Machine Learning Engineer Nanodegree

Capstone Project

Tae Jun Moon
November 23st, 2018

I. Definition

Project Overview

Convolutional neural networks have had a dramatic impact on the image recognition space. Even simple models are able to make highly accurate predictions on datasets like the MNIST database of handwritten digits.

The MNIST dataset, however, are fairly uniform and relatively simple. It also contains plenty of images per digits. Image recognition in real world does not provide well organized datset. Often datasets are not provided with enough data or quality of data may not be good.

There are datasets with limited amount of data and poor quality in reality. Approach to image recognition on these dataset should be different from the ones with well organized datasets. Some of the solution to this problem is to augment data or use transfer learning technique.

In this project we will attempt to train and test a dataset that contains relatively small amount of data with relatively low image resolution quality using Tiny-ImageNet dataset.

Problem Statement

The objective of this project is to develop and train a deep learning convolutional neural network that will be able to identify an image with similar quality. This will be challenging since the provided data on Tiny-ImageNet only includes 500 images per class for training dataset.

The goal of this project is to improve test accuracy of trained model by only using provided dataset and pretrained weights.

Metrics

Evalutation metrics for this project will be accuracy.

$$Accuracy = \frac{True\ Positive + True\ Negative}{Total}$$

II. Analysis

Data Exploration

Tiny-imagenet is a relatively small dataset compare to imagenet dataset.

Dataset	(ImageNet)	(Tiny-ImageNet)
Number of Classes	1,000	200
Number of training Images	50,000 per class	500 per class
Image Size	(256,256,3)	(64,64,3)

Tiny-ImageNet consist of 200 classes with 500 training images each. Each images are 64x64 pixels with RGB channel. Inputs for this dataset will be image and label.

Algorithms and Techniques

For this project we will use Convolutional Neural Network to train and test the model. We will also use image augmentation to increment the variety of training data and use transfer learning technique to improve the performance of the model. Then we will use normalization so the input value can have similar input range.

CNN

- AlexNet
- ResNet18

Augmentation

- Horizontal Flip
- Random Rotation

Transfer Learning

- Pretrained weight
- Fine Tuning

Benchmark

AlexNet was used as benchmark model for this project. We designed AlexNet to fit 64x64 images. Then we trained AlexNet using tiny-imagenet training dataset without any technique applied except pretrained weight. Performance of AlexNet marked accuracy of 35.88%.

Layer Name	Output Size (Input 64x64x3)	AlexNet
conv1	15x15x64	11x11, 64, stride=4, pad=2
max pool	7x7x64	3x3, stride=2
conv2	7x7x192	5x5, 192, stride=1, pad=2
max pool	3x3x192	3x3, stride=2
conv3	3x3x384	3x3, 384, stride=1, pad=1
conv4	3x3x256	3x3, 256, stride=1, pad=1
conv5	3x3x256	3x3, 256, stride=1, pad=1
max pool	1x1x256	3x3, stride=2
Fully Connected	4096	256x4096
Fully Connected	200	4096x200
Softmax		

III. Methodology

Data Preprocessing

Tiny-ImageNet data is not provided to use it right away. Train dataset is provided in organized form where each image classes are divided into separate folders. However, validation and test datasets are not provided in same format. Also, test dataset is only provided with image files. There are no labels provided with this dataset. Which means we cannot use test dataset unless we manually label it. Thus, in this project we will not use the test dataset, but we will split validation dataset into validation and test datasets.

	Train	Validation	Test
Original	500/class	50/class	50/class
Processed	500/class	25/class	25/class

We will use this same configuration to create 224x224 image dataset.

Detailed preprocessing step is included in run.sh file.

Implementation

We used ResNet18 as our classifier model.

Layer Name	Output Size (Input 224x224x3)	ResNet-18
conv1	112x112x64	7x7, 64, stride=2, pad=3
max pool	56x56x64	3x3, stride=2, pad=1
layer1	56x56x64	[3x3, 64] x 2, stride = 1
layer2	28x28x128	[3x3, 128] x2, stride = 2
layer3	14x14x256	[3x3, 256] x2, stride = 2
layer4	7x7x512	[3x3, 512] x2, stride = 2
average pool	1x1x512	Average Pooling(7x7)
Fully Connected	1000	512x1000
softmax	1000	

Above architecture is the original ResNet18. We have to change the architecture to fit image size of 64x64.

Layer Name	Output Size (Input 224x224x3)	Output Size (Input 64x64x3)	ResNet-18
conv1	112x112x64	32x32x64	7x7, 64, stride=2, pad=3
max pool	56x56x64	16x16x64	3x3, stride=2, pad=1
layer1	56x56x64	16x16x64	[3x3, 64] x 2, stride = 1
layer2	28x28x128	8x8x128	[3x3, 128] x2, stride = 2
layer3	14x14x256	4x4x256	[3x3, 256] x2, stride = 2
layer4	7x7x512	2x2x512	[3x3, 512] x2, stride = 2
average pool	1x1x512	1x1x512	Adaptive Average Pooling(1)
Fully Connected	200	200	512x200
Softmax	200	200	

If we use Adaptive Average Pooling, it will automatically adjust the output size of average pool layer to 1x1x512. Then connect this output to FC layer of 200 output which is the number of class in Tiny-ImageNet.

Below is the code implementation

```
#Load Resnet18 with pretrained weights
model_ft = models.resnet18(pretrained=True)
#Finetune Final few layers to adjust for tiny imagenet input
model_ft.avgpool = nn.AdaptiveAvgPool2d(1)
num_ftrs = model_ft.fc.in_features
model_ft.fc = nn.Linear(num_ftrs, 200)
```

Cross entropy loss was used as loss function, and SGD with momentum was used as optimizer

Learning rate: 0.001

Momentum: 0.9

```
#Loss Function
criterion = nn.CrossEntropyLoss()
# Observe that all parameters are being optimized
optimizer_ft = optim.SGD(model_ft.parameters(), lr=0.001, momentum=0.9)
```

For normalization we calculated the mean and std value of the dataset

```
transforms.Normalize([0.4802, 0.4481, 0.3975], [0.2302, 0.2265, 0.2262])
```

Following is the implementation we did

Model	Pretrained	Augmentation	Normalize	Input Size
ResNet18	X	X	X	64x64
ResNet18	ImageNet	X	X	64x64
ResNet18	ImageNet	X	O	64x64
ResNet18	ImageNet	RandomHorizontal Flip	O	64x64
ResNet18	ImageNet	RandomRotation	O	64x64
ResNet18	ImageNet	RandomHorizontal Flip & RandomRotation	O	64x64
ResNet18	ImageNet	RandomHorizontal Flip & RandomRotation	O	224x224
ResNet18-FineTuned	Model Above	RandomHorizontal Flip & RandomRotation	O	64x64

Refinement

First layer of ResNet18 has stride of 2 followed by maxpool layer with stride of 2. This reduces the information of the image in the early stage of CNN.

For fine tuning, we decided to reduce the kernel size to 3x3, stride to 1, and padding to 1. Then remove max pool layer to keep the output size.

Layer Name	Output Size (Input 64x64x3)	ResNet-18 FineTune
conv1	64x64x64	(3x3, 64, stride=1, pad=1)*
max pool	----------	(Removed)*
layer1	64x64x64	[3x3, 64] x 2, stride = 1
layer2	32x32x128	[3x3, 128] x2, stride = 2
layer3	16x16x256	[3x3, 256] x2, stride = 2
layer4	8x8x512	[3x3, 512] x2, stride = 2
average pool	1x1x512	Adaptive Average Pooling(1)

After fine tuning the layer, we train the model with 64x64 images.

```
#Load Resnet18
model_ft = models.resnet18()
#Finetune Final few layers to adjust for tiny imagenet input
model_ft.conv1 = nn.Conv2d(3,64, kernel_size=(3,3), stride=(1,1), padding=(1,1),
bias=False)
model_ft.maxpool = nn.Sequential()
model_ft.avgpool = nn.AdaptiveAvgPool2d(1)
model_ft.fc.out_features = 200
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model_ft = model_ft.to(device)

pretrained_dict = torch.load('./models/224/model_18_epoch.pt')
model_ft_dict = model_ft.state_dict()
first_layer_weight = model_ft_dict['conv1.weight']
pretrained_dict = {b[0]:b[1] for a,b in zip(model_ft_dict.items(),
pretrained_dict.items()) if a[1].size() == b[1].size()}
model_ft_dict.update(pretrained_dict)
model_ft.load_state_dict(model_ft_dict)
```

IV. Results

Model	Pretrained	Augmentation	Normalize	Input Size	Best Val Accuracy
ResNet18	X	X	X	64x64	21.04%
ResNet18	ImageNet	X	X	64x64	47.02%
ResNet18	ImageNet	X	O	64x64	47.32%
ResNet18	ImageNet	RandomHorizontal Flip	O	64x64	50.82%
ResNet18	ImageNet	RandomRotation	O	64x64	49.76%
ResNet18	ImageNet	RandomHorizontal Flip & RandomRotation	O	64x64	53.33%
ResNet18	ImageNet	RandomHorizontal Flip & RandomRotation	O	224x224	68.54%
ResNet18-FineTuned	Model Above	RandomHorizontal Flip & RandomRotation	O	64x64	68.46%

Model	Pretrained	Augmentation	Normalize	Input Size	Test Accuracy
Alex	ImageNet	X	X	64x64	35.88%
ResNet18	ImageNet	RandomHorizontal Flip & RandomRotation	O	64x64	53.58%
ResNet18	ImageNet	RandomHorizontal Flip & RandomRotation	O	224x224	69.62%
ResNet18-FineTuned	Model Above	RandomHorizontal Flip & RandomRotation	O	64x64	69.80%

V. Conclusion

From the result we can confirm few things.

1. Performance of ResNet18 over AlexNet
2. Validity of pretrained weight
3. Validity of Image Augmentation

AlexNet did not perform well compare to ResNet18 on same configuration.

Model	Pretrained	Augmentation	Normalize	Input Size	Best Val Accuracy
AlexNet	I	X	X	64x64	35.48%
ResNet18	ImageNet	X	X	64x64	47.02%

AlexNet is a simple forward CNN where as ResNet18 uses residual learning which refers to the addition of a skip connection to an existing stacked network. ResNet won the 2015 1st place on the ILSVRC 2015 classification task.

Model	Pretrained	Augmentation	Normalize	Input Size	Best Val Accuracy
ResNet18	X	X	X	64x64	21.04%
ResNet18	ImageNet	X	X	64x64	47.02%

There were also big difference in Validation Accuracy corresponding to existance of pretrained weight. Even though the number of classes and input size in ImageNet is different from Tiny-ImageNet, pretrained weight can be effective on initializing parameters.

Model	Pretrained	Augmentation	Normalize	Input Size	Best Val Accuracy
ResNet18	ImageNet	X	X	64x64	47.02%
ResNet18	ImageNet	X	O	64x64	47.32%

We could not confirm the validity of Normalizing. The best validation accuracy increased by 0.3%, but this cannot be confirmed if it's a meaningful increment or not. Since ResNet18 is a relatively shallow network, containing 18 layers, normalization may not be very effective, but it may be effective on deep networks.

Model	Pretrained	Augmentation	Normalize	Input Size	Best Val Accuracy
ResNet18	ImageNet	X	O	64x64	47.32%
ResNet18	ImageNet	RandomHorizontal Flip	O	64x64	50.82%
ResNet18	ImageNet	RandomRotation	O	64x64	49.76%
ResNet18	ImageNet	RandomHorizontal Flip & RandomRotation	O	64x64	53.33%

Validity of augmentation could be confirm. Compare to model without augmentation and model with augmentation, there are about 6% best validation accuracy difference. This implies that augmentation is effective on improving the performance of the model.

Model	Pretrained	Augmentation	Normalize	Input Size	Test Accuracy
ResNet18	ImageNet	RandomHorizontal Flip & RandomRotation	O	64x64	53.58%
ResNet18-FineTuned	Model Above	RandomHorizontal Flip & RandomRotation	O	64x64	69.80%

Finally, validity of transfer learning using pretrained weight and fine tuning can be confirmed. The initial test accuracy using 64x64 marked 53.58% whereas using pretrained weight from 224x224 image and fine tuning the first few layer to fit 64x64 image marked 69.80%.

Reflection

Through this project we could understand the difference in approach between image with low resolution and medium resolution. Size of the stride may cause too much information loss during convolution. Also it was interesting to observe effectiveness of pretrained weight from ImageNet, even though size and resolution were different. Difficult part of the project was that the amount of provided data was too small. Due to limit on amount of data, the final test accuracy could not mark above 70%. This may also be due to splitting validation dataset into half. Furthur study on image augmentation may help improving the performance of the model. Overall, expected results came out from this project.

Improvement

We only tried with same hyperparameter for this project to compare in equal state. Using different Network, optimizer, loss function, and augmentation method may improve the final result.

Reference

http://cs231n.stanford.edu/reports/2016/pdfs/411_Report.pdf

https://towardsdatascience.com/transfer-learning-946518f95666

https://medium.com/@14prakash/understanding-and-implementing-architectures-of-resnet-and-resnext-for-state-of-the-art-image-cf51669e1624

https://tiny-imagenet.herokuapp.com/