Minerva Suvanto
LT2326 Assignment 1
18.10.2021

Chinese character "detection"

In this report I have documented my work for the chinese character detection assignment for the course Machine learning for statistical NLP: Advanced LT2326. I have documented how I preprocessed the dataset, what architectures I chose for the models and how I trained them, and how I tested the resulting models and how the models performed.

The assignment is implemented as a jupyter notebook. The notebook runs on mltgpu and the path for the training and testing images can be specified in the first cell. The trained models can be found on mltgpu in /scratch/gussuvmi/

Data preparation

The format of the data after processing is a list of tuples, where each tuple represents an image and its bounding boxes. In the tuple the first element is a numpy array representation of the image, which I got using the matplotlib.image library. The second element is a numpy array of truth values for each pixel in the image, which I got using the matplotlib.path library.

The data is split into training (70%), validation (15%) and testing (15%) datasets. I use the sklearn.model_selection package to do this. I kept most of the images in the training set so that my models would hopefully learn better.

The processing of the images utilises a package joblib.Parallel when retrieving the bounding box information. This cuts back on the processing time significantly, from 30+ minutes to only about 10 minutes.

Models

LeNet

As my first model architecture I chose to do the LeNet architecture. The reason I chose this architecture was because I read that it is one of the most basic convolutional neural networks. Since I wanted to make a convolutional network for image processing, I thought the simplest option was a good way to get started.

The architecture of my model includes the following layers:

> Conv -> ReLU -> MaxPool -> Conv -> ReLU -> MaxPool -> Linear -> ReLU -> Linear -> Sigmoid

In the first convolutional layer I chose a kernel size of (5,5). I decided not to change the number of output channels. In the maxpool layer the kernel size is 5 so that I can downsample the data enough to keep memory on the GPU. The second convolutional layer is identical to the first one. The second maxpool layer, however, has a kernel size of 3. In the first linear layer I use the size of the output from the maxpool layer as the input features and

have an output feature size of 500. In the final linear layer I chose the number of output features to be 250. The output is run through a sigmoid activation function to get values between 0 and 1. The chosen values are mainly experimental.

Once the results are obtained, they have to be upsampled to be comparable with the real values. I do this outside the model. I use torch.nn.Upsample to do this and upsample the output back to a size of 2048*2048.

ResNet

For the second model I wanted to explore something with a bit more layers and I came across the ResNet architecture. I thought this looked simple enough for me to be able to implement and hoped that by having a deeper network I would have different results from my first model. I chose to make an 18 layer deep model.

The architecture of my model was the following:

> Conv-> BatchNorm -> ReLU -> MaxPool -> (ResLayer * 4) -> AvgPool -> Linear

Where the first ResLayer looks like this:

> (Conv-> BatchNorm -> ReLU -> Conv-> BatchNorm -> ReLU)

And the rest of the layers like this:

> (Conv-> BatchNorm -> ReLU -> Conv-> BatchNorm -> Conv-> BatchNorm -> ReLU) -> (Conv-> BatchNorm -> ReLU -> Conv-> BatchNorm -> ReLU)

Each layer consists of a “block” which is marked by the brackets. A layer consists of two blocks. An extra downsampling layer is added if the specified stride is not 1. In this case the first ResLayer has a stride of 1 so it does not have this downsampling layer. The rest of the layers have a stride of 2 in the first block and a stride of 1 in the second. That is why in the first block there is an extra convolutional layer but in the second one there isn’t.

In each layer the number of output channels is doubled, starting from 3 and ending in 24. The kernel size of the first convolutional layer is 5, in the residual layers it is 3 for all convolutional layers except the extra downsizing layer, where it is 1. The convolutional layers in the residual layers also include a padding of 1, except for the downsizing block. The values chosen are again experimental.

Training

I use DataLoader for the training data to get batches for the training loop. I chose to use Binary Cross Entropy as my loss function and Adam as the optimizer. My final values for the hyperparameters were 4 for batch size, 0.01 for the learning rate and 3 for epochs.

Minerva Suvanto
LT2326 Assignment 1
18.10.2021

I used the same loss function, optimizer and hyperparameters for training both models. The only difference between my training loops was that the ResNet training loop included a sigmoid outside of the model, this was a feature of the architecture. Upsampling for both models is done outside the model.

Testing and evaluation

I evaluated and tweaked my models slightly by testing them on the validation data first. I wanted to increase my batch size to 32 but that resulted in infinite CUDA out of memory errors so I had to stick with a batch size of 4. I tried a higher learning rate (0.001 -> 0.01) to see if that would improve the performance but it did not seem to make any difference.

To test how well my models performed, I decided to calculate the mean squared error. The results for the test data were the following:

LeNet
average MSE 0.0041
total MSE 0.13266362762078643
accuracy: 0.33068088938792545

ResNet
average MSE 114.4067
total MSE 3661.0144119262695
accuracy: 0.11528275969127814

The MSE for the LeNet model does not seem to be too bad but when looking at a few images it doesn't actually seem to detect bounding boxes really well, so in reality the model does not work well. I believe the reason why the MSE is so low is probably due to the fact that most parts of the images are just black (so pixels not are not in a bounding box). So the model only does well in detecting if a pixel is *not* in a bounding box.

The ResNet model works even more poorly and does not detect bounding box information correctly at all. It does seem to have learned something, it seems that it recognizes roads and darker spots in an image to not be in a bounding box. But the rest of the results are very messy, so I would not call this a good model either.

When looking at a few images, it seems that the first model has the same result for all images. The second model at least has some variance between images but the patterns are still similar. It is probably the case that there was not enough data to train the models on so they did not learn very much.

References

These are some of the articles I used as reference when implementing my models.

Minerva Suvanto
LT2326 Assignment 1
18.10.2021

LeNet:
https://www.pyimagesearch.com/2021/07/19/pytorch-training-your-first-convolutional-neural-network-cnn/
https://en.wikipedia.org/wiki/LeNet

ResNet:
https://jarvislabs.ai/blogs/resnet
https://www.run.ai/guides/deep-learning-for-computer-vision/pytorch-resnet/

General:
https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d
https://www.sicara.ai/blog/2019-10-31-convolutional-layer-convolution-kernel