Deep Residual Learning for Image Recognition (2015)

1. Palaash Agrawal 2018A3PS0565P
2. Rhythm Goel 2018A1PS0002P
3. Shrut Kharod 2018B4A8PS0600P

ABOUT THE PAPER

Title: Deep Residual Learning for Image Recognition

Authors and Affiliations:

1. Name: Kaiming He
   Affiliation:Microsoft Research, Current: Facebook AI Research (FAIR)
2. Name: Xiangyu Zhang
   Affiliation: Microsoft Research, Current: MEGVII Technology
3. Name: Shaoqing Ren
   Affiliation: Microsoft Research, Current: University of Science and Technology of China, 230026, Hefei, China
4. Jian Sun
   Affiliation: Microsoft Research, Current: MEGVII Technology

PAPER DESCRIPTION

Aim: To build a residual learning framework to ease training of substantially deeper networks than used before.

- Network depth is often regarded as one of the most crucial parameters towards training.
- Consequently, as the depth increases, so does the difficulty and error while training the network, as can be seen from the image alongside (taken from the literature itself).
- To ease the training procedure, the authors have tried to adapt a residual learning framework for the same.

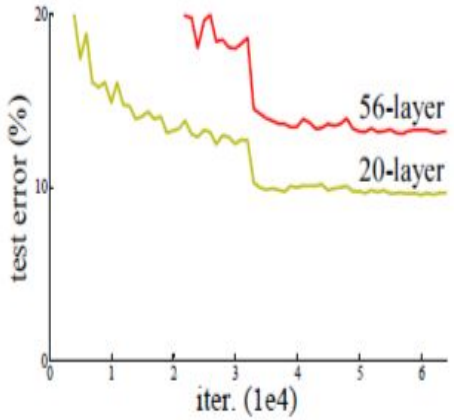

Fig 1: Test error of a “plain” network on CIFAR-10 Dataset

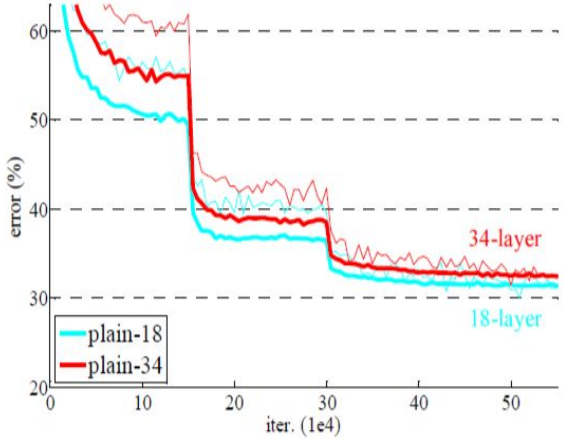

Fig 2: Test error of a “plain” network on ImageNet Dataset

PAPER DESCRIPTION

Methodology: Residual Learning Network

- An integral part of Residual Network is a Skip Connection, which are directly connected to the output.
- Instead of layers learning and trying to fit the underlying mapping, they are instead allowed to fit the residual mapping.
- The entire network can still be trained end-to-end by SGD with backpropagation, and can be easily implemented using common libraries.
- Shortcut connections used here are identity mappings.

PAPER DESCRIPTION

Outcome

- Extremely deep residual nets are easy to optimize (as can be seen from Figures 3 and 4) but the counterpart “plain” nets exhibit higher training error when the depth increases.
- Deep residual nets can easily enjoy accuracy gains from increased depth, producing results significantly better than previous networks.
- Accuracy wise results are shown in further sections.

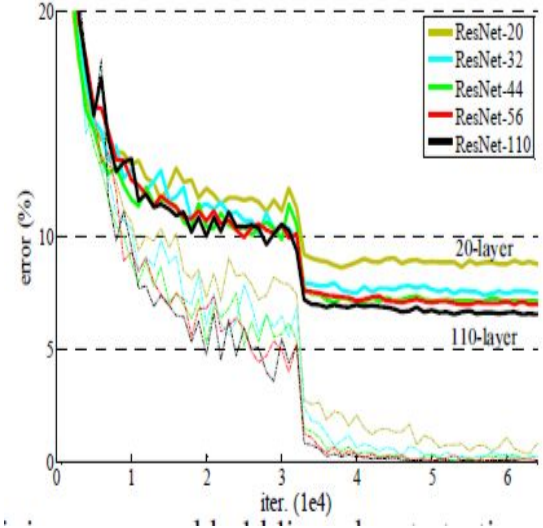

Fig 3: ResNet Results on CIFAR-10 Dataset

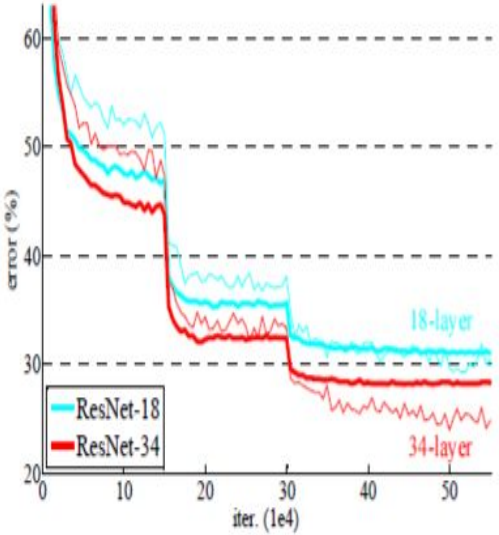

Fig 4: ResNet Results on ImageNet Dataset

BACKGROUND CONCEPTS

Need for a Residual Learning Framework

- Initial problem with increasing depth of networks: problem of vanishing/exploding gradients
  => The problem of Vanishing Gradient arises when a deep network is unable to propagate useful gradient information from the output back to the layers near the input end of the model, often due to the increased depth of the network.
  => Exploding gradients are a problem when large error gradients accumulate and result in very large updates to the network model weights during training, again propagating due to the increased depth of the network.
  => largely resolved by normalization of initial and intermediate layers.
- Problem of Degradation:
  => With increasing depth, accuracy gets saturated, and the degrades rapidly.
  => such degradation is not caused by overfitting, and adding more layers to a suitably deep model leads to higher training error.
  => Identity mappings were hypothesized to be the one solution towards this.

BACKGROUND CONCEPTS

Residual Learning

- Rather than expecting stacked layers to approximate H(x), they are explicitly let to approximate a residual function, F(x).
- The authors hypothesize that if the added layers can be constructed as identity mappings, a deeper model should have training error no greater than its shallower counterpart.
- If identity mappings are optimal, the solvers simply drive the weights of the multiple nonlinear layers towards zero to approach identity mappings, thereby rendering it easier for the solver to train this function.

Input to the first layer	x
Underlying Mapping	H(x)
Residual Function	F(x):= H(x) - x
Original Function	F(x) + x

Table 1: Symbols and Equation important for a Residual Learning Framework

BACKGROUND CONCEPTS

Identity Mappings with Skip Connections

- Building Block defined mathematically as:
  $y = F(x, \{W_i\}) + x$, or $y = F(x, \{W_i\}) + W_s\, x$
  where $F(x, \{W_i\})$ represents the residual mapping to be learned. W_s is used when dimensions of x and F are not equal.
- Here, $F = W_2\boldsymbol{\sigma}(W_1 x)$, where $\boldsymbol{\sigma}$ denotes ReLu
- These skip connections neither introduce extra parameters, nor do they add to the computational complexity.
- They prove advantageous because if any layer hurts the performance, it is simply skipped by regularization. This is how it helps in training very deep networks without vanishing/exploding gradient.

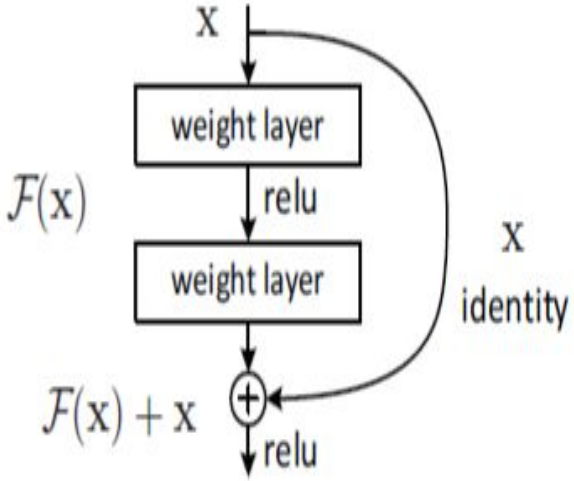

Fig 5: A building block for the Residual Learning Framework

BACKGROUND CONCEPTS

Network Architecture: Plain Network

- Mainly inspired by VGG Nets
- Filter size: 3x3
- Stride: 2
- Average Pooling
- Fully Connected Layers: 1000
- Weighted Layers: 34
- Fewer Filters and Lower Complexity as compared to VGG Nets.

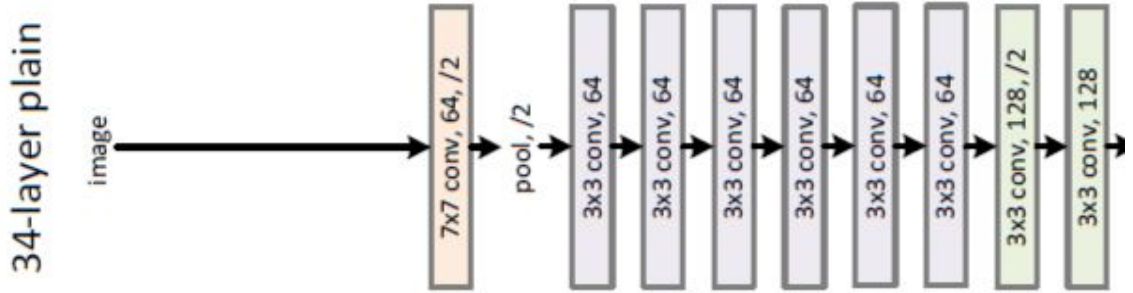

Fig 6: A segment of the Plain Network

BACKGROUND CONCEPTS

Network Architecture: Residual Network

- Based on the above Plain Network
- Inserting shortcut connections
- Solid Lines indicate direct identity shortcuts when input and output are of same dimensions.
- Dotted Line Shortcuts are for when the input and output dimensions do not match -
  => extra zero entries padded for increasing dimensions or,
  => The projection shortcut in is used to match dimensions.

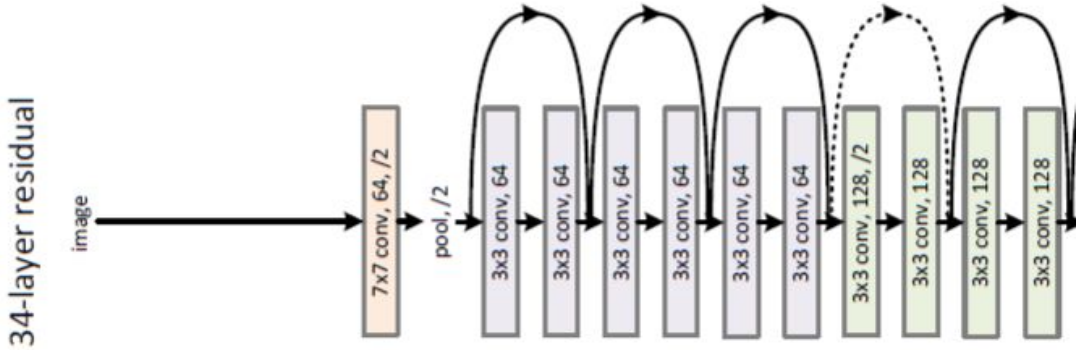

Fig 7: A segment of the Residual Network

DATASET DETAILS

ImageNet

- ImageNet is a large visual database designed for use in visual object recognition software research.
- It is a 2012 Classification Dataset and consists of 1000 classes, such as “cat” or “ strawberry”, etc.
- It has around 1.28 Million Training Images, 50,000 Validation Images, and 100,000 Test Images.

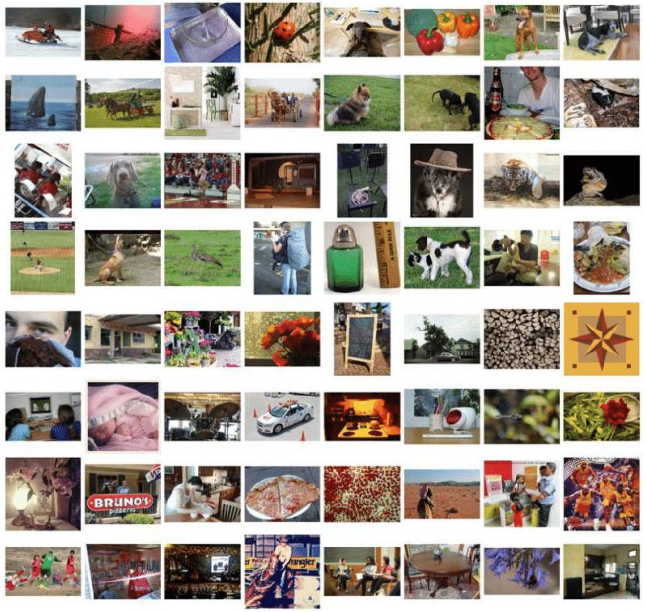

Fig 8: An Example of Images in the ImageNet Dataset

DATASET DETAILS

CIFAR-10

- CIFAR-10 is a set of images that can be used to teach a computer how to recognize objects.
- It is one of the most widely used datasets for Machine Learning an Computer Vision research.
- It has 32x32 colored images.
- CIFAR-10 Dataset consists of 10 classes, such as “airplanes” or “cars”, etc.
- It has roughly 60,000 Training Images, 10,000 Test Images, and 6000 images in each class.

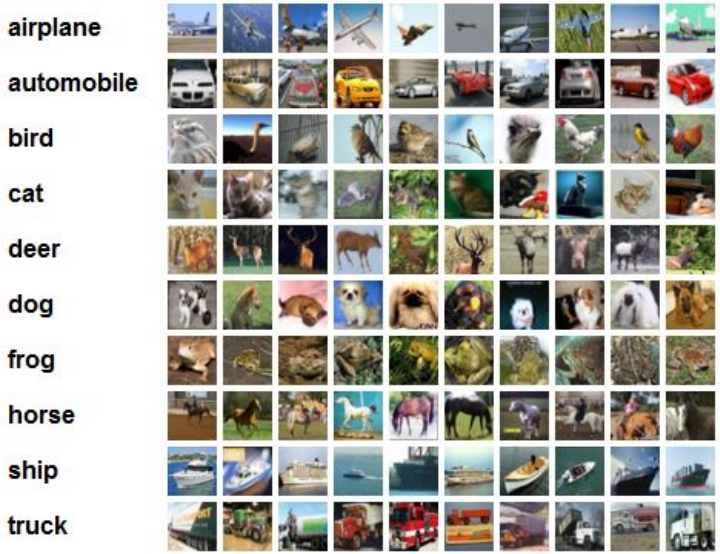

Fig 9: An Example of Images in the CIFAR-10 Dataset

IMPLEMENTATION TASKS

Allotted Tasks

- Our first task was to train a ResNet-50 architecture on the CIFAR-10 Dataset and use it for prediction on test data.
  => Progress: We have successfully built a ResNet-50 from scratch and trained it on the CIFAR-10 Dataset and used it for prediction, the results of which have been elaborated in further sections.

- Our second task was to use Genetic Algorithm to optimize and tune the hyperparameters of the network, and compare loss incurred with respect to that without Genetic Algorithm.
  => Progress: - we have successfully implemented the code for Genetic Algorithm and used it to optimize hyperparameters of our model.

IMPLEMENTATION DETAILS: Resnet Architecture

Task 1: Training and Predicting using a ResNet-50 on CIFAR-10 Dataset

The basic block is called the Residual block, which is basically a sum of two branches, one with Convolutional operations, and the other representing an identity mapping.

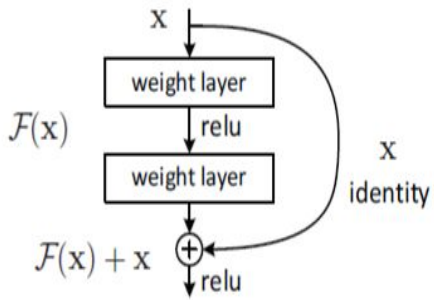

Fig 10: Architecture of Residual Block

```
class ResBlock(Module):
    def __init__(self,ni,nf,stride=1):
        self.convs = _conv_block(ni,nf,stride)
        self.idconv = noop if ni==nf else ConvLayer(ni,nf,1,act_cls=None)
        self.pool = noop if stride==1 else nn.AvgPool2d(2,ceil_mode=True)

    def forward(self,x):
        return F.relu(self.convs(x) + self.idconv(self.pool(x)))
```

Fig 11: Code for the Residual Block

Next, we need to put together all the blocks in sequence, which can be done using a single class. The entire flowline of a resnet stem is as follows.

ARCHITECTURE DETAILS (PSEUDO CODE)

1. Create a Resnet Stem (which does not contain any identity branches).
2. Create blocks with appropriate number of channels.
3. Final Layer (512 channels) will be average pooled.
4. FC layer, with as many neurons as number of classes.

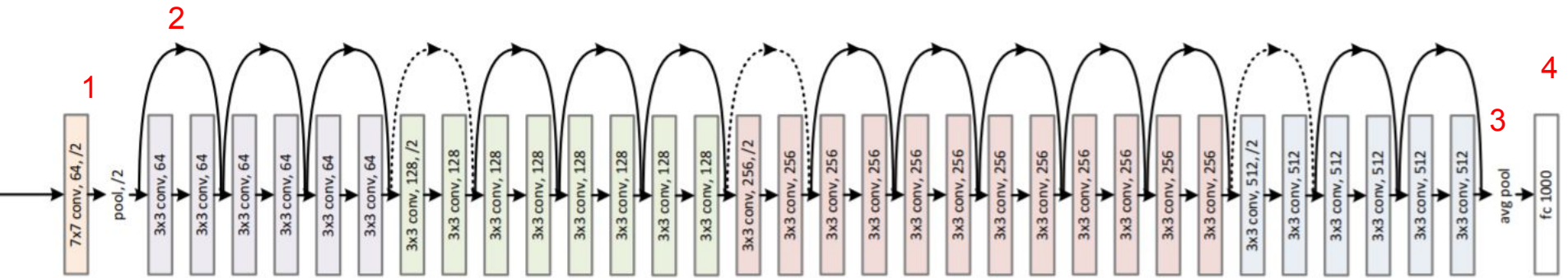

Fig 12: Network Architecture

TRAINING PROCEDURE DETAILS (PSEUDO CODE)

A cyclic training procedure is used for training (Leslie Smith, 2018). This includes starting with a low learning rate, gradually increasing in the middle of the cycle, then gradually annealing the learning rate. At the same time, momentum is first decreased, then increase gradually. The maximum learning rate is determined using a one cycle fit policy (Smith 2015).

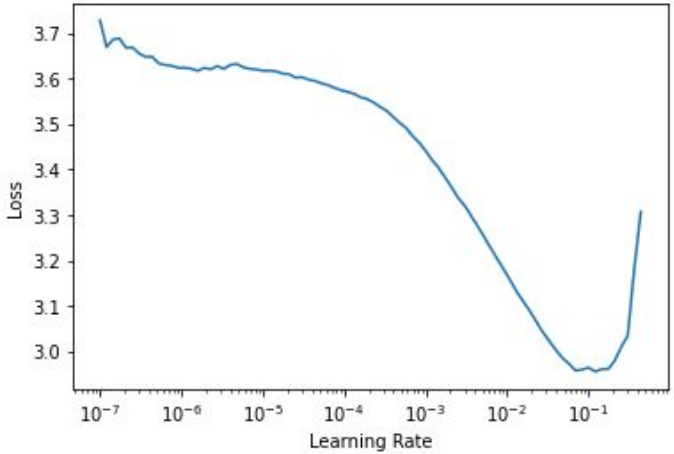

Fig 13: Learning rate; chosen as the smooth part initially (~3e-3).

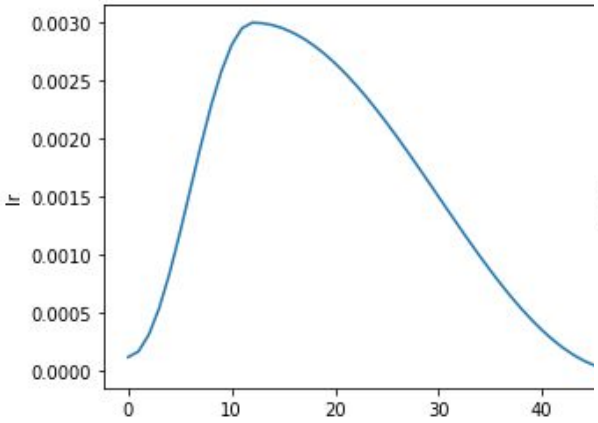

Fig 14: Learning Rate Schedule (Smith 2018)

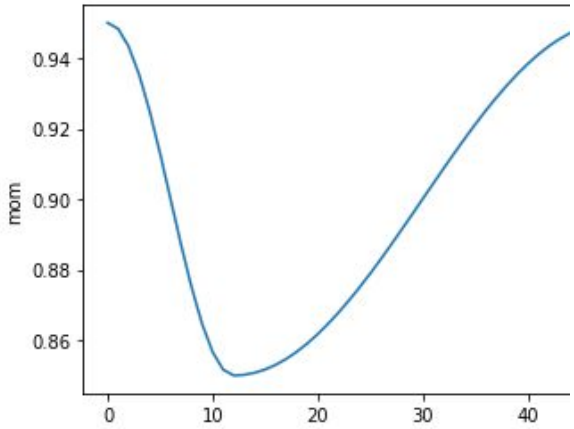

Fig 15: Corresponding momentum schedule

RESULTS

Task 1: Training and Predicting using a ResNet-50 on CIFAR-10 Dataset

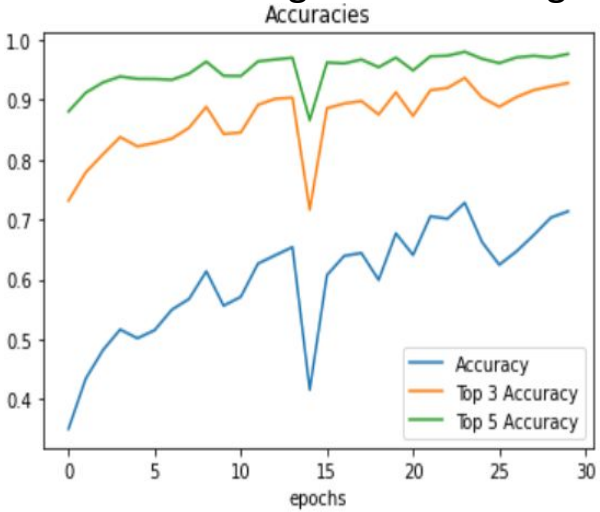

Fig 16: Accuracy vs # Epochs

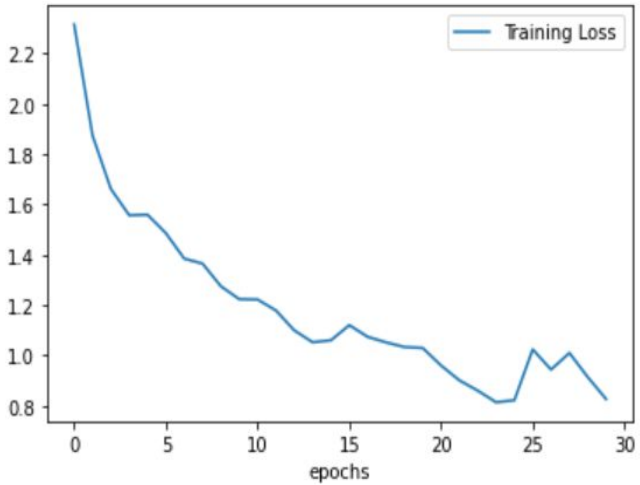

Fig 17: Training Loss vs # Epochs

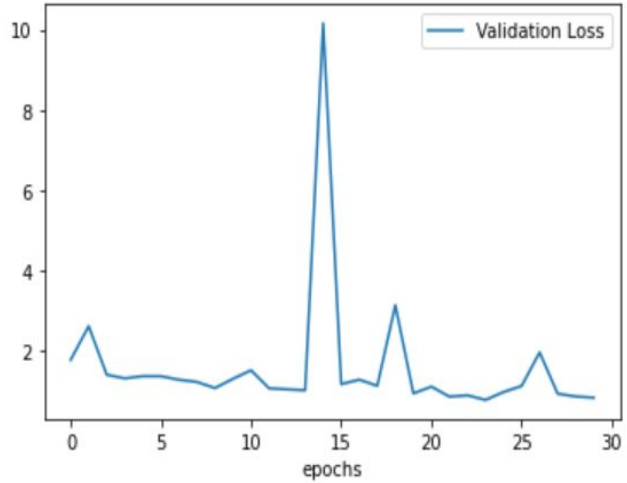

Fig 18: Validation Loss vs # Epochs

ANALYSIS

- The performance can be seen to grow steadily.
- There is a smooth upward trend in accuracy.
- The training loss is seen to decrease smoothly.
- The validation loss is however, more or less stagnant with respect to the range of the training loss.
- There are also certain spikes in the validation loss, which may be expected .

IMPLEMENTATION DETAILS: Genetic Algorithm

Continuous Genetic Algorithm has been used to optimize hyperparameters. We run the genetic algorithm to optimize three hyperparameters simultaneously

1. **Learning Rate** (lr)
2. **Weight Decay** (wd) (aka, the regularization parameter λ)
3. **Momentum** (mom)

It may be noted that the learning rate and momentum refer to the peak (max for lr, min for momentum) values in the cyclic hyperparameter schedules discussed earlier.

Genetic Algorithm explores these hyperparameters within the following limits

1. Learning Rate: (1e-6 → 1e-2)
2. Weight Decay: (0 → 0.1)
3. Momentum: (0.1 → 1.5)

These values have been chosen by a theoretical understanding of suitable hyperparameter values, beyond which model performances are likely to deteriorate.

IMPLEMENTATION DETAILS: Genetic Algorithm (PSEUDO CODE)

The following steps summarize the implementation of GA used for our purpose:

1. **Initialization:** Create an initial population of hyperparameters. Each population contains 10 individuals.
2. **Fitness Function:** Fitness is calculated for each of these individuals. In our case, the fitness is the final (top1) accuracy of the ResNet model trained using the selected hyperparameters.
3. **Selection**: We use the fittest half selection method, meaning the 5 fittest individuals are promoted to the next generation of population. The remaining are mutated.
4. **Mutation:** The remaining half of the population involves random reinitialization of some hyperparameters using a threshold based function.
5. **Pairing + Mating:** A part of population is crossed over at a single point (middle most node)
6. Thus at this point, the next generation is created. Calculate the fitness function on this population.

This cycle is repeated for 5 generations. Through this, we maximize the accuracy (fitness) of our model.

RESULTS

Task 2: Using Genetic Algorithm to optimize and tune the hyperparameters of the model

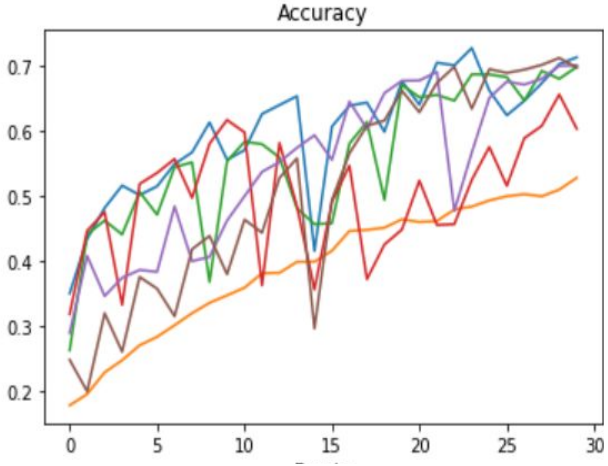

Fig 19: Accuracy vs # Epochs

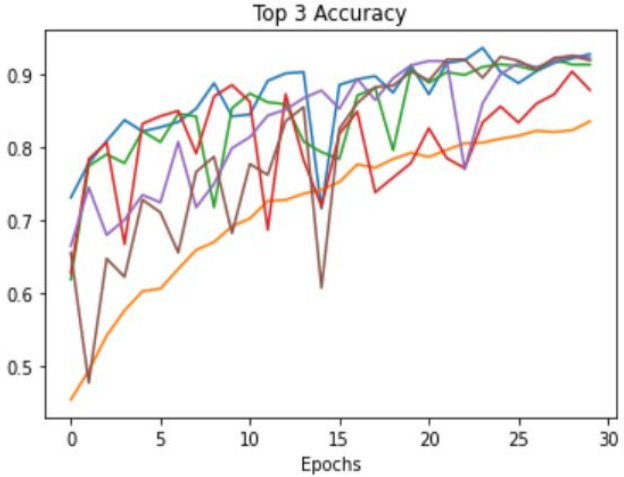

Fig 20: Top 3 Accuracy vs # Epochs

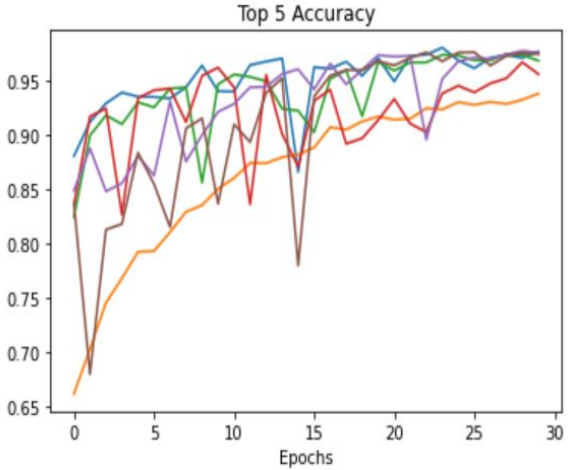

Fig 21: Top 5 Accuracy vs # Epochs

Note: Different colored graphs stand for different models which is mentioned in the next slide

RESULTS

Task 2: Using Genetic Algorithm to optimize and tune the hyperparameters of the model

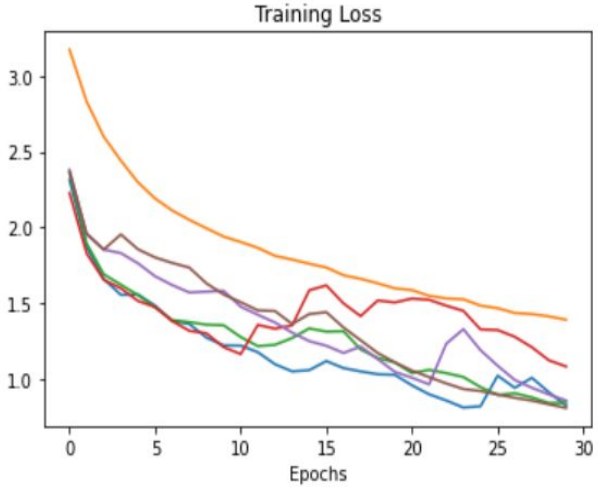

Fig 22: Training Loss vs # Epochs

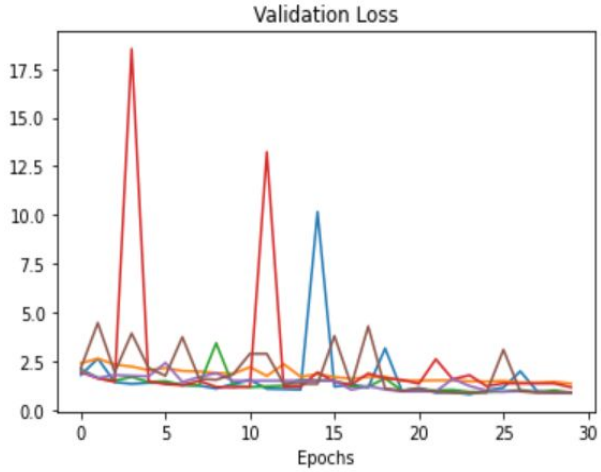

Fig 23: Validation Loss vs # Epochs

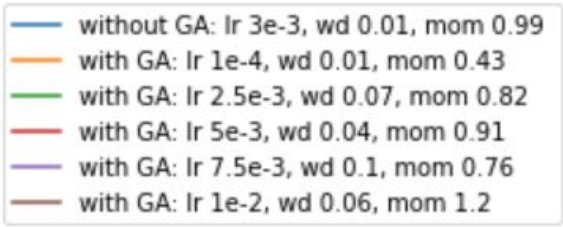

Fig 24: Labelings for all 5 graphs
Where, **wd**: weight decay
mom: Momentum
lr: Learning Rate

ANALYSIS

- The model accuracies with and without GAs are very close at the end. Model trained without GA outperforms the latter by a slight amount, but GA outperforms the model trained without GA slightly on the top3 accuracies.

- The training and validation losses for both cases are more or less same at the end of the training cycle. The loss saturates at the end of the training cycle (irrespective of the slight variations of the hyperparameters).

- A higher momentum usually leads to erratic spikes in the validation loss as well as the accuracies.

COMPARISON OF RESULTS WITH THE AUTHORS

method	top-1 err.	top-5 err.
VGG [41] (ILSVRC'14)	-	8.43†
GoogLeNet [44] (ILSVRC'14)	-	7.89
VGG [41] (v5)	24.4	7.1
PReLU-net [13]	21.59	5.71
BN-inception [16]	21.99	5.81
ResNet-34 B	21.84	5.71
ResNet-34 C	21.53	5.60
ResNet-50	20.74	5.25
ResNet-101	19.87	4.60
ResNet-152	**19.38**	**4.49**

Table 4. Error rates (%) of **single-model** results on the ImageNet validation set (except † reported on the test set).

Top 1 Accuracy:
Accuracy from the paper: **80.62%**
Accuracy of trained model: 74.68%

Top 5 Accuracy:
Accuracy from the paper: 95.51%
Accuracy of the trained model: **97.51%**

The trained model beats the paper's results on Top5 accuracy, and is comparably close on the Top1 accuracy.

Fig 24: Results of the Authors

CHALLENGES FACED IN IMPLEMENTATION

- GPU Memory issues: ResNet50 is a computationally expensive model. The problem particularly arises because we need to compare multiple models in the same environment, hence all models need to be evaluated in the same session. Solution: checkpoints introduced from time to time.

- GPU usage limits. Since each individual item in GA corresponds to one training cycle that may last anything from an hour to 3 hours, which is run multiple times corresponding to each generation, the total training time for the GA algorithm exceeds one day (24 hours), which is not usually provided by any freely available GPUS like COlab or Kaggle.

- Implementing ResNet in an object oriented programming manner was a challenge. We implemented it using our own experience, without copying from anywhere. Most online resources provide a crude, non-object oriented programming based implementation.

FUTURE SCOPE

Though GPUs are great tools to train Deep Learning models fast, our implementation depicts the drawbacks, and scopes of architectural advancements. A TPU implementation of the GA may help in accelerating the entire training procedure, and increase efficiency.

Since the results of training with and without GA are comparable, more advanced optimization techniques can be explored. These optimization techniques may optimize more complex model components such as layer design, hyperparameter schedules and even data augmentation transforms. This is not feasible with GA, since, as mentioned before, running a single generation of GA takes multiple hours, and is highly computationally expensive.

LEARNING OUTCOME

Below are listed a few learning outcomes we gained from working on this assignment:

- We learnt how to handle large datasets and integrating it with functional code for Deep Learning models using an object oriented programming approach.
- We were also able to obtain a comprehensive overview about the general working a Convolutional Neural Network, and in particular, combined with the the Residual Learning Framework.
- We also gained insights towards the problem of degradation of accuracy, and how to combat it.
- Lastly, we also learnt how to develop an understanding and derive insights from a research paper, and then subsequently work towards its implementation as a team.