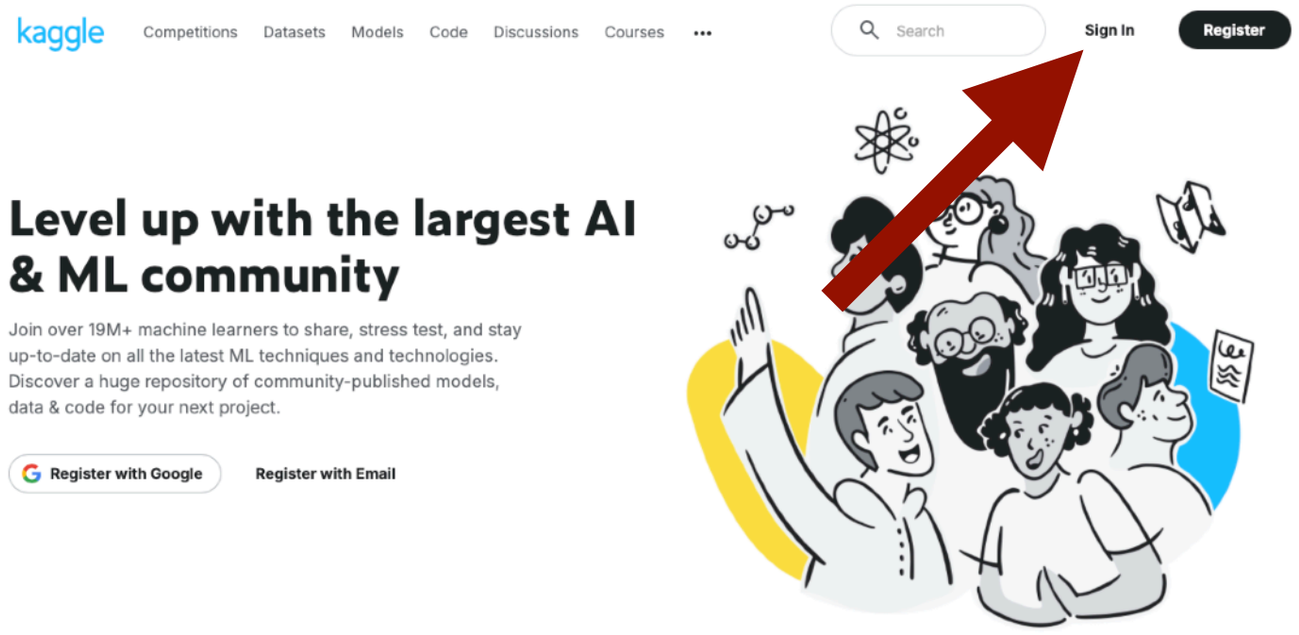
- Testing
  - glioma_tumor
  - meningioma_tumor
  - no_tumor
  - pituitary_tumor
- Training
  - glioma_tumor
  - meningioma_tumor
  - no_tumor
  - pituitary_tumor

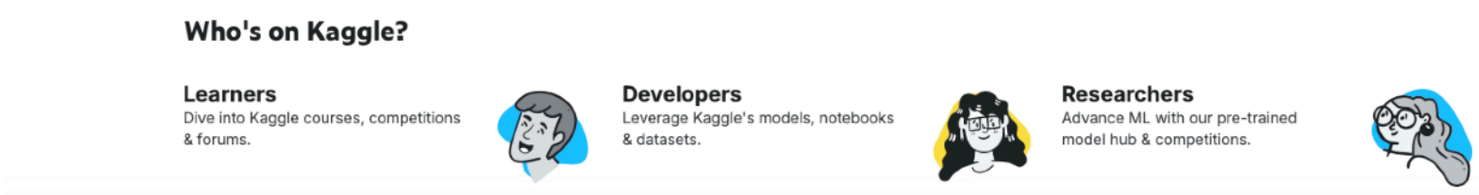


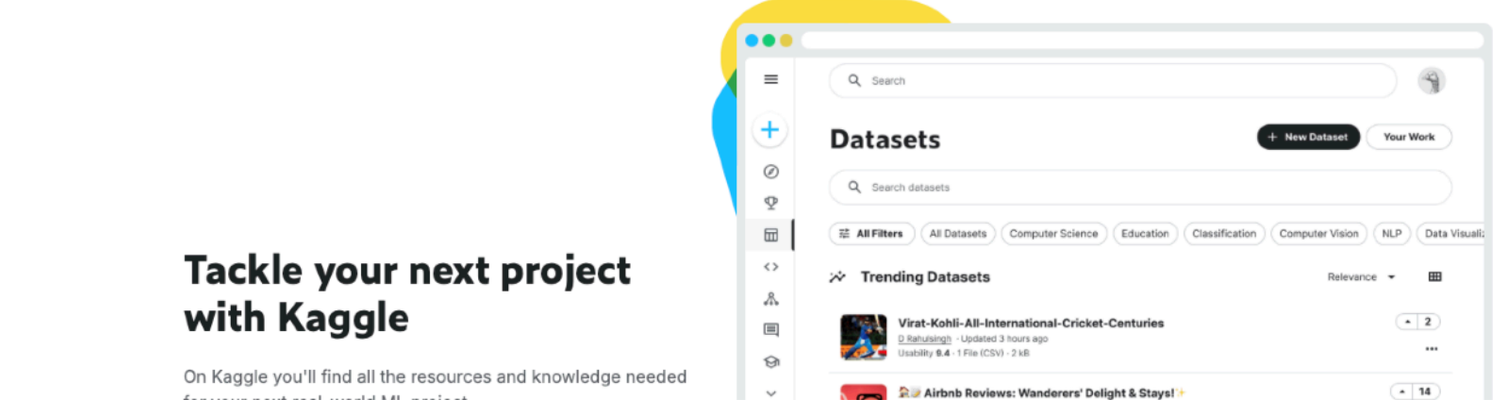
Simone.Gennai@cern.ch

How to login/register on kaggle

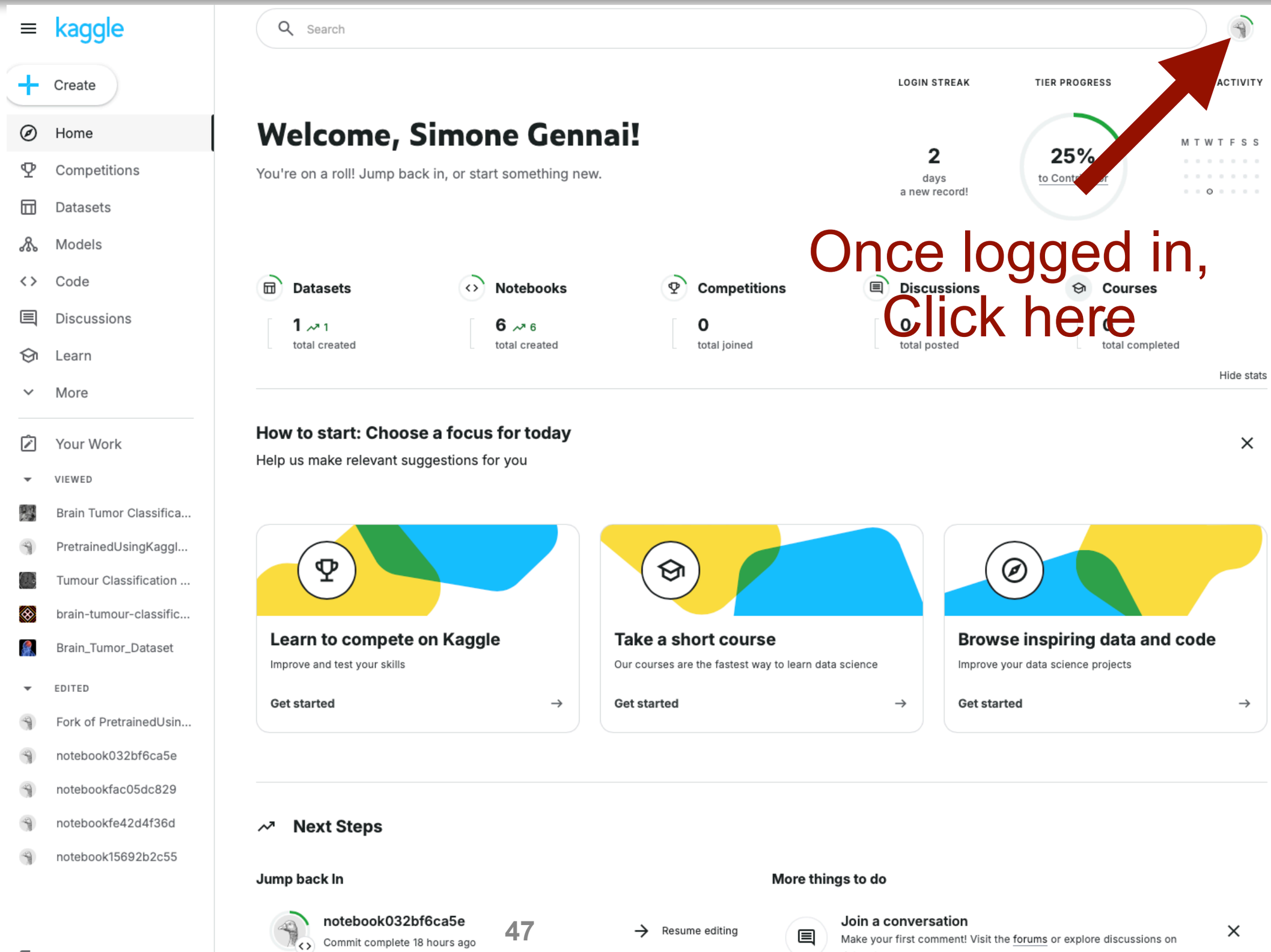
- Going to www.kaggle.com
  - Clicl con Sign In
  - You can use your google account



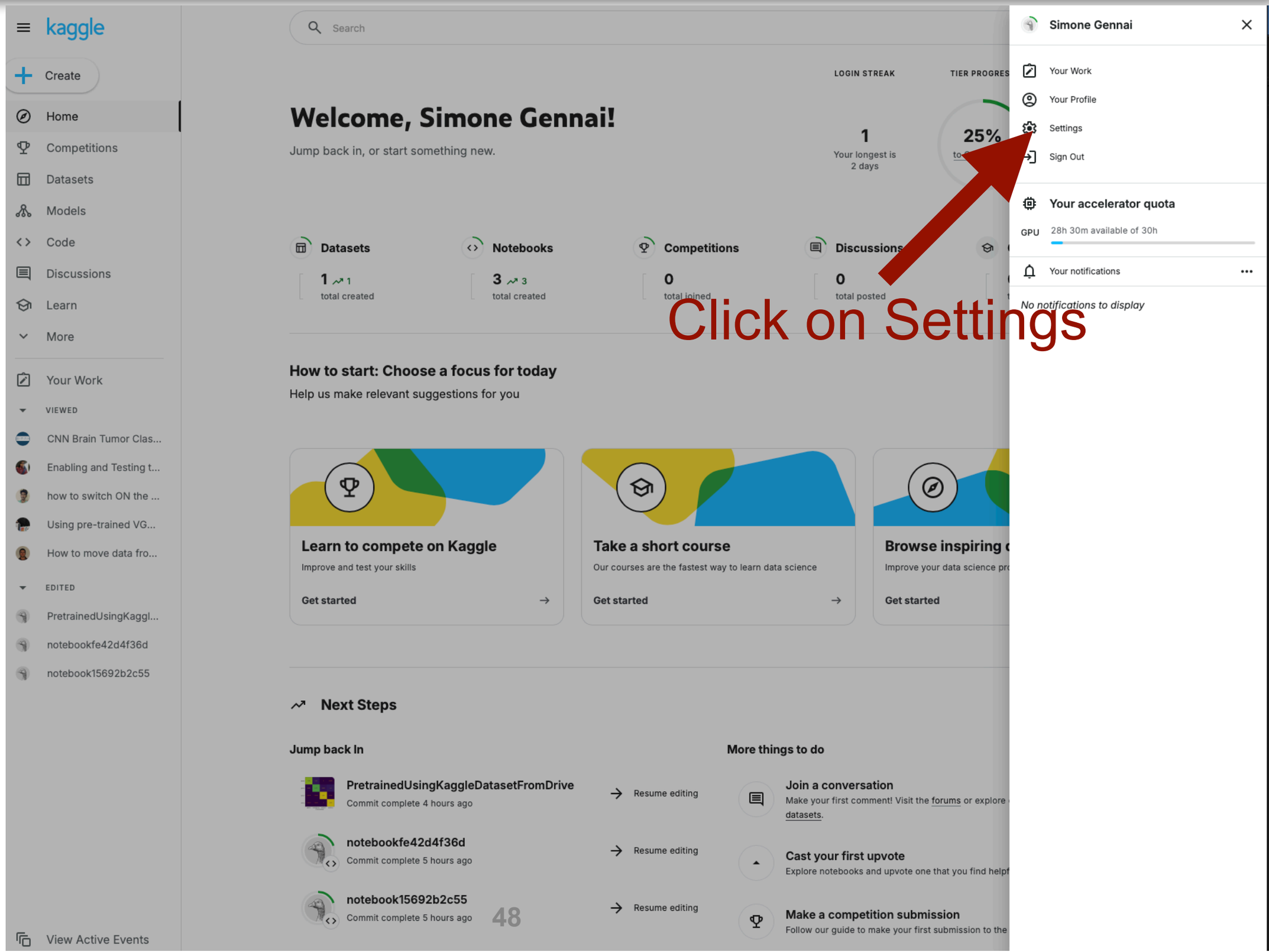




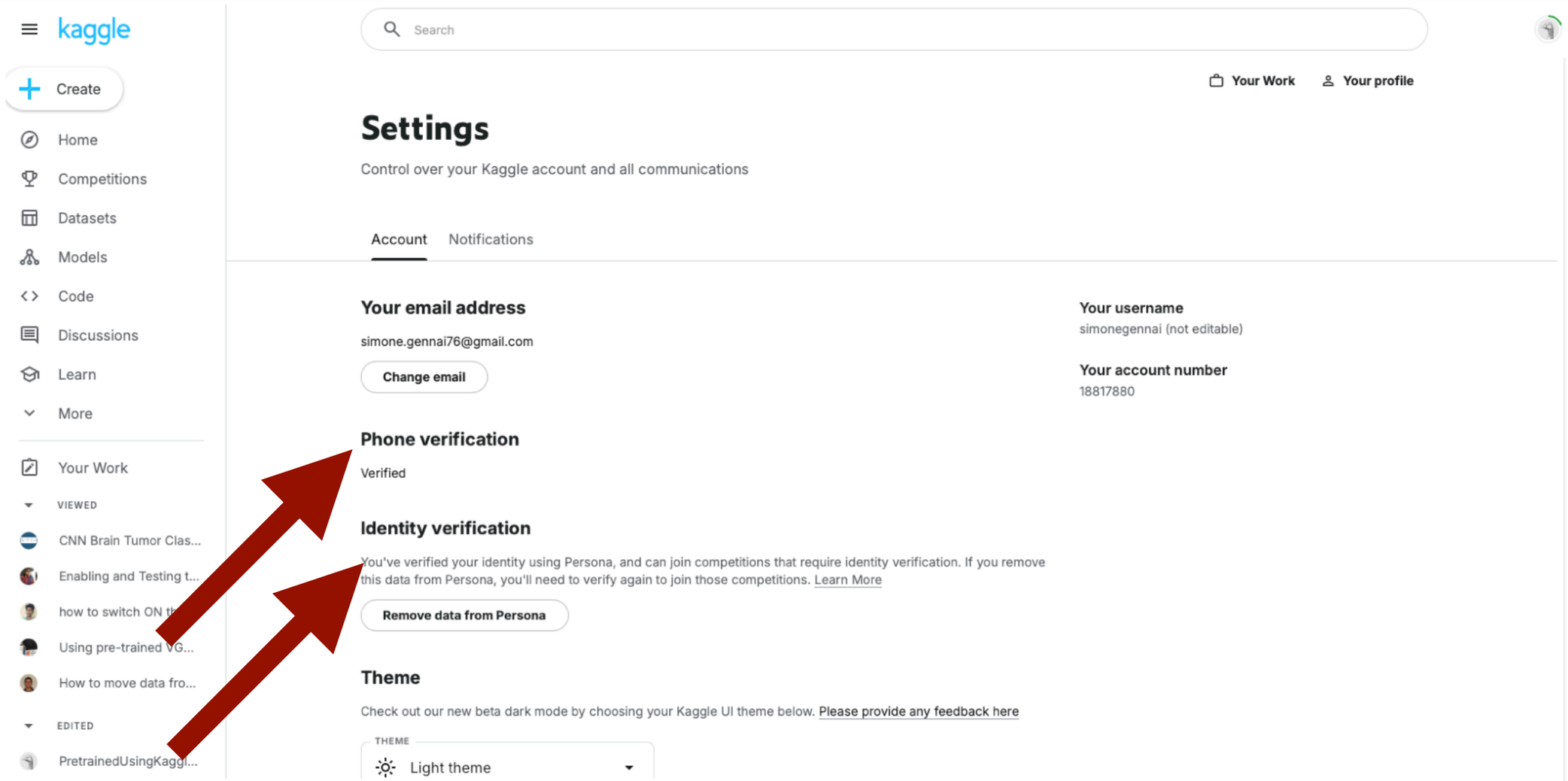
How to login/register on kaggle



How to login/register on kaggle

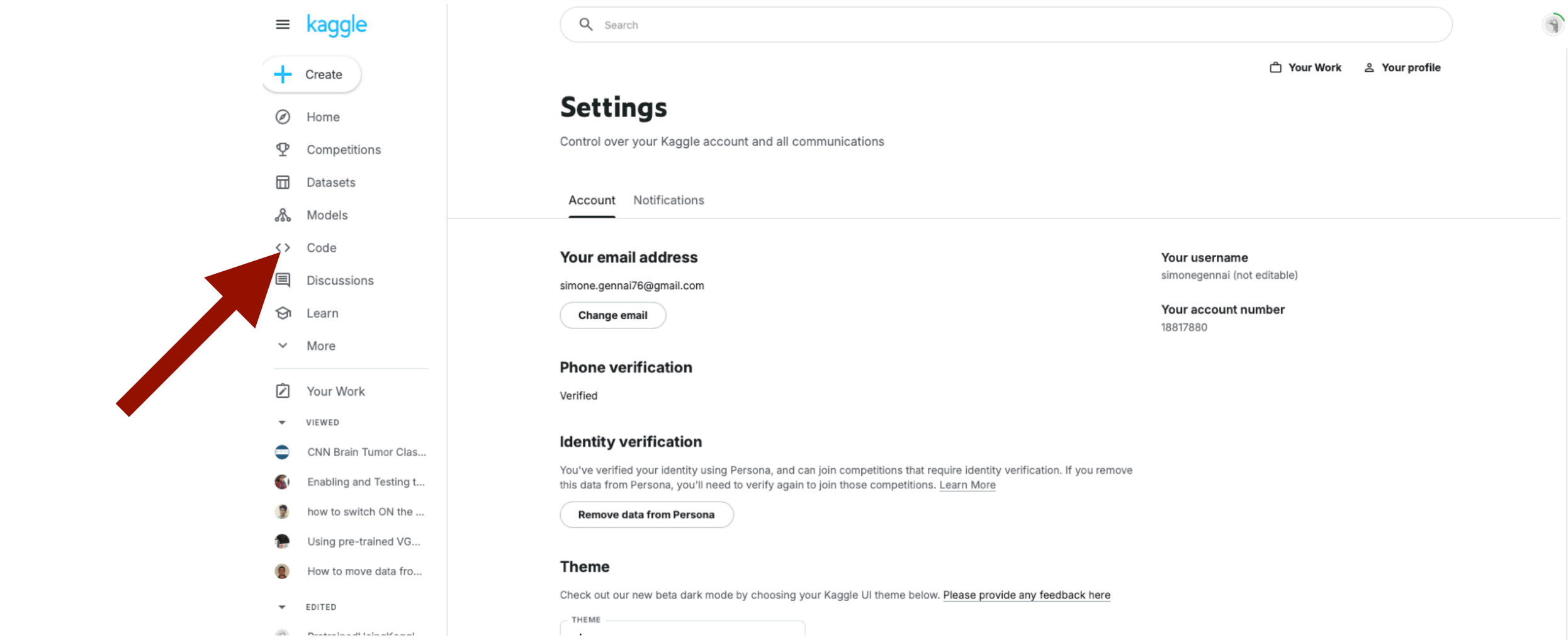


How to login/register on kaggle

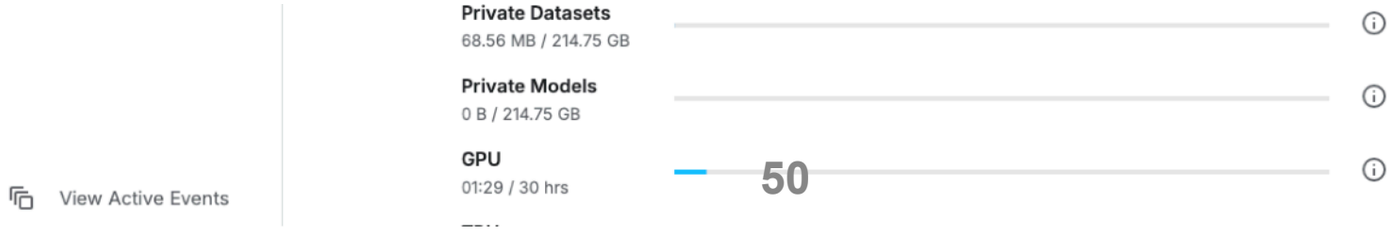


You need to perform phone verification and identity verification
in order to access 30h of GPUs per week

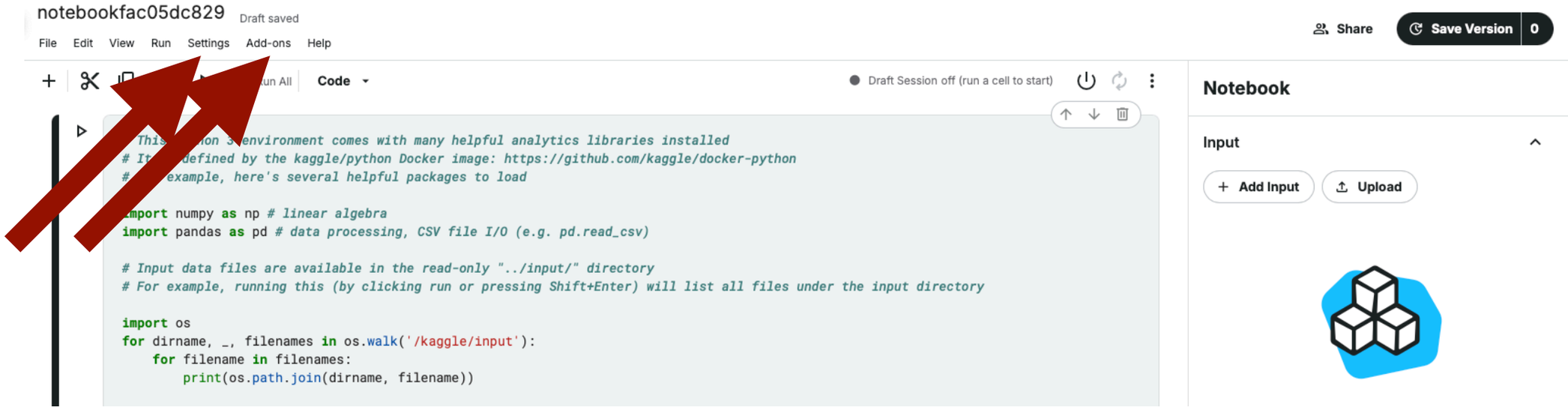
How to login/register on kaggle



Then you click on Code and then new notebook Under settings you can choose the accelerator



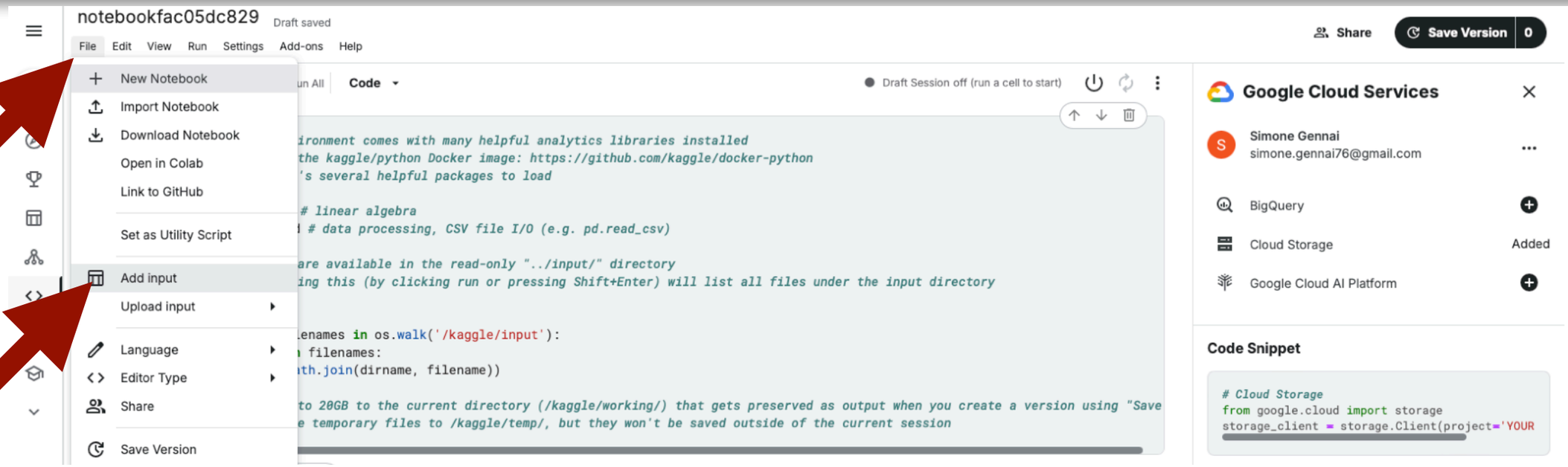
How to login/register on kaggle



- Under Add-ons click on Google Cloud services and link your google account
  - Add cloud storage among the possibilities

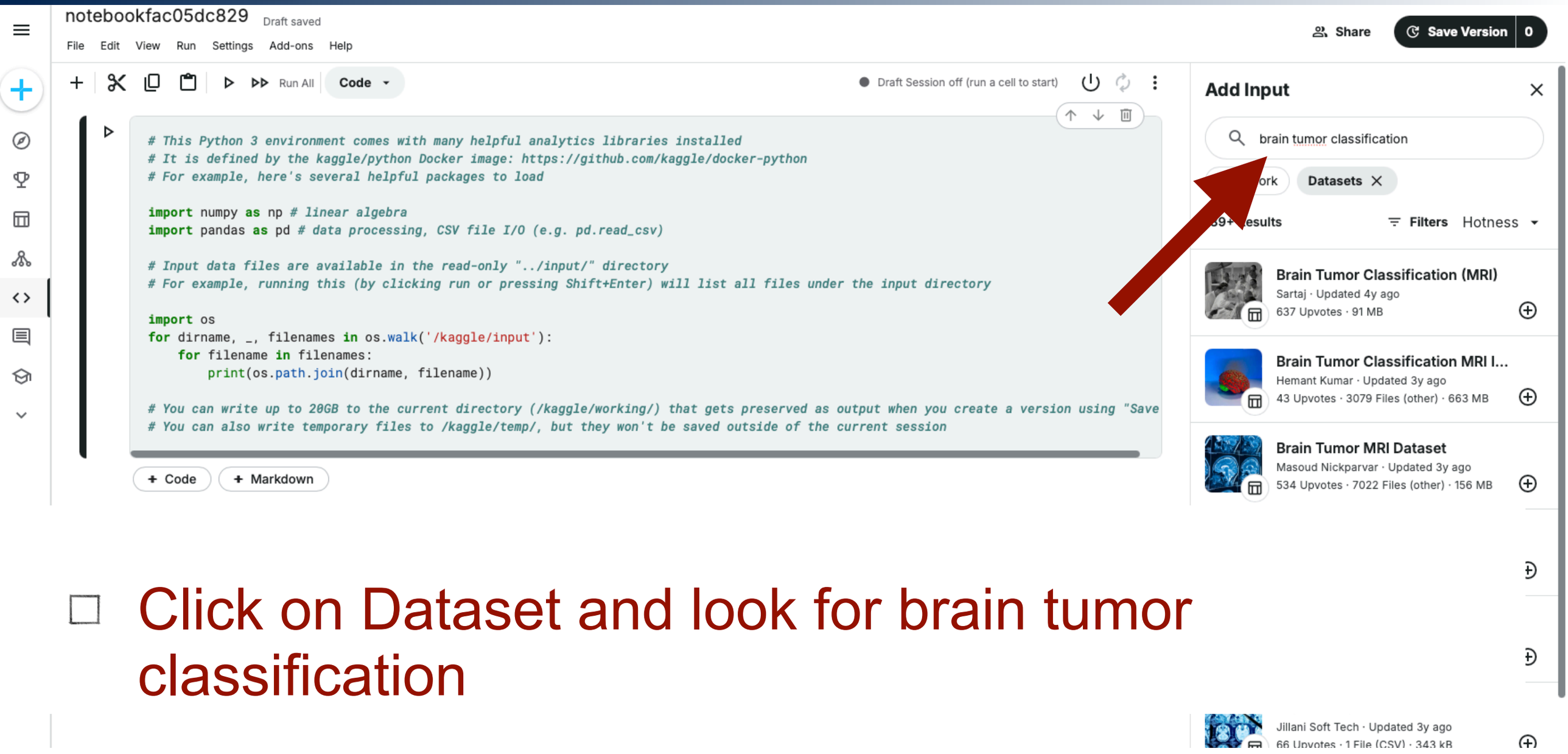
- This is needed in order to access the tensorflow based pre-trained models!

How to login/register on kaggle



- Finally you add the kaggle dataset with tumor images: from File select “Add Input”

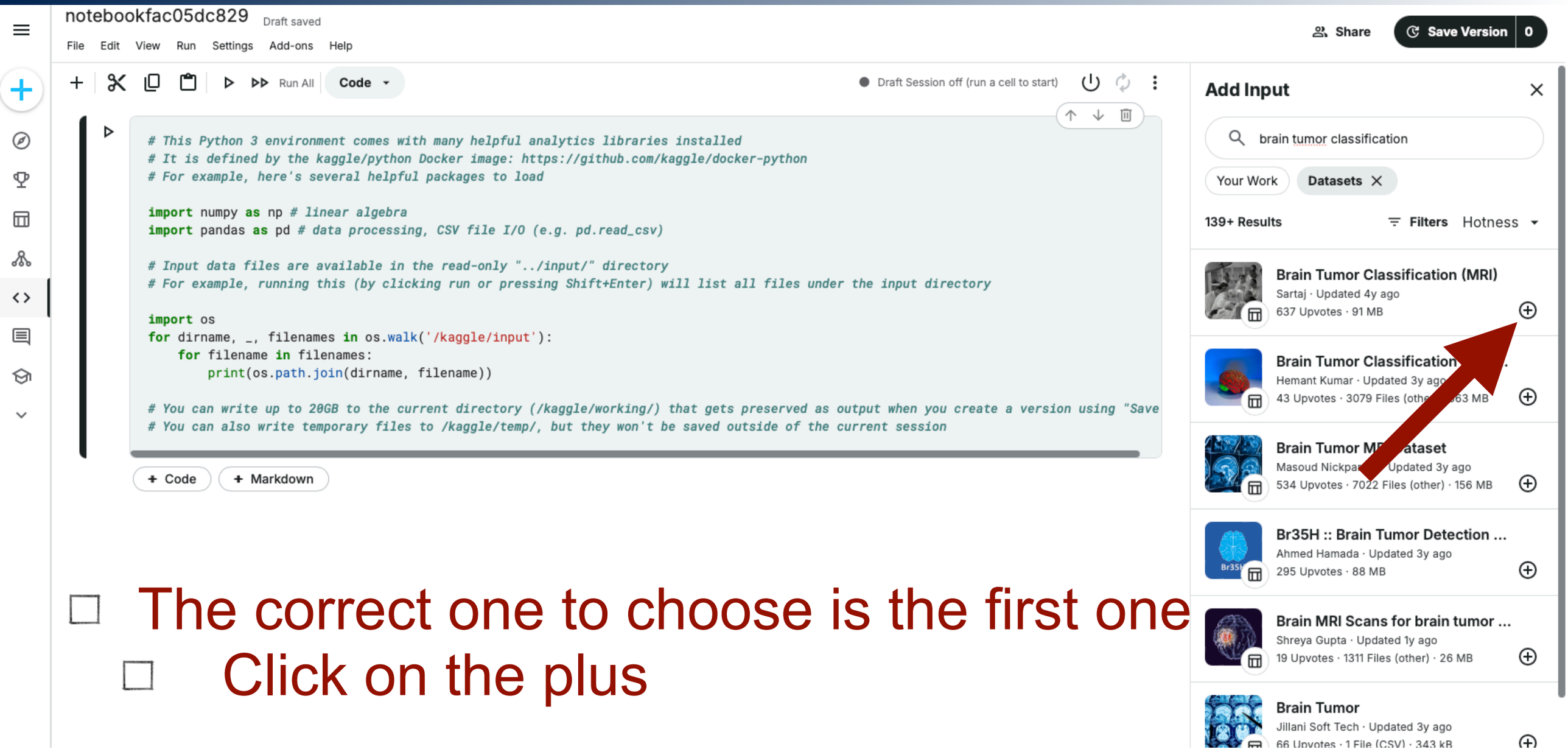
How to login/register on kaggle



- Click on Dataset and look for brain tumor classification

Simone.Gennai@cern.ch

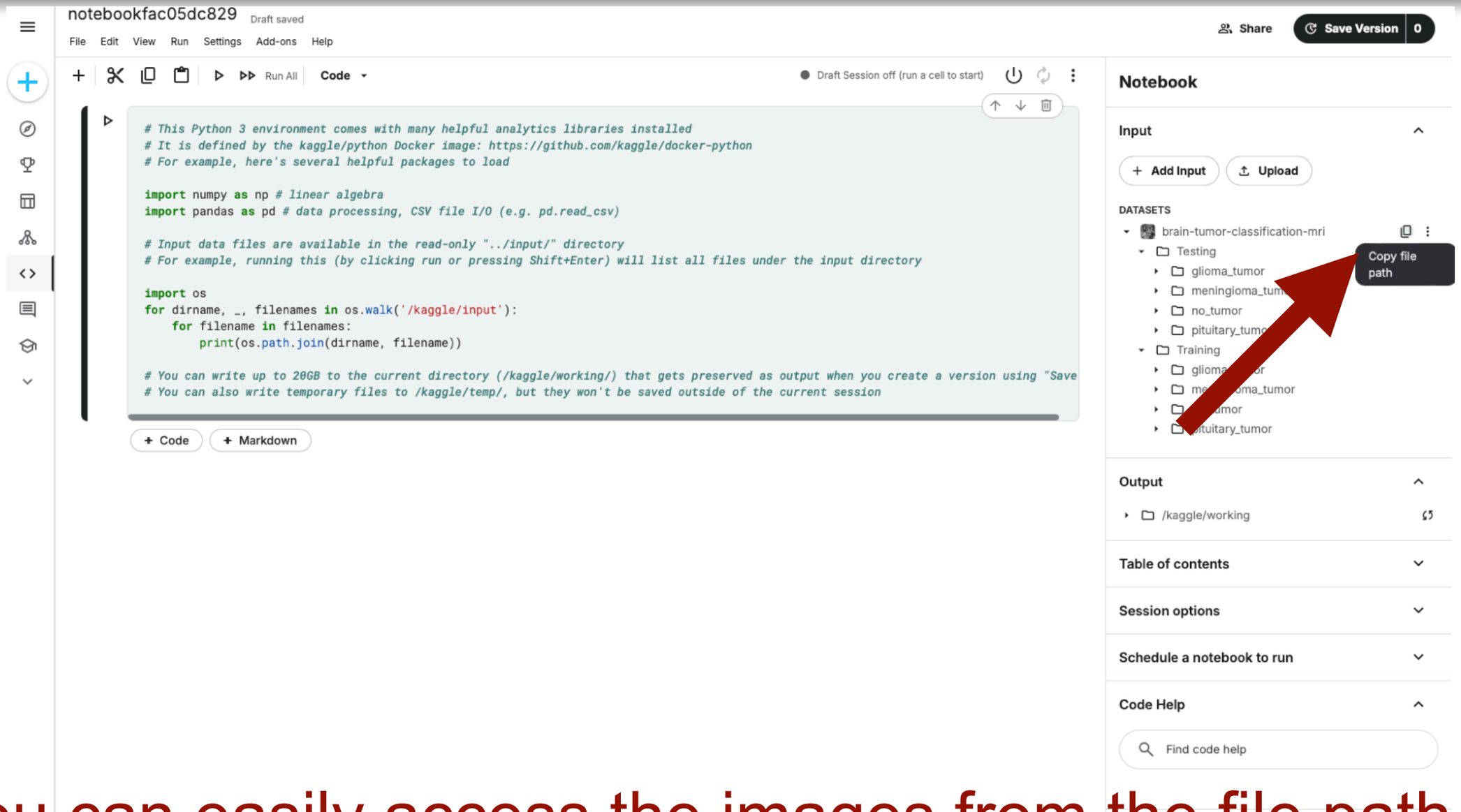
How to login/register on kaggle



- The correct one to choose is the first one
  - Click on the plus

Simone.Gennai@cern.ch

How to login/register on kaggle



You can easily access the images from the file path appearing at that link.

How to prepare the dataset

- Images are already divided between training and testing dataset
  - Each directory contains 4 sub directories with the 3 kinds of tumors and one without tumor
    - This means you will have to prepare a multiclassification model
- You need a package to “feed” the images to the training and prediction steps of the model.
  - Something that automatically takes care of the different categories
- I would suggest to use the package named : ImageDataGenerator
  - It can also be used for augmenting the dataset, rotating and flipping the images (even if I would not use it now as it takes longer …)
- For what regards the pre-trained models I would suggest to use these three:
  - MobileNetV2, VGG16, YOLOV8
    - The first two are made with tensorflow, the last one with PyTorch and compare results
- Finally, I would invite you to write your own “small" model with CNN in order to see the difference in performance

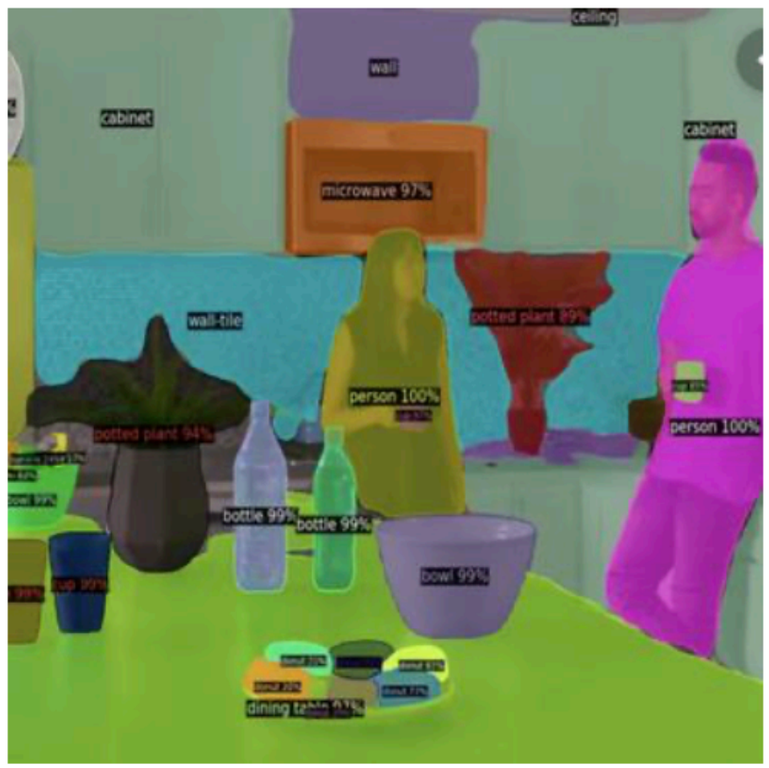
Simone.Gennai@cern.ch

Back up

Simone.Gennai@cern.ch

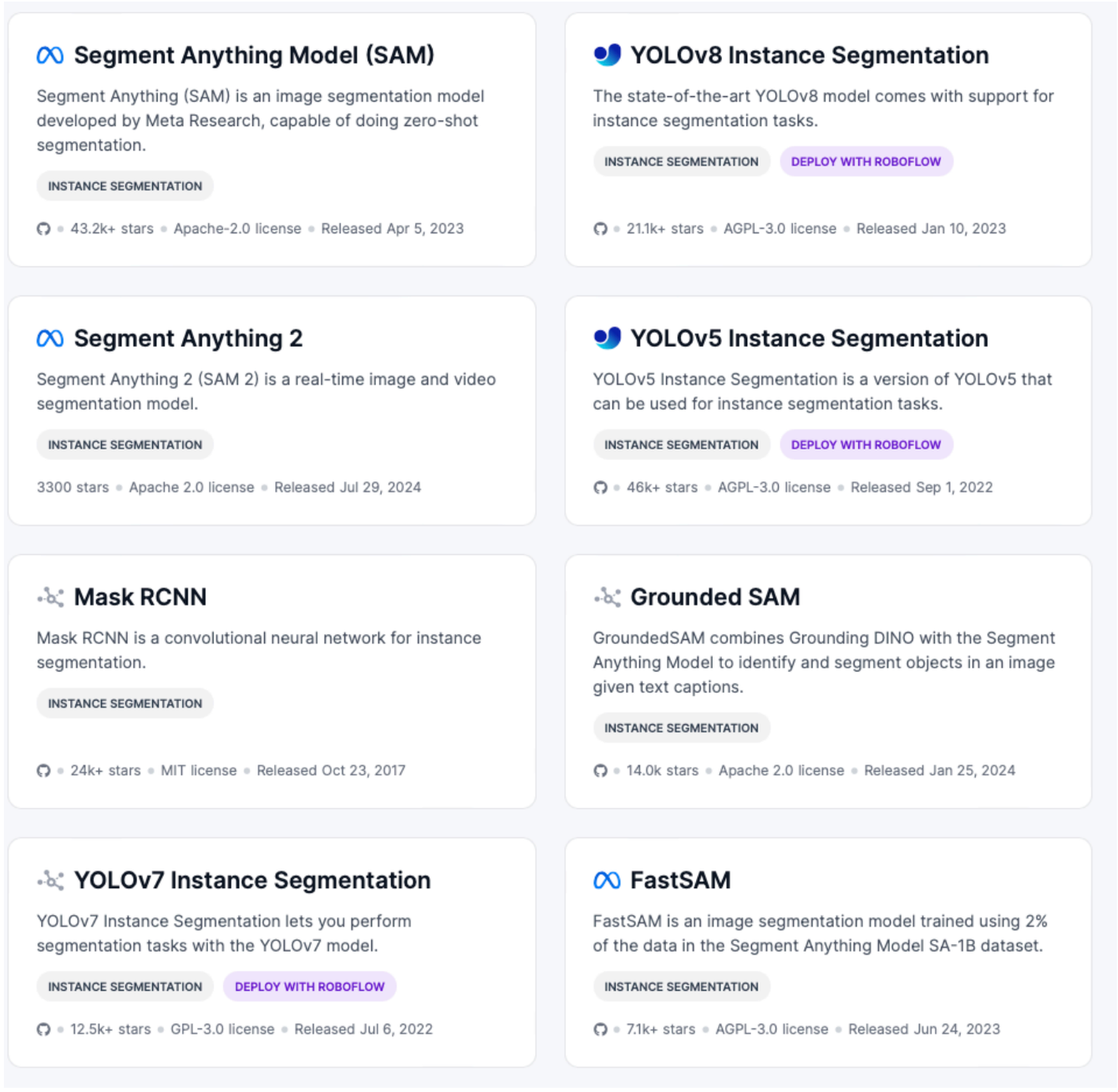
Pretrained models for segmentation

- We can have different types of segmentation:
  - Semantic segmentation: each pixel in a image is associated with a class
    - Nearby objects of the same class are merged together
  - Instance segmenation: we are only interested in defining the boundaries of the objects
    - We do not care if they are from different classes or the same one
  - Panoptic segmentation
    - It is the merging of the two above



Availabe models

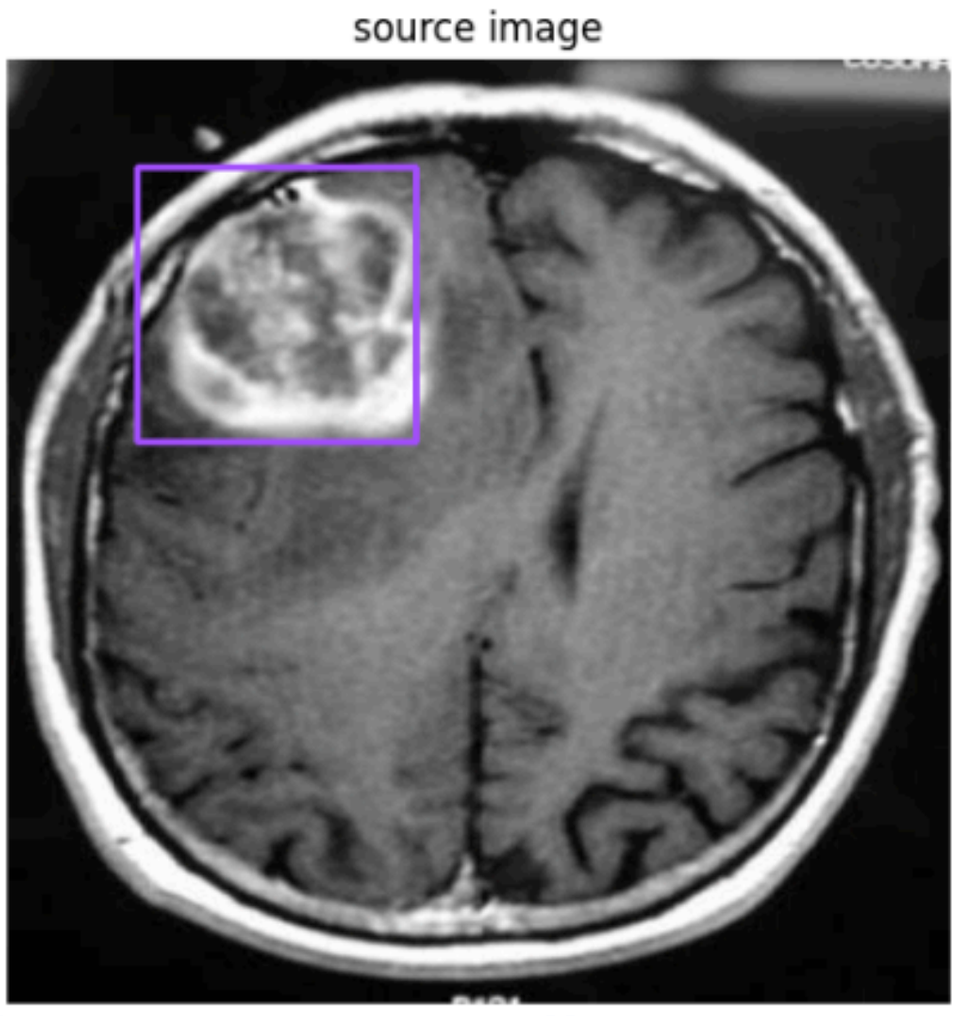
- There are many models available depending on what we want to do
  - YOLO, SAM, DINO, etc. etc.
  - The usage is very similar among them
- They create masks that can be overlaied to the original image to highlight the region of interest
  - This region can be send to a second model for classification

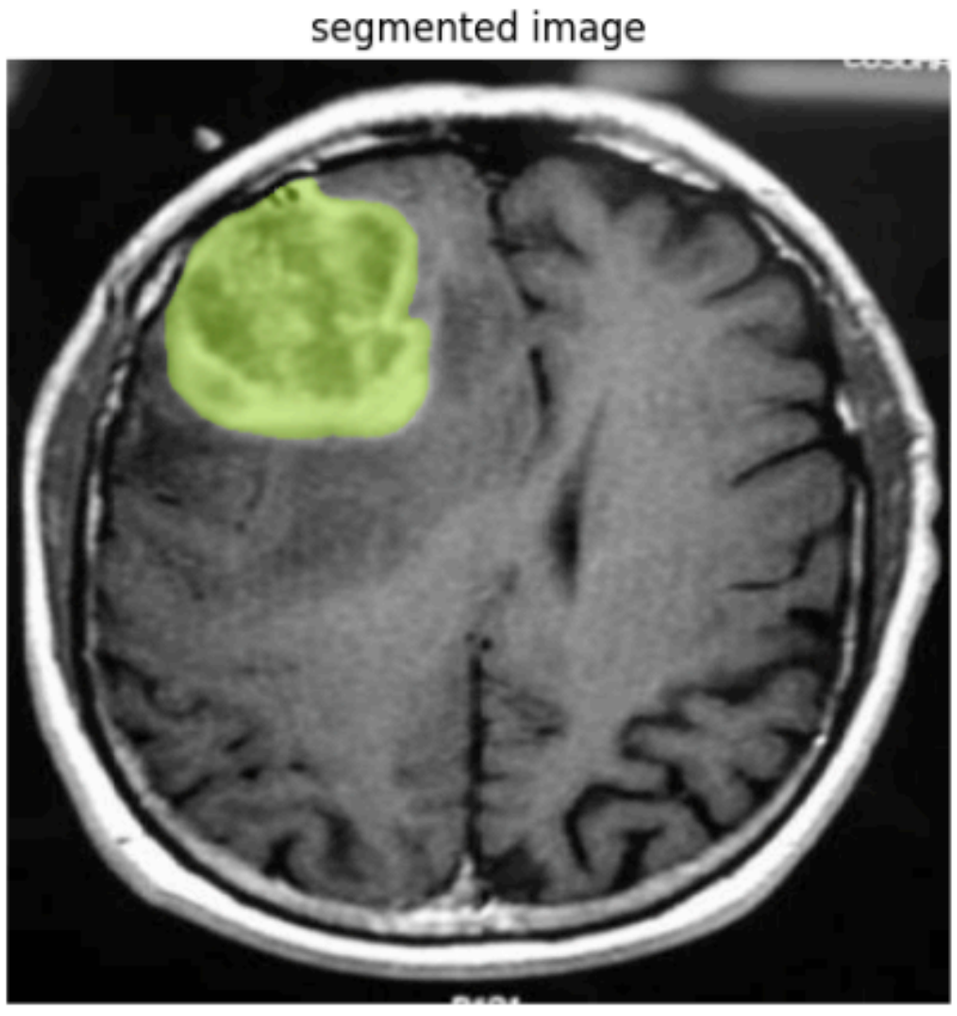


Simone.Gennai@cern.ch

Segmenting medical images

- One of the most useful (to me) use of ML is its application to medical images



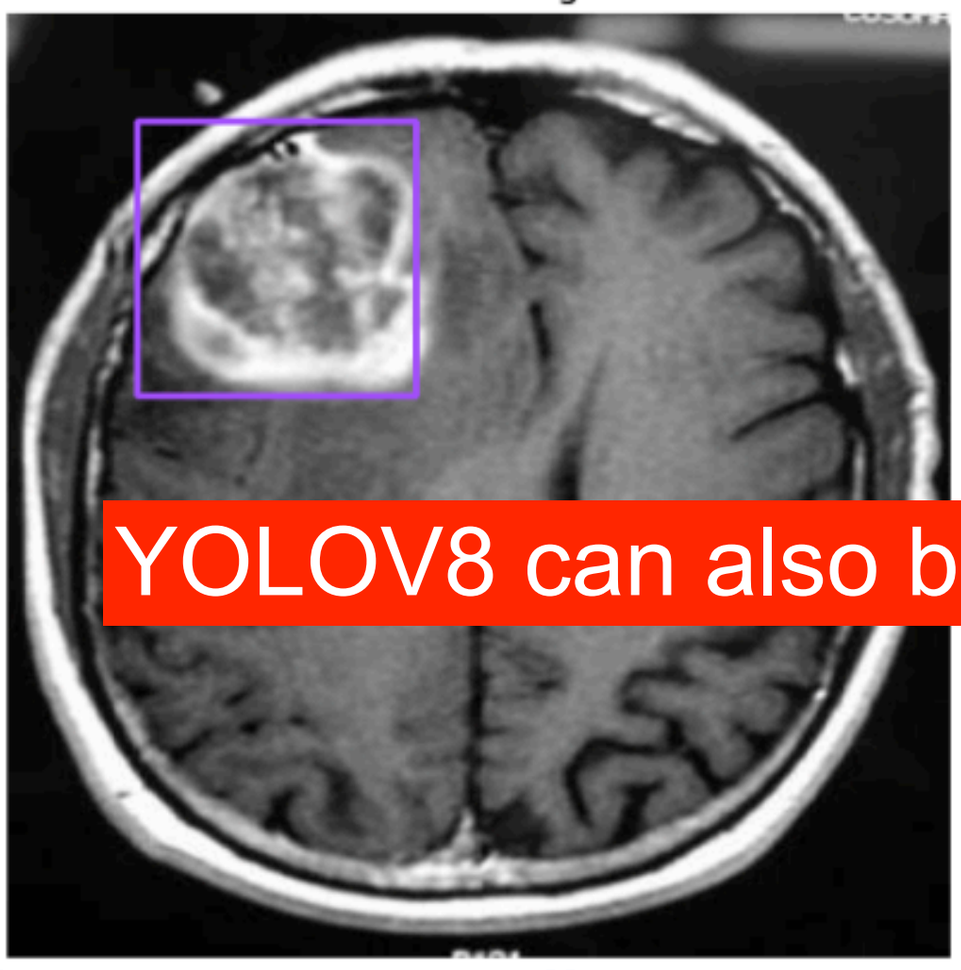


- of
- ected
- /
- tune
- e
- tuning on small image sample

Simone.Gennai@cern.ch

Segmenting medical images

- One of the most useful (to me) use of ML is its application to medical images
  - source image
  - segmented image
  - of
  - ected
  - /
  - tune
  - e
  - tuning on small image sample



YOLOV8 can also be used for Classification!!!!

Simone.Gennai@cern.ch