**Introduction**

Usually, when training a learning model for any task such as prediction, classification etc., a lot of work is being done to improve the model, mainly by model / architecture selection and hyper-parameter tuning. These changes are being examined relatively to one dataset, and the model that achieves the best results on the given dataset is being chosen.

This approach, even though being used often alone, might be problematic. If the given dataset is not diverse enough, the model’s generalization ability would likely stay limited, no matter how many hours we have spent on tuning the parameters and adjusting the network’s architecture. A great model with poor data is just not enough.

Therefore, a different approach to optimizing a model is being suggested in this exercise. Given a model, we want to improve its performance by changing only the dataset. This time we won’t be tuning the model, but we will perform actions to adjust the dataset. This method can solve the problem we addressed, as if we make the data diverse enough such that it is more likely to include real world like data, when testing the model in the real world, it should perform better.

**The Learning Task**

The task we address in this exercise is image classification. More specifically, we have a model that given an image of a roman number, outputs the number in the image.

The model being used in this exercise is ResNet50, which has 48 convolution layers. A final linear layer is added to the model for the specific classification task.

The model is being trained for a 100 epochs, using Cross Entropy loss and Adam optimizer, and we evaluate the model’s performance by observing the best validation data’s accuracy.

Maybe add some images about the network.

**The Data**

As our task is a classification task of roman numbers, the dataset is composed of images for each class. Each class contains ~207 images in average, and the images are represented as tensors of shape . Each entry in an image represents the pixel value, which ranges from 0 to 1 (the images are black and white).

Maybe add an image of a number.

Maybe add a histogram of the original data (number of images per class).

The original data is split to training set and validation set, such that the validation set contains a single image, and all other images are in the training set.

**Data Augmentations**

**Train-Validation Split**

The first thing we want to do is a smarter split of the data. We do not know how the data was split originally, but we do know that a single validation image from each class is not enough to understand whether a model can generalize.

Therefore we randomly split our data to separate training set and validation set, giving the validation set 20% of the data. In numbers, originally we had a total of 2077 images, so after the split we have 1658 training images and 419 testing images.

We train the model with this data and achieve an accuracy score of 0.852029.

**Data Transformations**

As we can see, the original model seems pretty good, and already achieves high accuracy on a dataset that wasn’t changed, but only split to train and validation sets.

Next we try augmenting the data by adding some transformed versions of each image. We perform randomize transformations on each image in order to mimic the real life, where data might be noisy.

We perform a serialized transformation as follows:

1. Random Rotation – We rotate the image in a random angle between degrees.
2. Gaussian Blur – We perform a random gaussian filter, which blurs the image using a kernel of random shape (can be 1, 3, 5, 7 or 9), and a convolution operation with a gaussian with random standard deviation (uniformly chosen between .
3. Random Resized Crop – We randomly crop a portion of the image (from 60% to 100% of the image) and then resize it to the original image shape.

We perform this transformation 4 times per image and end up with 8290 training images. We do not touch the validation images in this section.

We train the model with this data and achieve an accuracy score 0f 0.840095.

As we can see, the transformations did not help the model’s performance, but confused it. We performed transformations only on the training data, so that means that our transformation does not resemble real data well enough.

We will try a similar transformation, but a more gentle one:

1. Random Rotation – This time we rotate the image in a random angle between degrees.
2. Gaussian Blur – We perform the same random gaussian filter as before.
3. Random Resized Crop – This time we randomly crop a portion of the image from 90% to 100% of the image.

This time we achieve an accuracy score of .