Mini-Project 1: Residual Network Design

Madhu Vemmana (mv2283)
Mujahid Ali Quidwai(maq4265)
Sidharth Shyamsukha(ss14885)

Group Number - 62

Abstract

Neural nets are one of the best ways to classify images with high accuracy.On increasing the number of layers accuracy of the model increase up till a point and then starts diminishing due to feature reuse.Also increasing the number of layers leads to increasing the number of parameters which makes the model slower to train. To reduce the aforementioned problems we inside widen the neural network which also increases the accuracy and also applying data transformation and data augmentation techniques to contribute to the accuracy. We train our model using the CIFAR10 data set which has 50000 images for training.We also use hyper parameter optimization to help increase the accuracy while keeping the total number of parameters below the specified limit of 5M(4,977,226).

1 Introduction

ResNets (or Residual Networks) are one of the most commonly used models for image classification tasks.There are different types of Resent architectures with varying numbers of layers like Resnet-18, Resnet-52, Resnet-101, etc. As we increase the number of layers in our model we can handle more and more edge cases of image classification but also increase the complexity and training time of the model.A fraction increase in accuracy leads to doubling the number of layers which is counter productive for machines with slower hardware.Increasing the number of layers also causes various other problems like the vanishing gradient, degradation, etc. Changing the training techniques can help but cannot help in reducing the load on the machine.

A solution as proposed by Zagoruyko et al.[6], where they suggested widening the neural network by increasing the input layer size and reducing the number of total hidden layers. This implementation gives similar accuracy as a deep neural network without as many drawbacks.

A large data set cannot be trained as easily on a regular machine due to basic specification constraints and upgrading the machine can be costly so one solution is data augmentation which forms new and different examples to train data sets. We used various data transformation and augmentation techniques proposed by [5] like cropping the image, changing hue, saturation of image, adding some padding, cropping the image randomly, etc.Various fine tuning the hyper parameters like learning rate, decay will also help in reducing the loss and thereby increasing the overall accuracy of the model.

What data augmentation does is give artificial variation in the data and the main purpose of doing so is to train the model for different orientation,color profile of the images so it does not loose accuracy during the testing phase which might contain these sort of variations. Data augmentation also solves the issue of over fitting which is a major concern for neural networks.It is also very easy to implement and computationally cheap.

2 Methhodology

2.1 Data Augmentation

Data augmentation is the process of preprocessing the data before feeding it to the model. It increases the diversity of the data which allows the model to generalize well and results in higher accuracy. We have implemented the following kinds of transforms.

1. Normalization: We normalized the images by subtracting the mean and dividing by the standard deviation across each channel. The values of mean and standard deviation for CIFAR10 are available on the internet. As a result, the mean of the data across each channel is 0, and standard deviation is 1. Normalizing the data prevents the values from any one channel from disproportionately affecting the losses and gradients while training.
2. Random Crop: Crop the given image at a random location. We padded the given image by 4 pixels and then cropped them to a size of 32x32 to keep the dimensions intact. The reason for picking this transformation was: It generated the greatest task performance improvement was yielded by specific translations generated by the cropping method[5].

Figure 1: Padding with random crop

3. Random Horizontal Flip: Horizontally flip the given image randomly with a given probability. The probability we picked was 0.6. This method showed one of the greatest improvement because geometric augmentation schemes out performed photometric schemes.[5]

Figure 2: Random Horizontal Flip

4. Random Adjust Sharpness: Adjust the sharpness of the image randomly with a given probability. The sharpness factor determines if the image is blurred, normal or sharp. We've chosen the image to be blurred as out of the photometric schemes blurring the image produced the highest gain in accuracy.[5]

Figure 3: Random Adjust Sharpness

The following figure demonstrates the techniques mentioned above:

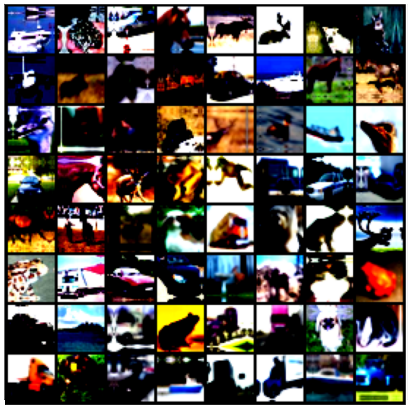

Figure 4: All the data Augmentation together

2.2 Optimizer

Gradient descent is one of the most popular algorithms to perform optimization. We tried out three optimizers and ended up picking the ADAM optimizer as the optimizer of choice. The optimizers and the hyperparemeters we considered were:

1. Stochastic Gradient Descent: Stochastic gradient descent (SGD) in performs a parameter update for each training example $x^{(i)}$ and label $y^{(i)}$: SGD does away with this redundancy by performing one update at a time. It is therefore usually much faster and can also be used to learn online[4]. The hyper parameters that we chose for stochastic. We chose a lower learning rate because lowering the learning rate will allow the network to train better, especially in case of fine tuning [2]. The most optimal learning rate(lr) and momentum that we found were:

$$lr = 0.001, momentum = 0.9$$

2. ADAM Optimizer:Adaptive Moment Estimation (Adam)[3] is another method that computes adaptive learning rates for each parameter. In addition to storing an exponentially decaying average of past squared gradients v_t Adam also keeps an exponentially decaying average of past gradients similar to momentum: the first moment (the mean) and the second moment (the uncentered variance) of the gradients respectively, hence the name of the method. The optimal values we used for learning rate(lr, $\beta_1, \beta_2, \epsilon$)were chosen to be. Experimenting with learning rate we found that a lower learning rate performs better.

$$lr = 0.001, \beta_1 = 0.9, \beta_2 = 0.9, \epsilon = 10^{-8}$$

2.3 Batch Size

Convolutional neural networks have shown superior accuracy in image classification, but to accurately train a CNN many hyperparameters need to be tune. Batch size is in an important parameter. There is a high correlation between the learning rate and the batch size, when the learning rates are high, the large batch size performs better than with small learning rates. We chose a small batch size with low learning rate.[2]

$$BatchSize = 64$$

2.4 Convolution Layers

The first instinct is to make the convolution layers thin and long. A thin network with multiple layers would provide a network that can be trainable for images. We found that this is not the case. A wide residual network architecture provides state-of-the-art results on several image data sets. A wide residual network of 16 layers can significantly outperform a lean 1000 layer network[6]. Widening of ResNet blocks provided a much more effective way of improving performance of residual networks compared to increasing their depth.

2.5 Regularization

We also used dropout in ResNet blocks: Since the residual networks are now wider with more parameters. Adding dropouts helped us improve the accuracy. Dropouts helped properly regularize the parameters in and outside the residual block thus preventing over fitting during training[6]. Regularization lower the complexity of a neural network model during training, and thus prevent the over fitting[1]. It penalizes the weights in case of drops out – it randomly drops the weights with the given probability thereby giving the other weights more weight age during training.

Dropout probability in Residual layers = 0.4
Dropout probability between layer 3 and layer 4 = 0.1
Dropout probability between layer 4 and Fully connected layer = 0.3

3 Results

3.1 Architecture

Parameters you can change :

N: # Residual Layers
B_i: # Residual blocks in Residual Layer *i*
C_i: # channels in Residual Layer *i*
F_i: Conv. kernel size in Residual Layer *i*
K_i: Skip connection kernel size in Residual Layer *i*
P: Average pool kernel size

Residual layers :

Layer 1 :
Layer 2 :
Layer 3 :
Layer 4 :

Residual blocks in Residual layer:

Layer 1 : 2
Layer 2 : 1
Layer 3 : 1
Layer 4 : 1

Channel in Residual layer:

Layer 1 : 64
Layer 2 : 128
Layer 3 : 256
Layer 4 : 512

Conv kernel size in residual layer

Layer 1 : 3x3
Layer 2 : 3x3
Layer 3 : 3x3
Layer 4 : 3x3

Skip kernel size in Residual layer I:

Layer 1 : 1x1
Layer 2 : 1x1
Layer 3 : 1x1
Layer 4 : 1x1

Average pool kernel size: 4x

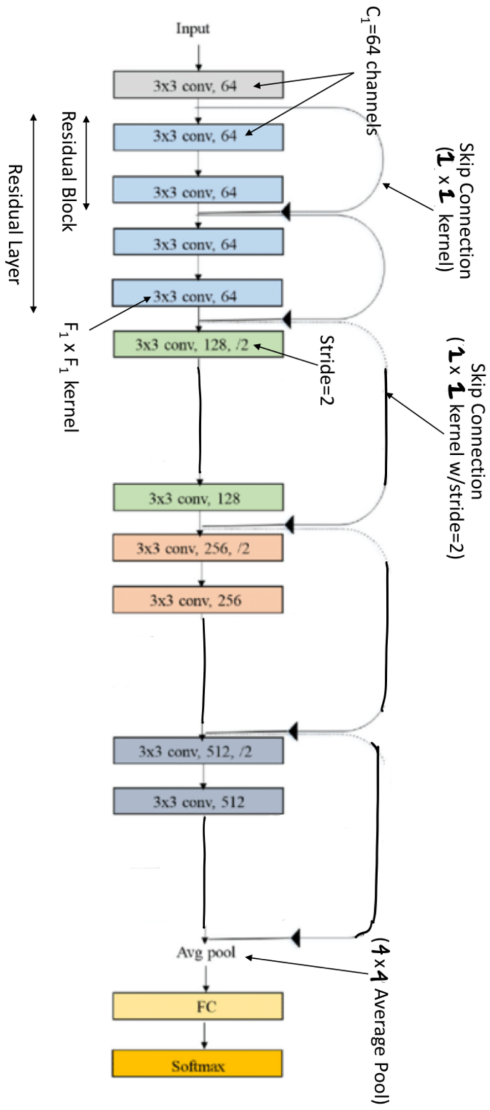

Figure 5: Model Architecture

3.2 Parameters

The accuracy of any particular model depends on the careful selection of the optimizer and the parameters that optimizer depends on, such as learning rate, momentum for Stochastic Gradient Descent and betas for Adam optimizer.

To find the optimum value of these hyperparameters, we vary these hyperparameters over a certain range and track the accuracy of the model on the validation dataset.

For instance, the following figure shows the performance of ResNet model with different learning rates for Adam optimizer. We can clearly see that model performs best at the learning rate of 0.0001.

```
==========================================================================
Layer (type:depth-idx)                             Param #
==========================================================================
├─Conv2d: 1-1                                      1,728
├─BatchNorm2d: 1-2                                 128
├─Sequential: 1-3                                  --
|    └─BasicBlock: 2-1                             --
|    |    └─Conv2d: 3-1                            36,864
|    |    └─BatchNorm2d: 3-2                       128
|    |    └─Conv2d: 3-3                            36,864
|    |    └─BatchNorm2d: 3-4                       128
|    |    └─Sequential: 3-5                        --
|    └─BasicBlock: 2-2                             --
|    |    └─Conv2d: 3-6                            36,864
|    |    └─BatchNorm2d: 3-7                       128
|    |    └─Conv2d: 3-8                            36,864
|    |    └─BatchNorm2d: 3-9                       128
|    |    └─Sequential: 3-10                       --
├─Sequential: 1-4                                  --
|    └─BasicBlock: 2-3                             --
|    |    └─Conv2d: 3-11                           73,728
|    |    └─BatchNorm2d: 3-12                      256
|    |    └─Conv2d: 3-13                           147,456
|    |    └─BatchNorm2d: 3-14                      256
|    |    └─Sequential: 3-15                       8,448
├─Sequential: 1-5                                  --
|    └─BasicBlock: 2-4                             --
|    |    └─Conv2d: 3-16                           294,912
|    |    └─BatchNorm2d: 3-17                      512
|    |    └─Conv2d: 3-18                           589,824
|    |    └─BatchNorm2d: 3-19                      512
|    |    └─Sequential: 3-20                       33,280
├─Sequential: 1-6                                  --
|    └─BasicBlock: 2-5                             --
|    |    └─Conv2d: 3-21                           1,179,648
|    |    └─BatchNorm2d: 3-22                      1,024
|    |    └─Conv2d: 3-23                           2,359,296
|    |    └─BatchNorm2d: 3-24                      1,024
|    |    └─Sequential: 3-25                       132,096
├─Linear: 1-7                                      5,130
==========================================================================
Total params: 4,977,226
Trainable params: 4,977,226
Non-trainable params: 0
==========================================================================
```

Figure 6: Count of Model Parameters

3.3 Training and Test loss vs Number of Epoch

3.4 Final test error achieved

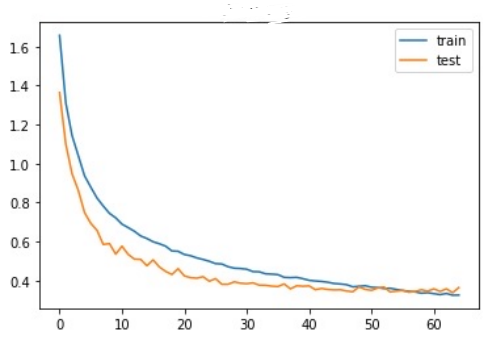

Figure 7: Error vs Epochs

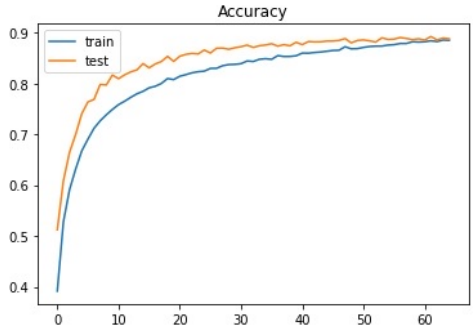

Figure 8: Accuracy vs Epochs

4 Conclusion

Our method has good generalization performance on CIFAR 10 dataset.The error rate in the model is $11\,\% \pm 2\,\%$ Testing the accuracy of our model for values from testing dataset.

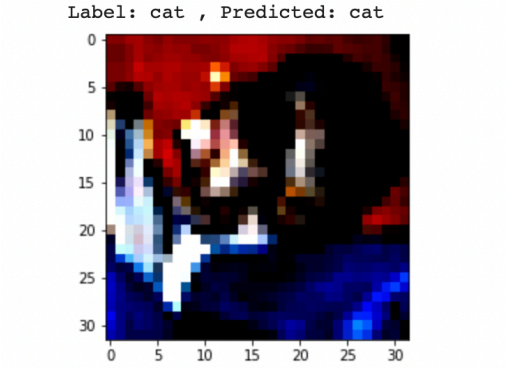

Figure 9: Predicted value - Cat , Actual value-Cat

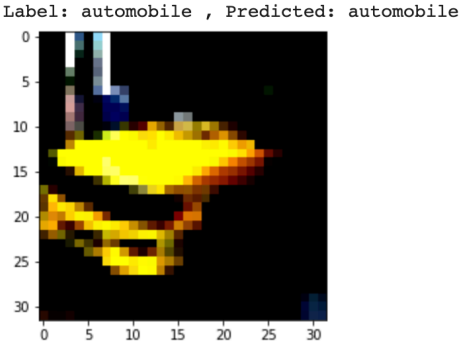

Figure 10: Predicted value - Automobile , Actual value-Automobile

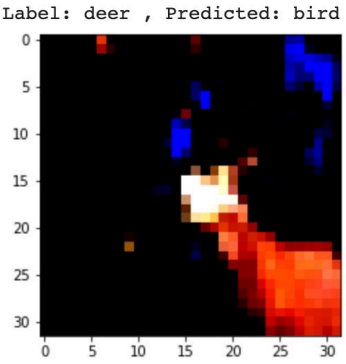

Figure 11: Predicted value - Deer , Actual value-Bird

5 Github Repo Link

5.1 `https://github.com/Ali-Maq/Deep_Learning`

References

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[2] Ibrahem Kandel and Mauro Castelli. The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset. *ICT Express*, 6(4):312–315, 2020.

[3] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[4] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[5] Luke Taylor and Geoff Nitschke. Improving deep learning with generic data augmentation. In *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1542–1547. IEEE, 2018.

[6] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.