BACKERT NOÉ, BANCHET ANTOINE : TP MACHINE LEARNING

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

Noé Backert, Antoine Banchet
noe.backert@etu.emse.fr, antoine.banchet@etu.emse.fr

July 4, 2024

1 The datasets: CIFAR-3 (4)

1.1 Question 1 (1)

What are the shape of the data? What are the min and max value of the pixels? Is it better to normalize the data to work in [0,1]? Display samples from the dataset

The shape of an image is 32 x 32 x 3. It is 32 by 32 pixels, and each pixel contain 3 values for RGB (Red Green Blue) values. For an image in Gray, it's only 32 x 32, because it's only a gray value. We have currently 18 000 images in our dataset.
Then, we have the following tensor size :
(18000, 32, 32, 3) = (nb_images, height, width, dimension(=RGB))
Min value of pixels: 0
Max value of pixels: 255
It is better for training to normalize data between [0, 1], then we're going to divide the values by 255 (there is no real visual differences when we plot the image normalized).

Figure 1: Five different images from the dataset in color

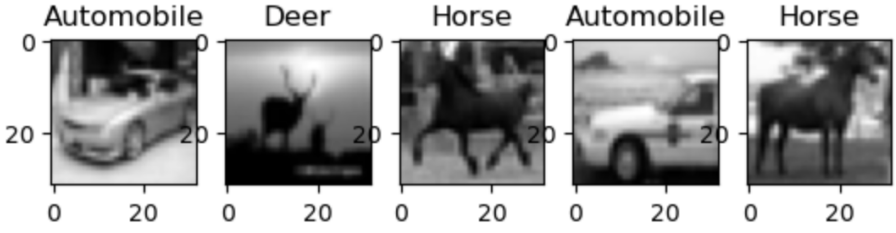

Figure 2: Five different images from the dataset in gray scale

1.2 Question 2 (2)

*Use the sklearn method **train_test_split** to split the dataset in one train set and one test set. Why this split is important in Machine Learning?*
This method randomly shuffles and divides the data into two subsets, typically with a specified ratio, where one subset is used for training the machine learning model, and the other is reserved for evaluating the model's performance.
This split is important to be able to avoid overfitting and being able to train and to test the model with enough data.

Using this function, we obtain the following shapes :

```
X_train.shape = (14400, 32, 32)        X_test.shape = (3600, 32, 32)
y_train.shape = (14400,)               y_test.shape = (3600,)
```

1.3 Question 3 (1)

Are the train and test sets well balanced (distribution of labels)? Why is it important for supervised Machine Learning?
We can find a good repartition of the labels in the sets for example :

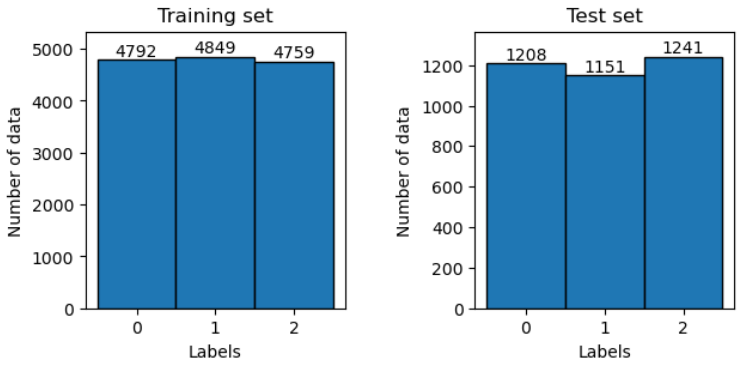

Figure 3: Data repartition between train and test datasets

Indeed, proper data distribution is crucial for supervised machine learning because our model needs to uniformly learn from all the data (the model shouldn't only know how to recognize a horse).

2 Dimensionality reduction with the PCA (4)

2.1 Question 1 (1)

Perform a Principal Component Analysis (PCA) with sklearn. You will need to reshape Try to keep different n_components.
We use the pca method **fit_transform** and numpy method **reshape** to reshape data to the good shape (18000, 1024) to have 18000 flattened images.

```
pca = PCA(n_components=k)
X_pca = pca.fit_transform(np.reshape(X_gray,(18000,32*32)))
```

We have to find an optimal n_components to decrease complexity but without losing too much informations.

2.2 Question 2 (2)

An interesting feature is PCA.explained_variance_ratio_ Explain these values according to your understanding of PCA and use these values to fit a relevant value for n_components.
The PCA.explained_variance_ration provides the proportion of the dataset's variance that is captured by each principal component.
The main vector of analysis is the cumulative explained variance ratio. This value gives us an estimate of how influential and important the values are. We can see an explanation of how this works on : jakevdp - In Depth:

Principal Component Analysis.
By plotting the cumulative explained variance, we can observe the point at which adding more components does not significantly increase the captured variance.

We choose to take the value where the cumulative variance is greater than 95%.

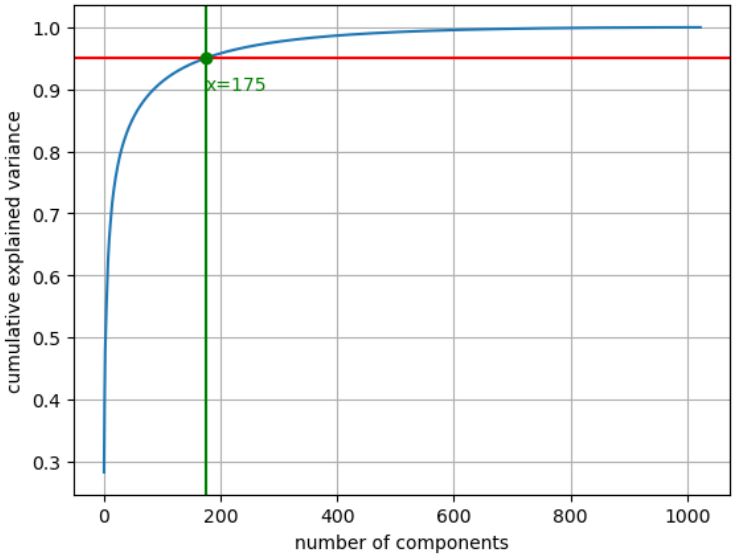

Figure 4: Cumulative explained variance to find the number of Components

Looking at the following graph, we see that with a value of 175, we would still have 95% of the cumulative variance.

2.3 Question 3 (1)

Display a CIFAR-3-GRAY picture with 5 values of n_components Here's the translation in English:

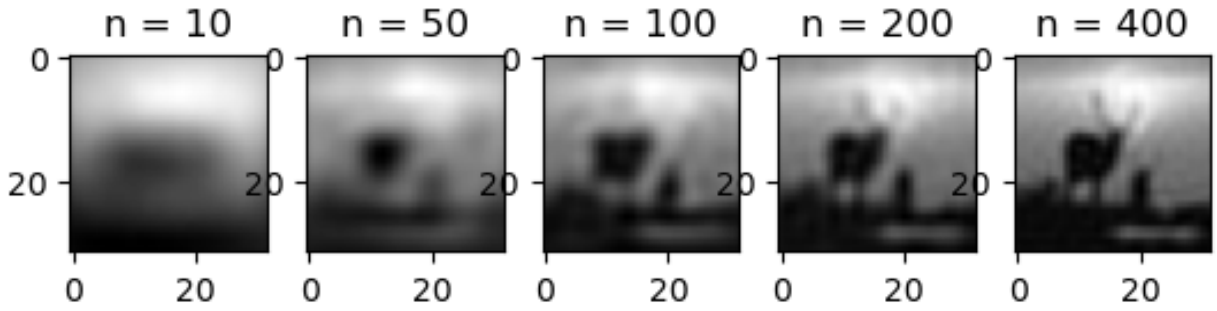

Figure 5: Five differents values of n_components

It is clear that when n=100, we do not recognize all the deer's information very well, whereas at 200, it is sufficient. The value of 175 appears to be consistent.

3 Supervised Machine Learning (28)

3.1 Logistic Regression, Gaussian Naive Bayes Classifier (6)

3.1.1 Question 1 (1)

What is the major difference between Naïve Bayes Classifier and Logistic Regression?
The major difference between Naïve Bayes Classifier and Logistic Regression lies in the way we try to guess the output.

- Naïve Bayes Classifier : The goal is to estimate $p(x|y)$ to deduce $p(y|x)$ with Bayes theorem. We are then learning the probability distribution of the data.
- Logistic Regression : The goal is to directly estimate $p(y|x)$. We are learning a decision boundary to separate classes based on the learned coefficients.

3.1.2 Question 2 (2)

With sklearn, train your LR and NBC models. Comment the results. If any, check how to modify the parameters and comment how it influences the results.

With no parameters, we have the following results :

```
Logistic Regression's score : 0.5977777777777777
GaussianNB's score : 0.5791666666666667
```

$\rightarrow$ Logistic Regression works well when the data is linearly separable. The accuracy score is moderately performant on your dataset. However, the score isn't exceptionally high, indicating that the data might not be well-suited for linear separation.

$\rightarrow$ GaussianNB tries to approximate $p(x|y)$ to deduce $p(y|x)$. To do so, the model assumes that the dataset follows a Gaussian distribution, but this is not always the case. The accuracy score obtained indicates that it is also performing moderately well, but slightly lower than Logistic Regression in this case.

1. The regularization parameter C for LR controls the regularization. Smaller C values indicate stronger regularization for overfitting.

2. The 'var_smoothing' parameter for NBC is used to avoid issues with zero variances in Gaussian Naïve Bayes.

3. The solver parameter for LR is the optimization algorithm. We can find ''lbfgs'', ''liblinear'', ''newton-cg'', ''sag'', and ''saga''.

var_smooth	Gaussian NB score	PCA NB score
1×10^{-1}	0.5486	0.3608
1×10^{-2}	0.5797	0.4031
1×10^{-3}	0.5825	0.5678
1×10^{-4}	0.5831	0.6119
1×10^{-5}	0.5831	0.6119
1×10^{-6}	0.5831	0.6119
1×10^{-9}	0.5831	0.6119

Table 1: Table 1: var_smooth influence

C	LR score	PCA LR score
1×10^{-1}	0.6019	0.6056
1×10^{-2}	0.6128	0.6058
1×10^{-3}	0.6100	0.6056
1×10^{-4}	0.5814	0.6056
1×10^{-5}	0.5297	0.6056
1×10^{-6}	0.4769	0.6053
1×10^{-9}	0.3247	0.6078

Table 2: Table 2: C influence

3.1.3 Question 3 (1)

With the score method, compute the accuracy of the model on the training and the test datasets. Why do we need to analyze the performance of the model at training and testing time?

Analyzing at both training and testing times is important because it helps detect issues like overfitting. By comparing training and testing performance, we are able to modify the model parameters to increase overall performance and mitigate overfitting.

3.1.4 Question 4 (2)

Same with a "compressed" version of CIFAR-3-GRAY with your PCA. Comment.

We can observe that with PCA, score are higher but not always optimal.

```
Logistic Regression's PCA score : 0.6013888888888889
GaussianNB's PCA score : 0.6127777777777778
```

3.2 Deep Learning: MLP (13)

3.2.1 Question 1 (1)

What is the size of the input tensor? What is the size of the output layer?

The size of the input tensor is a one-dimensional vector of 1024 because the image is 32x32, and it needs to be flattened into a vector. The size of the output tensor is 3 because the MLP is the classifier for the 3 classes (Automobile, Deer, and Horse) with a softmax function as the activation function to compute probabilities.

3.2.2 Question 2 (1)

How many epochs do you use? What does it mean? What is the batch_size? What does it means?

We use 30 epochs. An epoch is the number of times we iterate over the entire training dataset. A batch is a subset of the training data used during one iteration of training. Instead of updating the weights after processing the entire dataset or individual examples, the model is updated after processing a batch. In this case, the model is updated after processing 32 selected training pictures. This process is repeated until the entire training dataset has been processed, completing one epoch.

3.2.3 Question 3 (1)

Why do we need to define a validation set?

The validation set is used to obtain a performance measure after each epoch. It helps in reducing overfitting and is great to configure the model. It helps to understand the model. If the performance on the validation set starts to degrade while the performance on the training set improve, then it's that the model is overfitting.

3.2.4 Question 4 (2)

Pick the most important hyper-parameters you have to set to run the training process (e.g., optimizer...). Briefly explain why are they important (i.e. their influence).

The following hyper-parameters are very important:

- **Numbers of layers and units:** More we have layers and units more the capacity of the model is important. But if the model is too powerful it can leads to overfitting.
- **Number of Epochs:** Too few epochs may result in underfitting, but too many epochs may lead to overfitting.
- **Activation function:** Relu because it's avoiding vanishing gradient problem.
- **Optimizer:** The optimizer updates the learning rate during training. For example, Adam is known for adaptive learning rates, starting with a high learning rate that then decreases over time.
- **Loss:** Cross entropy loss because it's avoiding vanishing gradient problem.
- **Metrics:** Method of evaluating the model's performance, here accuracy, which represents the proportion of correctly classified instances out of the total instances.
- **Batch Size:** Larger batch sizes provide a smoother backward gradient propagation but may require more memory.

3.2.5 Question 5 (1)

Comment the training results.

```
 Layer (type)                Output Shape              Param #
=================================================================
 dense_4 (Dense)             (None, 123)               126075

 dense_5 (Dense)             (None, 3)                 372
=================================================================
Total params: 126447 (493.93 KB)
```

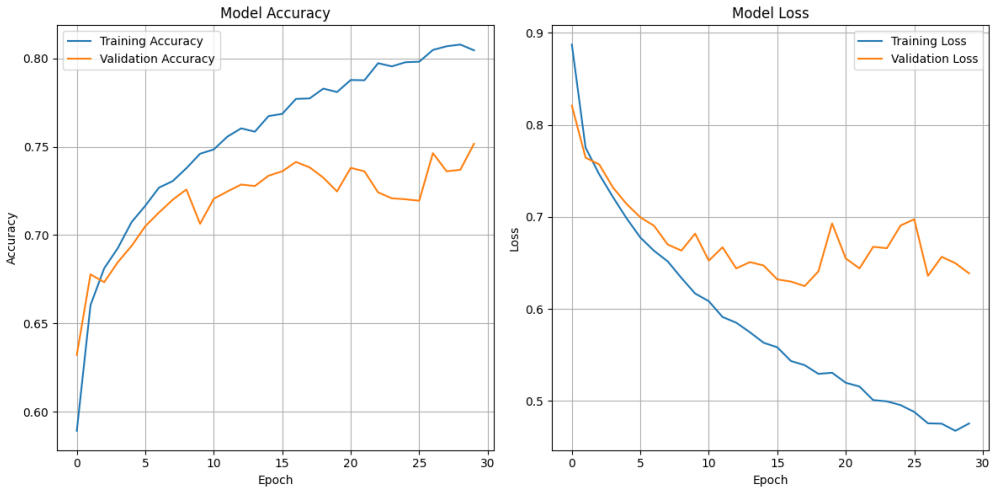

Figure 6: MLP result

First, we chose a very light model with only one hidden layer of 123 neurons.
If we only look at the training curve, we could think that the results seems good. Indeed, we're reaching a 80% accuracy and less that 0.5 loss. However, the validation score isn't that good which means that the model is overfitting from the epoch 10.

3.2.6 Question 6 (4)

Is there any overfitting? Why? If yes, what could be the causes? How to fix this issue? If you do not observe overfitting, how can you make your model overfit? Try and demonstrate the overfitting

Yes, as said at question 5, we can clearly see that the model overfits, because the difference between the training accuracy and the validation accuracy grow during the epochs. The possible causes are :

- Too many training epochs
- Model capacity
- Imbalanced of insufficient data

Here the data balance is good, and we have enough data. The training epochs seems good, but we're having a really small model. To improve the model's performance, we can **add dropout** layouts and **add dense layers** to achieve improving the model's complexity.

3.2.7 Question 7 (3)

According to this first performance, change the architecture of the MLP (change parameters, add/remove layers...) as well as hyper-parameters, explain why, what are the influence on the results... ?

```
 Layer (type)                Output Shape              Param #
=================================================================
 dense_6 (Dense)             (None, 123)               126075

 dropout_2 (Dropout)         (None, 123)               0

 dense_7 (Dense)             (None, 150)               18600

 dropout_3 (Dropout)         (None, 150)               0

 dense_8 (Dense)             (None, 3)                 453
=================================================================
Total params: 145128 (566.91 KB)
```

To improve the model, we add dropout layers, and add a dense layer to enhance model's complexity, we have the following result :

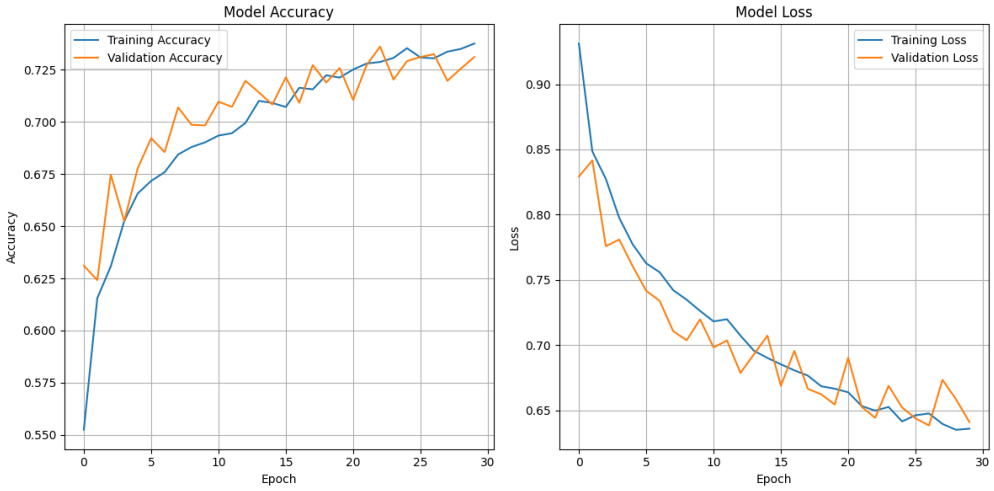

Figure 7: model 2 result

We can clearly see the difference, and it seems to have worked since the validations accuracy follows the training accuracy.

3.3 Deep Learning: CNN (11)

3.3.1 Question 1 (1)

What is the size of the input tensor? Why it is not the same as for your previous MLP model ?
The input tensor is a 32 x 32 x 3 tensor. This time, we have color images, and a CNN is able to take the image directly as input. We don't need to flatten the vector, contrary to an MLP. CNNs are more adapted for image classification.

3.3.2 Question 2 (1)

Comment the training results.
We start with the following model:

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 conv2d_3 (Conv2D)            (None, 30, 30, 32)        896

 max_pooling2d_2 (MaxPooling  (None, 15, 15, 32)       0
 2D)

 conv2d_4 (Conv2D)            (None, 13, 13, 64)        18496

 max_pooling2d_3 (MaxPooling  (None, 6, 6, 64)         0
 2D)

 conv2d_5 (Conv2D)            (None, 4, 4, 64)          36928

 flatten_1 (Flatten)          (None, 1024)              0

 dense_11 (Dense)             (None, 64)                65600

 dense_12 (Dense)             (None, 3)                 195

=================================================================
Total params: 122,115
Trainable params: 122,115
Non-trainable params: 0
_________________________________________________________________
```

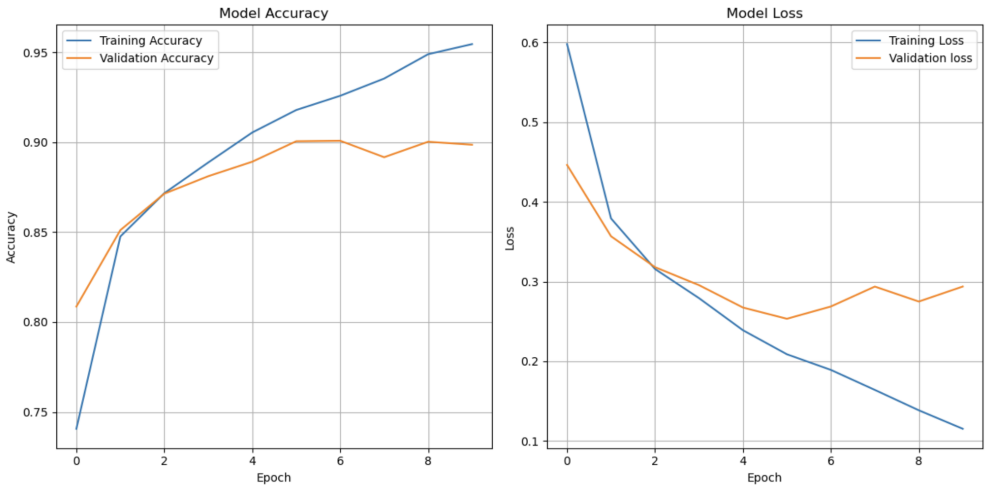

Figure 8: First CNN result

We have chosen a CNN with 3 convolutional layers. The first layer has 32 filters of size 3x3, while the second and third layers each have 64 filters of size 3x3. After the first and second convolutional layers, we add max pooling with a 2x2 window to achieve translation invariance. The activation function for all layers is ReLU.

The model is trained for 10 epochs using cross-entropy loss. However, the model is overfitting.

3.3.3 Question 3 (4)

Is there any overfitting? Why? If yes, what could be the causes? How to fix this issue? If you do not observe overfitting, how can you make your model overfit? Try and demonstrate the overfitting.

There is overfitting; the model is memorizing the data, possibly because there is no regularization, and dropout of neurons.

So, to address overfitting, we need to regularize the model by incorporating spatial dropout. Additionally, we apply the softmax activation function to the last layer to obtain probabilities.

3.3.4 Question 4 (3)

According to this first performance, change the architecture of the CNN (add/remove kernels, add/remove layers...) as well as hyper-parameters, explain why, what are the influence on the results... ?

Layer (type)	Output Shape	Param #
conv2d_9 (Conv2D)	(None, 30, 30, 32)	896
max_pooling2d_6 (MaxPooling 2D)	(None, 15, 15, 32)	0
spatial_dropout2d (SpatialD ropout2D)	(None, 15, 15, 32)	0
conv2d_10 (Conv2D)	(None, 13, 13, 64)	18496
max_pooling2d_7 (MaxPooling 2D)	(None, 6, 6, 64)	0
spatial_dropout2d_1 (Spatia lDropout2D)	(None, 6, 6, 64)	0
conv2d_11 (Conv2D)	(None, 4, 4, 64)	36928
max_pooling2d_8 (MaxPooling 2D)	(None, 2, 2, 64)	0
spatial_dropout2d_2 (Spatia lDropout2D)	(None, 2, 2, 64)	0
flatten_3 (Flatten)	(None, 256)	0
dense_15 (Dense)	(None, 64)	16448
dense_16 (Dense)	(None, 3)	195

Figure 9: CNN Model

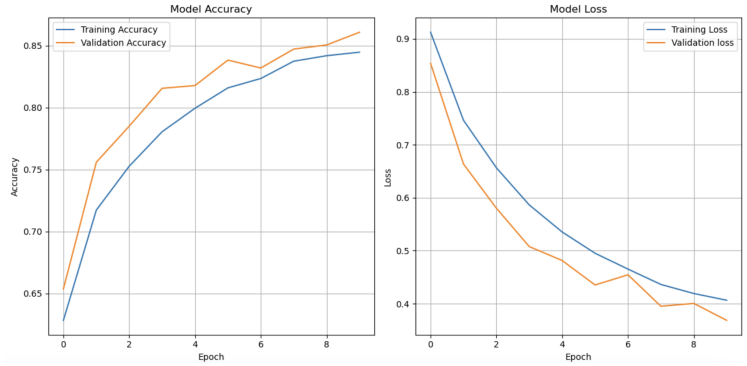

Figure 10: CNN results

We can observe that the issue of overfitting has been resolved with the implementation of spatial dropout.

```
test_loss, test_acc = model.evaluate(X_test_normalized_, y_test_, verbose=2)
```

```
113/113 - 1s - loss: 0.3685 - accuracy: 0.8608 - 577ms/epoch - 5ms/step
```

Performance is excellent, but could it be improved without overfitting ? We tried with 40 epochs.

```
113/113 - 1s - loss: 0.2755 - accuracy: 0.8931 - 565ms/epoch - 5ms/step
```

The validation loss at the end 0.8931 is a good result.

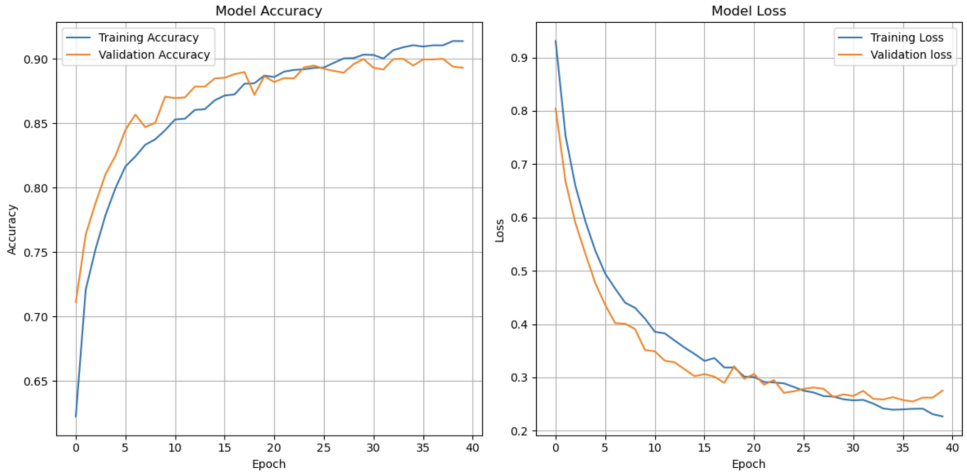

Figure 11: CNN results with 40 epochs

3.3.5 Question 5 (2)

Final comment on the MLP/CNN comparison.

The MLP performs better than NBC or LR with a validation accuracy > 0.7. It requires a flattened vector, and in our case, we use it exclusively with grayscale images because they are easier to learn. MLP can have a significant number of parameters, and the model tends to overfit rapidly without dropout.

On the other hand, CNN is specifically designed for image classification, achieving a validation accuracy > 0.8. It involves fewer weights to train for the same performance, and it allows for the use of color images. Therefore, for image classification in our case, CNN is the optimal choice.