Project 2 Report

COE 379L

Prof. Joe Stubbs

Owen Scott

6 November 2024

Data Preparation

Data preparation for this project was quite minimal. Images were resized to the expected model input dimensions; pixel values were remapped from the integer [0, 255] range to the floating point [0, 1] range; images were sometimes grayscaled depending on the model input dimensions. I did attempt to make models that generalized better using augmentations like horizontal flips, small zooms/translations/rotations, and brightness changes, but these augmentations always seemed to worsen model performance instead of improve it, so the model that I have selected for usage in the inference server did not use augmentation. That model leaves images as RGB, normalizes their pixel values to the [0, 1] range, and resizes the image to 150 by 150 pixels.

Model Design & Evaluation

I tried many base model architectures, many of which were suggested by the project description. At first, I tried grayscale and color ANNs. I also created the LeNet 5, Modified LeNet 5, and VGG-16-based models as suggested by the project document. I had familiarity with VGG-19 from prior machine learning work I've done, so I tried using that model too. At first, I trained ANNs, and achieved F1 scores of .36 (grayscale) and .66 (color). I think an F1 of .66 is impressive given the very basic model architecture, lack of convolutions, and low training time. At the very least, it showed that color was an important detail for model performance. Next, I trained LeNet5 (F1 .68), variants of it with augmentation in the training data, modified LeNet5 (F1 .97!), variants of it with augmentation in the training data, and VGG16/19 models (F1s of .94). Two notable takeaways: Modified LeNet5 performed very well with a modest training time and simple(ish) architecture; and; Augmentation always resulted in worse performance (or total model collapse). I concluded that I should stop trying augmentation (there's probably a way to do it well, but it would be longer training, and I was short on time), and I should focus on optimizing the modified LeNet5 model as much as possible. That model trains quickly enough that I could iterate many times with it. I asked ChatGPT about ways to improve my model architecture and training strategy and implemented variations of its suggestions. Eventually, my best-performing model was the seventh version of the modified LeNet 5 model, which uses batch normalization, a global average pooling layer instead of a flatten layer, and L2 regularization on the convolution layer. Training improvements included using label smoothing, weighted-average class weights for the loss function so that the model did not just pick one class every time (collapse), and reducing the learning rate when model accuracy plateaued. I am quite confident in the final variation of my model. It achieved an F1 score of .993 on the test dataset. It doesn't get much better than that.

Model Deployment & Inference

Server Endpoints

- GET /summary : Returns information about the model in JSON format
- POST /inference : Accepts an image binary and returns JSON like

{ "prediction": "damage" } or { "prediction": "no_damage" }

Running The Server Build

Note: Only linux/amd64 is supported by the image I published.

Do `docker-compose -f docker-compose.prod.yml up` to start the server.

Running A Local Build

Do `docker-compose -f docker-compose.local.yml up` to start the server.

Server Usage

By default, the server runs on port 5000. Feel free to change that to whatever works best for you by editing the docker-compose files.

Assuming the server is running on port 5000, the following `curl` examples work.

- GET /summary : `curl localhost:5000/summary`

- POST /inference : `curl -X POST -F "file=@/path/to/your/jpg" localhost:5000/inference`