Final report

2020310102 KunWoo Park
2019310762 Yoonkyung Jang
2020310107 Hoyoung Jeon

Abstract - **As a new classification platform, deep learning has recently received increasing attention from researchers and has been successfully applied to many domains. In this report, we review the current activity of image classification methodologies and techniques with some implementations in order to satisfy the goal of project. Image classification is a complex process which depends upon various factors. The main focus will be on advanced classification techniques which are used for improving classification accuracy. Additionally because existing deep neural network models are computationally expensive and memory intensive, hindering their deployment in devices with low memory resources or in application with strict latency requirements, we implement some techniques for compression of model also.**

I. INTRODUCTION

In our everyday life, classification helps us in taking decisions. The need for classification arises whenever an object is placed in a specific group or class depending upon the attributes corresponding to that object. Most of the industrial problems are classification problems. Scientists have devised advanced classification techniques for improving classification accuracy [1, 2]. Every single day numerous images are produced, which creates the necessity to classify them so that accessibility is easier and faster. The information processing which is done during the classification helps in image categorization into various groups.

Here, Deep Neural Networks (DNNs) have brought the state of the art performance in various image classification fields. DNNs usually have a large number of input features, as the consideration of more parameters usually improves the accuracy of the prediction. But also, it means the increasing computational cost and intensivity of memory.

In this report, we review the current activity of image classification methodologies and techniques with some implementations in order to satisfy the goal of project. We have below two goals in this projects.

- Improving the performance of model, as possible as we can, that has limitation on number of model parameters, 2 million

- Compress the model that we trained, so model size should smaller than 300Kbytes at least.

The main focus will be on advanced classification techniques which are used for improving classification accuracy such as learning rate scheduling, fixing train-test resolution, etc., and for compressing model such as distillation, pruning, etc.

II. IMAGE CLASSIFICATION TECHNIQUES TO IMPROVE MODEL PERFORMANCE

A. *Label smoothing*

A mislabeling occurs when there are incorrectly labeled images. The mislabeling occurs especially when the source of the image is from the Internet.

For example, when training a deep learning model that classifies dogs and cats, a cat image but labeled as a dog is trained. If the number of data is small, a person can visually check and correct it, but if the number of data is large, it is not possible.

When calculating the loss while training a deep learning model, the loss value is largely calculated for incorrectly labeled samples. This can cause problems during training.

One possible way to solve this problem is label smoothing. Label smoothing gives the label smoothly rather than 0 or 1 in order to reduce the effect of possible erroneous loss.

B. L2 Regularization

L1 regularization and L2 regularization are used to do regulation. This is a technique used to prevent overfitting.

The reason for dividing L1 and L2 is because of the norm used when applying regulation. The regularization model using L1 regularization is called Least absolute Shrinkage and Selection Operater(Lasso) Regression. L2 Regularization adds a regularization term to the Loss function to be minimized. Including the square of the weight is added to the existing cost function. Learn in a direction where the weight is not too large.L1 regularization can be expressed as followed

$$Cost = \frac{1}{n}\sum_{i=1}^{n}\{L(y_i, \hat{y}_i) + \frac{\lambda}{2}|w|\}$$

As the weight size is included in the existing cost function, it is learned in a direction that the weight is not too large. At this time λ is a constant such as the learning rate, and the closer to 0, the less effective the normalization.

C. Fixing train-test Resolution

As a way to solve this problem, there is a method of training a neural network. It is not a new architecture of a neural network, but a method to solve a problem simply by learning a neural network. This is a method to solve the problem of inconsistency between the distribution of training data and the distribution of test data because the size of the object viewed by the neural network during training and the size of the object viewed during testing are different.

Often, for image classification, a rectangular area is randomly selected from the image during the training time. Then this rectangular area is cropped and grew to a size of 224x224 to train this data. At the time of the test, a rectangular area with a size of 224x224 is selected based on the center point of the input image. After that, it is cropped and passed through the neural network. [3]

Because of this phenomenon, a discrepancy occurs between the distribution of the training data and the distribution of the testing data. This is called 'train-test resolution discrepancy'.

Figure 1 shows how well the fixing train-test resolution performs.

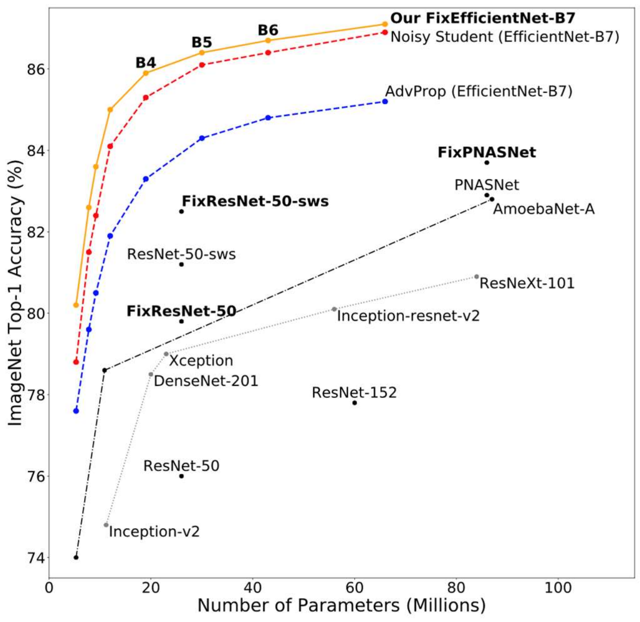


Figure 1. Comparing performance with Fixed Net

D. Resnet

ResNet is an algorithm developed by Microsoft. It is also an algorithm developed by Chinese researchers at the Beijing Research Institute. The original paper name of ResNet is "Deep Residual Learning for Image Recognition". Compared to 2014's GoogLeNet consisting of 22 layers, ResNet has 152 layers. It has a layer that is about 7 times deeper.

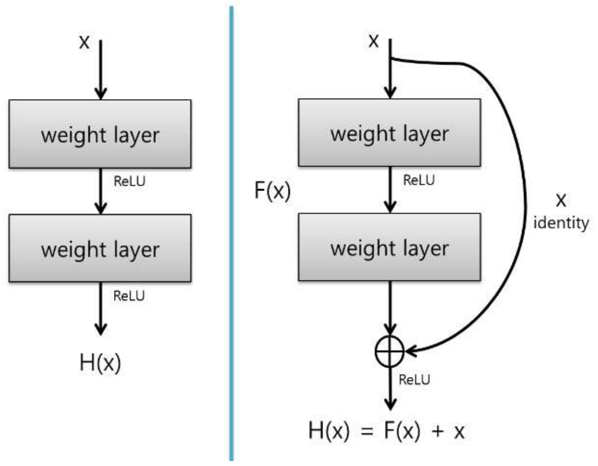


Figure 2. Residual block

Deeper layers do not necessarily improve performance. So, ResNet has a ‘Residual Block’. It has a shortcut to add the input value to the output value. The left part of Figure 2 shows original structure. The right part of Figure 2 shows residual block and the input tensor x becomes F(x) through two convolution layers. This process can be expressed as followed

$$F(x) = \tau(x \circledast w_1) \circledast w_2$$

Here $\circledast$ means convolution operation and τ is activation function(ReLU).

Originally, the convolutional neural network applies ReLU to the result of above equation and transfers it to the next layer. but residual learning adds x which is a shortcut link.so next equation shows this process. And in this equation, F(x) is called residual.

$$y = \tau(F(x) + x)$$

ResNet is similar to VGGNet in that it uses a 3*3 small kernel. But there are several different aspects.

First, it was deepened by acting on residual learning. While VGG-19 has 19 layers, ResNet tries several depths of 34, 50, 101, 152, 1202 layers. Figure 4. Shows 34-layer ResNet. Except for the first convolution, all use a 3*3 kernel. The solid line is the area where the size of the input and output feature maps are the same, and the dotted line is the area where the number of feature maps is different and matched with 1*1 convolution.

Second, all layers apply batch normalization to the convolution operation result and the ReLU. This eliminate the need to apply dropout.

Third, the three fully connected layers used by VGG are replaced with the global average pooling layer.

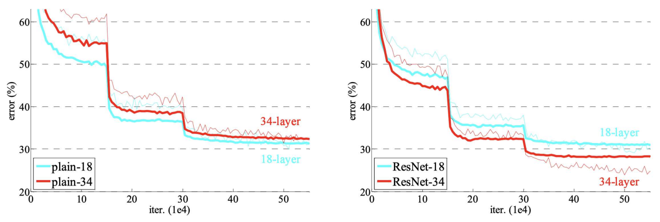


Figure 3. Error plot of each ResNet

when using a regular network, 18layer is better than 34 layers due to the vanishing gradient problem. but Figure 3. shows that when using ResNet,34 layers are better than 18 layers and by skip connection, vanishing gradient issue was fixed. There is not much difference between the 18 layer normal network and the 18 layer ResNet. This is because the vanishing gradient problem does not appear in shallow networks

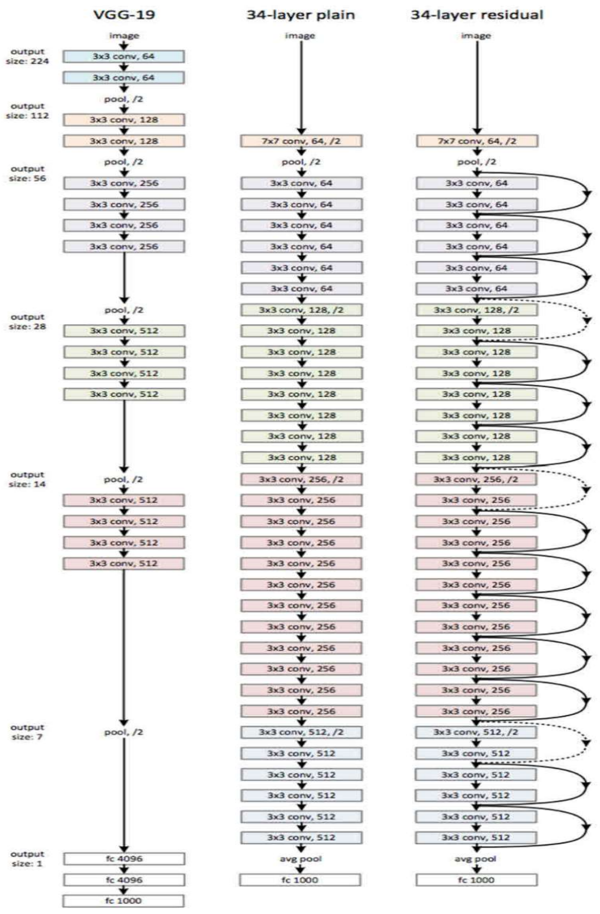


Figure 3. Structure of ResNet

E. Distillation

Knowledge distillation is a concept proposed in a paper titled "Distilling the Knowledge in a Neural Network" published at NIPS 2014. The purpose of knowledge distillation is to transfer the knowledge of a large network (Teacher network) that has been well-learned in advance to a small network (Student network) that wants to actually use it.

Deep learning models are generally wide and deep, so feature extraction is better when the number of parameters and computational amount is large, and accordingly, the performance of the model's purpose such as classification and object detection is improved. However, deep learning is simply "a model with good performance is a good model." It has exceeded the level of skill that can be said.

If a person can get the performance of a larger model with a small model, he can say that it is more efficient in terms of computing resources (GPU and CPU), energy (battery, etc.), and memory. For example, if you want to use an application that utilizes deep learning on a mobile phone, instead of connecting to an online cloud server and using resources such as GPU, you can make this model small enough and calculate it using only the mobile phone's CPU, in terms of various costs. Would be better.

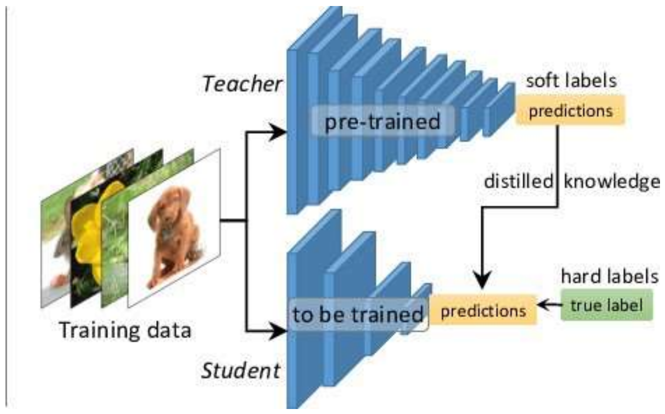


Figure 5. Overview of Distillation

In this way, knowledge distillation allows small networks to perform similarly to large networks. It aims to improve the performance of small networks by transferring knowledge of large networks to small networks in the learning process. Even with relatively few parameters, the performance of the model can be improved.

F. Pruning

Pruning is one of the ways to prevent overfitting. It is regularization by reducing the number of branches in the decision tree. Figure 6 shows pruning.

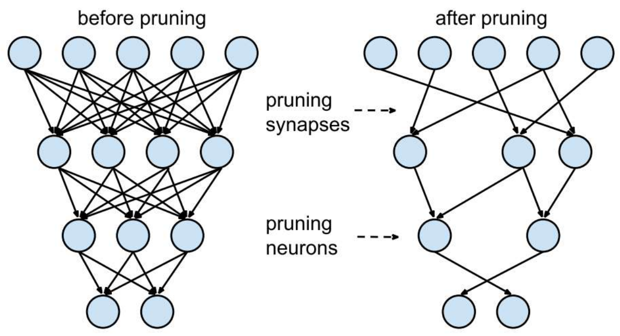


Figure 6. Pruning[4]

As a method of removing unnecessary parts after learning, there are various methods such as removal based on the size of the weight, attention head removal, and layer removal.

Removal based on the size of the weights is a technology that reduces the size of the network model by setting all small weights to 0 because the values required for actual inference among the weights of the existing neural network are resistant to relatively small values. . Subsequent research is proceeding with a method of fine-tuning the neural network in a way that can improve accuracy through retraining after weight pruning.

G. Model Quantization

The reason it is difficult to execute a deep learning model as it is in a mobile or embedded environment is that it has limitations such as memory, performance, and storage space, unlike a general PC. Therefore, depending on how to optimize the deep learning model, it is determined whether or not the optimal performance can be achieved even in an embedded environment.

Quantization is one way to do this. This is the process of reducing the internal composition or expression form of the Neural Network. Quantization is also applied to reduce the expression format from 32bit to 8bit. File sizes can be reduced by storing the minimum/maximum values of each layer and compressing each decimal value into an 8-bit integer representing the nearest real number in 256 linear sets within a certain range.

By executing a neural net with 8 bit input/output, you can save the computational resources required for inferencing calculations. Only 25% of the decimal point memory requirement is used to use an 8 bit value. You can avoid bottlenecks for better cache usage and RAM access, and you can get more DSP chips available. This quanitization-based decimal point operation can be particularly useful for IoT Embedded Systems.

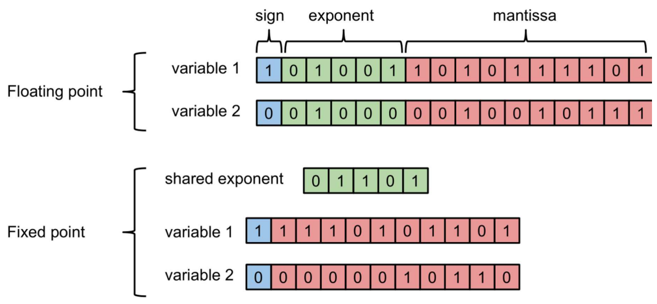


Figure 7. Quantization[5]

H. Normalization

Normalization has been an active area of research in deep learning. Benefits of using normalization is as followed. First, Normalization usually make networks to learn faster because normalization doesn`t allow weights to explode all over the place and restricts them to certain range.

Second, it reduces internal covariate shift.

Third, an unintended benefit of normalization is that it helps network in regularization.

Last, it makes network avoid local optimum.

We can restrict data scales to the extent we want by using normalization. The formula is as follows

$$\tilde{X} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

Normalization make features maintain the contribution of every feature. So, network could be unbiased.

I. Batch normalization

Batch normalization is a normalization that normalizes activations in network by mini-batch. In other words, batch normalization computes the mean and variance in the mini-batch and then subtracts the mean and divides the feature by its mini-batch standard deviation. This method restricts the activations to have zeros means and unit standard deviation. And Batch normalization rescales the normalized activations and adds a constant.

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \text{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \text{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Figure 8. Algorithm of Batch normalization

By above algorithm, We can see batch normalization algorithm.

Then what is advantages of using batch normalization? First of all, batch normalization makes networks train faster. Without batch normalization, if we set learning rate too high, gradient explode or vanish or get local minima. This happens because of parameter`s scale. But batch normalization was not affected by parameter`s scale at propagation. So, we can set high learning rate and that thing increases the speed at which networks train. And batch normalization provides a bit of regularization. This thing avoids to use weight regularization term and dropout. Dropout is good method but usually this make networks learn slowly.

J. Data augmentation

Recent advances in deep learning models have been attributed to vast amounts of data and diversity of data. But if we don`t have enough amounts of data, we will easily get overfitting problem that decrease performance. And it is too hard to get quality data. So, data augmentation is good method to us. Data augmentation is a strategy that enables user to increase the diversity of data without collecting new data. Data augmentation techniques such as cropping, padding, and horizontal flipping are commonly used to train large neural networks like Figure 9

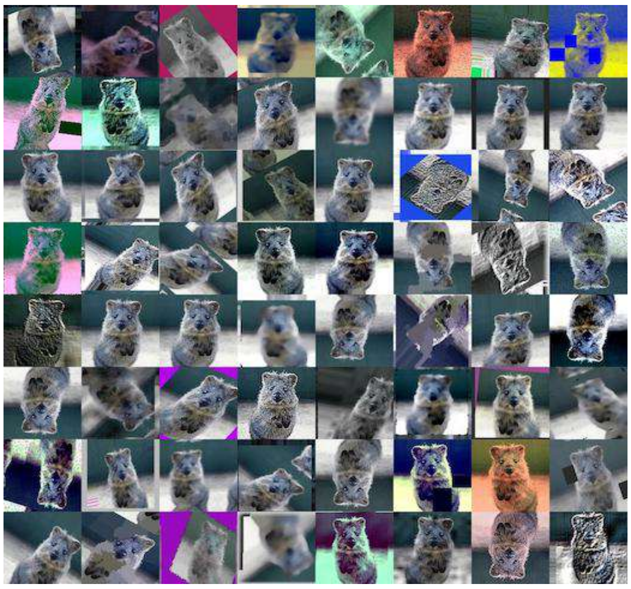


Figure 9. Example of Data augmentation

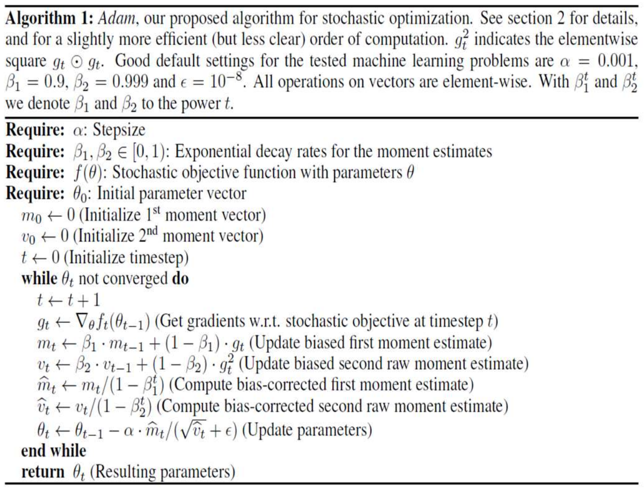
in fact, it is known that data augmentation is effective for model training. AlexNet introduce data augmentation for the first time. At that time AlexNet use heavily data augmentation. And VGGNet also use data augmentation techniques like preprocessing & augmentation. VGGNet decrease error rate from 10.4% to 7.1%.

K. Adam optimizer

Adam optimizer(Adaptive Moment estimation) is algorithm that find objective function`s minimum value and it is algorithm that combines the basic ideas of Momentum and RMSProp. Adam is different to classical stochastic gradient descent. It uses the squared gradients to scale the learning rate like RMSprop and it takes advantage of momentum by using moving average of the gradient instead of gradient itself like SGD with momentum.

The main advantage of the Adam method is that stepsize is not affected by gradient`s rescaling. Even if the gradient increases, the stepsize is bound, so it is possible to stably descend to stably descend for optimization no matter what objective function is used. In addition, stepsize can be adapted by referring to the size of the gradient in the past.

All of the experimental results showed that Adam performed overwhelmingly better than other techniques we can see this fig 9~10.and Adam`s pseudo-code is as follows.


Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

```
Require: $\alpha$: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters $\theta$
Require: $\theta_0$: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while $\theta_t$ not converged do
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep $t$)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return $\theta_t$ (Resulting parameters)
```

Figure 10. Algorithm of Adam optimizer

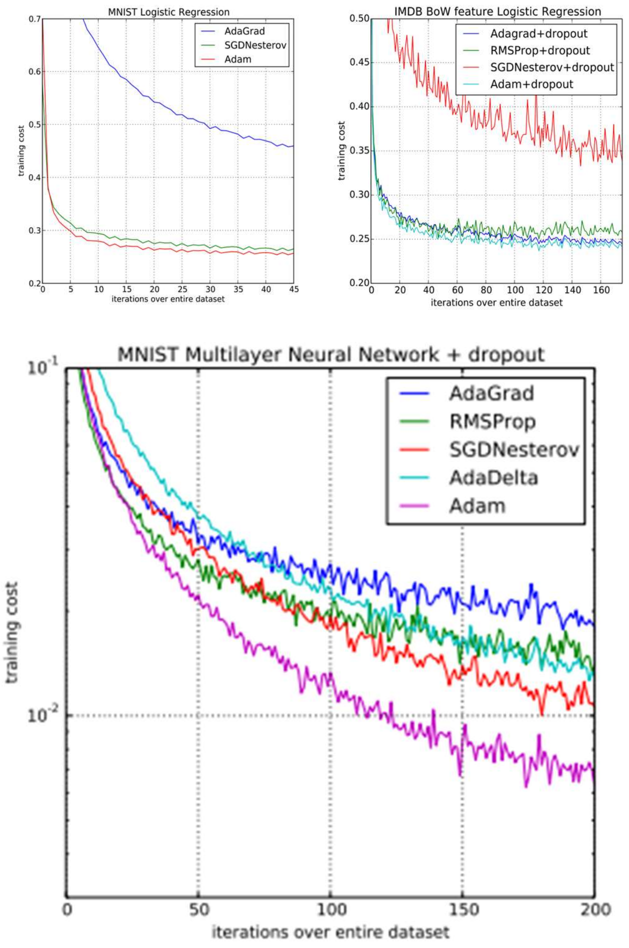


Figure 11. Comparing performance of optimizers with Adam

L. Learning rate scheduling

When training deep neural networks, to increase speed which network train, we reduce the learning rate. And we usually use pre-defined learning rate schedules to reduce the learning rate.

Learning rate schedules seek to adjust the learning rate during training by reducing the learning rate according to a pre-defined schedule. Common learning rate schedules include step-wise dacay and reduce on loss plateau decay

First, step-wise decay can be seen above formula. This formula`s parameter means that eta is learning rate.

$$\eta_t = \eta_{t-1}\,\gamma$$
$$\gamma = 0.1$$

And second, Reduce on loss plateau decay is reduce learning rate whenever loss plateaus. Here plateau mean that to reach a state or level of little or no growth or decline, especially to stop increasing progressing. And patience is after stop decreasing error, how many epochs to reduce the learning rate. If epoch is high ,then we must set patience value is high. And patience, decay factor is hyperparameter

M. Cosine annealing learning rate scheduling

Cosine annealing is a way to increase the accuracy by gradually adjusting the learning rate in the form of cosine. When training, the weight is not corrected by reflecting 100% of the difference between the error and the correct answer immediately, but training proceeds to the extent specified by the learning rate. This number usually range from 0.01 to roughly 0.000001, although it is possible to train with this value. However, it is possible to search for several optimum points while continuing to change them, or to find a better optimum point such as fine tuning

In addition, at each restart point, a different model may pop out, which can be used to further increase the accuracy of the ensemble.

In fact, Cosine annealing and model checkpoint ensemble method are often seen, but cosine annealing can be done by lowering the amount of training for only one cycle, whereas the ensemble method takes a long because it requires multiple epochs.

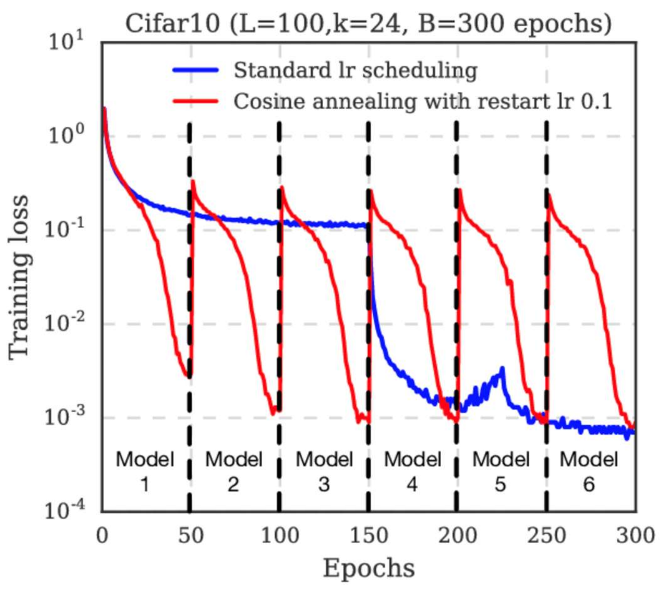


Figure 12. Cosine annealing train loss

III. EXPERIMENTAL RESULT AND ANALYSIS

As the introduction, Goals of this project are two things. First, improve the performance of model, as possible as we can, that has limitation on number of model parameters, 2 million. Another one is compress the model that we trained, so model size should smaller than 300Kbytes at least. We have been provided 5000 data samples which is composed of 10 classes(truck, airplane, etc.) in STL10, and has 500 samples for each class.

At the beginning of project, we should divide given dataset to training set and validation set, which are used for training the model and validating model literally. We used 4000 samples for train and 1000 samples for validation.

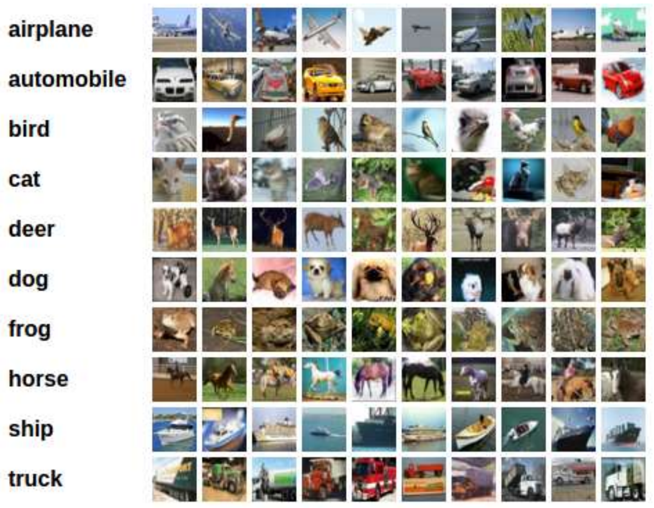


Figure13. STL 10

Before make our own model, we should check how high accuracy that we can be reached. So we tested with the pretrained models provided on pytorch, especially ResNext(based on ResNet). The highest top 1 accuracy on validation that we had reached with pretrained models is about 93% for ResNext101_32x8d.

Although we had reached highest validation accuracy with ResNext, the first model we made is inspired on VGG Net not on ResNet, because we thought that if the model is deeper, it would be harder to train. Figure 15 shows you the base structure of model that we made at first.

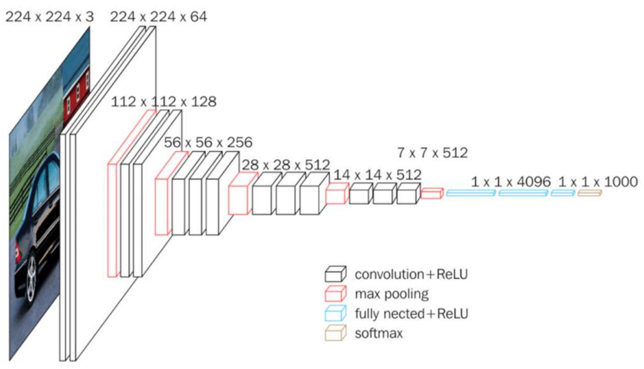


Figure 14. Architecture of VGG Net

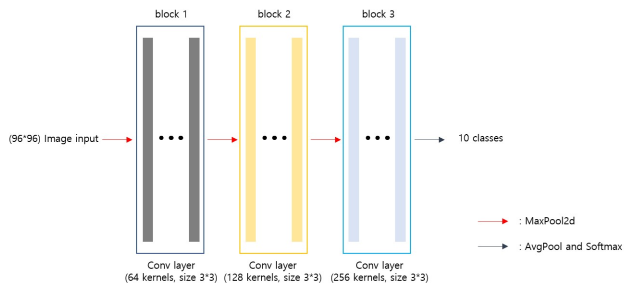

block 1
block 2
block 3
(96*96) Image input
10 classes
: MaxPool2d
: AvgPool and Softmax
Conv layer (64 kernels, size 3*3)
Conv layer (128 kernels, size 3*3)
Conv layer (256 kernels, size 3*3)

Figure15. Architecture of first customized network

We trained with changing 3 hyper parameters which are learning rate, coefficient of l2-regularizer, and batch size for 100 epochs each. But no matter how we change the hyper parameters, as shown on Table 1. performance did not get better. And we should change the base model structure to another model.

Learning rate	L2 coefficient	Batch size	validation acc(%)
1e-2	1e-3	64	41.01
		128	42.66
	1e-4	64	42.96
		128	44.98
1e-3	1e-3	64	42.71
		128	43.38
	1e-4	64	44.41
		128	45.76
1e-4	1e-3	64	44.95
		128	45.44
	1e-4	64	44.56
		128	43.25

Table 1. Result of first customized model

Model	Learning rate	L2 coefficient	validation acc(%)
ResNext101_32x8d	1e-3	1e-3	65.42
		1e-4	66.56
	1e-4	1e-3	65.54
		1e-4	68.08
LeNet-5	1e-3	1e-3	63.52
		1e-4	64.18
	1e-4	1e-3	64.00
		1e-4	64.24

Table2. Result of LeNet and ResNet (not use advanced technics)

According to Table 1., the learning rate value of 1e-2 is not useful so we rejected 1e-2 value in next experiments. Also, it seems good to use batch size as 128 generally, so we fixed batch size as 128 for every next experiment. We did the additional experiment by using LeNet and ResNet anyway, and though we didn't use any advanced technics for training, as shown on Table 2., results of LeNet and ResNet is better than first customized model. Additionally, we fixed L2 coefficient because it seems better using L2 coefficient as 1e-4 than 1e-3. After this, we decide base structure as ResNet and re-struct it to satisfy the condition of goal.

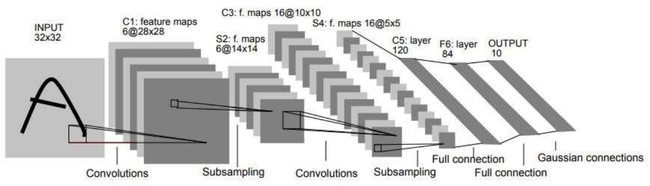
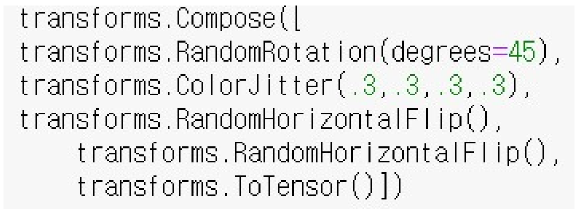


Figure16. Architecture of LeNet

500 samples for each class is not enough to get a good results. To improve performance we should modify our data. So, we augmented given data using following codes, 20 times.



```
transforms.Compose([
transforms.RandomRotation(degrees=45),
transforms.ColorJitter(.3,.3,.3,.3),
transforms.RandomHorizontalFlip(),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor()])
```

Figure 17. Data augmentation code

Here, some simple examples for each augmentation technic on Figure 18 – Figure 20.

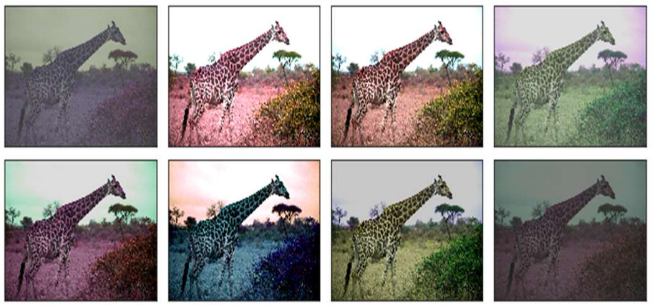


Figure 18. Color jitter

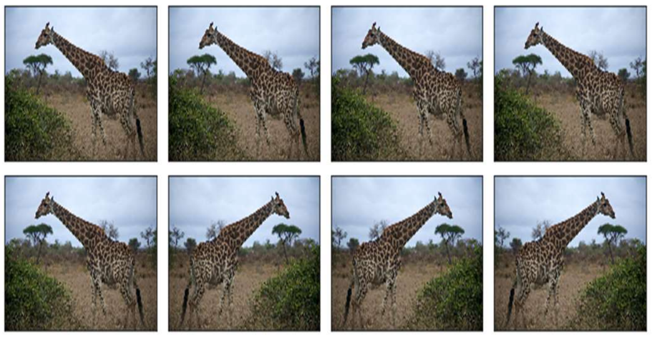


Figure 19. Random Horizontal flip

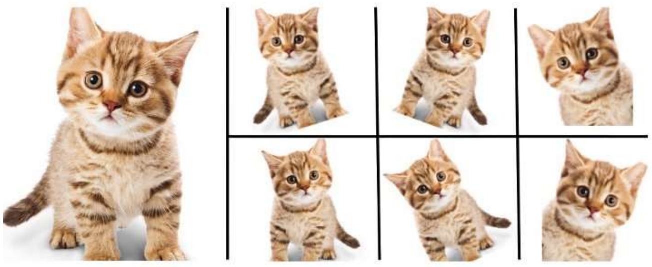


Figure 20. Random rotation

The results after augmented data is shown on Table 3.

Model	Learning rate	L2 coefficient	validation acc(%)
ResNext101_32x8d	1e-3	1e-3	67.41
		1e-4	68.23
	1e-4	1e-3	67.66
		1e-4	70.02
LeNet-5	1e-3	1e-3	65.24
		1e-4	66.83
	1e-4	1e-3	66.30
		1e-4	67.65

Table 3. Results After Data augmentation

In ResNet, the 512 kernel blocks is dominant on number of model parameters, so we delete 512 blocks and customize size of kernels and number of kernels. Overview of customized ResNet is following figure.

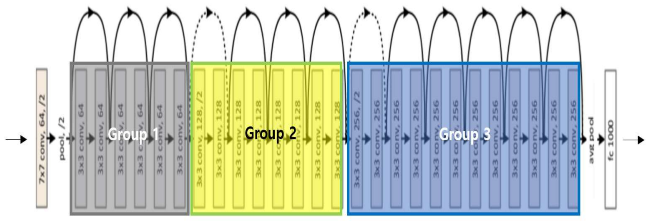


Figure 21. Customized ResNet

We changed the number of blocks of each group, size of kernel, and the number of kernels. After several experiments, we finally decided model with below parameters.

- First conv layer
  - kernel size : 4*4
  - stride : 1
- Group 1
  - Number of blocks : 4
  - Number of kernels : 30
  - Stride : 1
- Group 2
  - Number of blocks : 9
  - Number of kernels : 60
  - Stride : 2
- Group 3
  - Number of blocks : 3
  - Number of kernels : 90
  - Stride : 2
- Total parameters : 1996986 (under 2M)

We reached accuracy about 69.48% under 2M parameters.

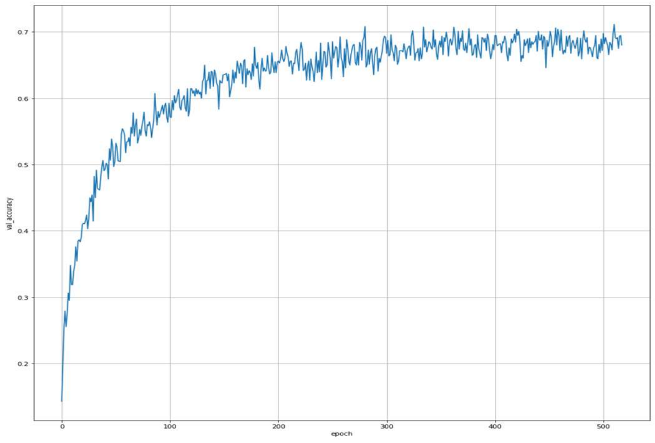


Figure 22. Result of final decided model

And now, we changed the activation function ReLU to Leacky ReLU which has negative slope 0.1. It increased accuracy about 1% as shown on Figure 24.

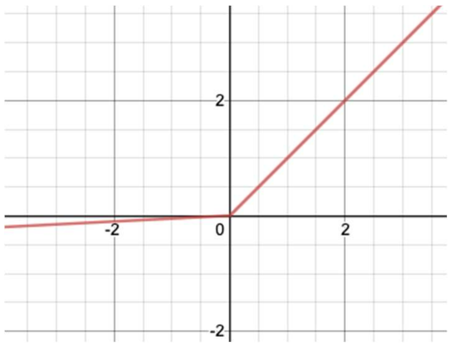


Figure 23. Leacy ReLU

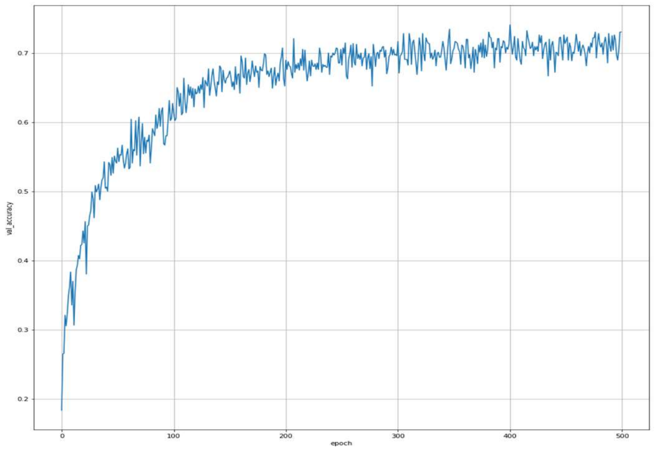


Figure 24. Accuracy after change activation function as leaky ReLU

The things left to do for us was applying advanced technics. First we apply learning rate scheduling. We used simple step reduce scheduler and with some experiments, we could decide step size as 50. Result of scheduling learning rate, shown as Figure 25, is improve accuracy about 2-3%.

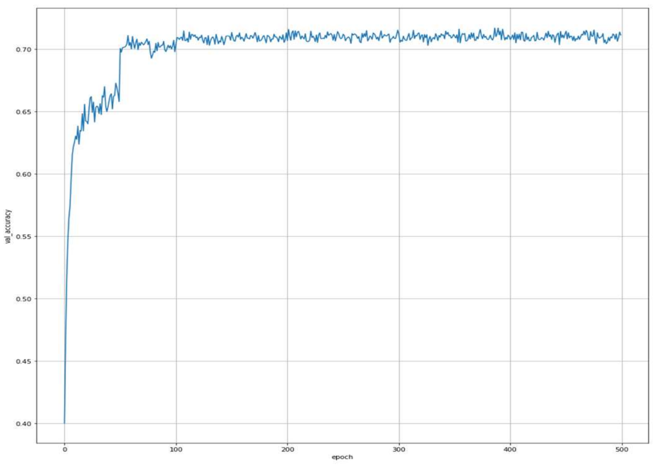


Figure 25. Result after apply Step LR scheduler

There is another advanced technic we used which name is fixing the train-test resolution discrepancy. So we trained 70*70 resolution for first 140 epochs and fine tuned 80 epochs for test which resolution is 120*120. This makes accuracy increased dramatically about 10%.

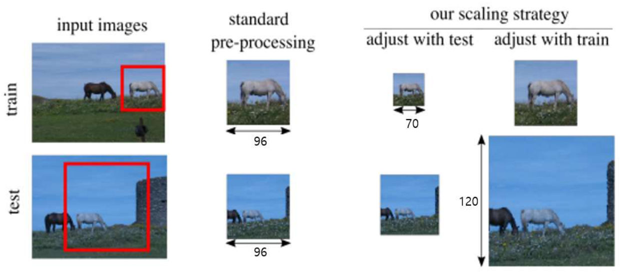


Figure 26. Overview of Fixing the train-test resolution discrepancy

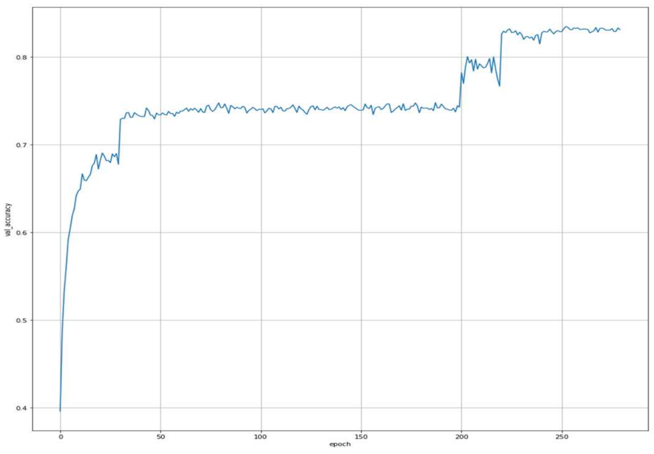


Figure 27. Result after apply Fixing the train-test resolution discrepancy

We thought that if scheduler can increase learning rate after converge local minimum then it can be possible to get out from local minimum. That is the base idea of cyclic learning rate scheduling. Also, it would be better that use smooth decreasing learning rate and decreasing slowly after every epoch than linearly decreasing. This one is the base idea of cosine annealing learning rate scheduling with warm restart. We thought it would get better performance if we use both of idea. So we designed our own customized scheduler shown as Figure 30.

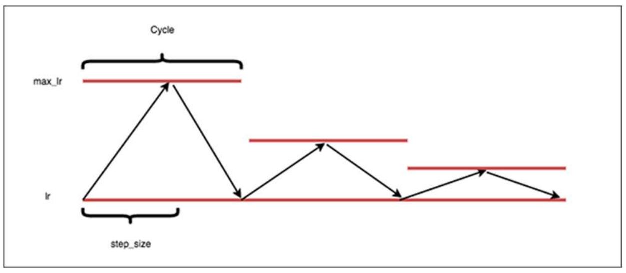


Figure 28. Cyclic LR scheduler

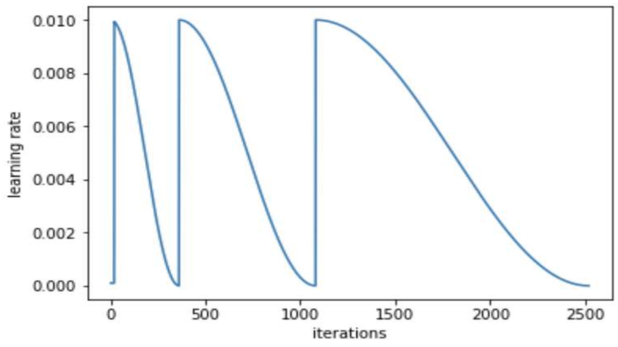


Figure 29. Cosine annealing LR scheduler with warm restart

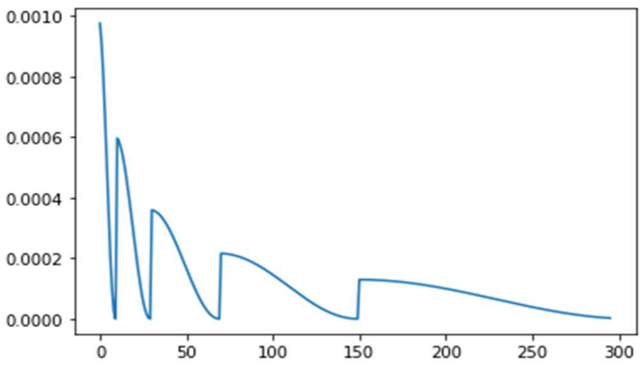


Figure 30. Customized LR scheduler

We start scheduling from 1e-3, and initially decrease it by cosine form during 20epochs. And the cycle of decreasing epoch is increases by multiply 2 to previous cycle. We apply it to fixing the train-test resolution discrepancy method, training model 300 epochs for 70*70 resolution and 150 epochs for 120*120 resolution. This make model more accurately about 1%.

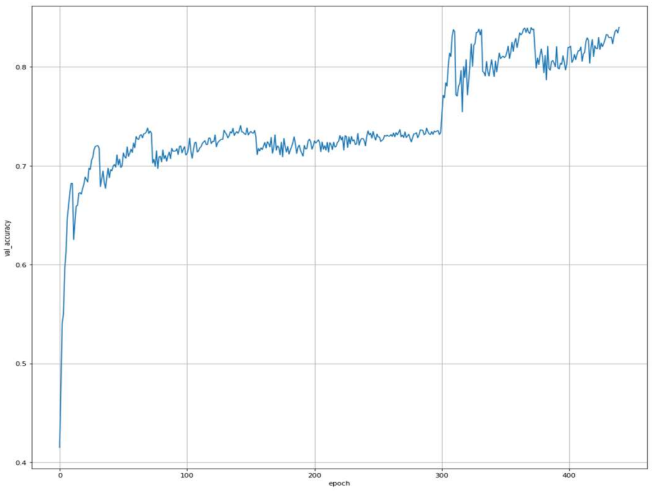


Figure 31. Result for applying customized LR scheduler

Since, there is one goal left which is compressing the model, we found a few options, pruning, quantization, distillation. But we didn't implement quantization because pytorch doesn't support quantization not gpu, but only cpu. So we started distillation first. To decide teacher model, we came up with the experiment that had done at beginning, about ResNext101_32x8d.

We used the pretrained model from torch.vision library, and trained 300 epochs for 70*70 resolution and 150 epochs for 120*120 resolution to fit our dataset. So, we could get accuracy about 94% for teacher model.

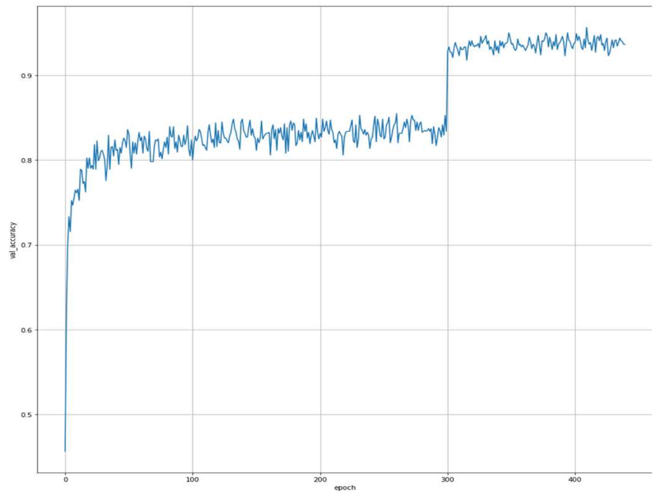


Figure 32. Accuracy for Teacher model

We trained for student model 200 epochs, with below loss function, and didn't apply any technics considering that if train environment has discrepancy between teacher and student model, it could cause instability to converge.

$$Distillation\ loss = KLdivergence\left(\log softmax\left(\frac{out_{student}}{T}\right), softmax\left(\frac{out_{teacher}}{T}\right)\right) * (T^2 * 2 * \alpha) + CrossEntropy(out_{student}, \text{labels}) * (1-\alpha)$$

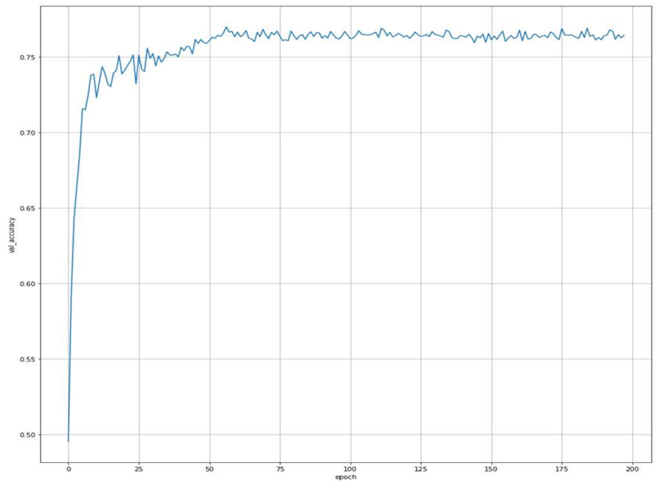
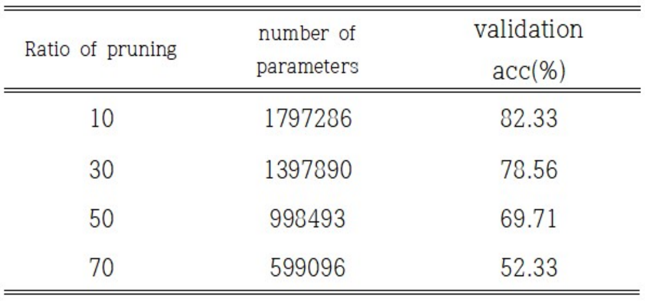


Figure 33. Accuracy for student model



Ratio of pruning	number of parameters	validation acc(%)
10	1797286	82.33
30	1397890	78.56
50	998493	69.71
70	599096	52.33

Table 4. Accuracy according to increasing ratio of pruning

According to Figure 33., we didn't make progress compared with training from scratch model that we made. So we used model that we trained before, accuracy about 84%, for applying pruning method. Shown as Table 4., which shows accuracy and number of parameters, prune 30% of network seems to be reasonable to final model.

IV. CONCULSION

In this report, we have implemented about the different image classification models such as LeNet, ResNet, customized ResNet, etc., and applied various technic to improve training or model performance such as LR scheduling, fixing the train-test resolution discrepancy method, etc. At the begin of this project, we know almost nothing about deep learning so we worried a lot. In fact, unfortunately, the final part of the project, model compression, was not implemented properly due to a lack of time because of various external factors, but this process still helped us get to know the various technologies of deep learning. We was able to learn the overall sense of training in field of deep learning by building the own model and modifying various hyperparameter with observing the results.

The area that we want for further study is the model compression since we did not implement technics clearly yet. So our further study is applying distillation with various student model, implementing quantization by using gpus, and compress the model from result of above methods using pruning method.

REFERENCES

[1] P. Gong, and P.J. Howarth, ―Land-use classification of SPOT HRV data using a cover frequency method,‖ International Journal of Remote Sensing, vol.13, no.8 pp.1459-1471,1992.

[2] M. Pal, P. M. Mather, ―An assessment of the effectiveness of decision tree methods for land cover classification,‖ Remote sensing of environment, vol. 86, no. 4 pp. 554-565, 2003.

[3] Touvron, H., Vedaldi, A., Douze, M., & Jégou, H. (2019). Fixing the train-test resolution discrepancy. In Advances in Neural Information Processing Systems (pp. 8252-8262).

[4] https://medium.com/@souvik.paul01/pruning-in-deep-learning-models-1067a19acd89

[5] https://heartbeat.fritz.ai/8-bit-quantization-and-tensorflow-lite-speeding-up-mobile-inference-with-low-precision-a882dfcafbbd