Technical Exercise

Field & Road Classifier

I) Data processing

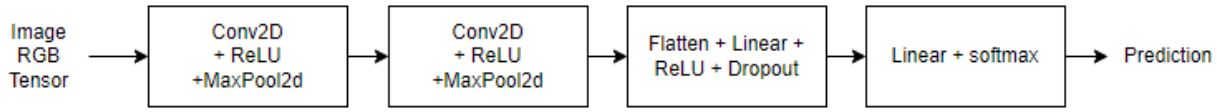
I started by processing the dataset into a usable format for the model by making a train and validation set out of the main images and keeping the test images for the final evaluation. Looking at the dataset I realized two things:

- 2 pictures were mislabeled and should have been road instead of field
- The road category is way more represented than the field category

So before doing anything, I corrected the image class. Then I used an 80/20 (train/val) data split on top of some undersampling to balance the dataset. I added random shuffling of the data during the split and controlled the randomness thanks to a seed because considering the small dataset, the results could vary a lot with the same model configuration but a different split. The dataset folder is reorganized into a "train" and "val" folder, all the images are renamed the following way: "name of the class"_"number".jpg so that I can use the name for the label in the Pytorch Dataloader. Some data augmentation and data processing techniques were also applied right before the training thanks to a transform in the loader. I first resize the image to a small resolution (224 x 336 most of the images have a rectangular format and are around this size, I also tested different resolutions), then I apply a few augmentations such as rotation (max ±15°) and horizontal flip (50% probability). I also tested other augmentations but didn't keep them considering the results were not improving. I then finish with a normalization of the RGB image (same as the one used for ImageNet Dataset which is an average on millions of images) and then transform the images into tensors.

II) Model Building

After making a few tests on the VGG16 architecture, I realized that the model might be too complex to apply transfer learning on the small dataset, considering that the number of features would probably allow the model to overfit. I then created a small CNN architecture with the following architecture:



The loss function I used is CrossEntropyLoss, I wanted to use this one to test balancing the weight of each class to avoid undersampling (to keep most of the data considering the dataset is already quite small) but it gave a worse result than undersampling so I sticked with my first solution. For the optimizer, I picked Adam since it gave the best results compared to others such as SGD and I tested different values of learning rate and L2 regularization (such as weight decay) to keep the best parameters.

III) Model Training

As mentioned, before I tested plenty of parameters with a small number of epochs to search for the best combination. I'm aware that my model could be improved by looking at how the validation loss isn't converging but I wasn't able to correct that within the time constraints. I kept the training around 26 epochs because it wasn't improving with more epochs. We can see the history of the loss value in the Figure 2. The training loss is very bumpy this is also might be because Adam optimizer and the low batch size (4). The model seems to be slightly overfitting if we look at the training loss average value and even the dropout wasn't enough to reduce it.

Figure 1: Training for 30 epochs

To evaluate the model, I used the test images and looked at the accuracy for all 10 images. The data is pre-processed the same way as during the training (without the augmentation) to help the model with a normalized image format. The maximum accuracy that I got was 80% but I could tell that some images were harder to correctly classify than others especially the desert one which is quite colorful and could look like flowers in the model's eyes (there are not lot of roads in the desert in the dataset). Or the other one that looks very dark when normalized and could look like a black road.

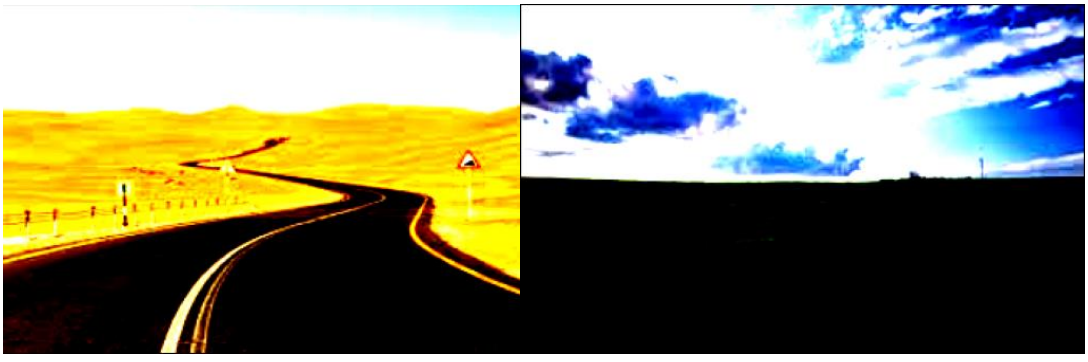


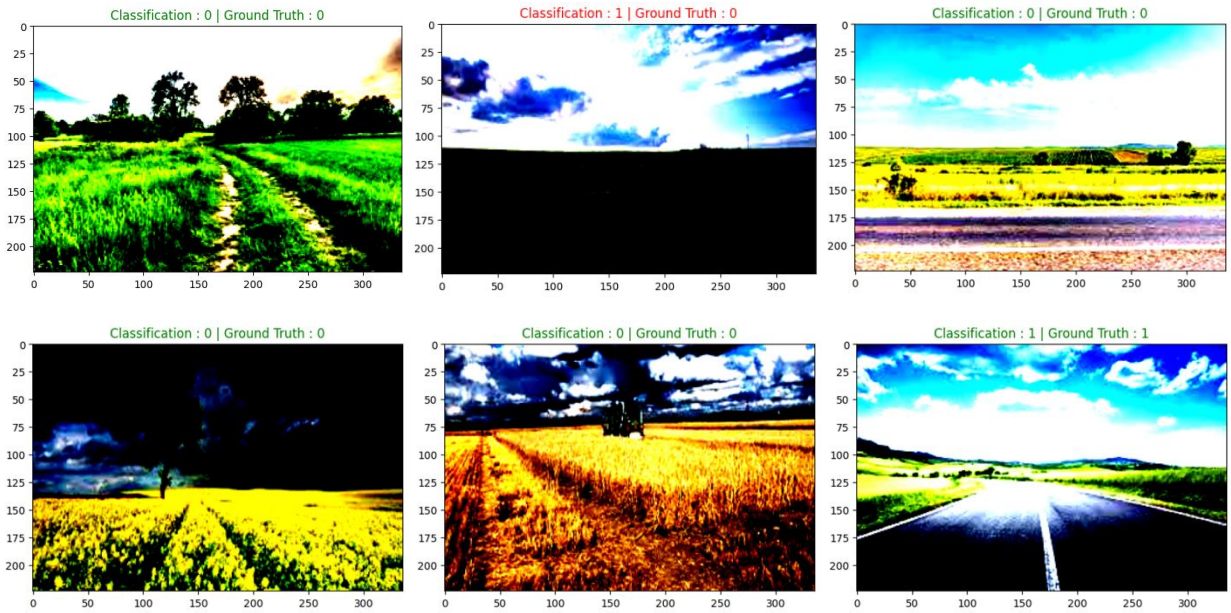
Figure 2: Test images wrongly classified

On top of that, there is one image that has a road in the foreground and a field in the background which is disturbing (Changing the class makes a difference of 10% in the accuracy value).

Figure 3: Test image that includes both classes

IV) Conclusion

Looking at the time constraints and the small dataset the model performances look relatively good but on a test with higher number of test images the precision value might differ a lot. Considering the loss isn't converging very low, it would be interesting to work on some changes especially looking at the model architecture to try and get better results by maybe simplifying the model even more or trying to use different hyperparameters. Ideally improving the dataset would probably help but it wasn't the point of this exercise. Also, more complex AI techniques could be implemented in the code but the small dataset is making the whole task harder (cross validation for example). I started the project thinking I would reuse a previous project (for sky segmentation) and adapt it to the problem but it took me probably more time to adapt the code than I would have to code it from the start. Here are a few inferences plotted after pre-processing:



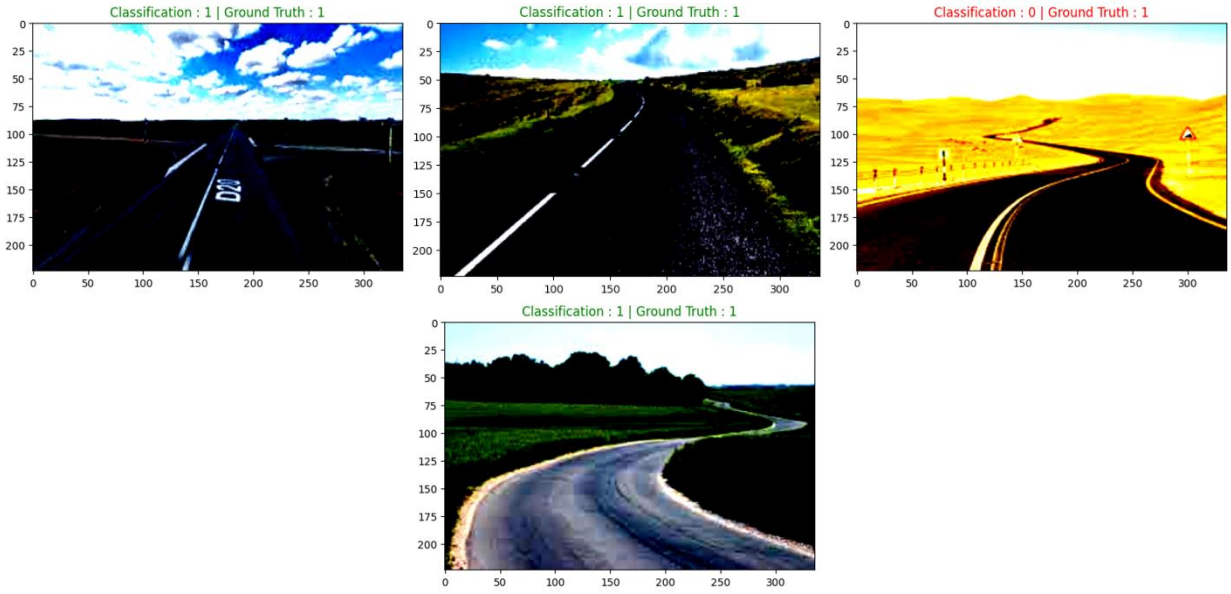


Figure 4: Test Image prediction