**Végső doksi Barni**

**Splitting the images into datasets**

The first vital step of organizing and processing the data is to spliti it into train, validation and test sets.

Considering our dataset size (circa 70k 256x256 images) loading all of them once into a huge list, or other data structure, then separate it into the desired sets would be very resource consuming, hence not too wise. Instead we indtroduce a rather easy way of data organization, which can be useful when dealing with large datasets.

The idea is straightforward. We give IDs to all data samples after this all data elements can be simply referred using its ID. Our goal is to organize not the dataset itself but the IDs since it is way cheaper regarding to resources (all IDs are numbers).

Our way to correlate IDs and images is done through the image names. All of the images are renamed incrementally from 1, thus all the names are numbers. These names are excellent IDs. After we have the ID set (numbers in 1 to image number ~70k range, incremented by one) it can be shuffled then split. The result is a Python dictionary with the following keys: “train”, “validation”, “test” and values of the corresponding id set. The train split is 80%, validation and test 10% each.

**DataGenerator**

The next issue is still pertained to the size of the data. We cannot load the entire set and send it through our network due to the lack of memory. Taking the formerly mentioned set size into consideration we can calculate the required storage for the images represented with float numbers (floats are necessary if we want to normalize our data). One image (gray channel for input, the other to for output) represented with floats: 256x256x3x4 = 0.78 MB thereby ~70k images take more than 50 (!) GB, so this way is not an alternative.

This is a common dilemma among image oriented deep learning problems. To overcome this vast dilemma DataGenerators are come in handy. The principal idea is to feed the network with chunks of data hence we do not need to store the entire set in the memory only a small part of it which we change time to time. There are lots of pre-implemented DataGenerators in deep learning and data processing libraries, still we decided to make our own, this way we can use it exactly to our cause. Instances of this class can be used with the keras \*\_generator functions. These functions are made directly for using networks with generators.

Our DataGenerator implementation makes use of the afore mentioned data organization practice. It loades the images in batches by the aid of the IDs. The output of the generator is a batch of input and output data from the adequate set, from which the generator is constructed.