Analysis of Convolutional Neural Networks and Multilayer Perceptron on CIFAR-10

Andrew Cheng
260707748
andrew.cheng@mail.mcgill.ca

Xianya Zhou
260729405
xianya.zhou@mail.mcgill.ca

Yutong Zhang
260727000
yutong.zhang2@mail.mcgill.ca

***Abstract*—Neural networks have recently demonstrated impressive classification performance on the CIFAR-10 dataset. In this paper, we illustrate the performance of two networks - the L-layer multilayer perceptron and the convolutional neural network - on CIFAR-10. We experimented with regularization techniques such as dropout and found it reduced overfitting but worsened convergence rates. Furthermore, we discovered adding more hidden layers to our multilayer perceptron did not increase accuracy but helped reduce overfitting. Finally, we investigated the general trends of accuracy as a function of the hyperparameters, the effects of different optimizers, and which classes were the hardest to classify.**

***Index Terms*—machine learning, computer vision, image classification, multilayer perceptron, convolutional neural network**

I. Introduction and Dataset

The *Canadian Institute For Advanced Research-10* dataset (CIFAR-10) first used by Krizhevsky et al. [1] is a well-known dataset for image multi-classification. Our goal is to implement an L-layer multilayer perceptron as well as a convolutional neural network (CNN) in order to learn the nuanced features of image data within CIFAR-10. Furthermore, we discovered that principle component analysis (PCA) did not improve our networks' performance. Similarly, the data augmentation techniques - rotations and shifts - did not help. Finally, we searched for the hyperparameters that optimize the performance of our networks.

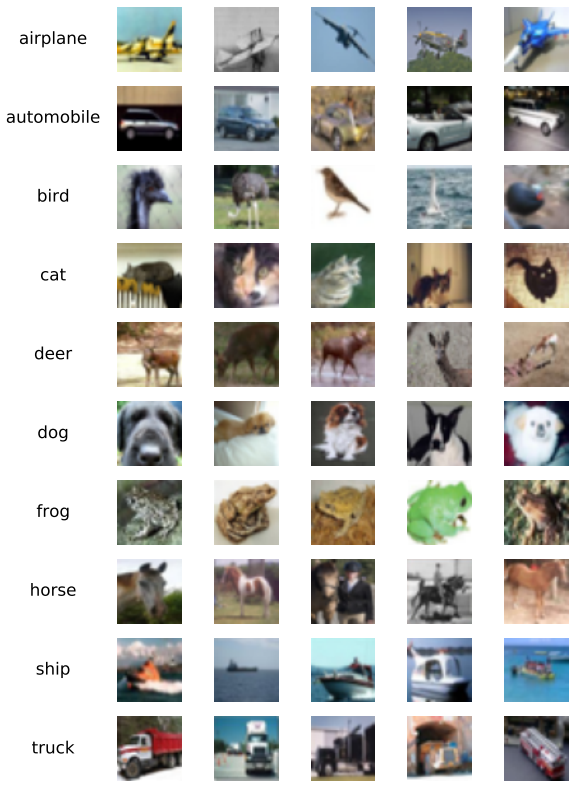

Fig. 1. Examples of 50 random images from each of 10 classes in the CIFAR-10 dataset.

II. Related Work

CIFAR-10 is widely used as a benchmark for state-of-the-art deep neural networks with a focus on computer vision. Recently, improvement in computational power and novel approaches have led to substantial progress of neural networks in terms of performances. In 2013, Network in Network, created by Lin et al. [2] achieved a groundbreaking 91.2% testing accuracy on the CIFAR-10 dataset achieved. By 2019, BiT-L by Kolesnikov et al. [3] achieved an astonishing 99.3% testing accuracy. These algorithms used creative approaches such as transferring representations of pre-trained data to achieve such results. CIFAR-10 is one of the many standard datasets that help illustrate the rapid advancement of neural networks.

III. Proposed Approach

A. Multilayer Perceptron

A multilayer perceptron (MLP) sequentially stacks (in most cases, fully connected) layers of neurons that computes

$$\hat{\mathbf{y}} = g^{[L]}\left(\mathbf{W}^{[L]}\left(\cdots g^{[1]}\left(\mathbf{W}^{[1]}\mathbf{X} + \mathbf{b}^{[1]}\right)\cdots\right) + \mathbf{b}^{[L]}\right),$$

where for each layer ℓ among L layers in total, $\mathbf{W}^{[\ell]}$, $\mathbf{b}^{[\ell]}$ and $g^{[\ell]}$ denote the weights, biases and its associated (nonlinear) activation function, respectively. The *Universal Approximation Theorem* states that an MLP with single hidden layer can approximate any continuous function with arbitrary accuracy. The required number of neurons, however, grows exponentially with increasing accuracy by the curse

of dimensionality. Neural networks, like MLP, are seen as black boxes that achieve high predictive power by sacrificing computational costs and interpretability of the model. Hence, we are interested in learning the best collection of parameters $\left\{\left(\mathbf{W}^{[\ell]}, \mathbf{b}^{[\ell]}\right)\right\}_{\ell=1}^{L}$ for our network that attains the lowest generalization error and that is computationally plausible. Alternatively, we try to minimize the *softmax cross-entropy loss* between the predicted class probabilities $\hat{\mathbf{y}}$ and the true labels $\mathbf{y}$, namely,

$$\begin{aligned}\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) &= \frac{1}{N} \sum_{k=1}^{N} \mathbf{y}_k \log \hat{\mathbf{y}}_k \\ &= \frac{1}{N} \left(-\mathbf{y}^{\intercal} \mathbf{Z}^{[L]} + \log \sum_{c=1}^{10} e^{\mathbf{Z}^{[L]}} \right),\end{aligned} \tag{1}$$

where N is the number of training samples. By applying the *forward propagation* and *backpropagation* (Rumelhart et al. [4]), as shown in Algorithm 1 and 2, repeatedly for some number of iterations (epochs), we could achieve our goal to train a successful model.

function `Forward(`$\mathbf{X}$`)`**:**
  $\mathbf{A}^{[0]} \leftarrow \mathbf{X}$;
  for *layer* $\ell \leftarrow 1$ **to** L **do**
    $\mathbf{Z}^{[\ell]} \leftarrow \mathbf{W}^{[\ell]} \mathbf{A}^{[\ell-1]} + \mathbf{b}^{[\ell]}$;
    $\mathbf{A}^{[\ell]} \leftarrow g^{[\ell]}\left(\mathbf{Z}^{[\ell]}\right)$;
  return $\mathbf{A}^{[L]}$;

Algorithm 1: Forward Propagation.

function `Backward(`$\hat{\mathbf{y}}, \mathbf{y}$`)`**:**
  for *layer* $\ell \leftarrow L$ **to** 1 **do**
    if $\ell = L$ **then**
      $\mathrm{d}\mathbf{Z}^{[\ell]} \leftarrow \hat{\mathbf{y}} - \mathbf{y}$;
    else
      $\mathrm{d}\mathbf{Z}^{[\ell]} \leftarrow \mathrm{d}\mathbf{A}^{[\ell]} \odot g'^{[\ell]}\left(\mathbf{Z}^{[\ell]}\right)$;
    $\mathrm{d}\mathbf{W}^{[\ell]} \leftarrow \left(\mathrm{d}\mathbf{A}^{[\ell-1]}\right)^{\intercal} \mathrm{d}\mathbf{Z}^{[\ell]} / N$;
    $\mathrm{d}\mathbf{b}^{[\ell]} \leftarrow \sum_{n=1}^{N} \mathrm{d}\mathbf{b}_n^{[\ell]} / N$;
    $\mathrm{d}\mathbf{A}^{[\ell-1]} \leftarrow \mathrm{d}\mathbf{A}^{[\ell]} \left(\mathbf{W}^{[\ell]}\right)^{\intercal}$;

Algorithm 2: Backpropagation. Notice that $\odot$ stands for element-wise multiplication.

As suggested by Glorot et al. [5], we initialize our weights according to the scaled uniform distribution

$$\mathbf{W}^{[\ell]} \sim \mathrm{Unif}\left(-\frac{\sqrt{6}}{\sqrt{n_\ell + n_{\ell+1}}}, \frac{\sqrt{6}}{\sqrt{n_\ell + n_{\ell+1}}} \right)$$

to speed up convergence rate of training, where n_ℓ is the number of neurons for layer ℓ. The overall training procedure of each epoch is described as follows:

(i) Feed the current batch of training data $\mathbf{X}$ into the network and obtain $\hat{\mathbf{y}}$ using forward propagation.
(ii) Calculate the softmax cross-entropy loss between $\hat{\mathbf{y}}$ and $\mathbf{y}$ with formula (1).
(iii) Back-propagate the loss to obtain its derivatives with respect to $\mathbf{W}^{[\ell]}$ and $\mathbf{b}^{[\ell]}$, namely, $\mathrm{d}\mathbf{W}^{[\ell]}$ and $\mathrm{d}\mathbf{b}^{[\ell]}$.
(iv) Update parameters $\mathbf{W}^{[\ell]}$ and $\mathbf{b}^{[\ell]}$ using their derivatives with a suitable optimization scheme. We choose to implement *mini-batch gradient descent with momentum*. Its update rule, for layer $\ell = 1, \cdots, L$, is

$$\begin{cases} \mathbf{v}_{\mathrm{d}\mathbf{W}^{[\ell]}} \leftarrow \beta \mathbf{v}_{\mathrm{d}\mathbf{W}^{[\ell]}} + (1-\beta)\, \mathrm{d}\mathbf{W}^{[\ell]} \\ \mathbf{W}^{[\ell]} \leftarrow \mathbf{W}^{[\ell]} - \alpha \mathbf{v}_{\mathrm{d}\mathbf{W}^{[\ell]}} \end{cases}, \tag{2}$$

$$\begin{cases} \mathbf{v}_{\mathrm{d}\mathbf{b}^{[\ell]}} \leftarrow \beta \mathbf{v}_{\mathrm{d}\mathbf{b}^{[\ell]}} + (1-\beta)\, \mathrm{d}\mathbf{b}^{[\ell]} \\ \mathbf{b}^{[\ell]} \leftarrow \mathbf{b}^{[\ell]} - \alpha \mathbf{v}_{\mathrm{d}\mathbf{b}^{[\ell]}} \end{cases}, \tag{3}$$

where the learning rate α and the momentum β are both hyperparameters that require fine-tuning.

Furthermore, we also implement Dropout (introduced by Srivastava et al. [6]), which randomly shuts down every neuron with a given probability p. In other words, we only keep on average $np_k = n(1-p)$ neurons active for training and set parameters of all inactive neurons to zero for a layer containing n neurons, where the keep probability p_k is also a hyperparameter we need to search. This technique is often used to reduce overfitting in practice.

B. Convolutional Neural Network

CNN or conv-nets is a special case of MLP with convolutional layers. AlexNet first introduced deep CNN into the field of image recognition. Its impressive performance on the ImageNet dataset [7] inspired the development of many other deep CNN architectures. The activation function for CNN is commonly a ReLU layer, and is subsequently followed by additional convolutions such as pooling layers, fully connected layers and normalization layers, referred to as hidden layers because their inputs and outputs are masked by the activation function and final convolution.

The objective of the Convolution Operation is to extract the high-level features such as edges, from the input image. We train the model by using backpropagation which is similar to MLP, shown in the previous subsection. Here for the images with 3-channel as in our dataset CIFAR-10, the Kernel has the same depth as that of the input image. Matrix multiplication is performed between Kn and In stack and all the results are summed with the bias to give us a squashed one-depth channel Convoluted Feature Output.

IV. Results

For the multilayer perceptron, we implemented a minimal learning framework that resembles mainstream libraries like PyTorch and Keras, allowing us to freely stack any number of fully connected and activation layers. Also, the code is written using PyTorch's `Tensor`, which helps us exploit the efficiency of atomic linear algebra operations (such as matrix multiplication) on GPU and brings massive speed-up. For the convolutional neural network, we used a 14-layer architecture with layer details specified in Table I for training and testing. To measure the performance under each setting of hyperparameters, we recorded the loss and accuracy with

Layer No.	Specification
1	Convolution (64 filters, $(3,3)$ kernel, same padding)
2	Max Pooling ($(2,2)$ pool) Activation (ReLU) Batch Normaliziation
3	Convolution (128 filters, $(3,3)$ kernel, same padding)
4	Max Pooling ($(2,2)$ pool) Activation (ReLU) Batch Normaliziation
5	Convolution (256 filters, $(3,3)$ kernel, same padding)
6	Max Pooling ($(2,2)$ pool) Activation (ReLU) Batch Normaliziation
7	Convolution (512 filters, $(3,3)$ kernel, same padding)
8	Max Pooling ($(2,2)$ pool) Activation (ReLU) Batch Normaliziation
9	Flatten
10	Fully Connected (128 units) Activation (ReLU) Dropout (keep probability 0.75) Batch Normaliziation
11	Fully Connected (256 units) Activation (ReLU) Dropout (keep probability 0.75) Batch Normaliziation
12	Fully Connected (512 units) Activation (ReLU) Dropout (keep probability 0.75) Batch Normaliziation
13	Fully Connected (1024 units) Activation (ReLU) Dropout (keep probability 0.75) Batch Normaliziation
14	Fully Connected (10 units) Activation (Softmax)

TABLE I
SPECIFICATION OF THE 14-LAYER ARCHITECTURE WE USED IN OUR CNN EXPERIMENTS.

respect to the number of epochs on both training and test sets. All experiments were carried out on a 3.6 GHz Intel i9-9900k CPU with 32 GB of RAM, and an NVIDIA GeForce RTX 2080 Ti GPU.

A. Multilayer Perceptron

Most of our MLP results are based on a network with only a single hidden layer for simplicity. In addition, we fixed some hyperparameters across all comparisons: batch size 64, the number of epochs 300, sigmoid activation function σ for the hidden layer, and momentum $\beta = 0.9$.

Fig. 2 compares the network performance with $n_h = 256$, $n_h = 512$, $n_h = 1024$ hidden neurons using the learning rate $\alpha = 0.05$ and no dropout. With a larger hidden layer, the overall training and test losses decrease while their accuracies increase as expected. But we also note that the performance change is not obvious, which indicates the "marginal return" of increasing hidden layer size only is rather small.

Another aspect of interest would be the relationship between performance and learning rate α. We tested $\alpha = 0.01$, $\alpha = 0.05$, $\alpha = 0.1$ using 1024 neurons in the hidden layer, and the result is shown in Fig. 3. In the long run, we expect $\alpha = 0.01$ to be the best learning rate among three choices because its

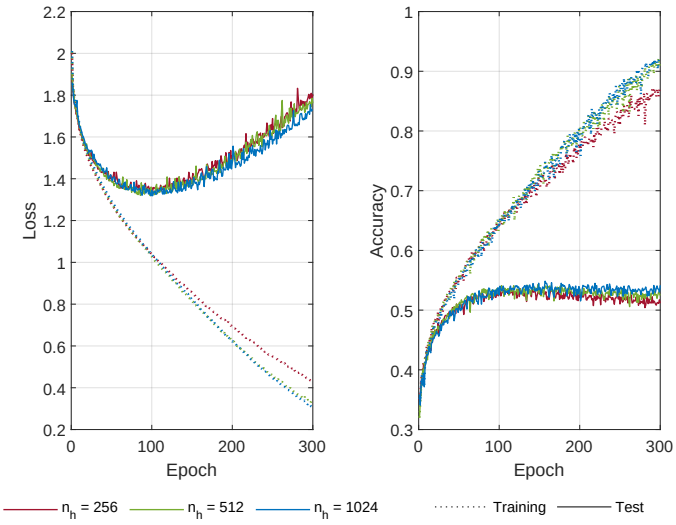

Fig. 2. Loss and accuracy vs. the number of epochs with different number of hidden layers for the MLP.

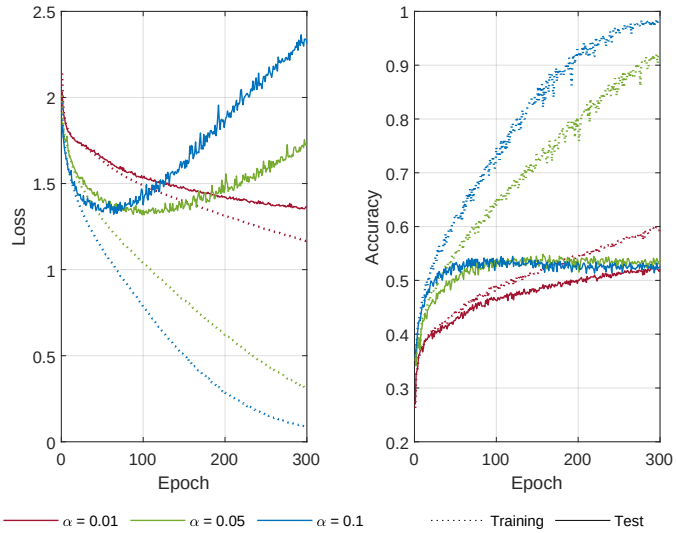

Fig. 3. Loss and accuracy vs. the number of epochs with different learning rates for the MLP.

loss still keeps decreasing after 300 epochs and overfitting is barely observed. Both $\alpha = 0.05$ and $\alpha = 0.1$, on the other hand, suffers from overfitting as their loss curves start to rise at roughly 100^{th} epoch. Within tight computation budget, we would still like to choose $\alpha = 0.05$ for its relatively less observed overfitting and sufficiently good performance on the test set.

Furthermore, we investigated how dropout technique performs and whether it could reduce overfitting of our model to the training data. As shown in Fig. 4, we applied no dropout, $p_k = 0.9$, $p_k = 0.75$, $p_k = 0.5$, respectively. We observe that with higher keep probability p_k (no dropout means $p_k = 1.0$), the softmax cross-entropy loss converges faster. On the other hand, however, the discrepancy between training and test accuracy becomes larger, which clearly implies overfitting.

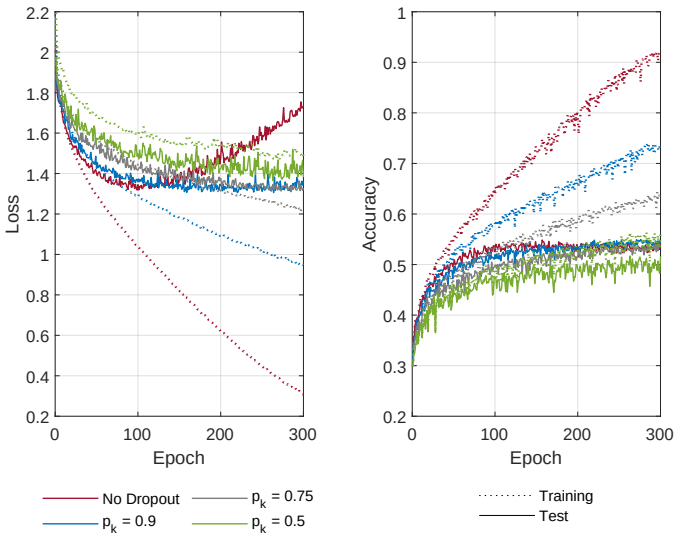

Fig. 4. Loss and accuracy vs. the number of epochs with different dropout keep probabilities for the MLP.

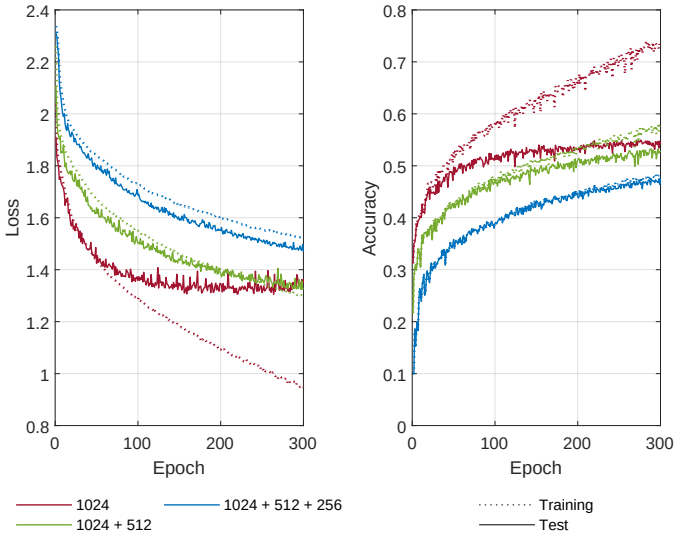

Fig. 5. Loss and accuracy vs. the number of epochs with different configurations of layers for the MLP.

Overall, we shall choose $p_k = 0.9$ in later experiments since it achieves highest average test accuracy yet reasonable training accuracy according to the plot.

Finally, we tried adding more hidden layers to our MLP, where the following configurations are tested:

- 1 hidden layer: $n_{h_1} = 1024$;
- 2 hidden layers: $n_{h_1} = 1024$, $n_{h_2} = 512$;
- 3 hidden layers: $n_{h_1} = 1024$, $n_{h_2} = 512$, $n_{h_3} = 256$.

The result is shown in Fig. 5. Quite unexpectedly, the overall model performance did not increase and convergence rate becomes slower with more layers. But on the bright side, we also observe smaller gaps between training and test accuracies or less overfitting.

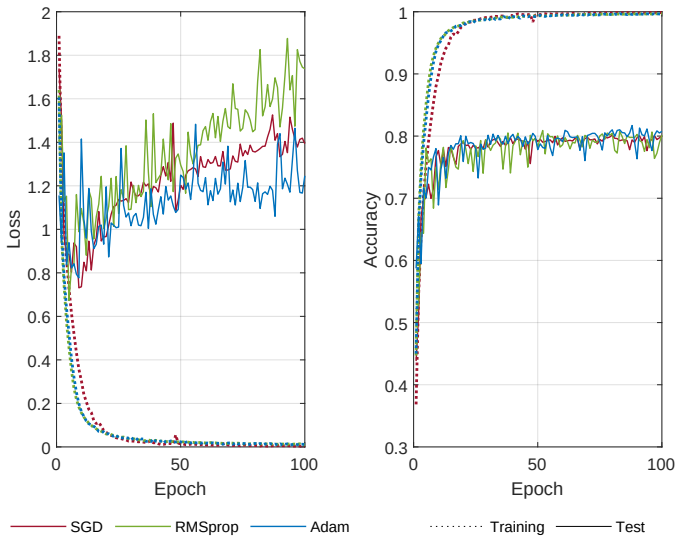

Fig. 6. Loss and accuracy vs. the number of epochs with different optimizers for the CNN. In terms of hyperparameter selection, we chose $\alpha = 0.01$, weight decay of 10^{-6}, $\beta = 0.9$, Nesterov momentum for SGD; $\alpha = 0.001$, $\rho = 0.9$ for RMSprop; $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ for Adam.

B. Convolutional Neural Network

We tested the performance of CNN under three different optimizers - SGD, RMSprop, and Adam (see Fig. 6). We found that Adam performed the best. By epoch 50, the training accuracy converges to almost 100% which further illustrates the capability of CNN to "memorize" the image training data. Also note increasing epochs does not necessarily increase testing accuracy, in fact, it can worsen testing performance. In particular, for RMSprop we see an increasing linear trend of the loss as a function of epochs. We can conclude the optimizer chosen evidently affects the performance of the network.

To delve into the actual misclassification rate of our model for each category in the CIFAR-10 dataset, we computed the confusion matrix for our Adam-optimized model which is shown as Fig. 7. Each diagonal entry identifies the number of correctly classified images under the given category, while off-diagonal entries specify how many images are misclassified between row and column classes. Among all 10 categories, *cat* and *dog* are two most wrongly classified as the model makes mistakes on $131 + 89 = 220$ images. This phenomenon is expected since cats and dogs are indeed very similar in appearance, and especially, the loss of subtle details caused by low-resolution images of CIFAR-10 makes it even harder to distinguish between those two classes.

V. Discussion and Conclusion

In this project, we explored the effects of different hyperparameter settings on the performance of MLP. We also analyzed how changes to the architecture of both networks affected their accuracy.

For MLP, we learned that increasing the number of hidden layers yielded little improvements in accuracy, but reduced overfitting. Also, the loss and accuracy had high variance with

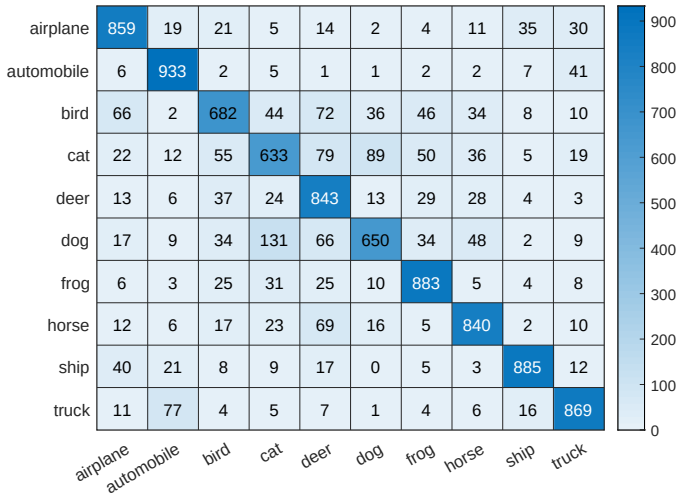

Fig. 7. Confusion matrix of our CNN model for each category in CIFAR-10. Diagonal entries represent the number of correctly classified images.

Model	Hyperparameters	Accuracy
2-Layer MLP	Hidden layer: 1024 Batch size = 64 $\alpha = 0.05$ $\beta = 0.9$ $p_k = 0.9$	55.34%
3-Layer MLP	Hidden layers: 1024, 512 Batch size = 64 $\alpha = 0.025$ $\beta = 0.9$ $p_k = 0.9$	53.89%
4-Layer MLP	Hidden layers: 1024, 512, 256 Batch size = 64 $\alpha = 0.025$ $\beta = 0.9$ $p_k = 0.9$	48.48%
14-Layer CNN	Architecture shown in Table I Batch size = 128 Adam optimizer $\alpha = 0.001$ $\beta_1 = 0.9$ $\beta_2 = 0.999$	81.10%
ResNet20v1 (He et al. [8])	Code available at: https://keras.io/examples/cifar10_resnet/	91.83%

TABLE II
BEST HYPERPARAMETERS FOR EACH OF OUR MODELS AND THEIR HIGHEST TEST SET ACCURACIES.

respect to different learning rates which implies that MLP is highly sensitive to its learning rate. We also observed that the soft-max cross entropy loss converges slower when the keep probability p_k lowers, but it also decreases overfitting. Therefore dropout sacrifices the speed of convergence for the mitigation of overfitting.

For CNN we examined different optimizers and found the choice of optimizer had substantial influence on the network. We also computed the confusion matrix (using Adam) to have a detailed understanding of the prediction in each category. Unsurprisingly, the misclassification rate between cat and dog was the highest while the performance of the network on other categories was much better.

In Table II, we also included the performance of a relatively newer model (without tuning any hyperparameters), ResNet20v1 (He et al. [8]), which is readily available at keras.io as Keras official documentation. This is an example to illustrate the significant improvements in performance and availability of neural networks during the last decade.

In conclusion, we understand the significance of each component to the model, as well as the robustness of the model in that only changing one factor leads to a small increase in the accuracy. We achieved a final accuracy of 55.34% with MLP and of 81.10% with CNN. For future investigations, we would like to perform more comprehensive experiments on more choices of hyperparameter settings, as well as other variations of the network architecture.

VI. Future Work

Neural networks are notorious for having many hyperparameters to tune. In turn, the majority of our work was devoted to manually tuning the network or relying on the grid search method. However, Snoek et al. [9] introduced Bayesian optimization approaches to automate machine learning algorithms including neural networks. In fact, they included a section on automating CNN's performance on CIFAR-10. One could further explore the techniques introduced by Snoek et al. and analyze their results on the multilayer perceptron.

VII. Statement of Contribution

Andrew examined the effects of image augmentation, built and tuned CNN, worked on the MLP model, and was responsible for the main part of the report. Xianya built and ran our CNN model using PyTorch. Yutong was responsible for building the MLP model from scratch and reported its results.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[2] M. Lin, Q. Chen, and S. Yan, "Network in network," 2013.

[3] A. Kolesnikov, L. Beyer, X. Zhai, J. Puigcerver, J. Yung, S. Gelly, and N. Houlsby, "Big transfer (bit): General visual representation learning," 2019.

[4] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[5] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, Y. W. Teh and M. Titterington, Eds., vol. 9. Chia Laguna Resort, Sardinia, Italy: PMLR, 13–15 May 2010, pp. 249–256.

[6] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105.

[8] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[9] J. Snoek, H. Larochelle, and R. P. Adams, “Practical bayesian optimization of machine learning algorithms,” 2012.