Performance Comparison and Hyperparameters Optimization of ResNet-34 on Three Input Data Formats

Yuting Shao
Northeastern University
San Francisco, CA
shao.yut@northeastern.edu

***Abstract*—In this project, I used a ResNet34-based image classification model for three distinct datasets, each with images compressed using different techniques: JPEG, linear, and logarithmic. The goal was to optimize the model's hyperparameters, including batch size, learning rate, and momentum, to achieve the highest test accuracy and compare the model's performance on the three input data formats. I implemented early stopping and conducted experiments to find the best configuration for each dataset. The conclusion is that the best hyperparameters of ResNet34 are the same for the three input data formats, and the testing accuracy of JPEG or linear is slightly higher than that of logarithmic. For JPEG and linear, the largest stable learning rate is 0.001, while for logarithmic, the largest stable learning rate is 0.01. By comparing with VGG16, I found that VGG16 has slightly better testing accuracy than ResNet34, but it takes more training time. The performance of VGG16 on different input formats has a consistent conclusion with that of ResNet34. Finally, I investigated the trained model's robustness to variations in intensity, color balance, and orientation.**

***Index Terms*—ResNet-34, image recognition, hyperparameters optimization, VGG16**

I. Introduction

Image classification is a critical task in computer vision with wide-ranging applications, from object recognition to autonomous driving. Image classification involves the categorization of objects in images into predefined classes. In this project, I focus on optimizing the performance of an image classification model on three different datasets with images compressed using different techniques: JPEG, linear, and logarithmic.

The goal of our project is to determine the optimal combination of hyperparameters of a ResNet34-based image classifier to achieve the highest test accuracy across the three input data formats. I use early stopping and conduct experiments to determine the best batch size, learning rate, and momentum for each dataset.

Image compression is a widely used technique in image processing, which aims to reduce the size of an image while maintaining its visual quality. JPEG is a lossy compression technique that is commonly used for images on the web. Linear is an uncompressed image format commonly used in scientific applications, while logarithmic compression is used in high dynamic range (HDR) photography to capture a broader range of luminance values.

This study is motivated by the need to understand the performance differences of ResNet34 on images compressed using different methods. I seek to determine whether the same hyperparameters can be used across different input data formats or whether it is necessary to adjust the hyperparameters to achieve optimal performance.

In this report, we present our findings on the performance of ResNet34 on the three input data formats, the optimal hyperparameters for each format, and the differences in performance between the three formats. I compared the model performance of ResNet34 and VGG16. Afterwards, I investigated the trained model's robustness to variations in intensity, color balance, and orientation.

II. Related work

In Ref. [1], a residual learning framework is introduced to facilitate training of substantially deeper neural networks compared to previous architectures. The authors reformulate the layers as learning residual functions with reference to the layer inputs, making the networks easier to optimize and gain accuracy from increased depth. They evaluated residual nets with up to 152 layers on the ImageNet dataset and achieved significant performance improvements. Additionally, they analyzed the performance on CIFAR-10 with 100 and 1000 layers. Due to their deep representations, these residual nets provide a substantial relative improvement on the COCO object detection dataset. This paper is directly relevant to the project as it introduces the ResNet architecture, specifically ResNet34, which is used as the primary model in the project.

Ref. [2] addresses the challenge of setting optimal hyperparameters for deep learning models. This process often requires substantial experience and can lead to unnecessarily long training times. The author proposes efficient ways to set hyper-parameters, including observing the training validation/test loss function for signs of underfitting and overfitting and providing guidelines for finding the optimal balance point. The paper also discusses adjusting the learning rate and momentum to speed up training and emphasizes the importance of balancing different types of regularization for each dataset and architecture. Weight decay is used as an example to show how the optimal value is closely linked with learning rates and momentums. This paper is relevant to the project because it

provides guidelines and techniques for setting crucial hyperparameters, such as learning rate, batch size, and momentum, which are important aspects of the project's optimization process.

Ref. [3] discusses the challenge of determining when to stop training a neural network to avoid overfitting, a technique known as "early stopping". The author investigates the trade-off between training time and generalization performance and proposes a systematic approach for selecting a stopping criterion. The paper's empirical investigation on multi-layer perceptrons reveals that slower stopping criteria can lead to small improvements in generalization but significantly increase training time. This paper is relevant to the project as it provides insights into the early stopping technique, which is employed in the project to optimize the number of epochs for training the model.

Ref. [4] focuses on the impact of convolutional network depth on its accuracy in large-scale image recognition tasks. The main contribution of this work is the evaluation of networks with increasing depth, using an architecture that employs very small (3x3) convolution filters. The authors demonstrate that by increasing the depth to 16-19 weight layers, their networks achieve significant improvements over prior state-of-the-art configurations. This paper is related to my project as it explores the use of deeper convolutional networks, specifically the VGG16 model, for image recognition tasks. By understanding the design and performance of VGG16, I can compare its performance with other models like ResNet34 in my experiments.

III. Methods

A. Dataset

I utilized a dataset that was collected by all students in the CS5330 course, which was then processed by the course instructors. The dataset includes three different image formats: jpg, tiff, and exr. I selected a small dataset with a minimum side length of 64 pixels. When reading in the images, I cropped the central area of each image with a length of 63 pixels. For the jpg and tiff images, I normalized the input to the range of [0, 1], while for the exr images, I kept the original values.

B. Model

I use a pretrained ResNet34 model as the image classifier whose architecture is shown in Fig. 1 and then optimize its hyperparameters for each dataset. I implement early stopping to find the optimal number of epochs for training. I then perform a series of experiments to find the best batch size, learning rate, and momentum for each dataset. Once the best hyperparameters are determined, I update the saved model with the best configuration. I use a pretrained VGG16 model to compare its performance with the ResNet34.

C. Early stopping

The best testing accuracy achieved during training is recorded as the highest testing accuracy of each experiment. The number of epochs is set to 50. If the testing accuracy increase is less than 0.005 for three consecutive epochs after the 10th epoch, early stopping will be triggered.

D. Variations to the image when investigate the robustness

I introduced three types of variations to the input images, which are intensity variation, color balance variation, and orientation variation. First, I applied an intensity variation by copying the original image and adding an intensity value of 50 to all the pixels, effectively brightening the image. Next, I adjusted the color balance of the image by independently modifying the intensity of the blue, green, and red channels. I increased the blue channel by 10, the green channel by 20, and the red channel by 30. This resulted in a color-shifted image with altered color balance. Lastly, I applied an orientation variation by rotating the image by 30 degrees around its center. This was achieved using OpenCV's getRotationMatrix2D and warpAffine functions. The resulting image has been transformed with variations in intensity, color balance, and orientation, allowing us to test the robustness of the trained ResNet34 model and VGG16 model against these variations.

IV. Experiments and results

A. ResNet34 performance on three input formats

The performance of the model with different hyperparameters on three input data formats is shown in Fig. 2. First, the model was trained on three different input formats: standard images in JPG format, linear images in TIFF format, and logarithm of linear images in EXR format. The batch size was set to 16, the learning rate was set to 0.001, and the momentum was set to 0.9. The training loss for each input format is plotted in Fig. 2(a). The training time was 2360 seconds. For the JPEG format DB (black line in Fig. 2(a)), the initial training loss after 1 epoch was 0.77596 and the early stopping was triggered after 11 epochs with a final training loss of 0.02441. For the linear format DB (blue line in Fig. 2(a)), the initial training loss after 1 epoch was 0.76653 and the early stopping was triggered after 16 epochs with a final training loss of 0.01626. The training time was 3626 seconds. For the logarithmic format DB (red line in Fig. 2), the initial training loss after 1 epoch was 0.86558 and the early stopping was triggered after 11 epochs with a final training loss of 0.031195. The training time was 2541 seconds. The corresponding validation accuracy is plotted in Fig. 2(b), and the corresponding testing accuracy is plotted in Fig. 2(c). From the testing accuracy in Fig. 2(c), we can see that the linear format DB has better performance, and both the testing accuracy of JPEG DB and linear DB are slightly higher than that of the logarithmic DB.

B. Optimization of batch size for ResNet34

Next, I attempted to optimize the hyperparameters for each input format database. Using the trained model from the previous experiment, I adjusted the hyperparameters and continued training to observe the effect on the best testing accuracy. The first hyperparameter I experimented with was the batch size. I ran the experiment on batch sizes of 16, 32, 64, and 128 for all three input format databases. The trends

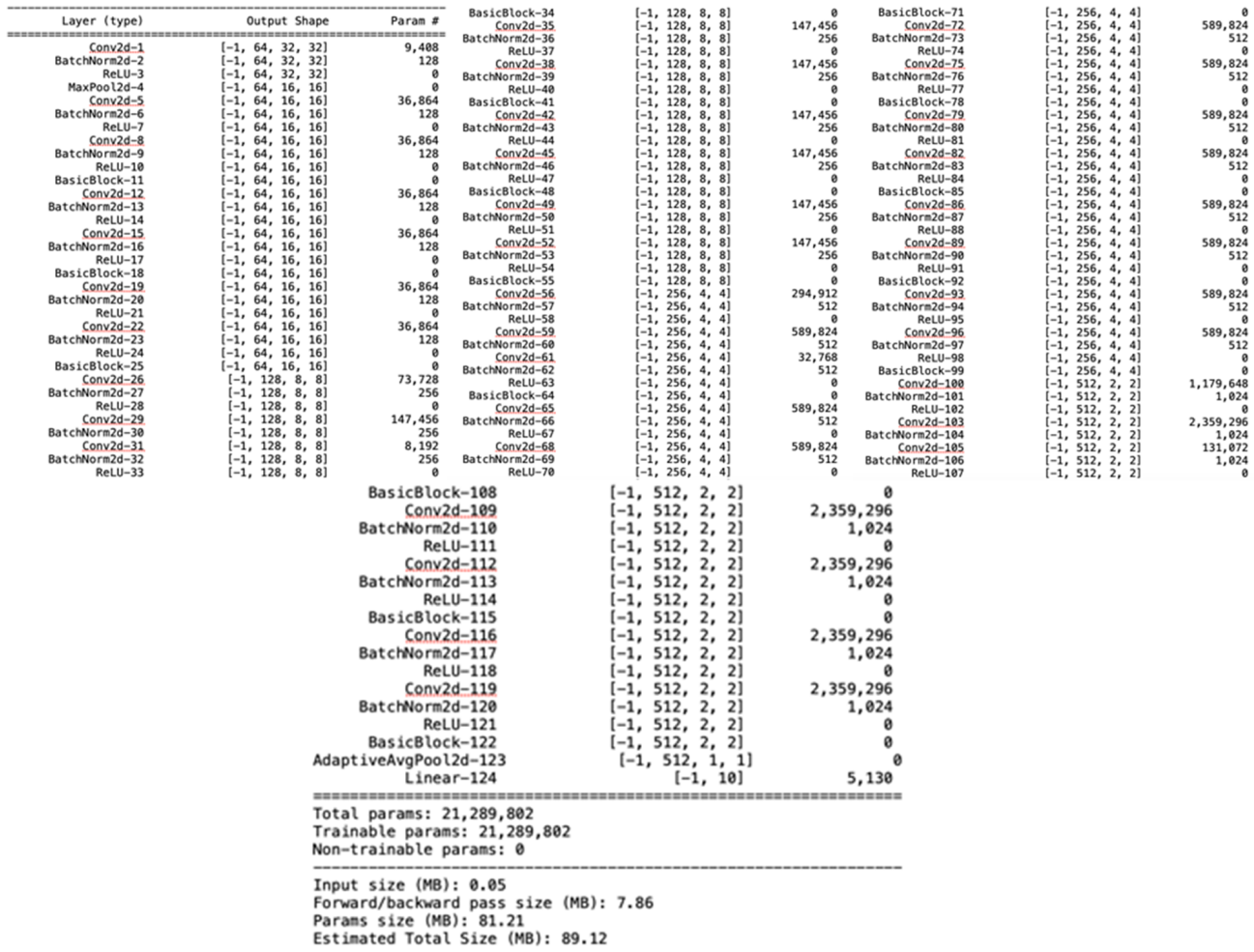

Fig. 1. The architecture of the ResNet34 model used in the project.

for the highest testing accuracy during training for all three input databases were the same (see Fig. 2(d)). The testing accuracy increased as the batch size changed from 16 to 32, but decreased as the batch size increased further to 64 and 128. The optimized batch size for all three input types was 32. The linear type database had the best performance, with a highest testing accuracy of 0.8685 at a batch size of 32, while the highest testing accuracy for JPEG was 0.8655, and for logarithmic, it was 0.864.

C. Optimization of learning rate for ResNet34

The second hyperparameter I experimented with is the learning rate (Fig. 2(e)). In this experiment, I ran the model on three input formats DB with learning rates of [1E-4, 1E-3, 1E-2, 1E-1]. The best testing accuracies were comparable when the learning rate was less than or equal to 1E-2. However, the testing accuracy decreased when the learning rate was 1E-1. Additionally, although there was good testing accuracy for JPEG and linear DB with a learning rate of 1E-2, the accuracy significantly decreased with the epochs, suggesting that 1E-2 is not a suitable learning rate for these formats. The initial good testing accuracy might be due to the use of a pretrained model with a learning rate of 1E-3 in these experiments. Therefore, for JPEG and linear DB, the suggested learning rate for training is 1E-3, while for logarithmic DB, the suggested learning rate can be 1E-2. However, using 1E-3 as the learning rate in logarithmic DB can increase the testing accuracy to 0.8625, while it is 0.862 when the learning rate is 1E-2.

D. Optimization of momentum for ResNet34

The third hyperparameter I experimented with is momentum (Fig. 2(f)). For this experiment, I ran the model on the three input formats DB, with momentum values of [0.5, 0.75, 0.9, 0.99]. There was not a significant difference in performance while changing the momentum from 0.5 to 0.99. Specifically, for the logarithmic DB, the best test accuracy increased from 0.862 to 0.8645 while using 0.99 as the momentum. For the linear DB, the best testing accuracy remained at 0.869 for all momentums. For the JPEG DB, the best testing accuracy was slightly higher while using 0.9 or 0.99 as the momentum, compared to 0.5 and 0.75. The best testing accuracy was slightly higher for the JPEG and linear DBs than for the logarithmic DB.

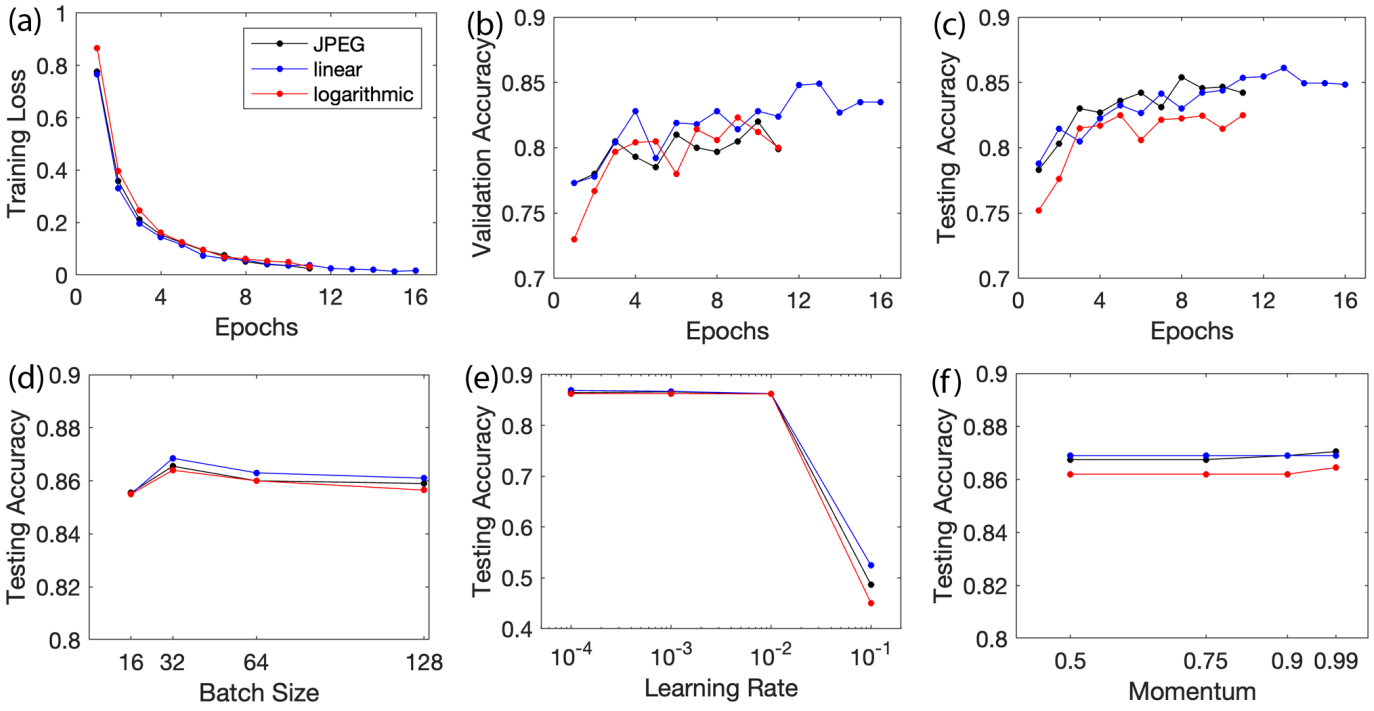

Fig. 2. The performance with different hyperparameters on three input data formats. The batch size is 16, learning rate is 1E-3 and momentum is 0.9 in (a), (b), and (c).

E. Comparison of ResNet34 performance on three input formats

The best testing accuracy for each database is shown in Table I. The optimized hyperparameters are consistent for the three input formats: a batch size of 32, a learning rate of 1E-3, and a momentum of 0.99. The testing accuracy on the JPEG database (0.8705) and the linear database (0.869) are slightly higher than that on the logarithmic database (0.8645).

TABLE I
BEST CONFIGURATION OF RESNET34 FOR EACH INPUT TYPE.

Input type	Batch Size	Learning Rate	Momentum	Testing Accuracy
JPEG	32	0.001	0.99	0.8705
linear	32	0.001	0.99	0.869
logarithmic	32	0.001	0.99	0.8645

F. Comparison of ResNet34 and VGG16 on JPEG DB

I have chosen to use the VGG16 model and compare its performance with that of ResNet34 on different input formats and databases. The architecture of the VGG16 model is shown in Fig. 3. Compared with the ResNet34 architecture (Fig. 1), VGG16 has fewer layers but more parameters. The performance comparison between VGG16 and ResNet34 on three input formats and databases is shown in Fig. 4. For JPEG and linear databases, the testing accuracy after one epoch using VGG16 is higher than that of ResNet34, and the model performance of VGG16 is slightly better after several epochs. For logarithmic databases, the testing accuracy after one epoch using VGG16 is lower than that of ResNet34, but the model performance of VGG is slightly better than that of ResNet34 after several epochs. When comparing the performance of the VGG16 model on three different input formats and databases (Fig. 4(d)), we can see that the linear database has the best performance and the JPEG database comes in second, which is consistent with the results of the ResNet34 model (Fig. 2(c)). The comparison of training time per epoch for ResNet34 and VGG16 is shown in Table II. VGG16 requires more training time per epoch than ResNet34.

TABLE II
TRAINING TIME PER EPOCH FOR RESNET34 AND VGG16. UNIT IS SECONDS.

Input type	ResNet34	VGG16
JPEG	215	330
linear	201	331
logarithmic	231	339

G. Robustness of ResNet34 and VGG16 to variations in intensity, color balance, and orientation

Table III shows the original testing accuracy and testing accuracy after intensity, color balance, or orientation variations of the ResNet34 and VGG16 models. The details of the variations are described in the Methods D section. The VGG16 model is more robust to variations than the ResNet34 model.

The logarithmic format shows poor robustness to variations in intensity and color balance due to the nature of the

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 64, 63, 63]           1,792
              ReLU-2         [-1, 64, 63, 63]               0
            Conv2d-3         [-1, 64, 63, 63]          36,928
              ReLU-4         [-1, 64, 63, 63]               0
         MaxPool2d-5         [-1, 64, 31, 31]               0
            Conv2d-6        [-1, 128, 31, 31]          73,856
              ReLU-7        [-1, 128, 31, 31]               0
            Conv2d-8        [-1, 128, 31, 31]         147,584
              ReLU-9        [-1, 128, 31, 31]               0
        MaxPool2d-10        [-1, 128, 15, 15]               0
           Conv2d-11        [-1, 256, 15, 15]         295,168
             ReLU-12        [-1, 256, 15, 15]               0
           Conv2d-13        [-1, 256, 15, 15]         590,080
             ReLU-14        [-1, 256, 15, 15]               0
           Conv2d-15        [-1, 256, 15, 15]         590,080
             ReLU-16        [-1, 256, 15, 15]               0
        MaxPool2d-17          [-1, 256, 7, 7]               0
           Conv2d-18          [-1, 512, 7, 7]       1,180,160
             ReLU-19          [-1, 512, 7, 7]               0
           Conv2d-20          [-1, 512, 7, 7]       2,359,808
             ReLU-21          [-1, 512, 7, 7]               0
           Conv2d-22          [-1, 512, 7, 7]       2,359,808
             ReLU-23          [-1, 512, 7, 7]               0
        MaxPool2d-24          [-1, 512, 3, 3]               0
           Conv2d-25          [-1, 512, 3, 3]       2,359,808
             ReLU-26          [-1, 512, 3, 3]               0
           Conv2d-27          [-1, 512, 3, 3]       2,359,808
             ReLU-28          [-1, 512, 3, 3]               0
           Conv2d-29          [-1, 512, 3, 3]       2,359,808
             ReLU-30          [-1, 512, 3, 3]               0
        MaxPool2d-31          [-1, 512, 1, 1]               0
AdaptiveAvgPool2d-32          [-1, 512, 7, 7]               0
           Linear-33             [-1, 4096]     102,764,544
             ReLU-34             [-1, 4096]               0
          Dropout-35             [-1, 4096]               0
           Linear-36             [-1, 4096]      16,781,312
             ReLU-37             [-1, 4096]               0
          Dropout-38             [-1, 4096]               0
           Linear-39               [-1, 10]          40,970
================================================================
Total params: 134,301,514
Trainable params: 134,301,514
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.05
Forward/backward pass size (MB): 16.70
Params size (MB): 512.32
Estimated Total Size (MB): 529.07
----------------------------------------------------------------
```

Fig. 3. The architecture of the VGG16 model used in the project.

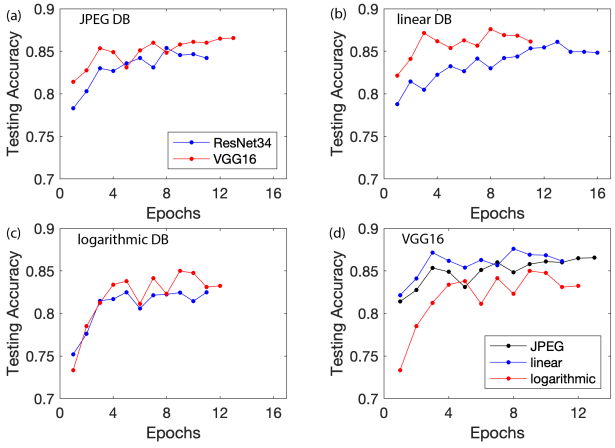

Fig. 4. The performance comparison between the VGG16 and ResNet34 on three input formats DB. (a) Comparison on JPEG. (b) Comparison on linear. (c) Comparison on log. (d) Performance of VGG16 on three DB.

logarithmic scale. In logarithmic scales, changes in intensity are represented as a relative ratio rather than an absolute difference. This means that a small change in intensity can lead to a significant change in the logarithmic value, which can have a substantial impact on performance.

Both the ResNet34 and VGG16 models are more robust to variations in intensity or color balance than variations in orientation on JPEG and linear DB. They are more robust to variations in orientation than variations in intensity or color balance on logarithmic DB. They are slightly more robust on the linear DB than on the JPEG DB for variations in intensity or color balance.

TABLE III
THE ORIGINAL TESTING ACCURACY AND TESTING ACCURACY AFTER VARIATIONS OF THE RESNET34 AND VGG16.

Input type and Variations	**ResNet34 Original**	**ResNet34 after variation**	**VGG16 Original**	**VGG16 after variation**
JPEG and Intensity	0.8705	0.816	0.8655	0.845
linear and Intensity	0.869	0.8685	0.8615	0.861
logarithmic and Intensity	0.862	0.1	0.8325	0.134
JPEG and Color Balance	0.8705	0.8595	0.8655	0.8625
linear and Color Balance	0.869	0.869	0.8615	0.8615
logarithmic and Color Balance	0.862	0.1115	0.8325	0.2165
JPEG and Orientation	0.8705	0.7535	0.8655	0.8165
linear and Orientation	0.869	0.7555	0.8615	0.8005
logarithmic and Orientation	0.862	0.6585	0.8325	0.747
JPEG and All	0.8705	0.6215	0.8655	0.791
linear and All	0.869	0.755	0.8615	0.8005
logarithmic and All	0.862	0.102	0.8325	0.1215

V. DISCUSSION AND SUMMARY

In this project, I focused on optimizing the performance of a ResNet34-based image classification model on three different datasets, each with images compressed using different techniques: JPEG, linear, and logarithmic. I optimized the model's hyperparameters, including batch size, learning rate, and momentum, to achieve the highest test accuracy and compared the model's performance on the three input data formats. I implemented early stopping and conducted experiments to find the best configuration for each dataset.

The experiments show that the same hyperparameters can be used across different input data formats, and the testing accuracy of JPEG or linear is slightly higher than that of logarithmic. For JPEG and linear, the largest stable learning rate is 0.001, while for logarithmic, the largest stable learning rate is 0.01. The optimized hyperparameters for all three input data formats are a batch size of 32, a learning rate of 1E-3, and a momentum of 0.99. The testing accuracy on the JPEG database (0.8705) and the linear database (0.869) is slightly higher than that on the logarithmic database (0.8645).

The findings suggest that the ResNet34 model with the optimized hyperparameters can achieve good performance on image classification tasks across different input data formats. By using early stopping and optimizing the hyperparameters, I can achieve a high testing accuracy with a relatively small

number of epochs. The hyperparameters I found can serve as a starting point for future work on similar image classification tasks.

After comparing VGG16 and ResNet34, it was found that VGG16 has slightly better testing accuracy but takes more training time. The performance of both models on different input formats is consistent. I also investigated the models' robustness to variations in intensity, color balance, and orientation. The results indicate that the models are more robust to variations in intensity or color balance than variations in orientation on JPEG and linear DB. They are more robust to variations in orientation than variations in intensity or color balance on logarithmic DB (logarithmic DB shows very poor robustness to variations in intensity and color balance). They are slightly more robust on the linear DB than on the JPEG DB for variations in intensity or color balance. These findings suggest that the models' robustness depends on the data format and the type of variation. Therefore, it is essential to consider the data format and the type of variation when evaluating the models' robustness.

In future work, I plan to investigate other deep learning models and architectures to see if they can achieve good performance on these datasets. Additionally, I will explore ways to further improve the models' performance and robustness.

This project provides insights into the performance of ResNet34 on images compressed using different techniques and experiments with hyperparameter optimization to achieve good performance on image classification tasks. It also compares the performance of VGG16 and ResNet34 and investigates the robustness of the trained model to intensity, color balance, and orientation.

REFERENCES

[1] He, K., Zhang, X., Ren, S., Sun, J. (2016). Deep Residual Learning for Image Recognition. CVPR 2016.

[2] Smith, L.N. (2018). A disciplined approach to neural network hyper-parameters: Part 1 - learning rate, batch size, momentum, and weight decay. ArXiv, abs/1803.09820.

[3] Prechelt, L. (1998). Early Stopping - But When? Neural Networks: Tricks of the Trade. Springer.

[4] Simonyan, K. and Zisserman, A. (2015). Very Deep Convolutional Networks for Large-Scale Image Recognition. ArXiv, abs/1409.1556.