CSE 352 Final Project Report

Jiaxin Dong, Zhedong Zhang, Weihao Yin

**Catalogue:**

Abstract

Techniques:

ResNet

Over-sampling/Under-Sampling

Transfer Learning (VGG)

Application Description

Research (Layers/ResNeXt)

Citations

**Abstract**

In this project, we defined a neural network architecture which can help us to do better classification on image data. We used ResNet as our base model with transfer learning and fine tuning to help us to distinguish rock paper scissors mimic by human hands. For further approach, we used some factory method on our data which is over-sampling.

**Motivation**

The neural network we defined can be utilized on hand figure processing, which can be utilized in many different ways. We can interpret the goal of our network architecture (distinguishing rock paper scissors mimic by human’s hands) as a fundamental step to develop a more complicated neural network to identify and classify different hand figures, which can be combined with other architectures to obtain significant benefits. For example, the advanced neural network will be able to identify and translate sign language into English words. The output can be fed into a natural language processing module to generate a human-readable sentence, and such application creates a way for people to communicate with deaf-mute without having any knowledge of sign language. Another promising application of our neural network is to create a “hand figure interface”. As it describes, the model will allow people to use hand figures as input to issue machine commands, which can be a valuable interface under specific scenarios, such as remote control.

**Techniques**

ResNet:

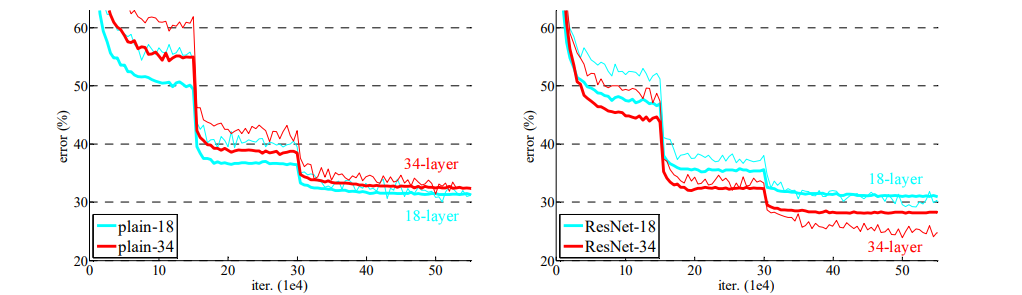
ResNet is a modern neural network architecture and has been defined in 2015. The reason why it has been invented is that, after Alexnet and VGG network published, all people is trying to develop their neural network in two ways: deeper or wider, or both. But nothing has really constructive under such circumstance. This bring us to a concept of Gradient Vanishing/Explosion. In 2014’s Kaggle competition, two networks stood out: 1st GoogleNet and 2nd VGG. VGG mainly talking about how neural network can have better performance with deep layers. GoogleNet is trying to create deep complex architecture that can prevent gradient vanishing and achieve high accuracy at the same time. The first version of GoogleNet (Inception v1) has two support classifier to help back propagation to avoid gradient vanishing (Szegedy, 2015). In the paper of ResNet, these four computer scientists created an experiment on VGG. And the result shows a highly disappointing result: on CIFAR-10, 56 layer network’s error was higher than 20 layer network.

ResNet prevent gradient vanishing by adding it’s own original parameter back after each convolution step:

Output = F(x) + x

(Where F is convolution layers and x is original weight)

At very end of this paper, authors designed an experiment on comparing plain 18/34 layer network with ResNet 18/34 network under same training circumstance. We can see from Figure 4 that although Resnet converge slower than normal network but it keep decreasing it’s error rate along with increasing of it’s layers.



ResNet is a framework that truly establish the concept of deep learning.

Feature Engineering:

In our project, we used several feature engineering techniques to create better representation of our data.

The first technique we used is Over/Under sampling. Our data only come with around no more than 2000 images. We choose around 100 of them as our test set and all classes (train/test) are approximately balanced. For a deeper neural network we need more images to maintain it’s functionality or we stuck in under-fitting issue. To avoid class imbalance, we random selected 20-40% of from each of our training images and make a copy of them and put them back into set. At this point, we have another 600-800 images to train our model. This technique is widely use in factory environment because extra data is not always available for us and we might limited to our expanse which can buy more accessories to data. One thing that need to notice is that Oversampling is not always easy as what we did in this project. It can be complex when encountering real life problem. For example, fraud detection and cancer prediction. Under these circumstances, data will not be balance any time and our model will spend more time on fitting positive samples others those significant negative samples. This time we do need to create our own negative samples to let our model perform better on such important predictions.

Transfer Learning & Fine Tuning:

Transfer learning is a technique that use other’s already pre-trained network and use it’s structure and general classify parameters to do further training. The main goal is to adjust it to our own task specific distribution job. It’s tough to learn from nothing. Training an accurate model needs time and resources (That’s why Google Colab is useful at some time). A normal deep NN training task can be extremely slow with CPU, but in GPU it will run much faster. Even though we have GPU and other hardware resources, we might not have a good skill on feature engineering and as a consequences out model doesn’t have a good representation of data we provide it. So the best way to approach a CNN task, is to borrowing learned network structure and parameters from others. VGG is a good resource of transfer learning, because we only need to modify the last few softmax layers.

Here we choose to use ResNet that already trained from Pytroch to do examination. First we add another layer of pytorch’s ResNet to let it classify to 3 classes. The accuracy rate of model is around 37%. This is close to normal guess expectation which is 33.3% which means this model does nothing special. Beside this output doesn’t make any sense because our model doesn’t know anything about our data. After we train 10 epoch on whole set we have, the accuracy rate raise up dramatically. After adjusting hyper parameters of learning rate, we eventually got a 99% accuracy on our test set.

**Application Description**

We did implement a ResNet model but for some reason, some of it’s layer can’t connect to each other (mat1 don’t match with mat2). We implement the normal basic block of ResNet 18/34, and the bottleneck block for ResNet 50/101/152. Our ResNet classes is trying to do a generic model which can fulfill all type of ResNet’s structure. It will make several layer based on the parameter (layer\_number) passed in when it’s object first been initiated. So we can use these layer number to control which version of ResNet we want to use.

Our data were extracted form a Kaggle website. It’s topic is “Rock Paper Scissors”. The main purpose of this dataset is to let model learn from images of human hands shape of Rock Paper and Scissors. And make a classification prediction of test image which shape is the hand represent. Image has total number around 1400 (Julien, 2019). We use torchvision.datasets to extract images into ndarrays. By using ImageFolder function from torchvision, it’s easy to extract images lays in different folders (data/scissors, data/rock, etc.). Then we pytorch’s dataloader to load all images with batch size of 8, shuffled and prepare to train/evaluate.

Our model was got directly from pytorch api. We use ResNet18 with pre-trained parameters. We choose Cross entropy loss as our loss function. We were thinking about binomial distribution form of maximum likelihoods but seems it only support binary classes. We also choose Stochastic Gradient descent as our optimize function.

For further detail of our implementation, please see CSE352\_Proj.ipynb.

**Research**

We compare our trained ResNet with non-tuned raw ResNet18 of pytorch. Without fine tuning, raw network gave a accuracy of 0%. This is because ResNet18’s last layer doesn’t have 3 classes as what we have. It might have several outputs like: cars, airplanes, flower, orange, etc. Our images might classify as hands but the class label we got doesn’t match what we want. Then I add another layer which modified it to output of 3 and it’s accuracy rate rais up to 27% - 37%. This number is normal guessing expectation.

E(xi) = xi/ sum(x’s)

In this case: 1/3

After our training process, our model accuracy rate increased to 98% on our testing set. This means our model did learn something from training set. But with epoch of 10 and no more than 1400 images, we are afraid that we have trained a overfitting model. The reason why we make this statement is because we didn’t do our feature extraction in a complex way. As data scientists, we know that data is the main driving force of our model. But our data in this case is very bias. All of them have a size of 300\*200 pixel which means our model don’t have ability to make correct decision on predicting different size or image with 0 padding noises. All of our images are with green background, this might confuse our model’s ability of dealing with channels (RGB).

**What We Learned:**

We learned the basic structure of ResNet. How it pass it’s parameter and perform addition to avoid gradient vanishing. And over-sampling of our train set. I still think that our feature exacting process is not enough. From what we observe, our data are all 300\*200 pixels png image, and all back ground are green. If we can copy and change some images color, rotate images, and add some self made image into training set. It might let our model to perform generically better.

Citation

He, K. (2015, December 10). *Deep Residual Learning for Image Recognition*. ArXiv.Org. https://arxiv.org/abs/1512.03385

Szegedy, C. (2015). *Going Deeper with Convolutions –*. Google Research. https://research.google/pubs/pub43022/

Julien de la Bruere-Terreault. (2019). *Rock-Paper-Scissors Images*. Kaggle.

https://www.kaggle.com/drgfreeman/rockpaperscissors