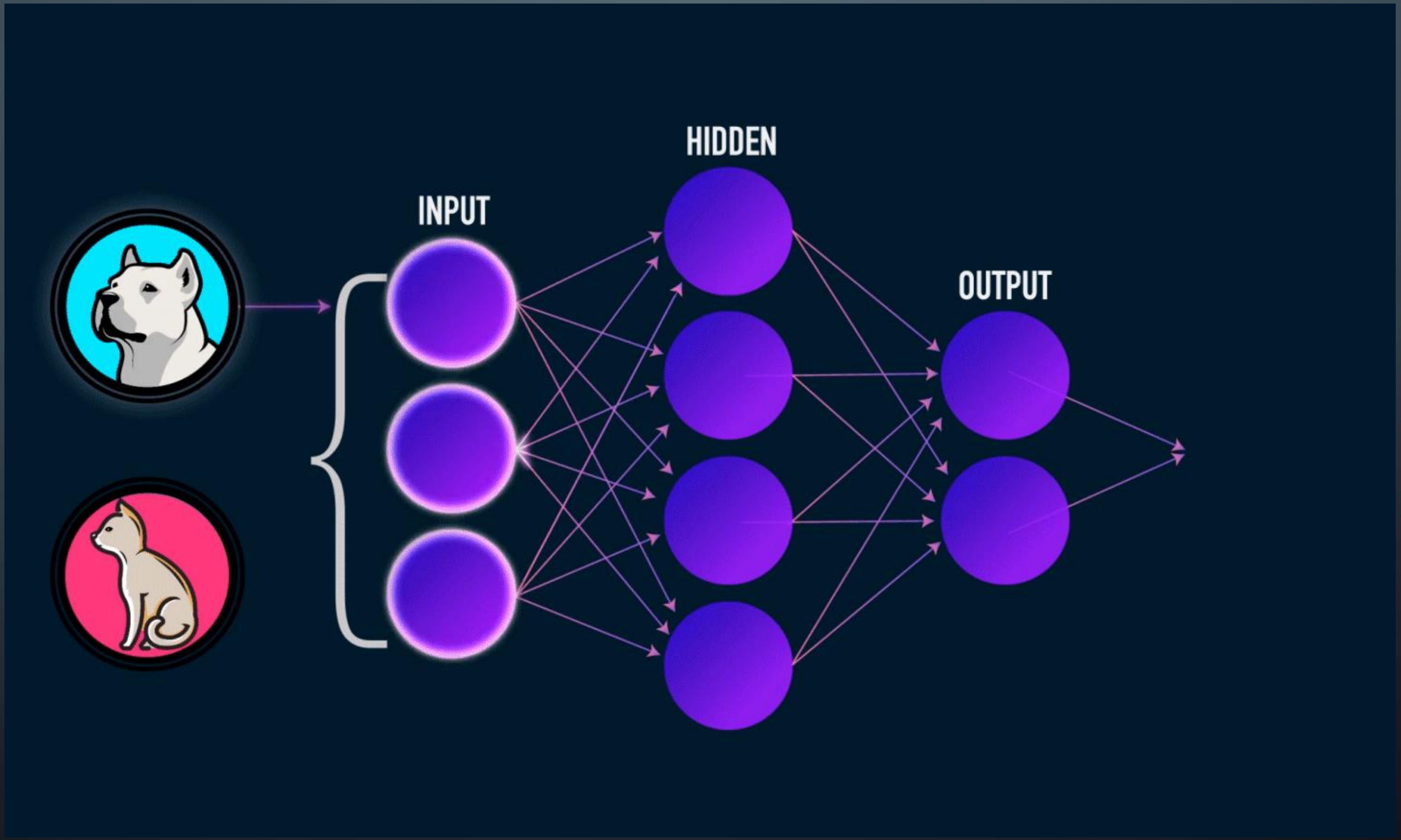
NEURAL NETWORKS 101

HOW TO PROGRAM ONE QUICKLY FROM SCRATCH THEN WITH TENSORFLOW



https://github.com/hube12/NeuralNetwork101

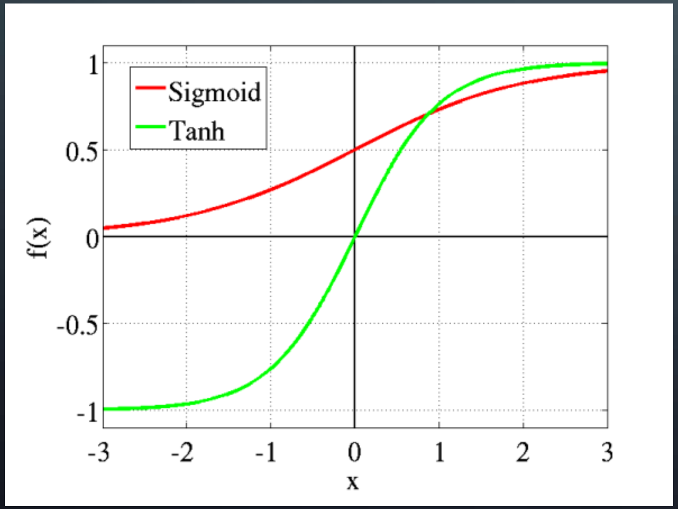
DISCLAIMER

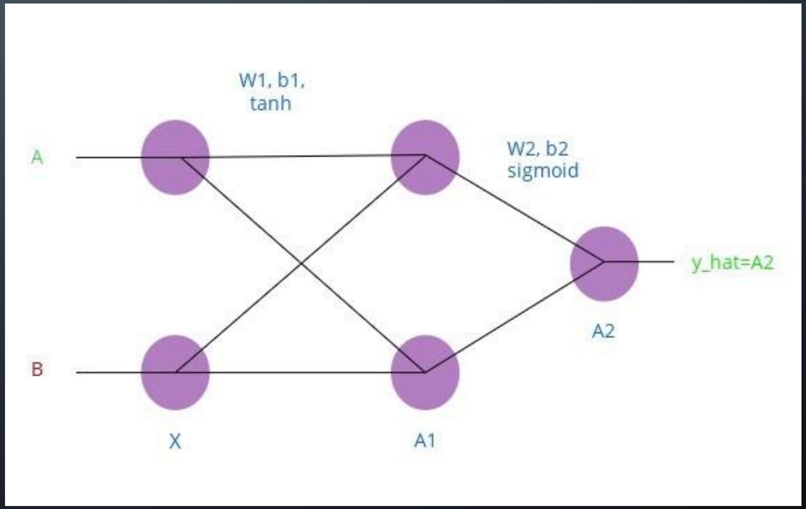
- I am not a neural network expert
- This is only intended for learning purpose (not refined nor applicable)
- The dataset used are conditionned
- We are going to see a limited set of neural network, so called CNN
- We dont have much time for this presentation

- OPEN Xor Neural Networks CO Open in Colab https://colab.research.google.com/github/hube12/NeuralNetwork101/blob/master/Jupyter%20Notebooks/neural%20network.ipynb

A NEURAL NETWORK FROM SCRATCH

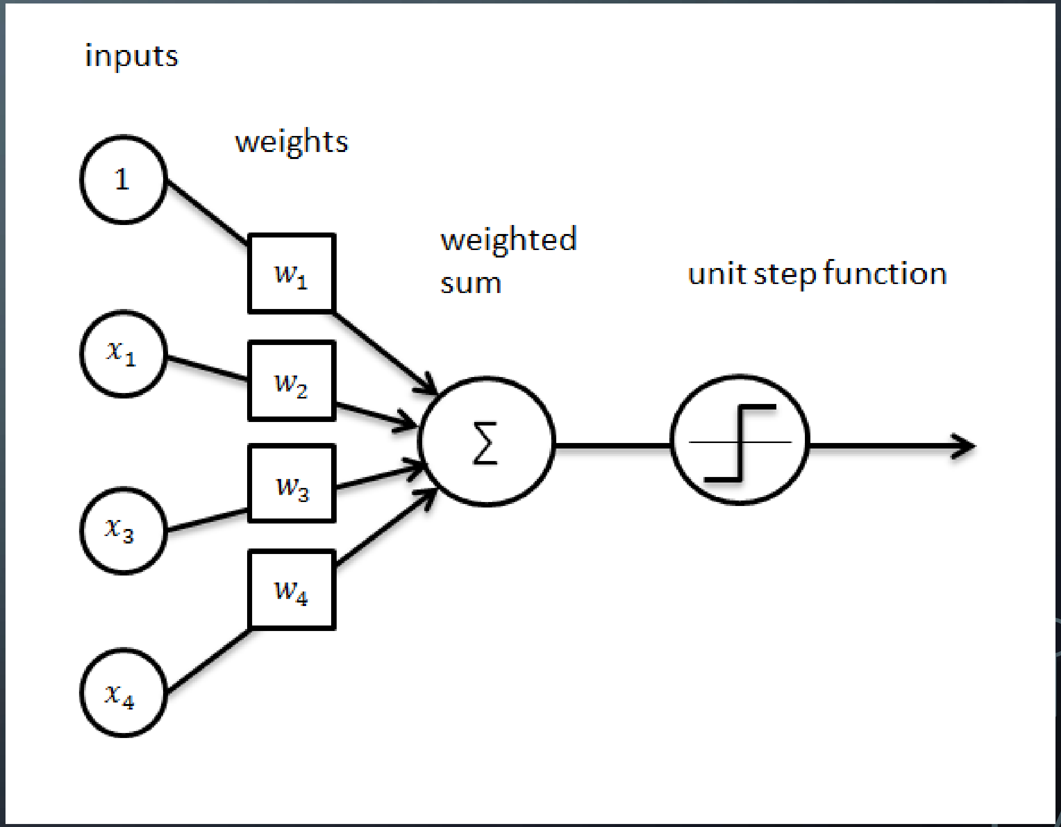
- How to make a neural network capable of understanding what is a xor function…
- We obviously have 2 input then we had one hidden layer of same size to not complexify and since xor output is only one class we have one final neuron
- We will use 2 activations functions : Tanh and sigmoid





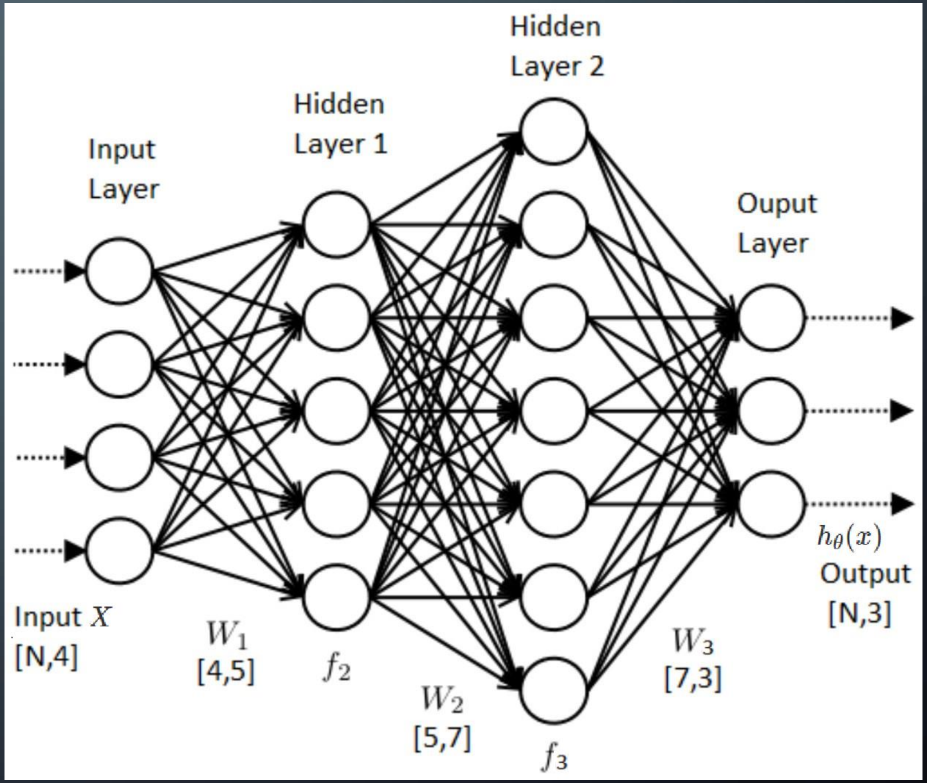
A NEURON

- For each neurons we compute the sum of the input times the weights given between neurons of different layers. We had a bias that is neuron dependant then we go through an activation function that say yes or no.

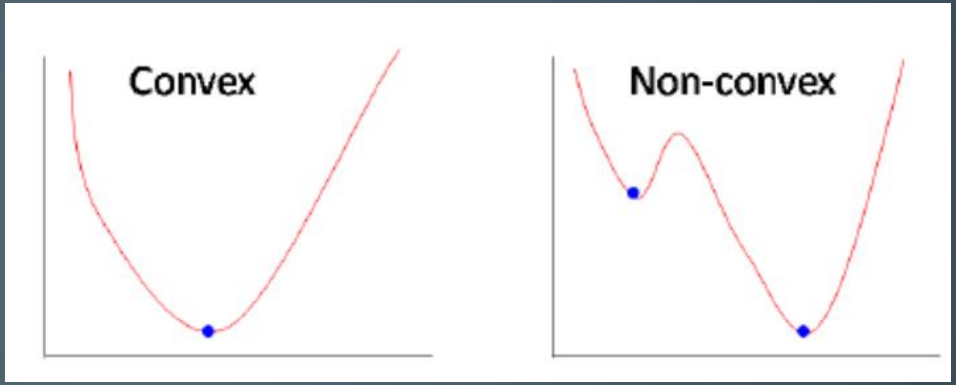


A LAYER

- Just putting together a lot of neurons then applying for each of them by sequence the same rules, we call that forwarding, meaning we go from the input layer to the output layer by just doing simple maths



COST FUNCTION



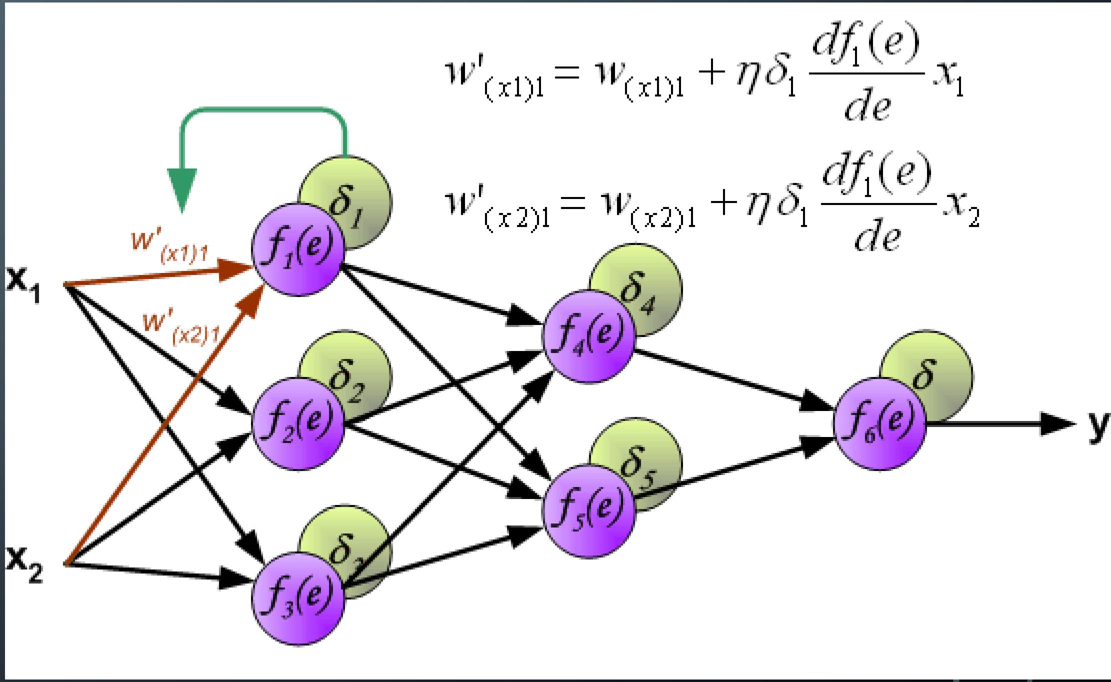
- Let’s start with defining the general equation for the cost function. This function represent the sum of the error, difference between the predicted value and the real (labeled) value. $J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\mathrm{Cost}(h_\theta(x^{(i)}), y^{(i)})$
- Our goal is to optimize the cost function so we need to find min ***J(ϑ)***. But Sigmoid function is a “non-convex” function (“*Image 15*”) which means that there are multiple local minimums. So it’s not guaranteed to converge (find) to the global minimum. What we need is “convex” function in order gradient descent algorithm to be able to find the global minimum (minimize J(θ)). In order to do that we use ***log*** function.

$$\mathrm{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$

BACKPROPAGATION

- What we want to do is minimize the cost function ***J(ϑ)*** using the optimal set of values for ***ϑ*** (weights). Backpropagation is a method we use in order to **compute the partial derivative of *J(ϑ)*.**
- We multiply the gradient by a learning rate factor to refine the search of the global minimum

$$
\begin{aligned}
&\text{Repeat } \{ \\
&\quad \theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \\
&\}
\end{aligned}
$$



INTRODUCTION TO MNIST

- The **MNIST database** (Modified National Institute of Standards and Technology database) is a large database of handwritten digits that is commonly used for training various image processing systems.
- http://yann.lecun.com/exdb/mnist/ is maybe the most famous

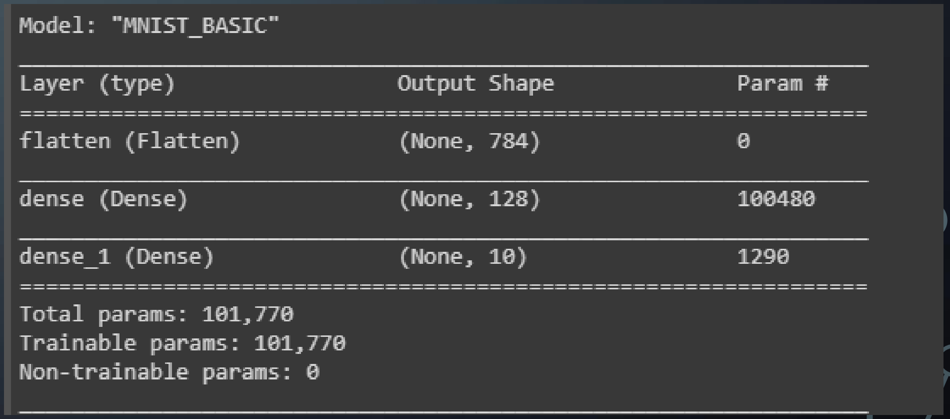
- OPEN MNIST BASICS Open in Colab https://colab.research.google.com/github/hube12/NeuralNetwork101/blob/master/Jupyter%20Notebooks/MNIST%20BASICS.ipynb

LETS LOAD THE DATA

We have about 60000 images of 28x28 in greyscale (one channel) for training and 10000 for testing.

LETS MAKE A SIMPLE MODEL

Only one hidden layer of 128 neurons, we still need to flatten the input so it can be used so we convert our 28x28 image to a 784 tensor then after the hidden layer we collect the only 10 possibles classes.



```
Model: "MNIST_BASIC"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 784)               0
_________________________________________________________________
dense (Dense)                (None, 128)               100480
_________________________________________________________________
dense_1 (Dense)              (None, 10)                1290
=================================================================
Total params: 101,770
Trainable params: 101,770
Non-trainable params: 0
_________________________________________________________________
```

RESULTS !

- Test loss: 0.11018784496150912
- Test accuracy: 0.9662
- Wow we got more than 96% accurate labels for only 12 epochs (meaning we inputed 12 times the data in the network so it effectively changed his weights only 12 times)
- Loss is also good: It is a summation of the errors made for each example in training or validation sets.

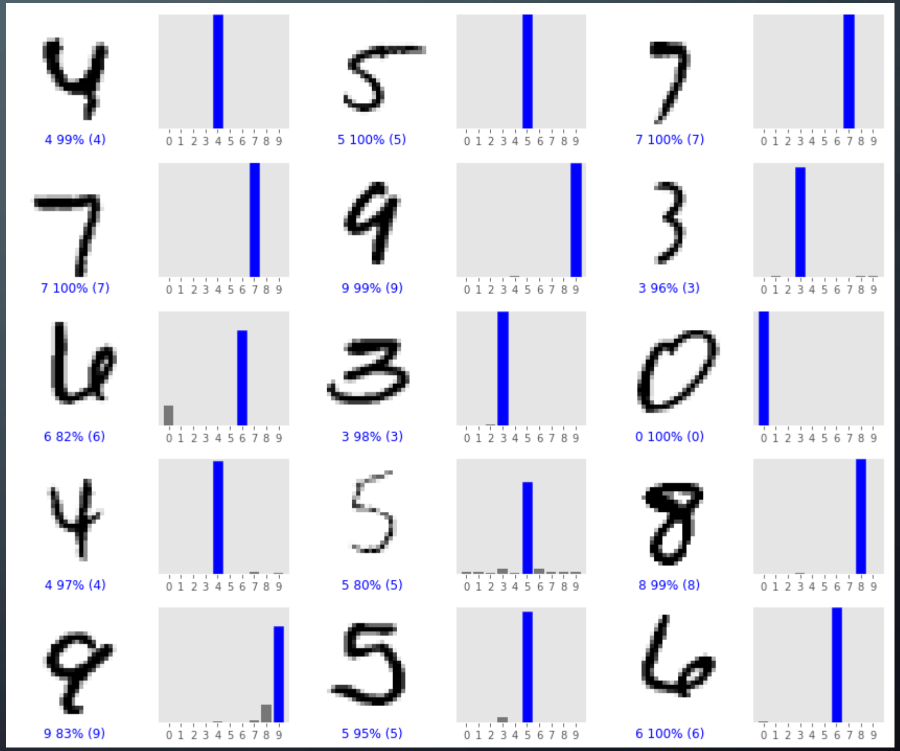
THE REALITY

Look we got it all right !

Awesome isnt it?

Actually it’s pretty BAD,

96.6% means 3.3 out of 100 will be wrong which in real life can have some serious consequences, plus the dataset is quite easy



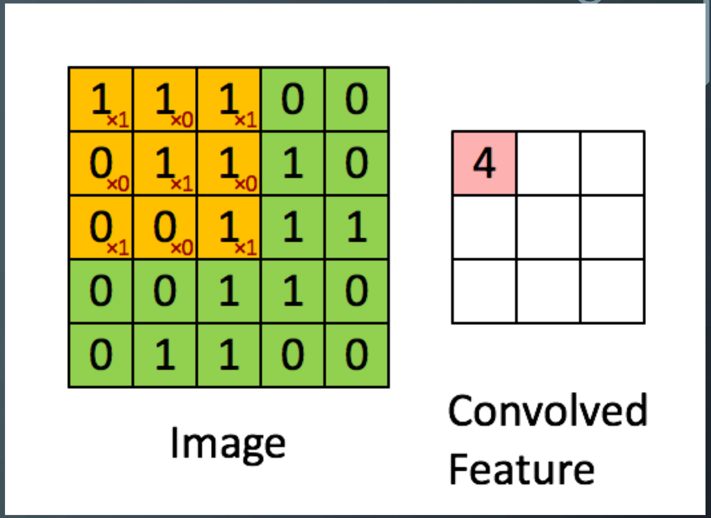
YOUR TURN

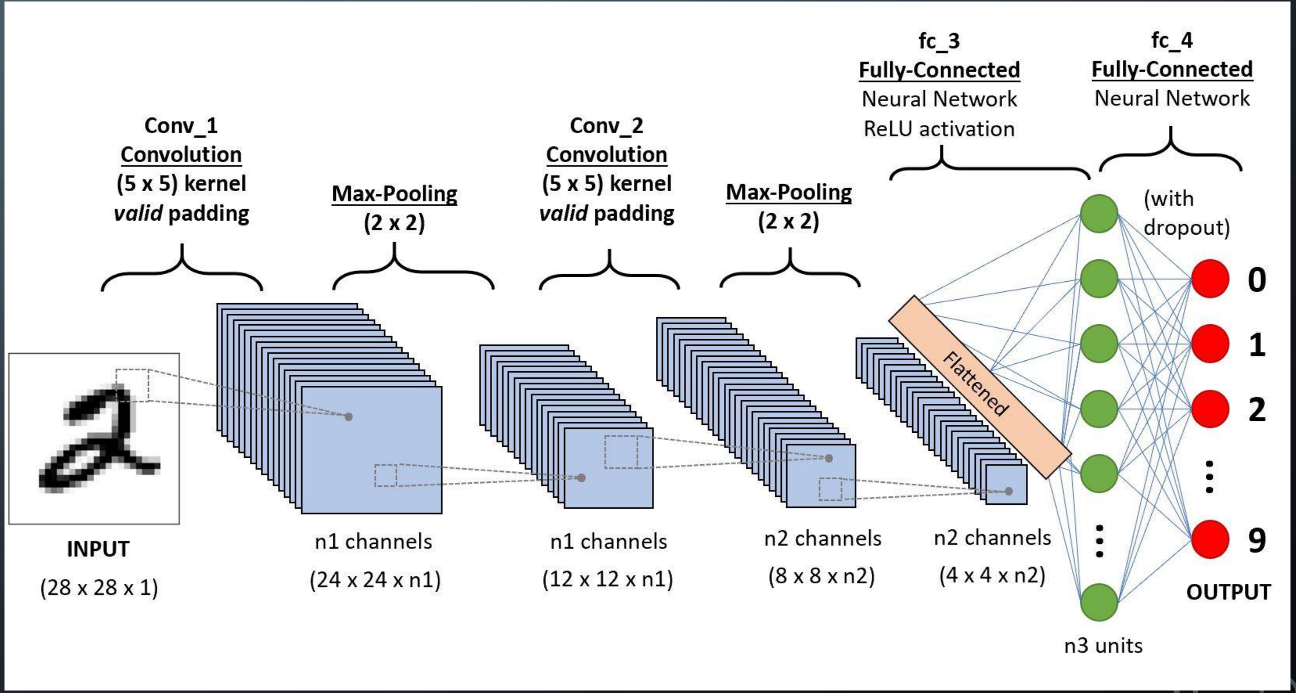
- Try to change this neural network to be a bit better
- For that just modify the model and that line :
- model.add(keras.layers.Dense(128, activation='relu'))
- Try to change 128 to more or less and change the activation function to some stated
- You can even try to duplicate that layer and add more hidden layers!

CONVOLUTIONAL NEURAL NETWORK

- A **Convolutional Neural Network (ConvNet/CNN)** is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other.

OPEN MNIST CNN Open in Colab
https://colab.research.google.com/github/hube12/NeuralNetwork101/blob/master/Jupyter%20Notebooks/MNIST%20CNN.ipynb





Conv_1 Convolution (5 x 5) kernel valid padding
Max-Pooling (2 x 2)
Conv_2 Convolution (5 x 5) kernel valid padding
Max-Pooling (2 x 2)
fc_3 Fully-Connected Neural Network ReLU activation
fc_4 Fully-Connected Neural Network
(with dropout)
Flattened
INPUT (28 x 28 x 1)
n1 channels (24 x 24 x n1)
n1 channels (12 x 12 x n1)
n2 channels (8 x 8 x n2)
n2 channels (4 x 4 x n2)
n3 units
OUTPUT
0
1
2
9

RESULTS !

Test loss: 0.027466179052060762

Test accuracy: 0.9914

Wow we truly are good here, that’s indeed way more like the result we should have from such dataset, only 9 out of 1000 errors and our losses are extremely low !

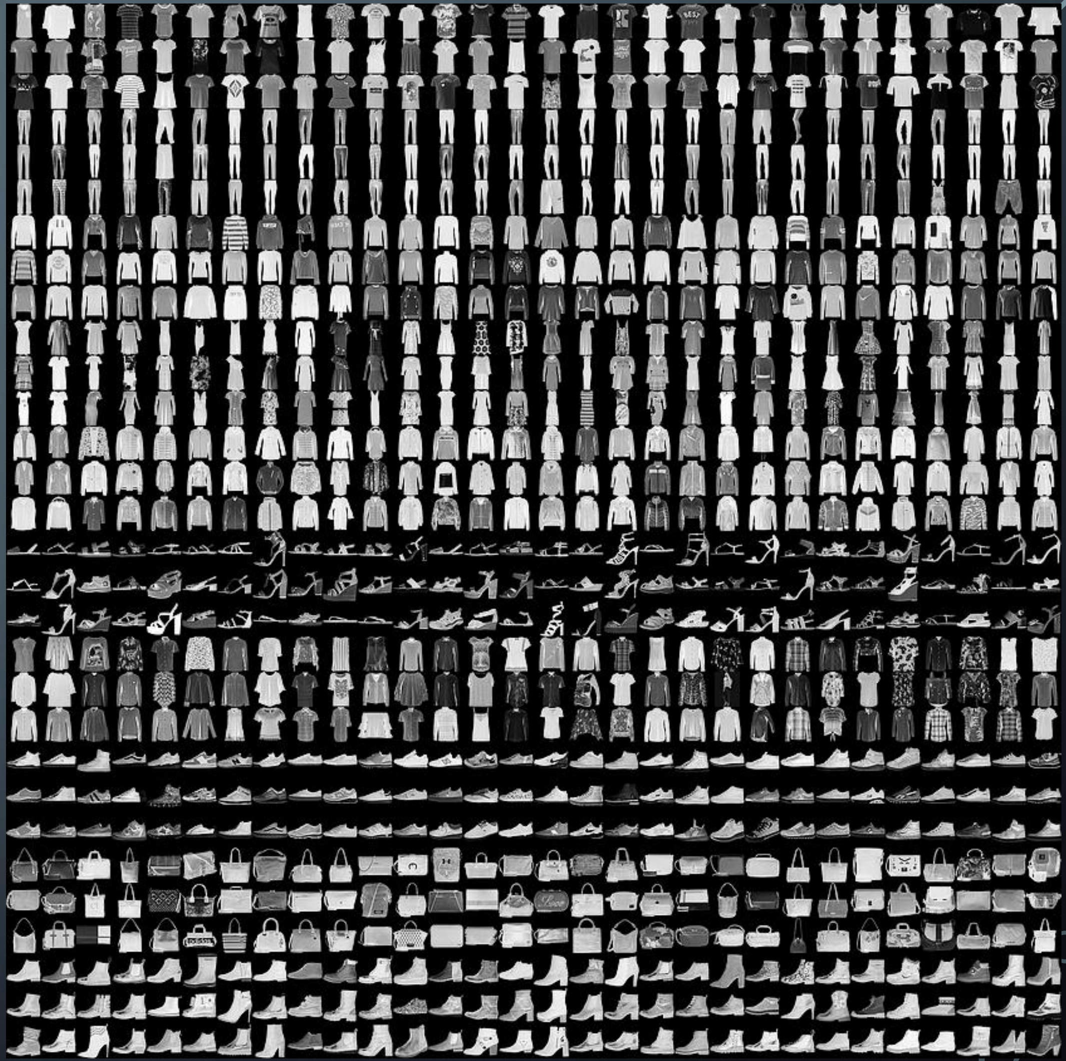
Of course this dataset is pretty good for CNN since its really feature dependent and CNN are good are getting the shape of an object in an image

FASHION MNIST: A LITTLE HARDER

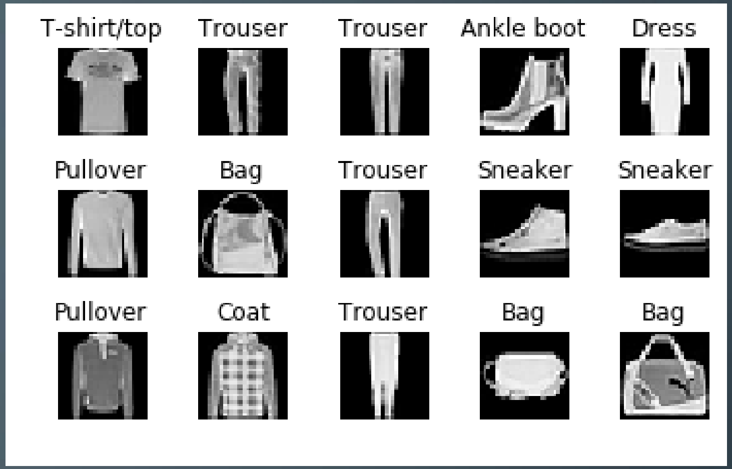
Fashion-MNIST is a dataset of Zalando's article images consisting of a training set of 60,000 examples and a test set of 10,000 examples.

Each example is a 28x28 grayscale image, associated with a label from 10 classes. We intend Fashion-MNIST to serve as a direct drop-in replacement for the original MNIST dataset for benchmarking machine learning algorithms.

It shares the same image size and structure of training and testing splits.



LOADING THE DATASET



- Open Open in Colab https://colab.research.google.com/github/hube12/NeuralNetwork101/blob/master/Jupyter%20Notebooks/FASHION%20MNIST%20CNN.ipynb and https://colab.research.google.com/github/hube12/NeuralNetwork101/blob/master/Jupyter%20Notebooks/FASHION%20MNIST%20CNN%202.ipynb
- (x_train, y_train), (x_test, y_test) = keras.datasets.fashion_mnist.load_data()
- As simple as that !

MODEL AND RESULT

- Let’s use the same CNN as for MNIST
- We obtain:
- Test loss: 0.21620800716280938
- Test accuracy: 0.9224
- Which is not bad considering how complicated the set is, to put in perspective the best on it is 0.967 on accuracy with WRN40-4 8.9M params

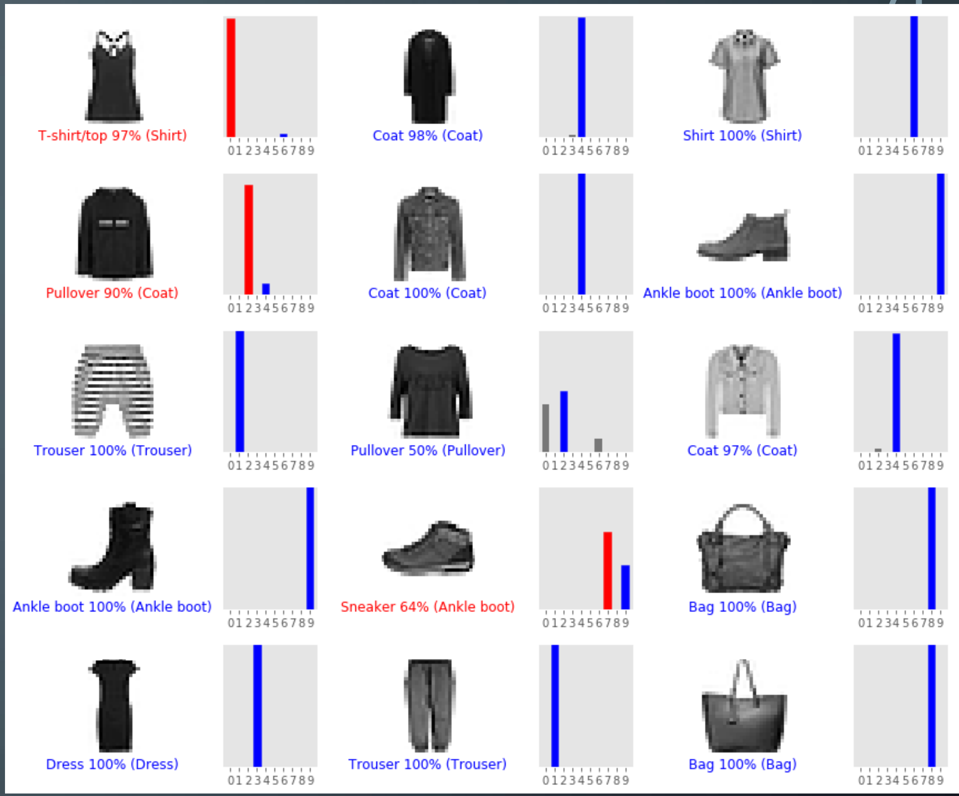
LET'S TRY THE SECOND ONE

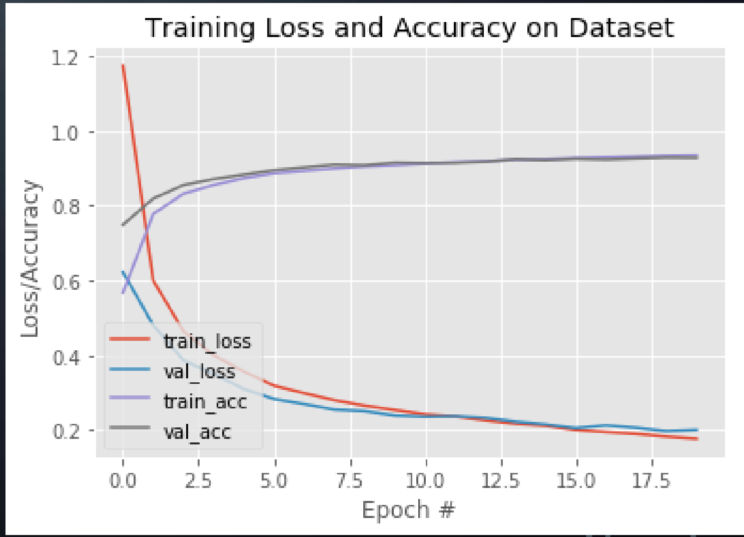
- We added two new layers and refined our flattening
- Test loss: 0.19887974030077457
- Test accuracy: 0.9311
- Still a bit low but we can hope that with

more epochs we can achieve 94 or 95%,

but we need to do something with the data

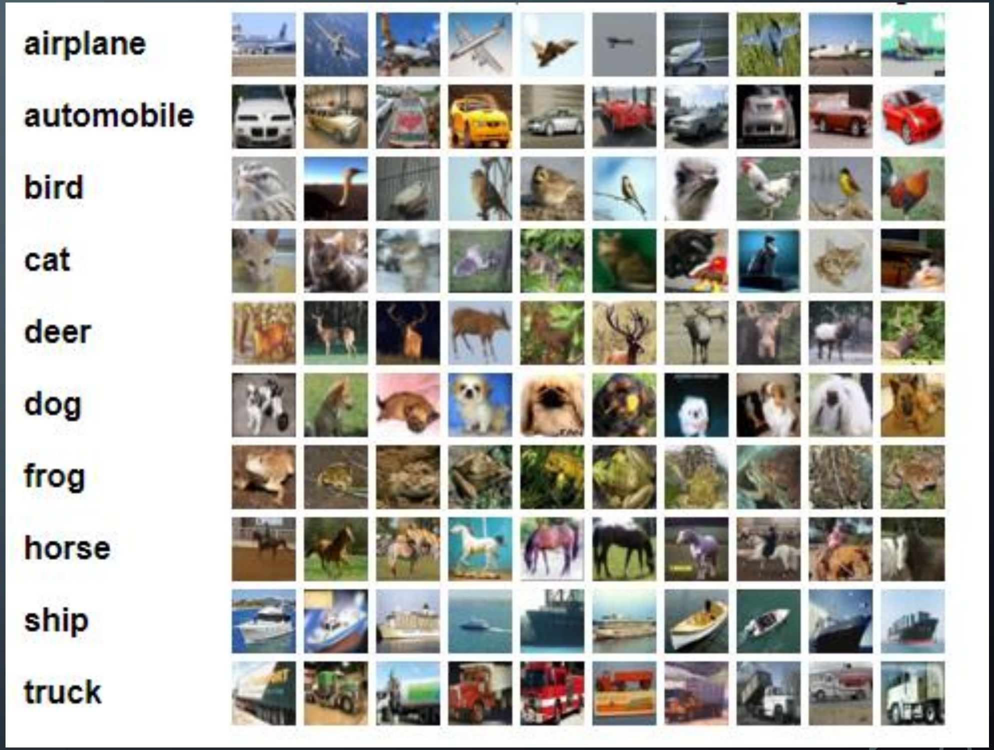
to go beyond, random flips and data augmentation !





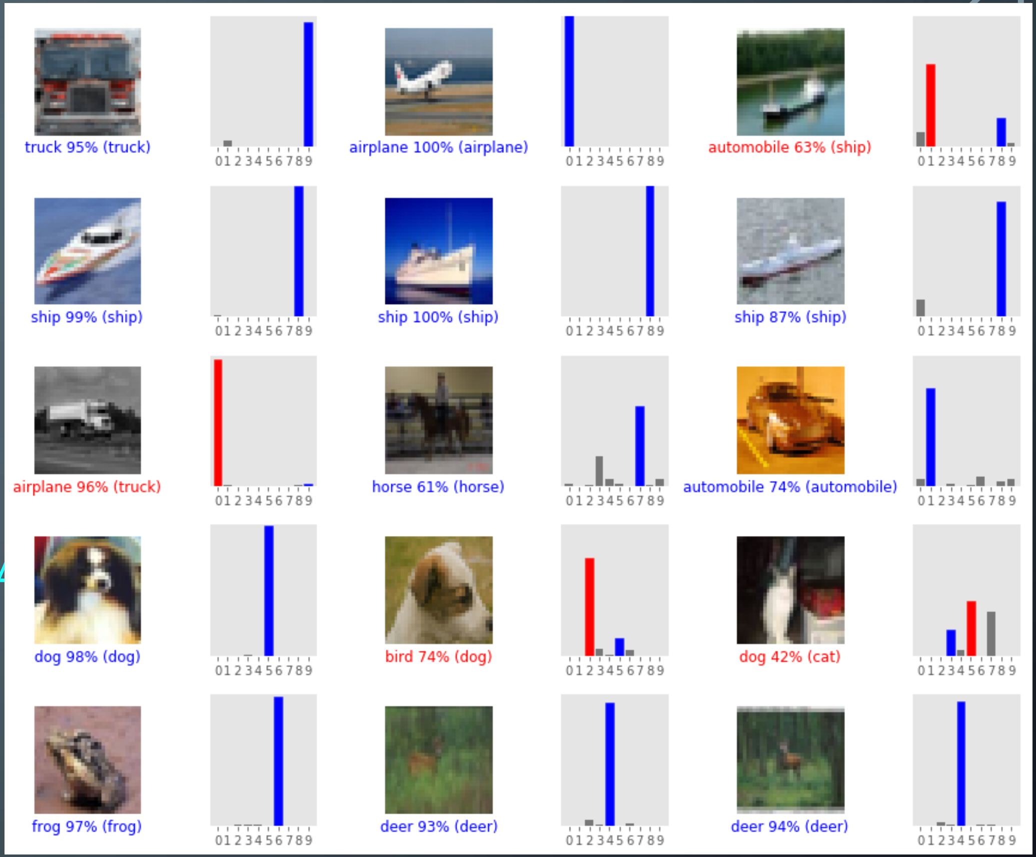
LAST DATASET: CIFAR10 (RGB !)

- The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.
- The classes are completely mutually exclusive. There is no overlap between automobiles and trucks. "Automobile" includes sedans, SUVs, things of that sort. "Truck" includes only big trucks. Neither includes pickup trucks.



RESULTS

- Open Open in Colab
  https://colab.research.google.com/github/hube12/
  /Jupyter%20Notebooks/CIFAR%20CNN.ipynb
- Test loss: 0.8869639729499816
- Test accuracy: 0.7111
- Ok our results are pretty bad, but in our defense we use colored images without converting them our doing some augmentation, plus we didnt optimize the model to fit the problem (i just made it fast enough to get a good enough result ;)


truck 95% (truck)
airplane 100% (airplane)
automobile 63% (ship)
ship 99% (ship)
ship 100% (ship)
ship 87% (ship)
airplane 96% (truck)
horse 61% (horse)
automobile 74% (automobile)
dog 98% (dog)
bird 74% (dog)
dog 42% (cat)
frog 97% (frog)
deer 93% (deer)
deer 94% (deer)