Computer vision tips and notions:

abv

## A typical model :

# Build the classification model

model = tf.keras.models.Sequential([tf.keras.layers.Flatten(),

tf.keras.layers.Dense(128, activation=tf.nn.relu),

tf.keras.layers.Dense(10, activation=tf.nn.softmax)])

* + Sequential: That defines a sequence of layers in the neural network.
  + Flatten: Remember earlier where our images were a 28x28 pixel matrix when you printed them out? Flatten just takes that square and turns it into a 1-dimensional array.
  + Dense: Adds a layer of neurons

Before passing the data into our model we should normalize the data ( since it’s gray scale image , its values are between 1 and 255 so divide all the data by 255 to get values between 0 and 1 ) , Then, we use first of all Flatten Layer to transform our 2D matrix 28\*28 ( the images ) into a 28^2 elms vector , and then we use our Dense layer to compute the relations between the image pixels ( we fixed arbitrary 128 neurons ) and since our result should be one of 10 classes of shoses , The last layer ( the output ) should have an input\_shape of 10 ,

* + Each layer of neurons need an activation function to tell them what to do. There are a lot of options, but just use these for now:
    - ReLU effectively means:

if x > 0:

return x

else:

return 0

* + - In other words, it only passes values 0 or greater to the next layer in the network.
* Softmax takes a list of values and scales these so the sum of all elements will be equal to 1. When applied to model outputs, you can think of the scaled values as the probability for that class. For example, in your classification model which has 10 units in the output dense layer, having the highest value at index = 4 means that the model is most confident that the input clothing image is a coat. If it is at index = 5, then it is a sandal, and so forth. See the short code block below which demonstrates these concepts. You can also watch this lecture if you want to know more about the Softmax function and how the values are computed.

## Upgrading the accuracy of our model

* We replace our hidden layer with 128 neurons by another dense one with 1024 neurons this time
  + We will notice that the time of the training will be increased
  + The accuracy will be increased too , which is great !

## Flatten Layer importance :

* If we attempt to execute our code without the flatten layer in the start of our model , the code will crush , SO it’s crucial to transform our matrix data into vector before going further

## The size of the output layer importance :

* If we attempt to fix the inut size of the final layer in a value differentof 10 , the code will crash and we wiil get an error
  + Explanantion : In the training data , We expect values between 0 and 9 .. SO the output shape must be equal to the class numbers

## Is increasing the number of layers is always important?

* For simple data like a 28\*28 gray images (without colors ) : it’s just a simple data , so adding new layers will not give us a significant amelioration to our model

## Increasing the epochs during the fitting impact :

* Very few times than the ideal one : a low accuracy in fitting + evaluating
* Ideal epochs times: a high accuracy in fitting + evaluating
* Very more times than the idea one : a high accuracy in fitting + a low one for the evaluating : OVERFITTING !

## Normalization Step importance

* If we pass the non-normalized data to our model , We will notice that we get significantly bigger loss values for every epoch iteration , and that should prove to us the importance of the normalization step

## Stopping the fitting executing once we get our desired accuracy :

* We can realize that thanks to a Callbakc that we will pass to model.fit method

class myCallback(tf.keras.callbacks.Callback):

def on\_epoch\_end(self, epoch, logs={}):

if(logs.get('accuracy') >= 0.6): # Experiment with changing this value

print("\nReached 60% accuracy so cancelling training!")

self.model.stop\_training = True

#Using the callback :

model.fit(training\_images, training\_labels, epochs=5, callbacks=[callbacks])

* + The execution stop when we will reach a 0.6 accuracy