Final Project

Introduction to Machine Learning Léo Dupire | ld2494

Method

Packages:

For this project, I used PyTorch as my Convolutional Neural Network (CNN) building framework. The packages that were used to complete the current project are shown below.

```
import os
import numpy as np
import random
import pandas as pd
import matplotlib.pyplot as plt
import cv2
import pickle as pkl

import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import Dataset, DataLoader
from torchvision import datasets, transforms
from torchvision.transforms import ToTensor
```

Data:

A custom LazyLoader was used to load the data into the runtime. However, instead of directly feeding the lazy loaded data into the models, I decided to run the LazyLoader once on each example of the train & test sets and saving these into corresponding torch tensors. I then saved these data into several .pt files. I could then load in the data all at once, instead of dealing with lazy loading latencies with every run of the model. This may seem to defeat the purpose of lazy loading but using the loader was still useful to extract the necessary files from the directory structure.

Models:

Several approaches were used during this project. These are highlighted below.

Model 1 & 2: A model using only the top RGB image as training input and another using the Depth image as training input. These models follow the exact same architecture, as shown below.

Conv2D
Conv2D
MaxPool2d
BatchNorm2d
Conv2D
Conv2D
MaxPool2d
Linear
ReLU
Linear
ReLU
Linear

We see a total of four convolutional layers and three fully connected layers, with some max pooling, batch normalization, and ReLU functions along the way.

Model 3: A model using a concatenation (6-layers) of the top RGB and depth images as input. This model follows the same general architecture as models 1 and 2, with adjusted input and output channels to accommodate the increased size of the training data. The idea here is for the model to learn patterns in both the RGB and depth images simultaneously.

Model 4: A network of three models. The first model is the top RGB image model as seen in model 1, above. The second model is the depth image model as seen in model 2, above. The third model takes the outputs of the first two models and merges their outputs into a singular, final output. The idea here is for the merger model to identify prediction strengths in both models 1 and 2, tuning its output to be an optimized version of the initial two models. The merger model's architecture is shown below.

Conv2D
BatchNorm2d
Conv2D
BatchNorm2d
LeakyReLU
Conv2D
BatchNorm2d
LeakyReLU
Linear
ReLU
Linear

We see a total of three convolutional layers and two fully connected layers, with some batch normalization and Leaky ReLU functions along the way.

With each of these models, I experimented with different learning rates, optimizers (SGD vs. Adam), activation functions, weight decays, numbers of layers, neurons per layer, and placement of batch normalization and dropout layers. I used leaky ReLU and batch normalization, among other efforts, to minimize exploding and vanishing gradients.

Experimental Results

I do not have the Kaggle test RMSEs for each of the models below. Given the limited daily submissions to Kaggle, I only submitted models that had good enough RMSEs on the training set. I set the threshold of submission consideration to be an RMSE score under that of my best score on Kaggle, since the training loss nearly always represents a floor loss for the test set.

	M1 (RGB)	M2 (Depth)	M3 (RGBD)	M4 (Merge)
Training RMSE	0.003626	0.007174	0.003951	0.022542
Test RMSE	0.00492	N/A	0.0084	N/A

Model 1: This model was trained only on the top RGB images. This was the best performing model, consistently outperforming the other three on the train RMSE. It got a training RMSE of 0.003626 and a test RMSE of 0.00492.

Model 2: This model was trained only on the depth images, getting a training RMSE of 0.007174.

Model 3: This model was trained only on the combined top RGB and depth images, getting a training RMSE of 0.003951 and a test RMSE of 0.0084.

Model 4: This model was trained only on the combined top RGB and depth images, getting a training RMSE of 0.022542.

Discussion

It was interesting to see that the RGB model performed the best, despite taking in only one input image compared to the RGBDepth and Merger models that took in two images. This is likely the product of sub-optimal hyperparameters for the latter models.

In addition, we see that the Depth model performed notably worse than the RGB model. My hypothesis is that the RGB model recognizes the white colored fingertips of the robotic hand. This helps the model better track the position of each individual fingertip whether the finger is extended or curled. The model doesn't have to learn the anatomy of what is a fingertip and where is it on a hand, rather it can simply look for a white circle-like shape. On the other hand, the Depth model does not have this fingertip "flag" to look for; it must infer where the finger tip is relative to the rest of the hand. Even if it learns to do this well, it is unlikely that it will surpass the RGB model that can do the same (same architecture) but with an extra aid of the white colored fingertips.

We can also see that the RGBDepth model performs somewhat comparably to the initial RGB model. The former has the same layer layout as the latter, except with different input channel, output channel, and kernel sizes and with twice (arguably) as much information, so it makes sense that it would perform similarly if not better. I spent less time training and tuning this model, so it is possible with more iterations, the RGBDepth model would outperform the vanilla RGB model.

Finally, the Merger model performed very poorly, significantly more poorly than the RGBDepth model, even though the two models ingest the same input information. I believe this to be the case as the RGBDepth model is analyzing all the data simultaneously, establishing a relationship between the depth image and the rgb image, one that strengthens its predictions accuracies. On the other hand, the Merger model compartmentalizes its learning, each compartment improving on its task, but working together poorly. This highlights the importance of being able to make inferences on multiple data at once rather than independently analyzing two inputs and trying to fit them together in the end.

Future Work

In the future, running hyperparamter search and tuning algorithms would likely produce better models for the given task. This includes optimizing nearly all model hyperparameters.

In my models, I only utilized the top RGB images and the depth images. This is not utilizing all the provided information. Going forward, I would either revisit one of the previous architectures (model 3 or 4) and adapt it to intake another input two images, or I would explore new architectures to accommodate the new influx of information .

References

- Github: https://github.com/leodup/CNN_robot_hand
- https://blog.paperspace.com/writing-cnns-from-scratch-in-pytorch/
- https://stackoverflow.com/questions/55894132/how-to-correct-unstable-loss-and-accuracy-during-training-binary-classificatio