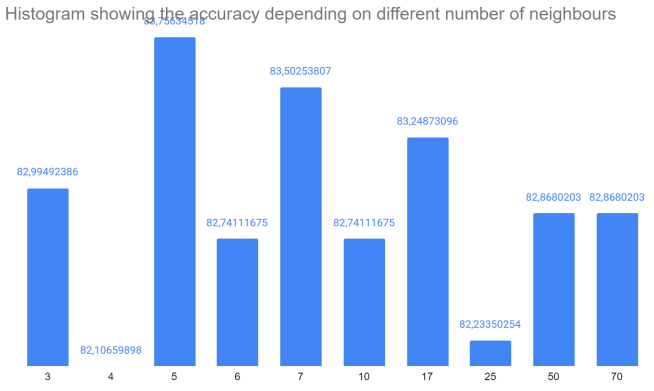
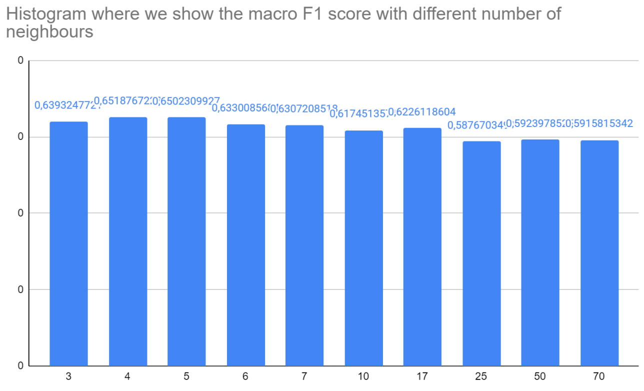
Milestone 2 Project Report : PCA, kNN, PyTorch neural network for classification (FMA Dataset)

1. k-Nearest Neighbors :

We implemented the kNN technique using the same strategy that we saw in class and in the exercise session. Then, for finding the best number of neighbors, we performed cross validation on kNN and got the following histograms.





Hence, after comparing our values, we seem to get the best trade-off in the number of neighbors being 5 with an accuracy equal to 83,75634518 and the macro f1 score to 0,6502309927.

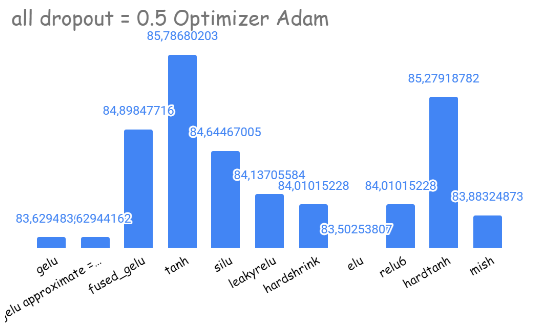
2. PCA

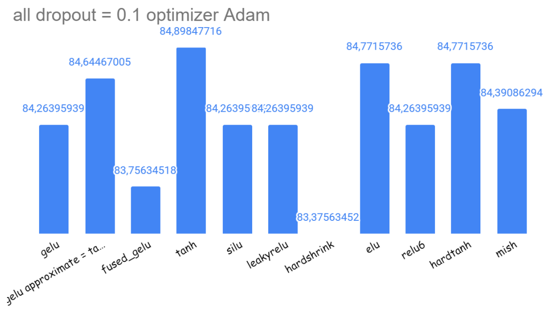
We implemented PCA the same way it was delivered to us in class.
First we tested PCA for kNN. We used the best “k” parameter to test the training time and performance, which is 5. The performance was the same, but the training time was not. Without PCA, it would approximately take us 50 seconds to perform kNN, while with PCA we would get 45 seconds.
Then, we tested PCA for linear regression. First, the training times were differing, as without PCA we would get 30 ms, and with PCA 69 ms. However the two training times are invisible to the human eye. Secondly, the performance was better with PCA (0.439) than without PCA (0.442).
Finally, we tested PCA for logistic regression. The results were similar to linear regression, first with the training time with PCA (12.953 seconds) being smaller than without PCA (15.089 seconds), and with the performance without PCA (accuracy equal to 80.20% and macro F1 score equal to 0.67) being slightly better than with PCA (accuracy equal to 79.95% and macro F1 score equal to 0.67).

3. Deep Network

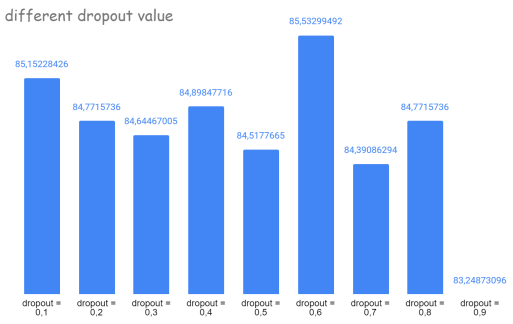
Because our model is a classification, we needed to use the softmax function and argmax to predict a result_class. In order to have the best accuracy, we tried to find the best architecture. In order to do this we tried several activation functions, optimizers, and ways to prevent overfitting.

The conditions of our test were : CrossEntropyLoss(), 3 layers, early stopping by setting a variable “patience” in order to prevent overfitting and a dropout.
We are not using weight decay since we are using 2 regularizations in order to prevent overfitting: dropout and early stopping. We are trying to see what is the best activation function for our layers (in the case we use the same for all our layers)

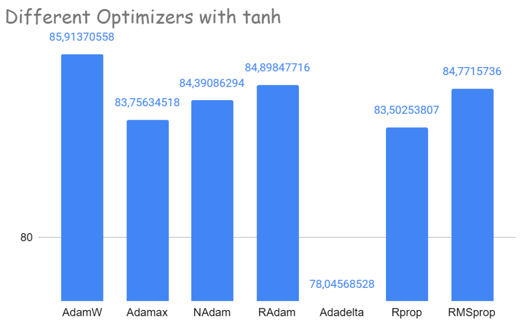




We will use tanh for now on for our dataset, let's try several dropout values



dropout = 0.6 seems the best, we then try different optimizers



AdamW optimizer seems to be the best for our model. So, our model uses 3 layers, with tanh as an activation function, dropout with p = 0.6 and AdamW was our optimizer. Also, we tried to add more layers to our model but the accuracy was decreasing, so we decided to have only 3 layers.

PS : Since we are doing dropout those accuracies are susceptible to change
For our task we prefer to use our model of deep Network because it gives the best accuracy over all the other models we coded, and thanks to our early stopping feature, it takes a little time to compute the result class (in only 21 seconds.)