MiniProject 2 - COMP551

Jérôme Genzling, Felipe Fontes and Sarah Al Taleb

March 10, 2023

Abstract

This project investigated the multilayer perceptron (MLP) for image classification using the CIFAR-10 dataset. Various hyper-parameters were tested, such as the number of layers, activation functions, and regularization techniques, to achieve the highest accuracy. A mini-batch stochastic gradient descent with early stopping was used to optimize training time. We also implemented CNNs and used pre-trained models from built-in libraries to compare their accuracy to the one from our algorithm. The study concluded that CNNs performs very well on classifying images as they can learn from the global image (relationships of the different pixels that form shapes etc..) whereas MLPs can only perform at best around 60% of accuracy. Indeed, all these models have to be fine-tuned and trained properly to achieve the best classification of this dataset. This can be done using transfer leaning, meaning re-using publicly available large trained models as we can reach high accuracy only changing the final layers of the model, keeping the millions of parameters already trained on large datasets.

1 Introduction

The goal of this project was to implement a multilayer perceptron (MLP) from scratch and use it to classify image data from the CIFAR-10 dataset. This multi-class classifier was first tuned regarding the different hyper-parameters used: the number of hidden layers, the activation function (ReLU, Leaky-ReLU or tanh), the regularization term as well as the learning rate and the mini-batch size of our Stochastic Gradient Descent (SGD) optimizer. We added on this latter an early-stopping process than stops fitting the data after 500 iterations without improvement on the testing set. We also explored the use of convolutional neural networks (CNNs) and pre-trained models from built-in libraries to compare their performance with our results.

Many models have used the CIFAR-10 dataset to work on multilayer perceptron models and convolutional neural network models. Nowadays, accuracies of 95% or even more are easily achievable with enough computation time and layers/number of parameters (Pandit et al. 2022, Obaid, Zeebaree, Ahmed, et al. 2020). That explain the creation of an even bigger and more challenging dataset, the CIFAR-100, following the same principle of multi-class classifier but this time with 100 different classes.

2 Data sets

The CIFAR-10 dataset used in this project is composed of 60,000 32x32 pixel colour images and one classification output. The outputs are made of 10 classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. The data set represents different images with different camera configurations with their corresponding class name. When analyzing the data distribution, we see an equal number of images in each class. There are 6,000 images in each category. This shows us that the dataset was well-balanced. During the data cleaning, we used normalization and one-hot encoding to preprocess the data. We also reshaped our data images into vectors and moved the distribution to 0.

(a) Visualization of some images and their corresponding class

3 Results

3.1 Model 1: MLP

We first started our study by implementing from scratch our MLP using a Mini-Batch SGD optimizer. This latter has an early-stopping process that stops the training of our model as soon as the best accuracy on the test set is not improved after 500 iterations. This way, we avoid the overfitting of our model on the training data. The first step was to determine the best learning parameter and batch size for our MLP. The initial weights were set following a uniform distribution (LeCun et al. 2002) where the standard deviation is

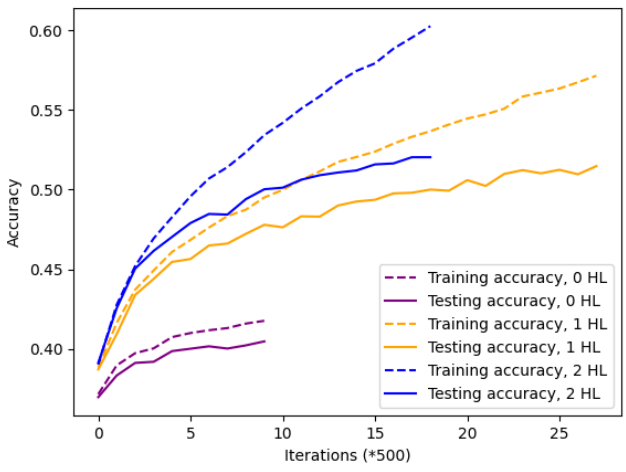
$$N^{-1/2}$$

with N the size of the incoming input of a specific layer. We implemented a full Grid Search to tune these two values and obtained the following values:

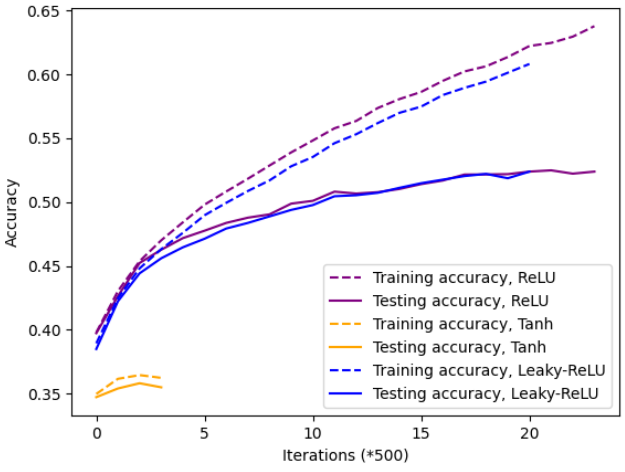
Size of Mini-Batch	Learning Rate				
	0.001	0.005	0.01	0.05	0.1
32	0.4712	0.5104	0.5065	0.5095	0.4533
64	0.4955	0.5151	0.5267	0.5152	0.5113
128	0.4859	0.5105	**0.5277**	0.5224	0.5149
256	0.4665	0.5175	0.5268	0.5145	0.5151

Table 1: Final Test Accuracy on each combination of hyper-parameters of an MLP with 2 hidden layers and ReLU Activation function

Looking at this tuning, we will keep the hyper-parameters as follows for the MLP part: a learning rate of 0.01 and a mini-batch size of 128. We then moved on to investigate the effect of the number of hidden layers on our performance. Looking at the following plot, we can directly see that as soon as we introduce non-linearity in our network with a hidden layer, our model gains a significant boost in accuracy. Indeed, we go from an initial testing accuracy of 0.4032 without any hidden layer to 0.5139 and 0.5222 adding some dense layers. This behaviour is quite expected as we do not expect to have a direct linear relationship between the values of the pixel of an image and the object that is on this image. We then understand why this non-linearity of MLP is key. Between one and two hidden layers, we see a small improvement that is also expected. As we have a second layer, we now have doubled the number of parameters and hence, increased the flexibility and so the capability of our model to distinguish the different classes of images. This of course comes with a higher risk of overfitting our training data, as we see the gap between the training and the testing accuracy getting larger.



(a) Learning curves with different hidden layers sizes



(b) Learning curves with different activation functions

The next test was then to try different activation functions for our hidden layers. We first started with ReLU but then wanted to investigate the differences using tanh and Leaky-ReLU. We then trained the same model with two hidden layers, only changing that activation function. Looking at the second graph, we see that both ReLU and Leaky-ReLU behave in the same manner and reach similar accuracy on the test set (0.5229 for ReLU and 0.5236 for Leaky-RELU). However, the tanh activation function makes the model worse as it no longer learns after 1500 iterations and is basically shut down by the early-stopping. The final accuracy of this latter is then 0.3565 on the test set, which is even worse than the linear model with no hidden layers. The similarity between ReLU and Lealy-ReLU was expected as they are really close to each other mathematically but the bad performance of the tanh function was not expected. Indeed, even after 10000 iterations, the

best testing accuracy we could get was 0.3608 so this bad performance is not due to the early-stopping. This could be due to the symmetry of the tanh function, meaning that it would be better for the model to learn from unsymmetrical activation functions. The ability of ReLU/Leaky-ReLU of killing the negative gradients might help the model to focus on the "right" pixels to determine the shape seen in the image.

We also investigated the width of our hidden layers, and the effect it has on the testing accuracy. Training the same model with only the width of our hidden layers being modified, we obtained the following testing accuracies.

2 Hidden Layers Learning Rate of 0.01	Width of the Hidden Layers				
	32	64	128	256	512
Testing Accuracy	0.4816	0.4854	0.5099	0.5228	0.5276

Table 2: Final Test Accuracy on different hidden layers width of an MLP with ReLU Activation function and Batch size of 128

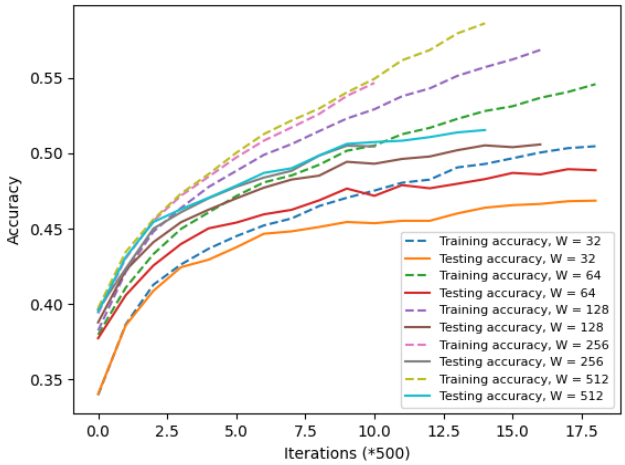


Figure 3: Learning curves with different widths

We observe that, as expected, the higher the number of our models' parameters, the more flexible it is, and so the better it can learn the data we present to it. But indeed, the accuracy increase is not that high compared to the number of parameters we are adding every time. We even see that between 256 and 512 neurons per hidden layer, the accuracy is no longer increasing that much compared to the computational cost added.

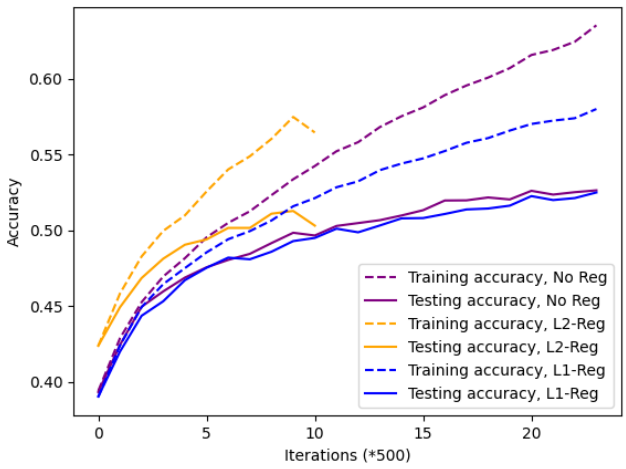
Our next test was then to introduce regularization to investigate its effect on the accuracy of our best MLP model so far (with 2 hidden layers). To do so, we compared the initial model with the ones where we added L1 and L2 regularization in the SGD. We first tuned the alpha rates of both L1 and L2 regularization to compare the best models to our reference MLP.

Regularization used	Regularization Rate			
	0.0001	0.001	0.01	0.1
L2	0.5093	0.1087	0.1	0.1
L1	0.5101	0.3714	0.2096	0.0999

Table 3: Final Test Accuracy on each combination of Regularization rate of an MLP with 2 hidden layers and ReLU Activation function, learning rate of 0.01 and Batch size of 128

We can directly observe that having a too large regularization rate may cause the gradient to be too reduced, preventing the model to learn correctly. Indeed, we need a rate small enough to allow the model to learn but not zero to correct the overfitting. Finally, we compared the results of our best models, where our reference MLP with 2 hidden layers has an accuracy of 0.5279, whereas adding L2-Regularization lowers it to 0.5087 and L1 to 0.5252. We directly see on the learning curves that the gap between the training and learning accuracy is reducing when we have regularization applied but this doesn't affect our final test accuracy. Another way of implementing regularization would have been to include dropout to prevent overfitting, even thought the early-stopping of our optimizer already help us that way.

Finally, we investigated the effect of normalizing the initial data of the CIFAR-10 dataset. Indeed, we compared the result of our MLP on normalized data versus the one on raw data where the pixels value lay between 0 and 255. The problem with using unnormalized data is that the gradient of our SGD is becoming very large quickly and then is exploding, becoming



(a) Learning curves with different regularization

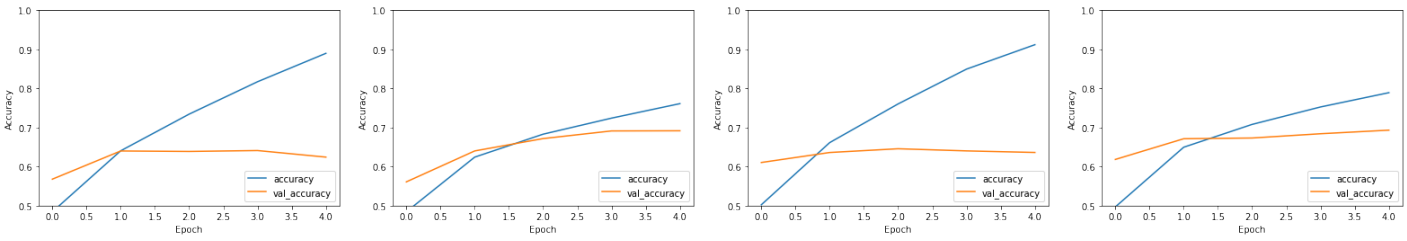
```
[7060.3202627   322.95839291  329.33459516]
[ 10120.54866658 367297.09243226 527843.35391022]
[9.00463178e+09 2.03118627e+08 2.87242433e+08]
[2.90456671e+15 1.11781657e+14 7.19278156e+14]
[8.15053127e+27 4.40226499e+27 1.38236359e+26]
[9.61661379e+50 0.00000000e+00 0.00000000e+00]
[0. 0. 0.]
[3.10339928e+23 2.87703044e+21 2.15796709e-01]
[7.99114427e+46 1.73966971e+37 1.43558319e+35]
[0. 0. 0.]
[4.66532995e+32 2.24370867e+32 2.33582416e+02]
[1.65252477e+66 1.92882590e+66 1.49785142e+42]
[6.31375872e+99 5.79422858e+98 1.34886163e+99]
[inf inf inf]
```

(b) Explosion of the gradient's norm using unnormalized data

only infinite values very quickly. To fix that, we need to reduce a lot the learning rate for that specific model. With a new learning rate of 1e-6 for that specific model, we can then compare the two models we have (with a grain of salt, since they do not share the same hyper-parameters) the best accuracy we had on the training set was 0.1317 only. This is definitely the worst model we had, showing the importance of normalizing the data. Indeed, either the gradient would blow up due to the large input values, or the model will learn really slowly since we need a small learning rate for the gradient to be computed.

3.2 Model 2: CNN

The accuracy results from the MLP were inferior to the results from CNN. The first CNN was based on a simple model with hidden layers composed of two convolutional layers with a 3x3 parameter set and two dense layers. All hidden layers used ReLU activation. For the output layer, we used softmax activation. In terms of performance after five epochs, the training accuracy was 0.9126, and the test accuracy was 0.6419. The significant difference between test and training accuracy and the extremely high training results are signs of overfitting. In order to adjust it, we decided to test two hyperparameters and compare them to the initial model. The first hyperparameter was pooling. In this case, we chose MaxPooling as our pooling function. As for the results, we saw a significant improvement in terms of the overfitting problem and the general test accuracy. The training accuracy was 0.7611 (0.1515 decrease), and the test accuracy was 0.6916 (0.0497 increase). The second hyperparameter was padding. We used padding='same' in order to maintain the output dimension similar to the input. However, the test accuracy decreased, and there were no improvements in overfitting. The test accuracy was 0.6362, and the training accuracy was 0.9120. We created one last model combining MaxPooling and padding, but there was no significant improvement. The test accuracy was 0.6933, and the training accuracy was 0.7892.

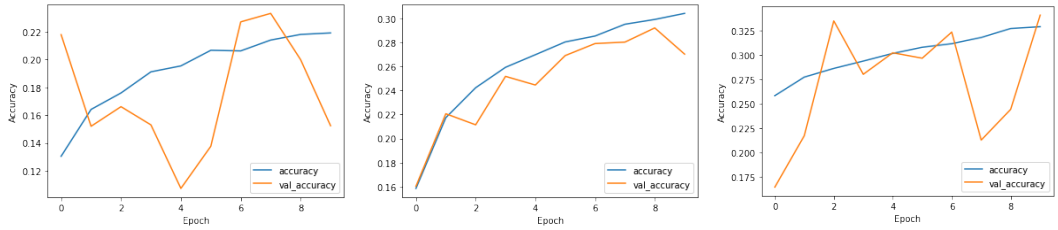


(a) Base model (b) MaxPooling (c) Padding (d) MaxPooling and Padding

3.3 Model 3: Pre-trained model

The results for the MLP were also inferior compared to the results obtained with the pre-trained model and a bit lower than the one from the CNN. We implemented the ResNet50 model on our dataset. ResNet50 is a deep convolutional neural network architecture, which contains 50 layers and is designed to classify images into 1,000 different categories. The model consists of a series of convolutional layers, followed by a global average pooling layer and a fully connected layer for classification. After freezing the convolutional layers and removing the last layer, we tried to add one dense output layer to the ResNet50 model. The best accuracy was 0.1524. We then tried adding one dense hidden layer to the model to achieve a testing accuracy of

0.2702. After that, we tried adding another dense hidden layer to the model, and we got an accuracy of 0.3407. So we can see the effect of the number of additional layers on the model, but the accuracy is still not satisfactory. It was best to add two fully connected hidden layers and one fully connected output layer. The hidden layers used ReLU activation and the output layer used softmax activation. We then tried to use batch normalization and dropout on each added dense layer in the model for regularization and to prevent overfitting along with upsample layers. The performance of the model was very slow but much better in terms of accuracy. We know that pre-trained models like ResNet and VGG have many layers, making them computationally expensive to train and evaluate. This could explain the slow training time. However, after only one epoch, we were able to get reasonable results for the loss and accuracy. For one epoch, using default hyper-parameters, the loss achieved was 1.1300 for training, with a corresponding training accuracy of 0.6224. The testing loss was 0.5674, and the testing accuracy was 0.8102.



(a) 1 output layer (b) 1 output layer + 1 hidden layer (c) 1 output layer + 2 hidden layers

4 Discussion and Conclusion

In this work, we saw that MLPs could be used to classify the images of the CIFAR-10 dataset. Playing with the different hyper-parameters, we achieve a test accuracy of around 52% which is good but not the best result overall. Indeed, MLPs are better than regular regression on the pixel values of the images as the non-linearity of hidden layers helps the models to recognize the images. However, the global shape of the image might not be recognized as the pixel are vectorized in a 1-D array, losing the global coherence of the picture. That may explain why our second choice, CNN models, performed very well. The parameter sharing and locality properties of CNN allow the model to gather information about the structure of the image, making the classification better. The test accuracy reached near 70%, a significant improvement to MLP. However, it is important to notice the extremely high training accuracy. If the correct hyper-parameters are not used, the model will overfit and impact its generalization capacity. In our test, MaxPooling was the best method to avoid that.

Regarding using existing pre-trained models such as ResNet, the high testing accuracy obtained is promising and suggests that the modified model has fitted well with the dataset. It is much higher than the one from the MLP and still better than the one obtained with the CNN. Unfortunately, the pre-trained model's training time was much greater than the one for the CNN and the MLP. We think the reason the pre-trained model took a lot of time to run even 1 epoch is due to the high quantity of layers and how large these layers are in the ResNet50 model. Also, since we increased the size of the images with the upsample layers, it is expected to take more time to fit the dataset. These factors may have influenced greatly the computation time.

We can also conclude that adding three fully connected layers (1 output layer and 2 hidden layers) with batch normalization and dropout after the base model, with upsample layers to increase the size of the input image, provided the best performance for the accuracy. These additional layers help the model to learn more complex datasets and avoid overfitting by regularization approaches such as batch normalization and dropout. Upsample layers also magnify the image sizes, which allows the large model to fit better the dataset. Therefore, we were able to get a fairly good accuracy for the pre-trained model as we were expecting it to be by adding a well thought number of fully connected layers to the ResNet model.

More experiments could be done with different pre-trained models, adding also more complex layers such as Batch Normalization, Dropout, and Pooling. Also, trying to have an even larger dataset using data augmentation could help getting better accuracies. In the end, we showed that CNNs are definitely the way to go to perform image recognition.

5 Statement of Contributions

Jérôme, Felipe, and Sarah shared the coding and writing parts equally. Jérôme took care of the MLP part, Felipe of the CNN part and Sarah of the pre-trained model part.

References

[LeC+02] Yann LeCun et al. “Efficient backprop”. In: *Neural networks: Tricks of the trade*. Springer, 2002, pp. 9–50.

[OZA+20] Kavi B Obaid, Subhi Zeebaree, Omar M Ahmed, et al. “Deep learning models based on image classification: a review”. In: *International Journal of Science and Business* 4.11 (2020), pp. 75–81.

[Pan+22] Suyesh Pandit et al. “A Review Paper on Image Classification for CIFAR-10 Dataset”. In: (2022).