ATIF, BERTRAND: TP MACHINE LEARNING

(MSE/ISMIN, 2A, 2023-2024)

Mohammed ATIF, Grégory Bertrand
mohammed.atif@etu.emse.fr, gregory.bertrand@etu.emse.fr

January 31, 2024

1 The datasets: CIFAR-3 (4)

1.1 Question 1 (1)

For this first section, we use X_cifar_grayscale to load the data. To extract the shape of the data, we use np.shape(X), and the obtained shape is (18000, 32, 32). Therefore, our data is formed by 18,000 images, each with a size of 32x32 pixels.

The min value of every pixel is 0 and the max value is 255.

The normalisation is important to continue the work on the data, since it gives a stable numeric data when our model is training, and also insures a faster convergence during training.

After performing data normalization on the X_gray array, we display the next two samples from our dataset.

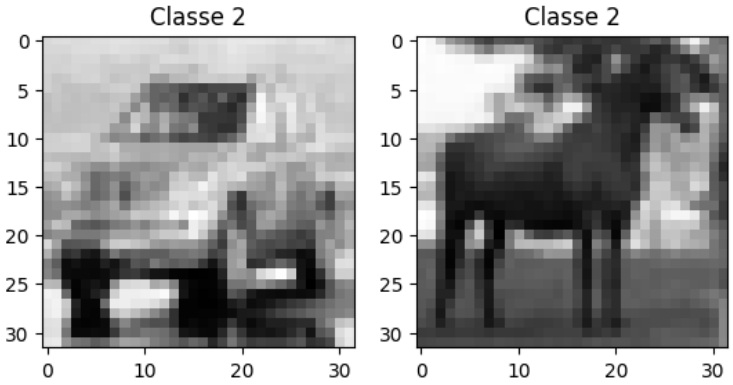

Figure 1: Samples from the dataset

1.2 Question 2 (2)

The data split is very useful for training the model. By dividing our data into testing and training sets, we can control the risk of overfitting or underfitting by adjusting hyperparameters, especially the test size. This allows us to evaluate the model's capability by generating outputs based on new data inputs 'X_test'.

For our model we have chose ,with the use of sklearn, to split our data to 30% for test data and 70% for training data

```
X_train, X_test, y_train, y_test = train_test_split(X_gray_normalized, y, test_size = 0.3, random_state = 42)
```

Figure 2: train_test_split method

1.3 Question 3 (1)

Our model is well balanced and every label has approximately the same distribution, This balance is very important for our model because if one of our classes has a larger distribution, the model will favor it and adapt its parameters based on the data from that class. This can lead to significantly inaccurate predictions.

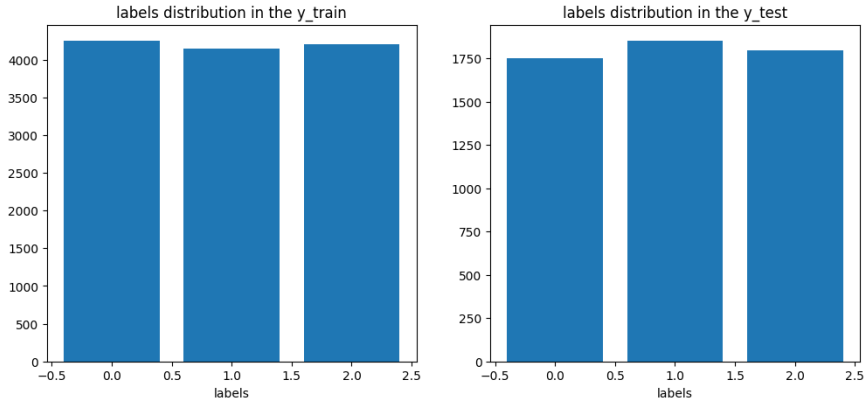

Figure 3: Distribution of labels

2 Dimensionality reduction with the PCA (4)

2.1 Question 1 (1)

Our pictures have a size of 1024 pixels. Normally, we work in $\mathbb{R}^{1024}$. To apply PCA, we are going to reduce this dimension, so we chose to perform different n_components [70, 130, 250, 700, 950] and we show the decompressed images,

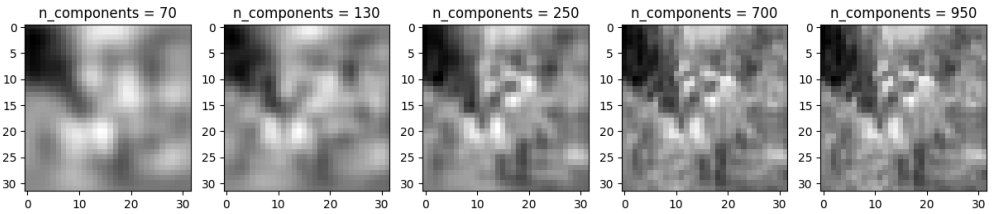

Figure 4: Picture with 5 values

2.2 Question 2 (2)

For n_components = 1024, the PCA.explained_variance_ratio_ gives us the following results.

[2.82356419e-01 1.20829347e-01 7.55195364e-02 ... 7.77817131e-07 7.00027664e-07 6.69533910e-07]

Explanation: These numbers represent the values of each principal component PC_i, where i belongs to the set $\{1, 1024\}$. These values are a linear combination of original variables and represent the proportion of the total variance for each principal component. As we observe, the values in the row decrease from 2.82356419e-01 (the value of the first principal component PC_1) to 6.69533910e-07. This implies that PC_1 contains more information about the distribution of our data. For example, our first value is 2.82356419e-01, meaning that PC_1 explains 28% of the total variance of our data.

Our goal is to retain the maximum percentage of the total variance. However, since each principal component maintains a value, we can determine a relevant value of $n\_components$ by calculating the cumulative variance in order to capture 99% of the total variance. So we had the idea to plot the cumulative variance of values obtained PCA.explained_variance_ratio_ in function of n_components.

So if we use our plot, we can se that for a total variance of 95% the best n_components is 180

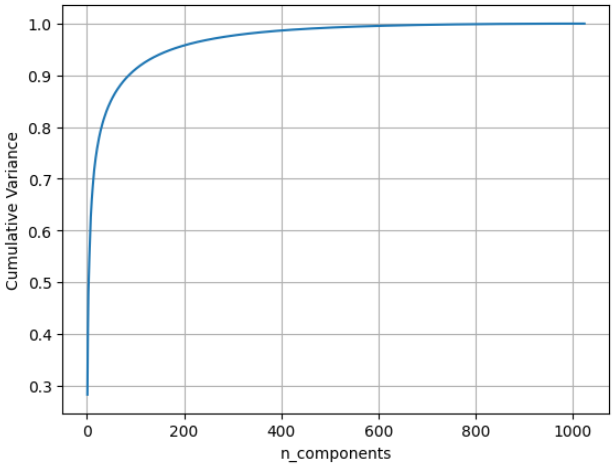

Figure 5: Cumulative variance

2.3 Question 3 (1)

We place in a liste "n_components_val" 5 values that we want to try : 1,30,70,180,300

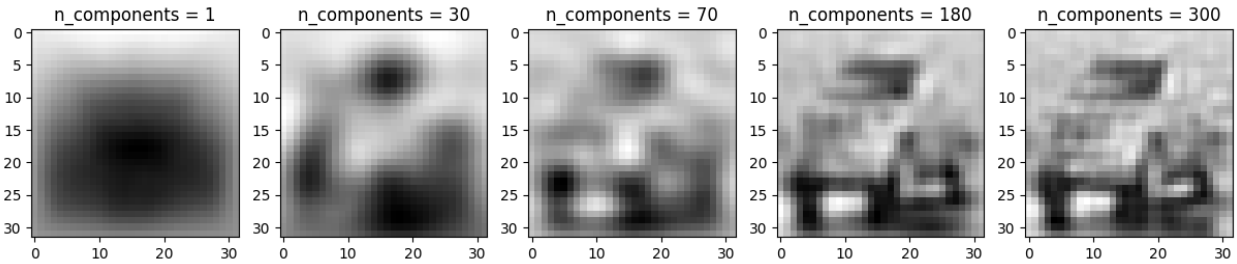

Figure 6: pictures with 5 n_components values

3 Supervised Machine Learning (28)

3.1 Logistic Regression, Gaussian Naive Bayes Classifier (6)

3.1.1 Question 1 (1)

In our case, we are trying to predict the class of each of our images. Naïve Bayes Classifier is an algorithm that calculats the probability of sample to belong to a class based on Bayes' theorem and it suppose that features are conditionally independent given the class. Logistic Regression is used for binary classification and in contrary to Naïve Bayes Classifier it models the linear relationship between features and the log-odds of the target variable.

3.1.2 Question 2 (2)

When we train our two models *LR and NBC* using test data with a size of 25%, we decide to use confusion matrix to measure the performance of our two models

After adding the parameters $C = 0.4$ and penalty $=' l2'$ for Logistic Regression (LR) and var_smoothing $= 1e - 9$ for Naive Bayes Classifier (NBC), we observe the following performance.

Confusion matrix show ass the true positive and true negative predictions, after we change the parameters by adding a strong regularization 'C=0.4' and L2 penalty the performance of LR becomes better, but with a very large value of var_smoothing the performance of NBC didn't change

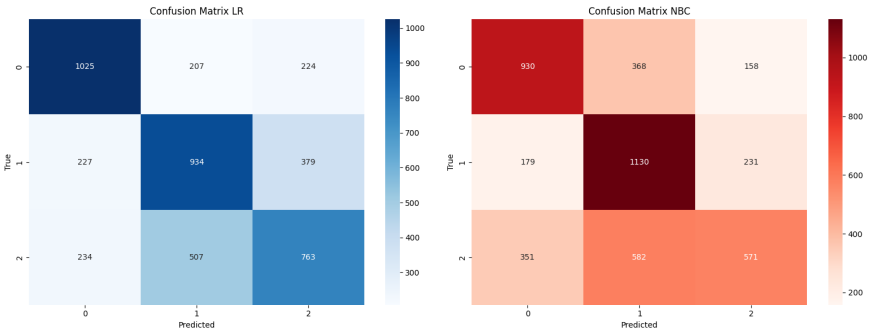

Figure 7: confusion matrix with normal parameters

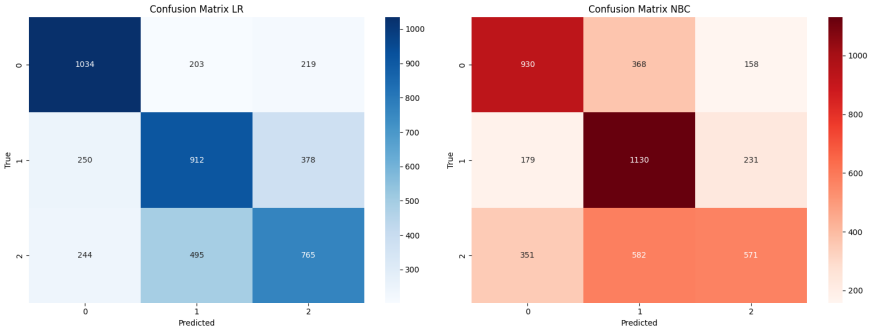

Figure 8: confusion matrix after changing parameters

3.1.3 Question 3 (1)

After computing the Logistic Regression model on the training and test datasets, we obtain the following results:

- Accuracy on the training set: 0.6421481481481481
- Accuracy on the test set: 0.6048888888888889

Similarly, for GaussianNB, we obtain:

- Accuracy on the training set: 0.5862222222222222
- Accuracy on the test set: 0.5846666666666667

Analyzing the performance of the model at training and testing time helps in selecting the best hyperparameters of the model. It also assists in detecting overfitting in cases where the accuracy on training is much higher than the accuracy on the test set. In our case, the accuracies are very close, indicating that we do not have overfitting.

3.1.4 Question 4 (2)

Using “compressed” version of CIFAR-3-GRAY we have the following values:

For LR model:

- Accuracy on the training set: 0.6256349206349207
- Accuracy on the test set: 0.6090740740740741

For NBC model:

- Accuracy on the training set: 0.6194444444444445
- Accuracy on the test set: 0.6140740740740741

Comment :

Impact of PCA on Logistic Regression Model:

After applying PCA, the accuracy has decreased compared to the accuracy without PCA. This could be attributed to the reduction in the number of informative features in the data, which was selected through the use of PCA.

Impact of PCA on Gaussian Naïve Bayes Model:

After applying PCA, the accuracy has increased compared to the accuracy without PCA. This is because the NBC assumes independence between features, and PCA can help decorrelate the principal components, creating independence between features.

3.2 Deep Learning: MLP (13)

3.2.1 Question 1 (1)

The size of the input tensor is [18000,32,32]

The size of the output layer is 3

3.2.2 Question 2 (1)

We use epochs=50, this means that the training data (X_train, y_train) will be used 50 times.

The batch_size is the number of of training examples utilised in one iteration, for example, if batch_size = 50 means that our model updates its weights after processing 50 samples.

3.2.3 Question 3 (1)

The validation set gives us the opportunity to work with an independent dataset different from the model training data. Thus, if we provide this dataset to our model, we can evaluate its performance on new input data. Additionally, we can obtain preliminary information on whether overfitting is occurring or not. It also provides us with the opportunity to control and improve our hyperparameters.

3.2.4 Question 4 (2)

1.Optimizer: determines the rules that we are going to follow to update weights during model training.

2.Dropout : help to turn off some units in our MLP during training with a probability p.

3.Number of Epochs as it is explained in question 4.1

4.Loss Function : in our case we use Categorical Crossentropy that is usually used in multi-class classification problems.

5.Regularization : In our case, regularization is applied to the layer with 128 neurons with the regularization L2, it penalises the weights with large values.

3.2.5 Question 5 (1)

We start our model with one layer of 230 neurons and the following hyperparameters.

Table 1: Hyperparameters

Hyperparameter	value
Dropout	no dropout
epochs	50
test_size	0.2
Optimizer(adam)	0.0001
Regularization	no regularization

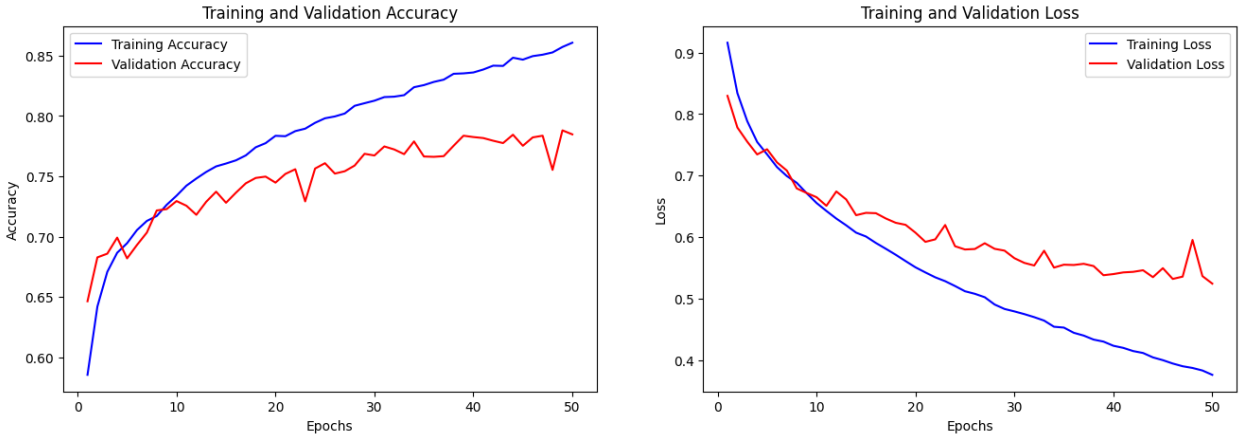

Figure 9: Evolution of the loss and the accuracy over epochs

Total params: 236443 (923.61 KB)

Trainable params: 236443 (923.61 KB)

Non-trainable params: 0 (0.00 Byte)

Train_loss : 0.3764423727989197

Val_loss : 0.5246177315711975

Train_acc : 0.8606944680213928

Val_acc : 0.7847222089767456

Our training loss is decreasing to reach approximately 0.37, and our validation loss is decreasing to the value of 0.52.

The training accuracy is increasing to 0.86, and the validation accuracy is increasing to 0.78. This is a good thing since the amount of well-predicted data is increasing.

But we observe a gap between train and val for the loss and accuracy, which means that our model is overfitting.

3.2.6 Question 6 (4)

Our model is overfitting, the causes could be the following: First we use only one layer with 230 neurons wich could be not enought, also we didn't use any dropout or any regularization algorithm, so it is possible that our model is adapted a lot to training data and can't generalize, Also we remarque a fluctuation in the both validation loss and accuracy.

How to fix the issue?

To address overfitting, we attempted to mitigate the issue by adding new hidden layers. Additionally, we incorporated L2 regularization, among other types of regularizations, to introduce a penalty term into the loss function. We employed the Adam optimization algorithm with a learning rate of 0.0001 to mitigate fluctuations and expedite convergence. Furthermore, we adjusted dropout rates to minimize the gap between validation and test accuracy/loss.

3.2.7 Question 7 (3)

In our improved model we decide to add a two hidden layers with 50 and 140 neurons in addition to the first hidden layer with 210 neurons, and we apply all the modifications mentioned before on hyperparameters.

Table 2: Hyperparameters

Hyperparameter	value
Dropout	0.5(first layer) 0.4(second layer) 0.3(Third layer)
epochs	50
test_size	0.2
Optimizer(adam)	0.0001
Regularization	L2(0.0001)
Batch_size	default value 32

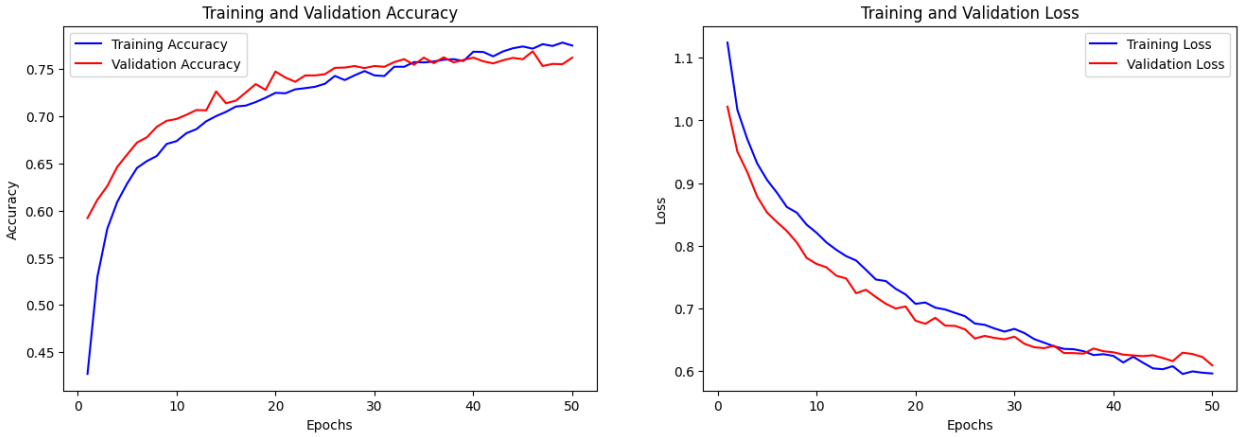

Figure 10: Evolution of the loss and the accuracy over epochs

Total params: 251993 (984.35 KB)

Trainable params: 251993 (984.35 KB)

Non-trainable params: 0 (0.00 Byte)

Train_loss : 0.5959163308143616

Val_loss : 0.6088330745697021

Train_acc : 0.7747222185134888

Val_acc : 0.7619444727897644

Using dropout and L2 regularization, we fixed the issue of overfitting, also we add two layers to add more precision to our model.

3.3 Deep Learning: CNN (11)

3.3.1 Question 1 (1)

The size of my input tensor is [18000,32,32,3], It is not as for the MLP model, because we are working now with a colored images in our dataset.

3.3.2 Question 2 (1)

In our first model, we used 3 convolutional layers. The first one has 32 filters with a size of (3,3), while the second and third ones have 64 filters each. Pooling is typically done with a local maximum of size (2,2).

The activation function is ReLU. We utilized 'valid' padding. As an optimizer, we chose Adam with a learning rate of 0.0001. The number of epochs is set to 50, and no dropout or regularizers were applied.

Table 3: Hyperparameters

Hyperparameter	value
epochs	50
test_size	0.2
Optimizer(adam)	0.0001
Regularization	no regularization

Our model has the following performances.

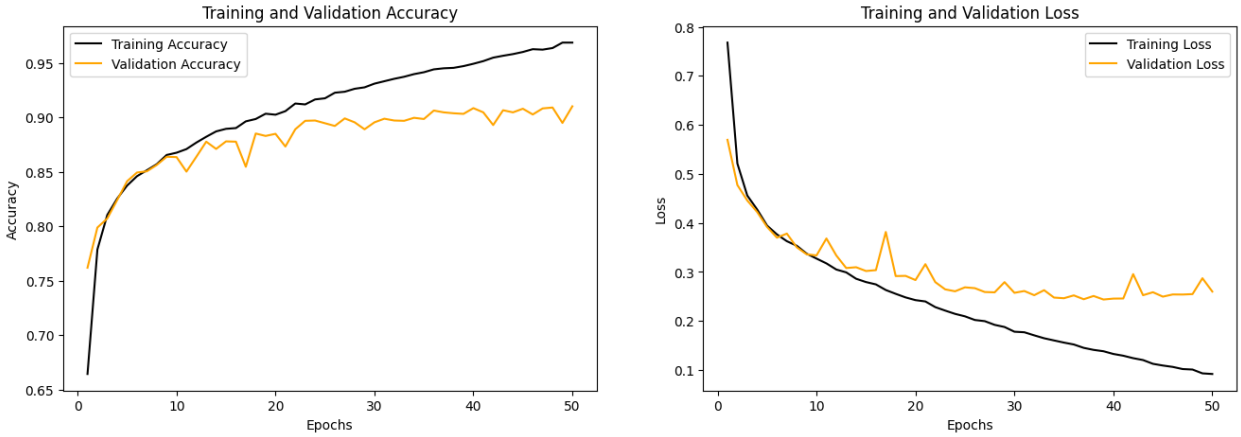

Figure 11: Evolution of the loss and the accuracy over epochs

Total params: 122115 (477.01 KB)

Trainable params: 122115 (477.01 KB)

Non-trainable params: 0 (0.00 Byte)

Our training results are:

Train_loss : 0.08233938366174698

Val_loss : 0.28575628995895386

Train_acc : 0.9728472232818604

Val_acc : 0.9072222113609314

The results of the first model are relatively good, as our training loss is very low and training accuracy is close to 100%. However, we noticed that this is not the case for the validation results, and we observe a significant gap between validation and training values. The cause of this discrepancy is that our model struggles to generalize, leading to overfitting.

3.3.3 Question 3 (4)

In our case, we are experiencing overfitting, as there is a large gap between training and validation values.

The causes of the overfitting may include using too many parameters in our CNN. Additionally, we didn't implement any regularization techniques, and the issue could be exacerbated by a high learning rate.

To address the problem, we will attempt to add dropouts and incorporate regularization techniques such as L1 or L2. We will also test our model with different learning rates. It's worth considering changing the activation functions in our CNN as well.

3.3.4 Question 4 (3)

Based on our first CNN model, we define our second CNN model by just adding the following hyperparameters.

Table 4: Hyperparameters

Hyperparameter	value
epochs	50
test_size	0.2
Optimizer(adam)	0.0001
Regularization	L2(0.01)
batch_size	120
Dropout	0.55

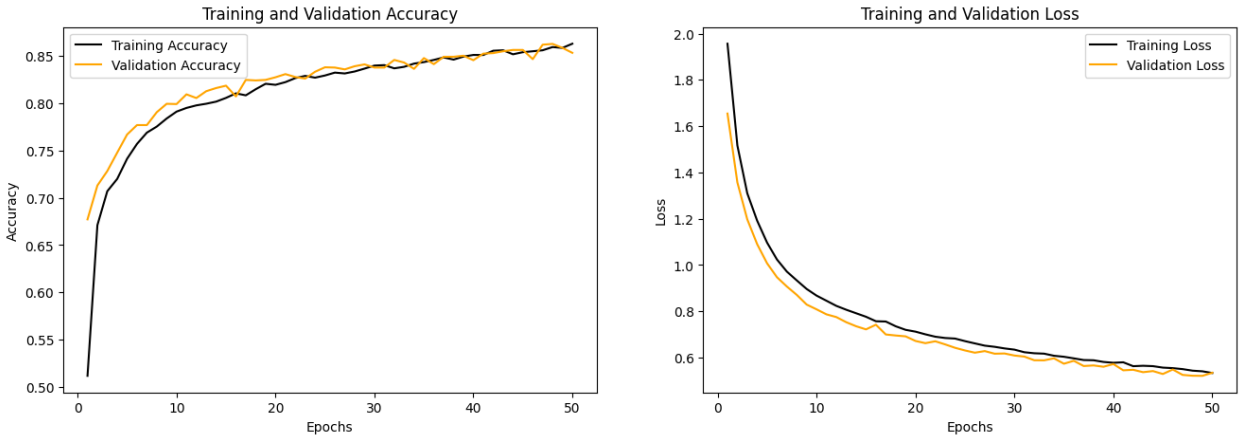

Figure 12: Evolution of the loss and the accuracy over epochs

Total params: 122115 (477.01 KB)

Trainable params: 122115 (477.01 KB)

Non-trainable params: 0 (0.00 Byte)

Train_loss : 0.5333437323570251

Val_loss : 0.5345288515090942

Train_acc : 0.8629860877990723

val_acc : 0.8533333539962769

By adding dropout and L2 regularization, we addressed the issue of overfitting. Additionally, we introduced two layers to enhance the precision of our model. Moreover, adjusting the batch size helped reduce the variability of weight updates, leading to a more stable optimization process.

3.3.5 Question 5 (2)

Considering our two models, CNN and MLP, we observe a significant difference in the performance of the two. CNN excels in image classification, and it is easier for this model to address the overfitting issue. Its use of convolutional filters makes it straightforward to extract spatially structured data. On the other hand, for MLP, it is more challenging to mitigate overfitting and achieve good generalization. It struggles, particularly with large-volume datasets and complex tasks like image classification. Additionally, it requires a larger number of parameters than CNN, which is one of the causes of overfitting.