COMP551 Mini-Project2 Report

Leila Wang, Raphael Wang, Oliver Wang

Abstract

This project aims to compare the performance of various types of Multilayer Perceptron models using CIFAR-10. In our experiments, we implemented different MLPs and explored the change in performance caused by changes in the number of layers, regularization, and activation functions. We also built CNN models with 2 fully connected layers and part of pre-trained models and tested their performance. We conclude that The performance of MLP models tends to increase as the number of layers increases, the ReLU activation function is more suitable than the others, and normalized data can raise the accuracy. Furthermore. The first CNN model has a 0.6562 accuracy but struggled with overfitting. Among others that were modified from pre-trained models, VGG16 has the highest accuracy which was also inhibited by overfitting, however.

1 Introduction

The CIFAR-10 is a well-known dataset for testing the image classification ability of MLP and CNN models.[1] The latest research shows that the best model to classify the CIFAR10 dataset is ViT-H/14 with 99.5 accuracy which is even higher than human recognition [3]. However, to test the fundamental properties of MLP and CNN models, we constructed an MLP model from scratch that can add layers and choose an activation function and regularization and a CNN model using pytorch. In the experiments, we first evaluated how the number of hidden layers affects the accuracy by comparing 0, 1, and 2 hidden-layer MLP models with the same activation function. Then, we explore the effect of changing activation functions on the accuracy, including ReLU, leaky ReLU, and tanh. Also, we tested the effect of L1 and L2 regularization and unnormalized data. Moreover, we compared the performance of a 2-layer CNN with 2 fully connected layers to the MLP models and modified pre-trained models by adding fully connected layers and observed its performance. The result showed that the accuracy would increase if there are more layers in the MLP model; MLP models with ReLU outperform others with tanh and leaky ReLU; MLP with L1 and L2 regularization may not be better than the original model due to improper hyperparameters; models with normalized input demonstrates better performance than those with unnormalized input. Our 2-layer CNN model has a 0.6562 accuracy but tends to overfit the training data. Compared to it, other CNN models constructed using pre-trained models have an overall lower accuracy, suffering from overfitting as well.

2 Datasets

CIFAR-10 has 60,000 32x32 color images with pixel values [0, 255] and is divided into 10 classes with 6,000 images per class, each of the images belongs to one of the following 10 classes: [0 airplanes, 1 cars, 2 birds, 3 cats, 4 deer, 5 dogs, 6 frogs, 7 horses, 8 ships, and 9 trucks]. In order to fit the input images to the fully-connected MLP model, all images are reshaped into 1-dimensional vectors of 3072 (32×32) pixels. The pixel values of the training and test set are normalized to [0, 1] for data preprocessing. Figure 1 illustrates some examples of the CIFAR-10 images.

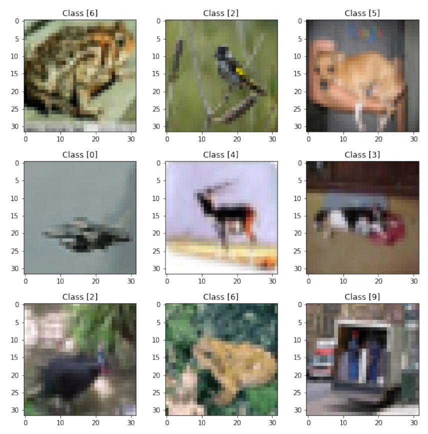


Figure 1: Some examples of the CIFAR-10 images with labels.

There are 10,000 test images and 50,000 training images. The Canadian Institute for Advanced Research (CIFAR) developed CIFAR-10, which has been extensively utilized in the computer vision community to assess the effectiveness of various image classification algorithms.

The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class as it is demonstrated in Figure 2. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class.[1]

3 Results

For all the MLP models below, the hyper-parameters (learning rate, λ for L1 and L2 regularization) were

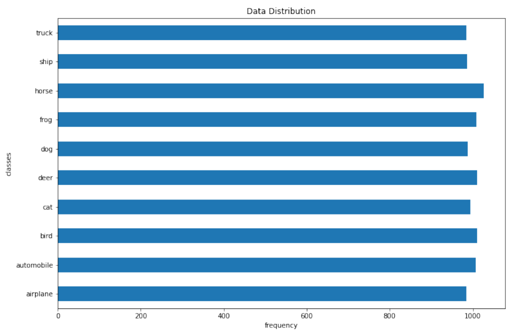


Figure 2: The test data distribution over the 10 labels.

chosen after tuning. Hyper-parameter tuning was conducted using `optuna` package. We set 100 trials of different values of hyper-parameters (e.g., learning rate) for every model. The trial that achieved the best accuracy after 5 iterations were chosen. Figure 3 illustrates an example of the result of hyper-parameter tuning the 2-hidden-layer-model with L_1 regularization. It turns out that trial 13 with learning rate $= 0.0739$ and $L_1\lambda = 1.4458e-06$ reached the highest accuracy of 0.1831 after 5 iterations.

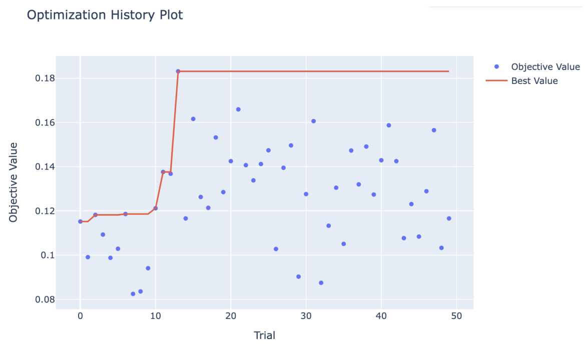


Figure 3: Example of Hyper-Parameter Tuning: L1 Regularization Model

3.1 Task3.1

To assess the impact of the number of layers on our data, we built 3 models each using the ReLU activation function and 256 units for all layers. After hyper-parameter tuning, the 0-hidden-layer-model with the learning rate of 0.0586 achieved the best accuracy of 0.1437 after 5 iterations; the 1-hidden-layer-model with the learning rate of 0.0369 achieved the best accuracy of 0.1747 after 5 iterations; the 2-hidden-layer-model with the learning rate of 0.0792 achieved the best accuracy of 0.1803 after 5 iterations.

We observed an increased accuracy as the number of hidden layers increases. Over 1000 iterations, the 0-hidden-layer-model achieved a test accuracy of 0.3578; the 1-hidden-layer-model achieved a test accuracy of 0.4101 and the 2-hidden-layer-model achieved a test accuracy of 0.4188. Figure 4 shows the testing accuracy over 1000 iterations of the 3 models, it clearly shows that the test accuracy of the model with 2 hidden layers is slightly greater than the model with 1 hidden layer, and both of them significantly outperformed the model with 0 hidden layer.

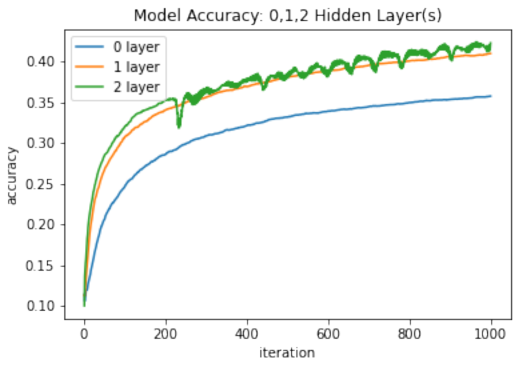


Figure 4: Model performance comparison for 0, 1, 2 hidden layer(s): Accuracy vs. Iterations

Nonlinearity is introduced in the model with 2 and 3 layers via ReLU. This use of nonlinear transformations of the predictors increases the model's flexibility and allows it to capture more complex relationships between the predictors and the response, thereby, increasing the accuracy. Depth also increases accuracy, this agrees with what we learned from class, as the number of layers in a network increases, the network can learn increasingly abstract and hierarchical representations of the data.

3.2 Task3.2

To experiment with the impact of different activation functions on our data classification, we created an MLP model with 2 hidden layers each having 256 units with ReLU activations as above, and two different copies of it in which the activations are now tanh and Leaky-ReLU. After hyper-parameter tuning, the tanh model with the learning rate of 0.0819 achieved the best accuracy of 0.1561 after 5 iterations; the Leaky-ReLU model with the learning rate of 0.0825 achieved the best accuracy of 0.1736 after 5 iterations.

We observe a reduced accuracy for the two hidden layers with the `tanh` activation function (0.3797) and Leaky-ReLU activation function (0.4109). Figure 5 illustrates the test accuracy over 1000 iterations of the three models. It is clear that the ReLU model outperforms the Leaky-ReLU model, and the Leaky-ReLU model outperforms the tanh model.

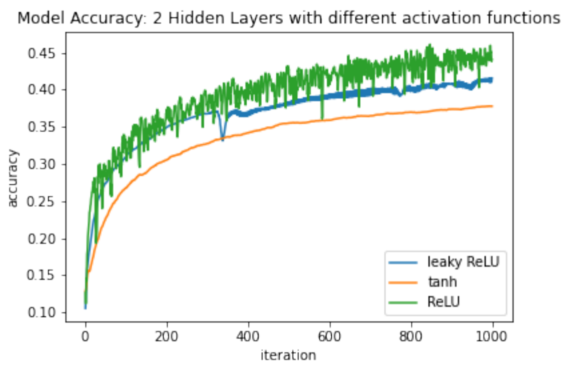


Figure 5: Model performance comparison for 2 hidden layers with ReLU, Leaky-ReLU and Tanh: Accuracy vs. Iterations

ReLU is a more robust activation function that is best suited for the CIFAR-10 dataset, as evidenced by the result of our experiment. ReLU's ability to generate sparse representations in the network by returning zero values for negative inputs may be one factor. Forcing the network to concentrate on the most crucial input features, can assist decrease overfitting and increase generalization. Moreover, unlike tanh, ReLU does not saturate in the positive zone. ReLU may therefore create larger outputs and is more sensitive to input changes, which helps speed up network learning and prevent the vanishing gradient problem. The Leaky-ReLU activation function, which is similar to ReLU thus obtained a similar accuracy as ReLU, but it allows a small slope for negative values. This can help to avoid the problem, where the gradient of the ReLU function becomes zero for negative inputs.

3.3 Task3.3

We also experimented with the effect of L1 and L2 regularization on network performance. To test this, we created an MLP model with 2 hidden layers each having 256 units with ReLU activations as above, and independently added L1 and L2 regularization to the network. After hyper-parameter tuning, the L1 regularization model with the learning rate of 0.0740 and $L_1\lambda = 1.4458e-06$ achieved the best accuracy of 0.1831 after 5 iterations; the L2 regularization model with the learning rate of 0.0662 and $L_2\lambda = 0.006363$ achieved the best accuracy of 0.1783 after 5 iterations.

We observed that the accuracy for the model with L1 regularization is 0.3768 and the accuracy for the model with L2 regularization is 0.4083 over 1000 iterations. Figure 6 illustrates the test accuracy over 1000 iterations of the three models. Comparing them to the original MLP model, we found out that adding regularization did not improve the performance. This is largely due to an ineffective choice of hyperparameters.

According to the data below, L2 regularization outperforms L1 regularisation fitting our dataset. One explanation could be that L2 promotes the model to spread the weight values across all of the characteristics more evenly. As a result, the decision boundary becomes smoother and more generalized, which can assist in lowering overfitting. A further explanation could be that L2 regularization is easier to optimize during backpropagation than L1 regularization because the gradient of the L2 penalty term is continuous and differentiable, whereas the gradient of the L1 penalty term is not continuous and has a non-differentiable point at zero.

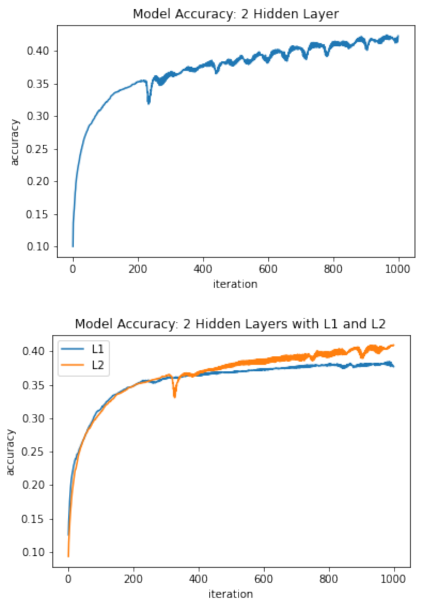


Figure 6: Model performance comparison of 2 hidden layer models with no regularization, L1 regularization, and L2 regularization.

3.4 Task3.4

We created an MLP model with 2 hidden layers each having 256 units with ReLU activations as above. However, this time, we trained it with unnormalized images. We set the learning rate equal to 0.02 and trained the unnormalized data for 200 iterations. Figure 7 shows the model performance for the model. We observe there is a much less smooth convergence as the accuracy over iteration becomes very unstable. The accuracy only converges to somewhere around 0.2250 over 200 iterations, which is much lower than the performance result with normalized data in section `Task3.1`. Note that we also observed that the test accuracy does not noticeably improve after training the model for more iterations as it always stays between 0.2 to 0.3.

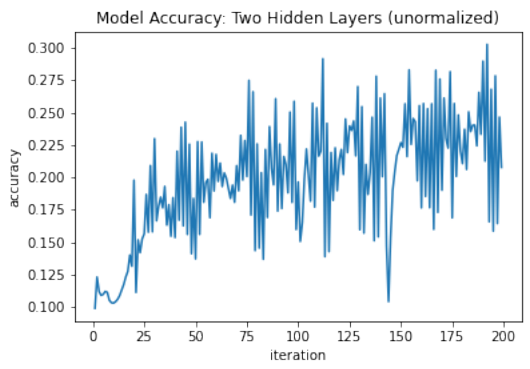


Figure 7: Model performance for the unnormalized dataset: Accuracy vs. Iterations

This is expected as the inputs have different scales or ranges so the optimization algorithm may struggle to find the optimal weights and biases that lead to accurate predictions. Also with normalization, there could exist a covariate shift because the distribution of the input data varied between the training and test sets. Furthermore, without rescaling the input values to a more reasonable range, the outlier will have significantly more impact on the network's performance.

3.5 Task3.5

The next step is to build a convolutional neural network (CNN) with two convolutional layers, and two fully-connected layers using keras. The convolutional layers are with filter size equal to 64 and kernel size equal to (4×4). To create fully linked layers, we settle on 256 units. ReLU is applied to all layers. Figure 8 demonstrates the model accuracy and loss for training and test data over 11 epochs. When compared to the MLP models, the test accuracy after 11 epochs is roughly 0.6562 (shown in Figure 9), which is a much higher accuracy. From the result below, we can see that epoch 11 yielded the highest accuracy and the lowest cost, large epoch led to a degradation problem. (One of the possible reasons could be overfitting. The model tends to overfit with the increase in depth.)

This shows that CNNs are effective for image classification tasks. This is because they are designed

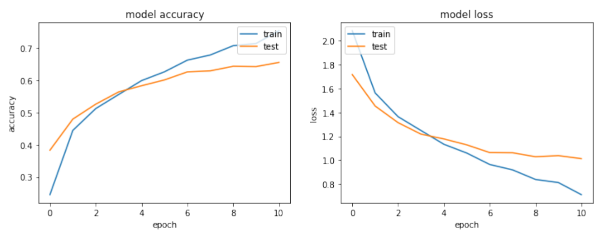
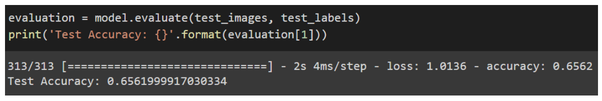


Figure 8: Model performance of the CNN model Accuracy vs. Epoch and Loss vs. Epoch



```
evaluation = model.evaluate(test_images, test_labels)
print('Test Accuracy: {}'.format(evaluation[1]))

313/313 [==============================] - 2s 4ms/step - loss: 1.0136 - accuracy: 0.6562
Test Accuracy: 0.6561999917030334
```

Figure 9: Accuracy over 11 epochs

to extract meaningful features from images through convolutional (and pooling) layers. They can learn spatial features that are invariant to transformations, making them more effective at detecting patterns in the image. In contrast, MLPs do not have spatial invariance and are not as effective at learning image features. Therefore, the CNNs typically outperform the MLPs in image classification tasks.

3.6 Task3.6 AND Comparing Different Pre-Trained Models

First, we choose 2 models: ResNet and VGG: [ResNet50, ResNet10, VGG16, and VGG19] as pre-trained models for `Task3.6`. The models are performed with a batch size of 64, and all convolutional layers are frozen. Each model contains 5 fully connected layers with 1024, 512, 256, 128 and 64 nodes, and a ReLU activation function. Figure 11 to figure 15 illustrate the network performance of the five different pre-trained models. We discovered that the VGG model outperforms the ResNet model in terms of accuracy. VGG16 has the best accuracy and attains its peak in the lowest amount of epoch. The ResNet model performs best with 50 layers. One possible explanation is that having too many layers might induce "vanishing gradients" or "exploding gradients," causing the network to degenerate and perform badly. (Another intriguing observation is that the loss stabilized throughout the whole x-axis when we used full batch gradient descent in ResNet50, however, the loss soared in ResNet50 using SGD after 60 epochs. This is also to be expected because just a portion of the training instances are utilized to calculate the loss value, which means it will likely be noisier and more erratic than the loss computed using the entire dataset. The loss function may oscillate more as a result of this.)

Training time grows as the number of convolutional and dense layers increases because a bigger network with more layers has more parameters to learn, necessitating greater computation and memory use during training. When all other variables are held constant, we discovered that ResNet50 has a quicker training time than VGG16 or a simple CNN. This is due to ResNet50's usage of skip connections, which helps to lessen the network's parameter number and calculation time during both forward and backward passes.

After determining the best model to use and how many convolutional layers to use (VGG16), we carefully examined how many dense layers to use and discovered that 5 dense layers work best. Figure 10 shows the network performance of VGG16 with 1 dense layer, and figure 11 illustrates the network performance of VGG16 with 5 dense layers. We may conclude that 5 dense layers could reach a higher accuracy for a smaller amount of epochs significantly, as it can be seen with a sharper gradient. VGG16 performed better than the CNN in Task3.5 because it required fewer epochs to achieve accuracy above 0.6. VGG16 has a nearly 20% increase in accuracy compared to the best MLP model in Task3.1 with two dense hidden layers, which is to be expected given that it uses convolutional layers to extract spatial features from images, which are more appropriate for image classification than the fully-connected layers used in MLP.

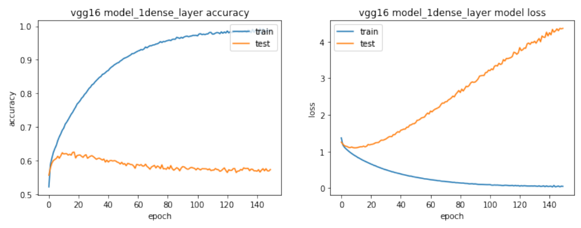


Figure 10: VGG16 with 1 dense layer: Accuracy vs. Epoch and Loss vs. Epoch

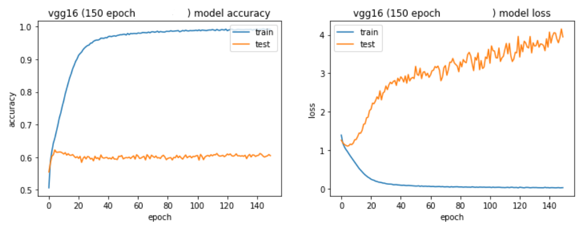


Figure 11: VGG16: Accuracy vs. Epoch and Loss vs. Epoch

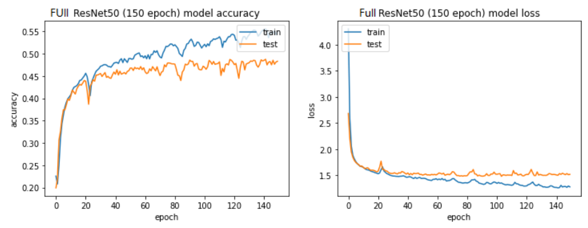


Figure 12: ResNet50 Full Batch: Accuracy vs. Epoch and Loss vs. Epoch

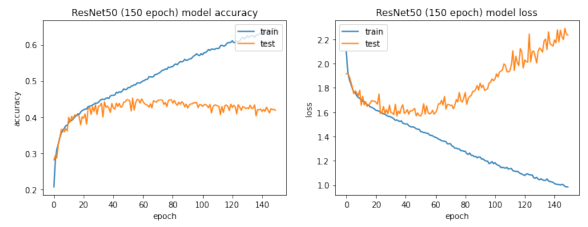


Figure 13: ResNet50 with Batch Size 64: Accuracy vs. Epoch and Loss vs. Epoch

3.7 The Effect of Width of the MLP

We are also interested in the effect of width on neural network performance. To experiment, we created an MLP model with 2 hidden layers, but with 64 and 64 units in each layer. The learning rate was set to 0.05 and regularization was not included in this model. Figure 16 shows that the test accuracy of this model converges to 0.3962 over 1000 iterations. Compared with the 2-hidden-layer-model (0.4188) in section `Task3.1`, we can conclude that MLP models with larger widths generalize to better network performance.

This is expected because in general, increasing the width of a layer in a neural network can increase its representational capacity (larger weights dimensions), allowing it to learn more complex features and patterns in the data. This can lead to improved performance, especially when dealing with complex datasets. However, increasing the width too much might also lead to overfitting.

4 Discussion and Conclusion

In this study, we mostly used MLP and CNNs to train our dataset. Future applications of our data could include the use of residual networks (ResNets), dense networks (DenseNets), and wide residual networks (WRNs). We anticipate that these models will perform better than the conventional CNN because ResNets and WRNs employ residual connections, enabling the network to skip some levels and learn more complicated features, while DenseNets use dense connections, enabling each layer to have access to the

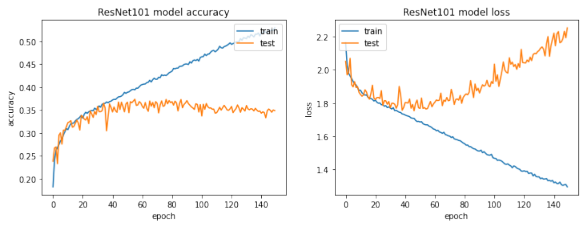


Figure 14: ResNet101: Accuracy vs. Epoch and Loss vs. Epoch

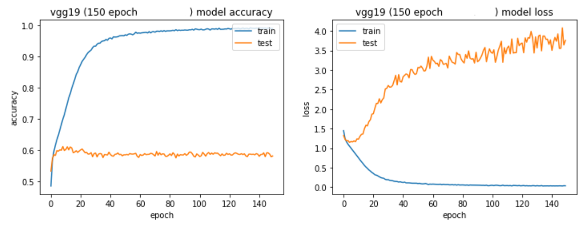


Figure 15: VGG19: Accuracy vs. Epoch and Loss vs. Epoch

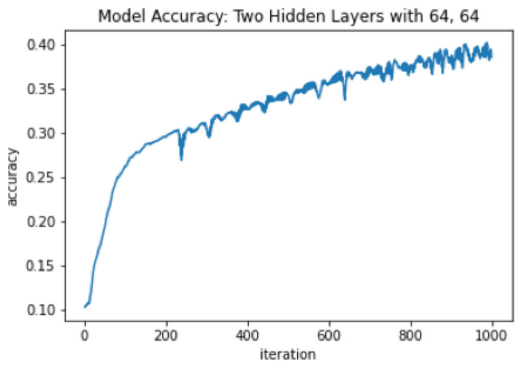


Figure 16: Model performance 2 hidden layers with 64 neurons: Accuracy vs. Iteration

feature maps of all preceding layers, enhancing the network's capability to learn complex features. [2] Also, rather than manually testing the parameter, we can make use of several hyperparameter tuning programs. Keras Tuner, Hyperopt, and Ray Tune are some of these packages.

5 Statement of Contributions

Every group member cooperated and contributed to each part of this project.

References

[1] Alex Krizhevsky. *Learning Multiple Layers of Features from Tiny Images*, `https://www.cs.toronto.edu/~kriz/cifar.html`. 2009.

[2] Gao Huang et al. "Densely Connected Convolutional Networks". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 2261–2269. DOI: `10.1109/CVPR.2017.243`.

[3] Alexey Dosovitskiy et al. "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale". In: *CoRR* abs/2010.11929 (2020). arXiv: `2010.11929`. URL: `https://arxiv.org/abs/2010.11929`.