HW2 Report

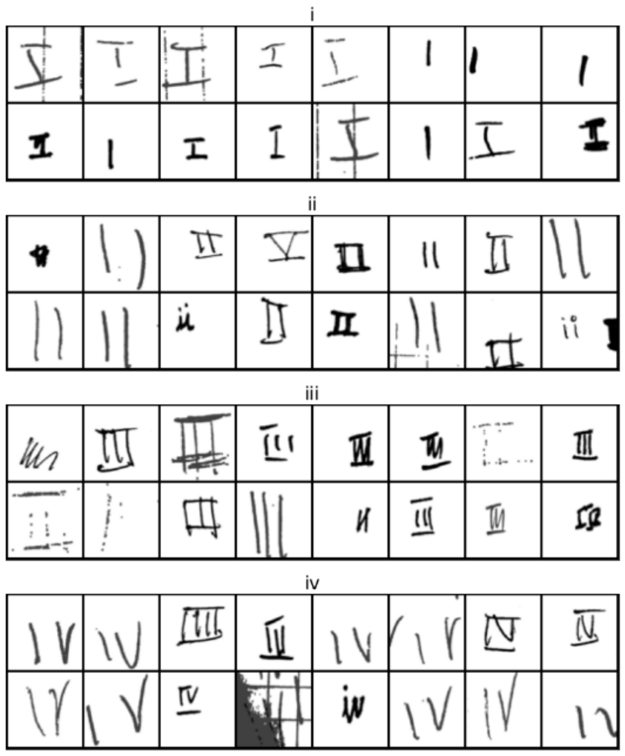
Authors: Omer Madmon, Omer Nahum
GitHub repository: https://github.com/omer6nahum/LabHW2

Introduction

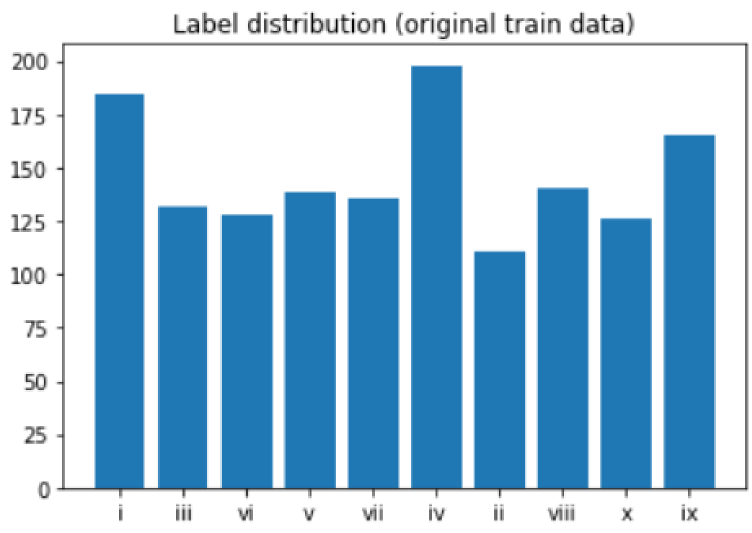
In this task we are asked to improve the performance of a (static given) ResNet50 model in roman digits classification, by only modifying and enriching the dataset it is trained on. In particular, we have access to ~2000 images and we are asked to create our own training and validation sets, under the constraint that the two sets together have at most 10,000 images.

Exploratory Data Analysis

We have started the exploratory data analysis by manually looking at random examples from each class:

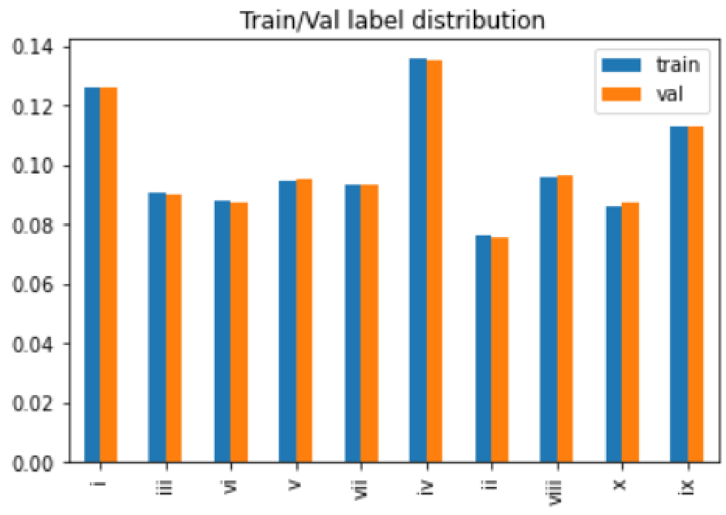


Then we have visualized the label distribution within the dataset:

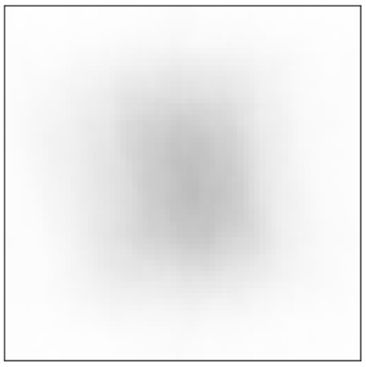


As it can be seen, the distribution is close to uniform, and we will preserve it in the generated train and validation sets.

Next, we have created a random 70%-30% split of train and validation, and verified the label distributions are similar:



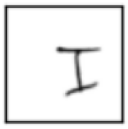
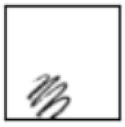
To get a better understanding of the data and its structure, we have seen that most images have white background and a black (usually) handwritten digit in the middle of it. This property can be seen also by visualizing the eigenvector corresponding to the first principal component of the data:



Methods

After having a basic train and validation sets, our aim is to modify and enrich the training set in a way that should help the model learn meaningful patterns in the data, while leaving the validation set untouched. We will measure our performance with respect to the validation accuracy.

Observations that motivated us while choosing methods to modify the training set:

- Structure - white background, black (usually) handwritten digit in the middle of it.
- Mixed labels - some of the images are labeled by a wrong label. For example:
   (labeled as III)
- Unclear images - some of the images are more doodles than digits. For example:
  
- The model lacks regularization methods such as weight decay and early stopping.

Inspired by these observations, we have decided on a high level training set modification pipeline which consists of two main phases:

1. Filtering - we will filter out images which are prone to mislead the model in train time (due to incoherent digit in the image or wrong label).
2. Augmentation - we will use various augmentation techniques to create a richer and larger set of images from the remaining images.

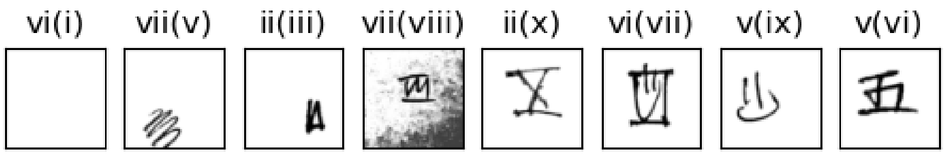
We will try different filtering and augmentation techniques. We have also tried adding some other phases to the pipeline described above which did not yield any improvement in results on the validation set - see 'other methods' subsection below.

Filtering

In order to identify images which are prone to mislead the model, we have trained a separate resnet50 model (prior to the model which we aim to optimize its performance). This model was trained on relatively small number of epochs (to prevent overfitting) and we used its predicted probabilities on the train set to identify these images based on different criteria:

- Filtering-1 - remove an image (with true label y) if the predicted probability P(y) is less than some threshold c. The motivation for this criterion is that we expect that images with low predicted probability of the correct label might be images with a wrong true label.
- Filtering-2 - remove an image if its predictions' entropy is larger than some threshold c. High entropy indicates uncertainty of the model with respect to the image (entropy's maximizer is a uniform distribution), and we expect that images with high entropy are "too confusing" and may not be coherent digits at all.

The following figures visualize examples of images that were filtered out by these filtering policies:



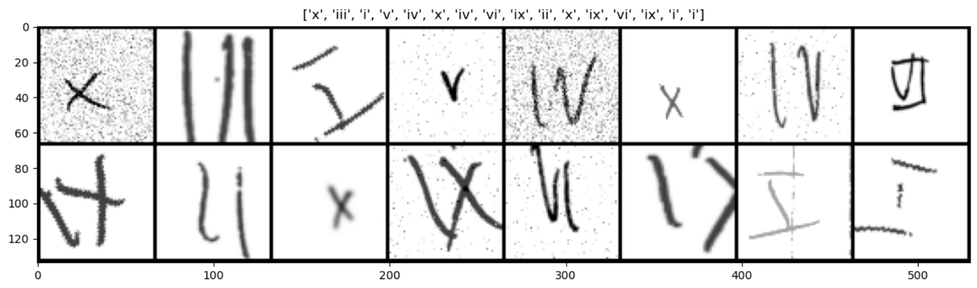
The header of each image contains its true label as it appears in the data, and the label predicted by the resnet50 model in the following format: true_label(predicted_label)
Note that, for instance, the second image is an incoherent image, and the fifth image is an image with a wrong true label (II instead of X). The exact thresholds of each filtering technique have been tuned manually, based on visualization of the selected images and the final model's results on the validation set.

Augmentation

After filtering irrelevant images, we have enriched the training set by applying the following augmentation techniques on different images in the dataset:

1. Blur
2. Perspective
3. Rotation (15-45 degrees).
    - Note that rotation in a higher degree might change the meaning of the image.
4. Affine transformation
5. Auto-contrast
6. Gaussian Noise
    - We added grayscale noise, despite the fact that the image has 3 channels.
7. Masked Noise (same as gaussian noise, but applied only on a fraction of pixels in the image)
    - Note that adding noise in columns might change the digit in the numer (for example: II might be changed to III).
8. Brighten/darken
9. Random crop
    - Note that we are aware of the fact that cropping might end up with an image that no longer fits to its label (for example: cropping one 'I' out of an image of label 'IV' can change the true label from IV to V). We tuned the parameters accordingly.

Most augmentations are implemented in pytorch. The following figure provides examples of images from the training data after augmentations have been applied:

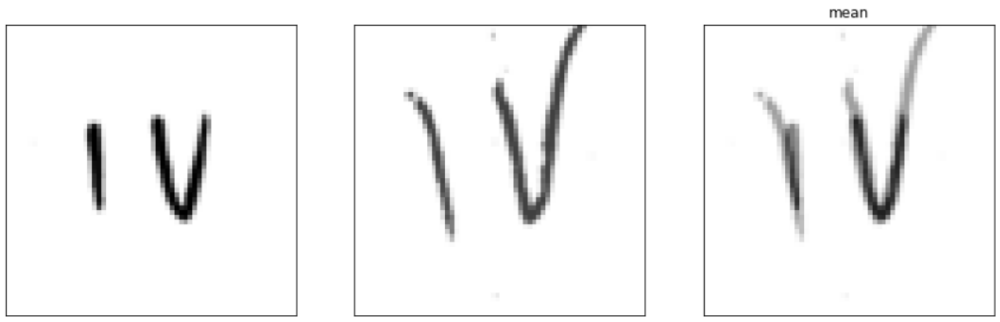

['x', 'iii', 'i', 'v', 'iv', 'x', 'iv', 'vi', 'ix', 'ii', 'x', 'ix', 'vi', 'ix', 'i', 'i']
0
20
40
60
80
100
120
0
100
200
300
400
500

Note that there are some other common augmentation techniques that are usually used in image augmentation which are expected to be problematic for our specific task.

Other Methods

1. <u>Label flipping</u> - In contrast to the filtering phase, where we have filtered out images with wrong labels, we have tried to randomly flip a small portion of the labels of the augmented data in order to introduce regularization and increase the difficulty of the model training. Note that if we do leave wrongly labeled images and then perform augmentation, we might end up having multiple images with the same flipping, which might mislead the model and encourage it to learn wrong patterns while training.
2. <u>Average of correlated images</u> - in order to generate more images in each class based on existing images, we have implemented the following logic:
   a. For each class, find the top 50 correlated pairs of images.
   b. For each such pair, create a new image by simply averaging the two correlated images.

   The following figure provides an example of two correlated images in class 'IV' and their resulting average, which was added to the training set:

   We have tried to add this phase both before and after the augmentation phase. When tried after the augmentation phase there was no significant effect on the results (as it only increased the training set by 500 out of ~10,000 images). When applied before the augmentation phase, results were slightly worse.
3. <u>Classes balancing</u> - After performing the entire pipeline, we balanced the classes by a simple random under-sampling technique. When using this phase, we increased the number of augmented images, in order to keep the total number of images as ~ 10k.

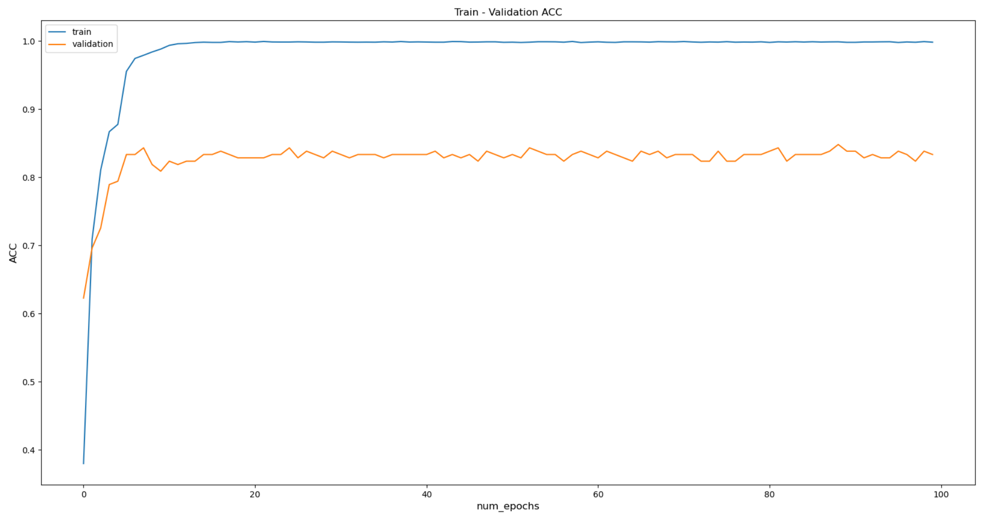
Results

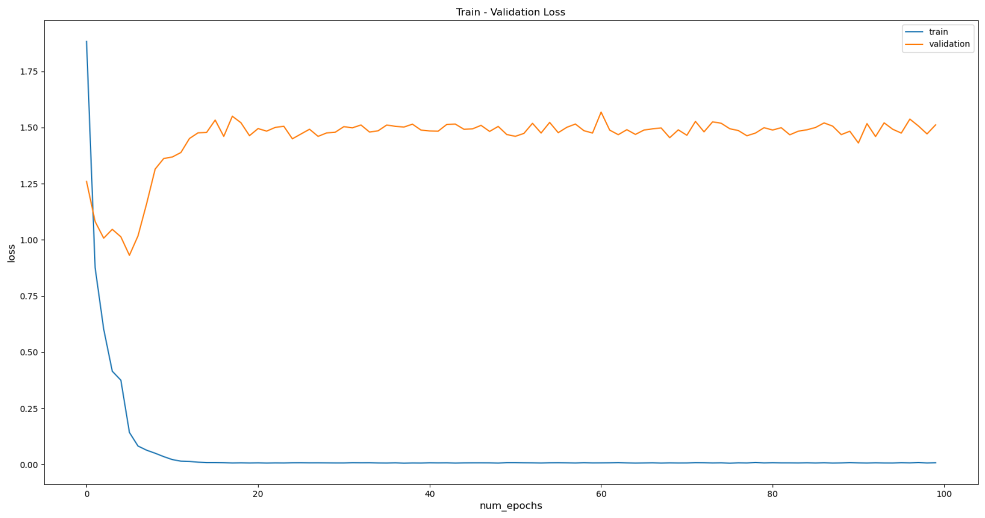
The following table displays the results of the final resnet50 model after being trained for 30 epochs on a training set which was created by different methods, as described in previous sections.

Method	Train Accuracy	Validation Accuracy
Unmodified training set	0.965	0.773
Augmentations only	0.999	0.806
Filtering 1 OR filtering 2 + Augmentations	0.997	0.830

As it can be seen from the table above, it seems like both the augmentations and the filtering phases have contributed to the improvement in the model's performance on the validation set. Other methods have been tried as well (see 'Other Methods' subsection above) but these haven't improved performance.

Our submitted model will be trained on a bigger fraction of base train set with the same augmentation and filtering pipeline and parameters. In addition, we will submit a model trained for 100 epochs rather than the 30 epochs reported above.
The following 2 plots (train-validation accuracy and train-validation loss) are obtained from the training process of the submitted model:


Train - Validation ACC
train
validation
ACC
num_epochs


Train - Validation Loss
train
validation
loss
num_epochs