MAIS 202 Assignment 4 Deliverable

Group 2

Noa Kemp, Sarvasv Arora, Saumyaa Verma

April 4, 2021

Implementation of the model

We used the VGG-16, a classic CNN architecture, with slight modifications. In each VGG block, we included a batch normalization layer that improves the training speed by normalizing the activations from the Conv2d layers. It also provides a slight regularization effect, thereby preventing our model to overfit the training data.

Our model is made up of four VGG blocks, each consisting of up to three Conv2d layers. Further, each VGG block is separated by a max pooling layer.
The VGG layers are followed by two fully connected layers and a softmax layer that outputs the predictions.
In total, the model had 87.5 M trainable parameters.

The loss function used was cross entropy loss and the optimizer was ADAM, which is pretty standard for deep learning tasks.

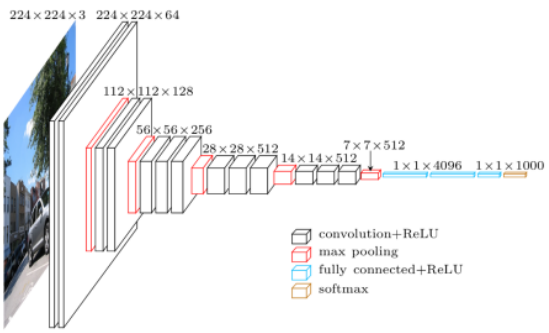

Figure 1: Source: Papers With Code

We trained the model, first for 9 epochs (version 1) and then for 10 additional epochs (version 2).
For each training instance, we used a batch size of 32 images and adaptive learning rate initialized at 10^{-3}.

Results

The version 1 of the model achieved an accuracy of 98%, whereas the version 2 (which was trained for 10 more epochs) has an increased accuracy of 98.4% on the training set.

On the test set (as per the private leaderboard), version 1 achieved 96.7% whereas the version 2 achieved a slightly lower accuracy of 96.6%, which shows that it had started to overfit.

We also experimented by making predictions on preprocessed test data that had reduced noise and almost clear background, however as expected, the model didn't perform well on that and achieved just 73.5% accuracy. This was bound to happen since we didn't train the model on such a data distribution.

Challenges

The biggest challenge was to remove the background of the images, as it was our first time preprocessing images using OpenCV (or any related module).

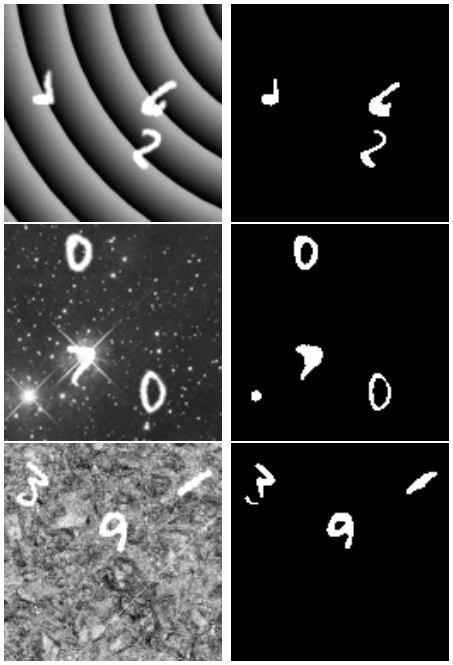

Figure 2: Some images before and after preprocessing

After having spent a quite impressive amount of hours on that issue, we figured out that we could replace all black pixels with RGB value < 70 (this number was found after multiple trials) by white; and then replace all pixels that aren't white by black. Doing so results in an image with the handwritten digit in white and the background in black. For the background to not have white 'noise,' denoising was also done before making any color changes.
Unfortunately, the way we implemented the color change was not efficient — it ran in $O(n^3)$. Moreover, Colab didn't allow us to run an environment overnight, and adding to it the time constraint, this idea of preprocessing couldn't be used to train the model from scratch.

Another challenge was to constantly keep a backup of the model while training it, considering that we could loose all the process in case the environment crashes. To address this, we used the Pytorch Lightning framework that offers automatic checkpointing as well as a nice API to facilitate hassle free training.

Lastly, we also planned to use augmented data, which could have potentially increased the final accuracy by a margin of $\approx 2\%$. However, due to time constraints, we couldn't do so.

Conclusion[1]

The model performs pretty well on the test set with an accuracy of $> 95\%$.

Apart from that, we are contended that we could learn more about the workflow of a Kaggle competition, data preprocessing, and other things like collaborating virtually and managing to not lose the model due to Colab RAM issues.

Individual contribution

Noa and Saumyaa worked on data visualization, data preprocessing and creating the dataset class. They also extensively carried experiments to denoise images and get better pictures for the model to train on. This idea couldn't be implemented on time however.

Noa also tried creating an augmented dataset.

Sarvasv mostly worked on the model implementation side (a.k.a. matching tensor dimensions :p), making predictions, calculating accuracies and organizing the code for the final submission.

[1] Github: @nrbkempMAIS202_A4_Competition