Deep Learning for Image Recognition: Convolutional Neural Networks (CNNs) and Multilayer Perceptron (MLP)

Zainab Almheiri, Renato Salinas, Mohammad Amini

***Abstract*—Convolutional neural networks are state-of-the-art computer visions and they are widely used in image recognition. In this paper, we apply and compare the performance of two algorithms, multilayer perceptron and convolutional neural networks. These models are developed to predict 10 different classes of images using the CIFAR-10 dataset. There are many deep Convolutional neural networks that achieve better performance for image classification including Xception, ResNet50, InceptionV3, DenseNet, and GoogLeNet (Inceptionv1). However, these models require heavy computation (large space of RAM). Due to limited access to resources during the pandemic period (COVID 19) and some equipment malfunctions, we implemented a simple yet efficient architecture of Convolutional neural networks that achieved 80% accuracy. First, We implemented image prepossessing to maintain model performance. Then, we developed convolutional neural networks using PyTorch and TensorFlow and compared their performance.The results show that the developed convolutional neural networks (CNN-3) using TensorFlow and regularization techniques such as drop out and data augmentation achieved the highest performance than the other models.The results also demonstrated the superiority of convolutional neural networks over multilayer perceptron.**

***Keywords*—Image Recognition, Deep learning, convolutional neural network, multilayer perceptron**

I. Introduction

Convolutional neural networks (CNNs) are commonly used in image recognition, speech recognition, natural language processing, and video analysis. CNNs have achieved high performance in these areas [1]–[3]. It is a neural network with multiple and deep numbers of hidden layers (from 5 to 25) that detect patterns in a given dataset. Additionally, CNNs extract different features of the input at different level (layer), it extracts more detailed features as it goes deeper [1].

The results show the superiority of CNN models over MLP in terms of accuracy. MLP gave the worst result as expected, not only due to the MLP training procedures, but also due to the fact that it was created from scratch with limited resources. CNN with PyTorch gave an average result even with the Adam algorithm, and CNN with TensorFlor surpassed our expectations when training it with this data.

The rest of the paper is organized as follows: Section II summarizes Related Work, Section III discusses IFAR-10 Dataset and preprocessing. Section IV presents Mathematical representation and setup of the applied models. Section V elaborates on Model Implementation. Section VI summarizes the results with the experiment setting. We conclude the paper in section VII. Finally, in section VIII, we explain the contribution of the teammates.

II. Related Work

In [4] , the authors trained a two-layer convolutional Deep Belief Network (DBN) on CIFAR-10 dataset, and achieved 78.9% on the test set. Additionally, In [5], the authors achieved 74.5% using a sparse-coding scheme.

III. Dataset and preprocessing

In this study, we use the CIFAR-10 dataset for image recognition task. We apply preprocessing method which is an essential step to maintain high model performance. This section discuses the applied dataset, and preprocessing method.

A. *CIFAR-10 Dataset*

The CIFAR-10 dataset contains two separated files, training dataset to train the implemented model, and test dataset to evaluate the developed model. Training dataset consists of 50000 training images, whereas test dataset consists of 10000 test images. The CIFAR-10 dataset contains 60000 colored images, each image is represented as 32x32 of pixel values. The main task of the implemented model is to classify these images into 10 classes, airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck.

B. *Preprocessing*

In this paper, we implemented, for PyTorch, a simple preprocessing, and for TensorFlow, a simple and a complicated image preprocessing to see how this improves the model performance. For PyTorch, we loaded the images using a dataloader and normalized then the data with a given mean of 0.5, and a given standard distribution of 0.5.

For TensorFlow, in the simple image preprocessing, we applied the following procedures, (1) image reshape to [width] x [height]x [pixel], (2) one hot encoding for labels, (3) Data normalization for pixels to be between 0 and 1, and finally (4) splitting training dataset to validate the model and choose the hypermeters. Whereas, in complicated procedures, we used the OpenCV package for image preprocessing. The procedures of complicated image preprocessing are as follows (1) convert image into grayscale, (2) image thresholding (covert an image into a binary image (e.g., assign a pixel with either a value of 0 or 1), (3) find contours (using find contours()). This method is very useful for object detection and recognition, (4) contour features, which does not consider the rotation of an object, where the start coordinate, and width and height of rectangular are (x,y) and (w,h), respectively. Then we applied the same implemented steps of simple preprocessing. Simple image preprocessing was selected as it showed better performance on developed models.

Furthermore, due to implementing Multilayer Perceptron from scratch, we need to take care of each element that contributes to model building. And it does not operate well if the input, output, and initial weight data are not configured to fit the network architecture. Also, there are two common problems called saturation and zero values that we need to take into account when we are introducing the initial elements of our MLP, therefore, based on above we chose the below policies: Initial link weights: to avoiding zero value, it should be random and small. To do that, we sample the weights from a normal probability distribution around zero and with the standard deviation that related to the number of incoming links into a perceptron, so simply we have:

$$\frac{1}{\sqrt{(\#\text{ of incoming links })}} \tag{1}$$

Inputs: inputs should be normalized to be small, but not zero to avoid zero value. In our model to scale the inputs, we divide them by 255 to bring them into the range of 0-1. Still, our goal is 0.01 – 0.99 to avid the problems so by multiply normalize input to 0.99 and then add by 0.01 we reach our goal. Outputs: the output range should be within what the activation function can produce. Because our baseline model is a logistic sigmoid function to avoid the problems, we set the range between 0.01 – 0.99. Because we have 10 classes in our CIFAR-10 dataset, so we have an array with 10 elements we set the goal predicted label to 0.99, and the reset are 0.01 for each prediction. Finally for simplicity of implementing our Multiplayer Perceptron model we converted the data set to a CSV file by keeping all 3072 RBG features for each instance and then replaced categorical labels by numbers.

IV. Setup Design of the Applied Models

A. Model 1: CNNs

We first developped CNNs using PyTorch (CNN-1), and the results show that the model achieves around 60% accuracy on CIFAR-10 dataset. Afterwards, we applied CNNs with a different setup design using TensorFlow (CNN-2). Data augmentation was then developed to improve the model performance (CNN-3).The results show that the model accuracy has improved compared to the first CNNs using PyTorch. For CNNs, there are four main hyperparameters, which are size and number of filters, stride, and padding. Different hyperparameters were assigned for each CNN. This section discusses the setup design of the developed models in each platform.

- Setup Design of CNN-1 in PyTorch: The first CNN-1 was developed by following the approach of mixing the negative log likelihood loss and the log(softmax(x) function to the n-dimensional input Tensor, using the following formula:

$$\operatorname{loss}(x, \text{ class }) = -x[\text{ class }] + \log\left(\sum_j \exp(x[j])\right) \tag{2}$$

In order to optimize it, we used Adam: A Method for Stochastic Optimization. This algorithm is for first-order gradient-based optimization. This method seemed optimal as it has little memory requirements, which makes it efficient. After using this algorithm and comparing it with a regular Stochastic Gradient Descent algorithm with momentum, Adam does better in when training with this data. After such implementation, the overall performance ends up giving around 60% accuracy instead of 53% from the Stochastic Gradient Descent. On the other hand, the batch size was kept as 4, since increasing the batch size, specially when using SGD (Stochastic gradient descent), showed a loss in accuracy. Meanwhile increasing the number of loops over the dataset while training the network did not give different results regarding accuracy but a much lower loss when trying the Adam algorithm as seen in Table II.

- Setup Design of CNN-2 in TensorFlow: The final CNN-2 is developed after training and validating the algorithms at different hyperparameters including padding, number and size of the filter, and stride. The results show better performance in terms of accuracy when implementing padding where the output dimensions are the as same the input dimensions, (see Fig. 1 and Fig. 2). In Fig.1, the validation accuracy has reached to around 68% , whereas 73% in Fig.2 with padding setup. Additionally, regularization technique such as drop out is implemented to avoid overfitting [6].

- Setup Design of CNN-3 in TensorFlow: CNN-3 is developed following the same setup design as CNN-2. However, another technique of regularization is applied called data augmentation. Data augmentation is widely used technique in deep learning, particularly in image recognition task. This technique enhances the training of convolutional neural network [7]. The results reveal that data augmentation could improve the model performance. In other words, the model performance has increased from CNN-2 = 73% to CNN-3 = 80%, (see Fig.3).

B. Model 2: Multilayer Perceptron

There are two main parts due to build a Multilayer Perceptron:

1. Feedforward: In general, the goal of doing feedforward is to predict labels related to each input so we have:

$$\hat{y}_k = g\left(\sum_m W_{k,m} h\left(\sum_d V_{m,d} x_d\right)\right) \tag{3}$$

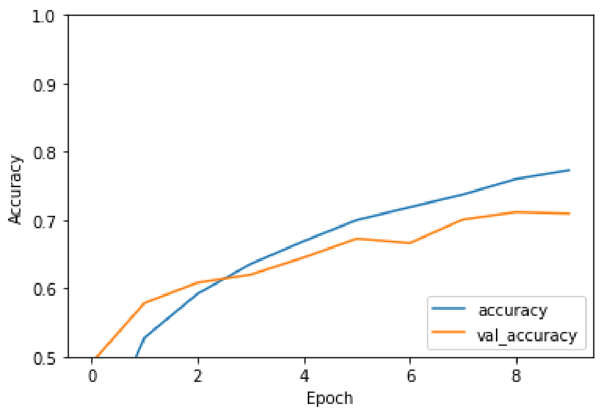


Fig. 1: CNN-2 validation performance without padding (68%).

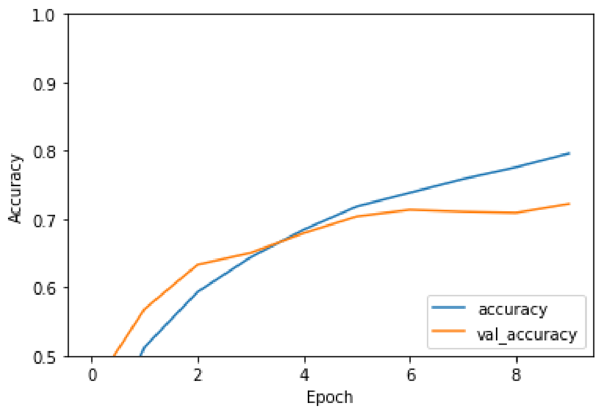


Fig. 2: CNN-2 validation performance with padding (73%).

Where contains inputs, link weights and activation function. It is good to mention that for our model we deal with biases in preprocessing phase. For our model we implemented logistic sigmoid, ReLU and Leaky ReLU activation function for hidden layers and SoftMax for output layer. But our baseline is Logistic Sigmoid activation function, to use the others we need to reshape the data after using them.

2. Backpropagation: There are some methods to backpropagate cost function between perceptrons. For our baseline model we used SGD method for link weights updates as follow:

$$\Delta W_{ho} = \alpha * C_o * \text{sigmoid}\left(O_o\right) * \left(1 - \text{sigmoid}\left(O_o\right)\right) \cdot O_h^{\top} \quad (4)$$

Where C is cost function of the model, O is output of the model and o, h express output layer and hidden layer respectively.

To chose the number of per

V. Proposed Methods

Many other CNN approaches might achieve higher accuracy and better performance than the applied models in this paper. Although the applied models have simplistic architecture, they are effective and have shown good performance. We suspect a deeper architecture of CNNs including GoogleNet, InceptionV3, Xception, VGG16, DenseNet, and MobileNet could achieved higher accuracy. GoogleNet is developed following

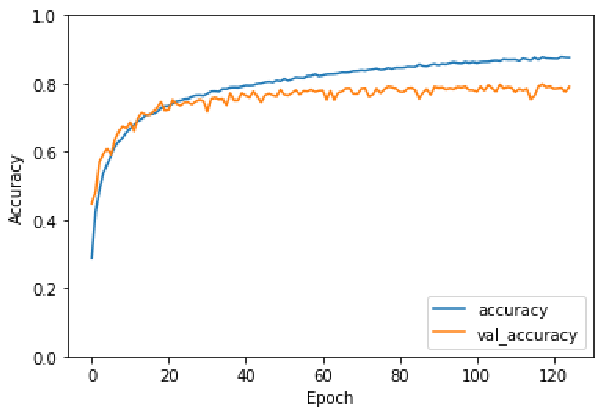


Fig. 3: Validation performance of CNN-3 (80%).

the proposed architecture by [8]. The code is written in google colab but it needed a large size of RAM, the allowed space was not enough to execute the code. Due to the limited in time, resources and some equipment malfunction during the pandemic period, we wrote the code and did not reach to the final result (see appendix for more details about the architecture of GoogleNet).

VI. Results

A. Experiment Setting

Jupyter notebook and Google colab are used as running environment to implement all the models. Three different RAM and CPU are used. Two methods of Cross validation are applied including hold-out and k-fold.

B. Result of the the developed Models

In this section, we present the performance of each model including training time, loss, and accuracy of CNN-1,CNN-2,CNN-3, and MLP. Finally, we present the final performance of these models on the test CIFAR-10 dataset. For our MLP we experimented the model with two and three layers including different number of perceptrons. in one experiment with two layer we achieved 42% accuracy and by changing to three layer and increase number of epochs we observed much better performance. Our best accuracy experiment for MLP is 57%.

choosing the number of perceptrons and layers is depending on the dataset and policies that we design to implement.

TABLE I: Performance of CNN-1 while tuning hyperparameters with 2 epoch, Adam's algorithm, final accuracy 60%.

Epoch	Training Time (sec)	Training loss
1	90	1.223
2	89	1.157

VII. Conclusion and Future Recommendations

In this study, we applied on of the powerful models that is recommended for image recognition, CNNs. We applied two different CNNs using PyTorch and TensorFlow. The

TABLE II: Performance of CNN-1 while tuning hyperparameters with 100 epoch, Adam's algorithm, final accuracy 60%.

Epoch	Training Time (sec)	Training loss
1	89	1.378
2	90	1.262
...	...	...
100	92	0.625

resules showed that the developed CNN-2 using TensorFlow is achieved higher accuracy (73%) than the one developed in PyTorch, CNN-1 (60%). Additionally, MLP was developed and compored to CNNs. The results proved that CNN-2 achivied the highest accuracy. Whereas, CNN-1 and MLP have achieved almost the same accuracy, 60%.

For future recommendations, we suggest using more sophisticated CNNs including Xception, ResNet50, InceptionV3, DenseNet, and GoogLeNet (Inceptionv1). They would give better accuracy. We also suggest conducting other preprocessing techniques without converting the image into grayscale. As reportes in this paper, colored images gave better performance.

VIII. Statement of Contribution

We all contributed in the division of the preprocessing and feature extraction. For the models, we divided the models equally and helped each other whenever we had questions during our weekly online meetings. We all contributed in the report and putting things together for the evaluation.

References

[1] S. Hijazi, R. Kumar, and C. Rowen, "Using convolutional neural networks for image recognition," *Cadence Design Systems Inc.: San Jose, CA, USA*, pp. 1–12, 2015.

[2] Z. Liang, A. Powell, I. Ersoy, M. Poostchi, K. Silamut, K. Palaniappan, P. Guo, M. A. Hossain, A. Sameer, R. J. Maude, *et al.*, "Cnn-based image analysis for malaria diagnosis," in *2016 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pp. 493–496, IEEE, 2016.

[3] L. Shang, Q. Yang, J. Wang, S. Li, and W. Lei, "Detection of rail surface defects based on cnn image recognition and classification," in *2018 20th International Conference on Advanced Communication Technology (ICACT)*, pp. 45–51, IEEE, 2018.

[4] A. Krizhevsky and G. Hinton, "Convolutional deep belief networks on cifar-10," *Unpublished manuscript*, vol. 40, no. 7, pp. 1–9, 2010.

[5] K. Yu and T. Zhang, "Improved local coordinate coding using local tangents," 2010.

[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. Hernández-García and P. König, "Further advantages of data augmentation on convolutional neural networks," in *International Conference on Artificial Neural Networks*, pp. 95–103, Springer, 2018.

[8] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

TABLE V: Performance of MLP for 10 epochs, and learning rate 0.1.

Epoch	Training Time (sec)	Training loss
1	120	4.943703
2	123	3.030600
3	127	2.749477
4	131	1.684069
5	132	1.575203
6	133	1.466832
7	133	1.360362
8	134	1.358724
9	133	1.254246
10	133	1.273482

TABLE VI: Performance of the Final Models Using test dataset.

Model	Test Accuracy
MLP	57%
CNN-1	60%
CNN-2	73%
CNN-3	**80%**

TABLE III: Performance of CNN-2 while tuning hyperparameters.

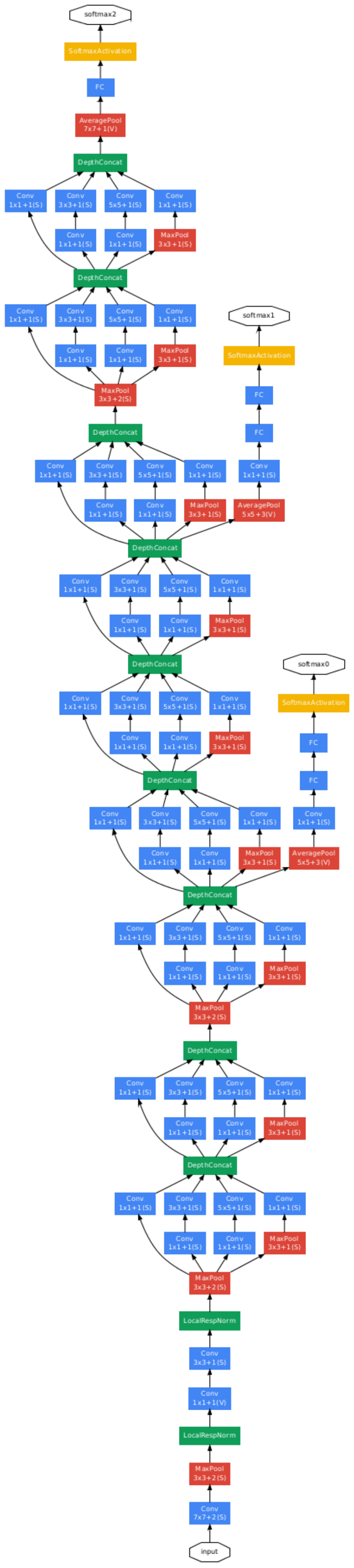
Epoch	Training Time (sec)	Training loss	Training Accuracy	Validation loss	Validation Accuracy
1	188	1.7717	0.3474	1.5410	0.4495
2	185	1.3658	0.5123	1.1983	0.5681
3	186	1.1548	0.5932	1.0055	0.6331
.....	.....	.....	.....	.....	.....
10	187	0.5930	0.7961	0.8364	0.7223

TABLE IV: Performance of CNN-3 while tuning hyperparameters.

Epoch	Training Time (sec)	Training loss	Training Accuracy	Validation loss	Validation Accuracy
1	105	1.9185	0.2875	1.5065	0.4486
2	106	1.5851	0.4245	1.4834	0.4827
3	105	1.4240	0.4877	1.1814	0.5720
.....	.....	.....	.....	.....	.....
125	106	0.3636	0.8770	0.7696	0.7903

IX. Appendix

The deep architecture of GoogleNet.


softmax2
SoftmaxActivation
FC
AveragePool
7x7+1(V)
DepthConcat
Conv
1x1+1(S)
Conv
3x3+1(S)
Conv
5x5+1(S)
Conv
1x1+1(S)
Conv
1x1+1(S)
Conv
1x1+1(S)
MaxPool
3x3+1(S)
DepthConcat
Conv
1x1+1(S)
Conv
3x3+1(S)
Conv
5x5+1(S)
Conv
1x1+1(S)
softmax1
Conv
1x1+1(S)
Conv
1x1+1(S)
MaxPool
3x3+1(S)
SoftmaxActivation
MaxPool
3x3+2(S)
FC
DepthConcat
FC
Conv
1x1+1(S)
Conv
3x3+1(S)
Conv
5x5+1(S)
Conv
1x1+1(S)
Conv
1x1+1(S)
Conv
1x1+1(S)
Conv
1x1+1(S)
MaxPool
3x3+1(S)
AveragePool
5x5+3(V)
DepthConcat
Conv
1x1+1(S)
Conv
3x3+1(S)
Conv
5x5+1(S)
Conv
1x1+1(S)
Conv
1x1+1(S)
Conv
1x1+1(S)
MaxPool
3x3+1(S)
DepthConcat
softmax0
Conv
1x1+1(S)
Conv
3x3+1(S)
Conv
5x5+1(S)
Conv
1x1+1(S)
SoftmaxActivation
Conv
1x1+1(S)
Conv
1x1+1(S)
MaxPool
3x3+1(S)
FC
DepthConcat
FC
Conv
1x1+1(S)
Conv
3x3+1(S)
Conv
5x5+1(S)
Conv
1x1+1(S)
Conv
1x1+1(S)
Conv
1x1+1(S)
Conv
1x1+1(S)
MaxPool
3x3+1(S)
AveragePool
5x5+3(V)
DepthConcat
Conv
1x1+1(S)
Conv
3x3+1(S)
Conv
5x5+1(S)
Conv
1x1+1(S)
Conv
1x1+1(S)
Conv
1x1+1(S)
MaxPool
3x3+1(S)
MaxPool
3x3+2(S)
DepthConcat
Conv
1x1+1(S)
Conv
3x3+1(S)
Conv
5x5+1(S)
Conv
1x1+1(S)
Conv
1x1+1(S)
Conv
1x1+1(S)
MaxPool
3x3+1(S)
DepthConcat
Conv
1x1+1(S)
Conv
3x3+1(S)
Conv
5x5+1(S)
Conv
1x1+1(S)
Conv
1x1+1(S)
Conv
1x1+1(S)
MaxPool
3x3+1(S)
MaxPool
3x3+2(S)
LocalRespNorm
Conv
3x3+1(S)
Conv
1x1+1(V)
LocalRespNorm
MaxPool
3x3+2(S)
Conv
7x7+2(S)
input