Name: Nadu Toma
Group: 241

Brain Anomaly Detection

Project report

1. Project description

This project was assigned as part of the Machine Learning course of the University of Bucharest, with the goal of training a model to correctly classify grayscale images depicting brain CT scans as either normal or abnormal. The project was organized around a Kaggle competition, with the mean F1 score being the metric based on which all submissions would be ranked. Training, validation, test datasets and their corresponding labels were all provided, except for the test labels, since the final scoring would be performed assuming no other information other than the images themselves were given beforehand.

2. Preprocessing and Data Augmentation

2.1. Loading the data

Using the *pandas* library, the labels associated with each image were loaded, while the most efficient library I found for reading the images from the disk and into memory was *opencv*. As a result, significant memory was freed up compared to *pillow* or other alternatives. After all the images are fully loaded into a python array, the whole data structure is converted to a *numpy* array, which is used by all the other libraries with machine learning model implementations that I chose. Most of the time, the provided samples are downsized from 224x224 pixels to 120x120 pixels, so that the scripts could be executed locally.

2.2. Data splitting

The validation set contains a lot more abnormal samples than could be found in the training set, which I thought would skew the dataset and it is the reason for which I used the *train_test_split* function from *sklearn* on the datasets which some of the models trained on.

2.3. Augmentation

Since the abnormal scan samples were so much fewer than the normal scan samples in both the training and validation datasets, I decided to try and augment the abnormal

images. Browsing through the images, I noticed that the exposure and scale were very consistent, which is why I only tried to rotate the chosen set slightly, around the center (the number of degrees for each image was generated randomly from the interval [-10, 10]). This method worked better on some models than on others.

3. Machine Learning algorithms

For this project I experimented with a wide variety of models, fine-tuning their hyperparameters as I saw fit: Naive Bayes, K-Nearest Neighbors, Support Vector Machines and Convolutional Neural Networks. All of the confusion matrices used in this report follow the convention of the *i*th row corresponding to the real label *i* and the *jth* column corresponding to the predicted label *j*.

3.1. Naive Bayes

In terms of data preprocessing, the Naive Bayes algorithm can work better or worse, depending upon the number of values that each parameter can take. This translates to fitting the whole 0-255 pixel space into various 'bins', by picking the closest 'bin' value to the right of a given number. I experimented with different *numberOfBins* values and all augmentation and dataset splitting options available, as follows:

No data augmentation, default train-validation dataset split.

- Number of bins = 2, Accuracy = 86.25%, Confusion matrix:

1724	0
275	1

- Number of bins = 4, Accuracy = 86.7%, Confusion matrix:

1621	103
163	113

- Number of bins = 6, Accuracy = 60%, Confusion matrix:

994	730
70	206

- Number of bins = 8, Accuracy = 61.55%, Confusion matrix:

1028	696
73	203

- Number of bins = 10, Accuracy = 65.45%, Confusion matrix:

1118	606
85	191

Data augmentation, default train-validation dataset split.

- Number of bins = 2, Accuracy = 13.8%, Confusion matrix:

0	1724
0	276

- Number of bins = 4, Accuracy = 21.9%, Confusion matrix:

193	1531
31	245

- Number of bins = 6, Accuracy = 58.7%, Confusion matrix:

1724	0
275	1

- Number of bins = 8, Accuracy = 60.5%, Confusion matrix:

1005	719
71	205

No data augmentation, random train-validation dataset split.

- Number of bins = 2, Accuracy = 86.25%, Confusion matrix:

1724	0
275	1

- Number of bins = 4, Accuracy = 86.7%, Confusion matrix:

1621	103
163	113

- Number of bins = 6, Accuracy = 60%, Confusion matrix:

994	730
70	206

- Number of bins = 8, Accuracy = 61.55%, Confusion matrix:

1028	696
73	203

Data augmentation, random train-validation dataset split.

- Number of bins = 2, Accuracy = 13.8%, Confusion matrix:

0	1724
0	276

- Number of bins = 4, Accuracy = 21.9%, Confusion matrix:

193	1531
31	245

- Number of bins = 6, Accuracy = 58.65%, Confusion matrix:

967	757
70	206

- Number of bins = 8, Accuracy = 60.5%, Confusion matrix:

1005	719
71	205

The variant with the highest score on the platform turned out to be the one having had 4 bins applied on the dataset, without any augmentation or random splitting. With this model I achieved my final highest score in the competition.

- Number of bins = 4, Accuracy = 86.7%, Precision (normal class) = 0.9, Recall (normal class) = 0.94, Precision (abnormal class) = 0.52, Recall (abnormal class) = 0.4, Confusion matrix:

1621	103
163	113

3.2. K-Nearest Neighbors

As with the previous model, I tried many combinations of K values, data augmentation and dataset splits. Data preprocessing was conducted using normalization with either the *L1* or *L2* distances and the former yielded much better results. Below is the outcome based on the variables outlined so far.

No data augmentation, default train-validation dataset split, *L1*.

- K = 1, Accuracy = 84.1%, Confusion matrix:

1597	127
191	85

- K = 3, Accuracy = 85.4%, Confusion matrix:

1664	60
231	45

- K = 5, Accuracy = 86.4%, Confusion matrix:

1688	36
235	41

- K = 7, Accuracy = 86.6%, Confusion matrix:

1703	21
247	29

- K = 9, Accuracy = 86.7%, Confusion matrix:

1712	12
251	25

Data augmentation, default train-validation dataset split, *L1*

- K = 1, Accuracy = 80.4%, Confusion matrix:

1475	249
142	134

- K = 3, Accuracy = 76.1%, Confusion matrix:

1353	371
107	169

- K = 5, Accuracy = 73.5%, Confusion matrix:

1278	446
84	192

- K = 7, Accuracy =72.2%, Confusion matrix:

1244	480
75	201

- K = 9, Accuracy = 70.8%, Confusion matrix:

1215	509
74	202

No data augmentation, random train-validation dataset split, *L1*.

- K = 1, Accuracy = 83.2%, Confusion matrix:

2677	207
363	153

- K = 3, Accuracy = 84.5%, Confusion matrix:

2776	108
416	100

- K = 5, Accuracy = 85%, Confusion matrix:

2810	74
436	80

- K = 7, Accuracy = 85%, Confusion matrix:

2838	46
463	53

- K = 9, Accuracy = 85.1%, Confusion matrix:

2850	33
472	44

Data augmentation, random train-validation dataset split, *L1*.

- K = 1, Accuracy = 79.2%, Confusion matrix:

2470	430
275	225

- K = 3, Accuracy = 75.8%, Confusion matrix:

2286	614
208	292

- K = 5, Accuracy = 72.5%, Confusion matrix:

2151	749
186	314

- K = 7, Accuracy = 71.4%, Confusion matrix:

2103	797
174	326

Data augmentation, default train-validation dataset split, *L2*

- K = 1, Accuracy = 72.9%, Confusion matrix:

1316	408
134	142

- K = 3, Accuracy = 66.6%, Confusion matrix:

1153	571
97	179

- K = 5, Accuracy = 62.6%, Confusion matrix:

1061	663
84	192

- K = 7, Accuracy = 61.2%, Confusion matrix:

1027	697
78	198

- K = 9, Accuracy = 60%, Confusion matrix:

991	733
67	209

Unfortunately, data augmentation and the dataset splitting didn't seem to be helping much, as the best performing model has K=9 without applying said techniques.

- K = 9, Accuracy = 86.7%, Precision (normal class) = 0.87, Recall (normal class) = 0.99, Precision (abnormal class) = 0.67, Recall (abnormal class) = 0.09, Confusion matrix:

1712	12
251	25

3.3. Support Vector Machines

The approach used with this model was quite similar to the previous two, in the sense that multiple experiments were conducted using different values for the hyperparameter *C* ('the penalty parameter'), both the *L1* and *L2* distances in the normalization process, data augmentation and training-validation dataset splitting, as well as various resolutions of the images.

No data augmentation, default train-validation dataset split, *L1*.

- C = 0.0625, Accuracy = 62.7%, Confusion matrix:

1051	673
73	203

- C = 0.125, Accuracy = 63.8%, Confusion matrix:

1077	647
76	200

- C = 0.25, Accuracy = 65.1%, Confusion matrix:

1107	617
81	195

- C = 0.5, Accuracy = 67.5%, Confusion matrix:

1162	562
88	188

- C = 1, Accuracy = 69.9%, Confusion matrix:

1223	501
101	175

- C = 2, Accuracy = 71.4%, Confusion matrix:

1266	458
113	163

- C = 4, Accuracy = 72.8%, Confusion matrix:

1307	417
126	150

Data augmentation, random train-validation dataset split, *L1*.

- C = 0.0625, Accuracy = 64.7%, Confusion matrix:

1829	1095
105	371

- C = 0.125, Accuracy = 65.1%, Confusion matrix:

1859	1065
119	357

- C = 0.25, Accuracy = 65.8%, Confusion matrix:

1886	1038
123	353

- C = 0.5, Accuracy = 66.3%, Confusion matrix:

1920	1004
139	337

- C = 1, Accuracy = 66.6%, Confusion matrix:

1939	985
148	328

- C = 2, Accuracy = 66.9%, Confusion matrix:

1962	962
161	315

- C = 4, Accuracy = 66.7%, Confusion matrix:

1973	951
178	298

No data augmentation, default train-validation dataset split, *L2*.

- C = 0.0625, Accuracy = 62.7%, Confusion matrix:

1051	673
73	203

- C = 0.125, Accuracy = 63.8%, Confusion matrix:

1077	647
76	200

- C = 0.25, Accuracy = 65.1%, Confusion matrix:

1107	617
81	195

- C = 0.5, Accuracy = 67.5%, Confusion matrix:

1162	562
88	188

- C = 1, Accuracy = 69.9%, Confusion matrix:

1223	501
101	175

- C = 2, Accuracy = 71.4%, Confusion matrix:

1266	458
113	163

- C = 4, Accuracy = 72.8%, Confusion matrix:

1307	417
126	150

No data augmentation, default train-validation dataset split, *L2,* 60x60 resolution.

- C = 0.0625, Accuracy = 62.8%, Confusion matrix:

1050	674
69	207

- C = 0.125, Accuracy = 63.7%, Confusion matrix:

1076	648
77	199

- C = 0.25, Accuracy = 64.9%, Confusion matrix:

1104	620
82	194

- C = 0.5, Accuracy = 66.8%, Confusion matrix:

1143	581
83	193

- C = 1, Accuracy = 68.2%, Confusion matrix:

1183	541
94	182

- C = 2, Accuracy = 68.9%, Confusion matrix:

1204	520
102	174

- C = 4, Accuracy = 70.2%, Confusion matrix:

1237	487
109	167

As it were the case with the KNN models, neither data augmentation, nor random dataset splitting were of use. The best performing variant's *C* hyperparameter based on the metrics outlined above is equal to 16. However, the best F1 score in the competition was achieved using a SVM with *C=0.5*.

- C = 0.5, Accuracy = 67.5%, Precision (normal class) = 0.92, Recall (normal class) = 0.67, Precision (abnormal class) = 0.25, Recall = 0.68, Confusion matrix:

1162	562
88	188

3.4. Majority voting using Naive Bayes, K-Nearest Neighbors and Support Vector Machines

Using the best performing models so far, I attempted to use the majority voting technique, which means that the dataset would be processed by each model individually and the final label would be decided based on two matching predictions.

Analyzing the previous experiments, I decided not to augment or randomly split the dataset.

- Number of bins = 4, K = 9, C = 0.5, Normalization using the *L1* distance, Accuracy = 72.95%, Precision (normal class) = 0.92, Recall (normal class) = 0.74, Precision (abnormal class) = 0.27, Recall (abnormal class) = 0.605, Confusion matrix:

1292	432
109	167

Clearly, this method didn't make a significant difference neither in the accuracy, nor in the F1 score.

3.5. Convolutional Neural Networks

Data augmentation and random splitting were all used in combination with varying epoch sizes and different learning rates for both of the architectures which will be described in detail below.

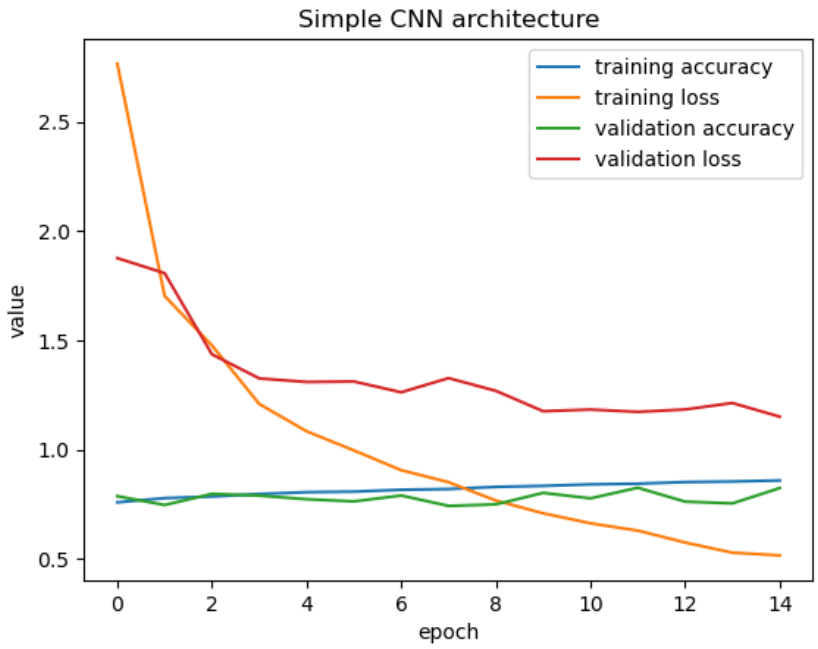
3.5.1. The first architecture

I started with quite a rudimentary architecture, making use of a single convolutional input layer of 8 filters of size 3, with padding, followed by a max pooling layer, with the default parameters. Then, flattening was applied in preparation for the fully connected output layer, which used the *softmax* activation function. The *ADAM* optimizer was used in the compilation process and both *loss* and *accuracy* were recorded.

No data augmentation, default train-validation dataset split.

- Learning rate = 1e-5, Accuracy = 82.4%, Confusion matrix:

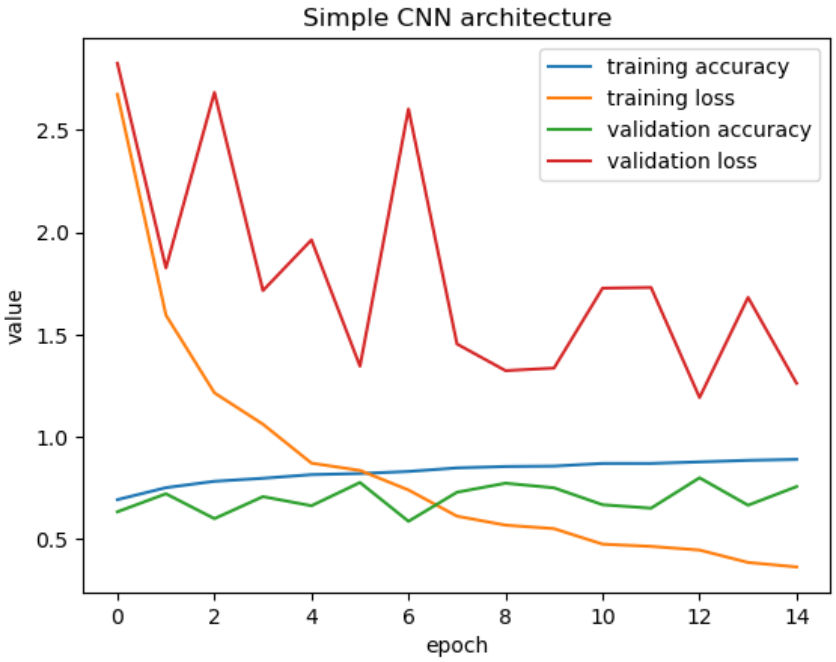
1581	413
208	68



Data augmentation, default train-validation dataset split.

- Learning rate = 1e-5, Accuracy = 75.7%, Confusion matrix:

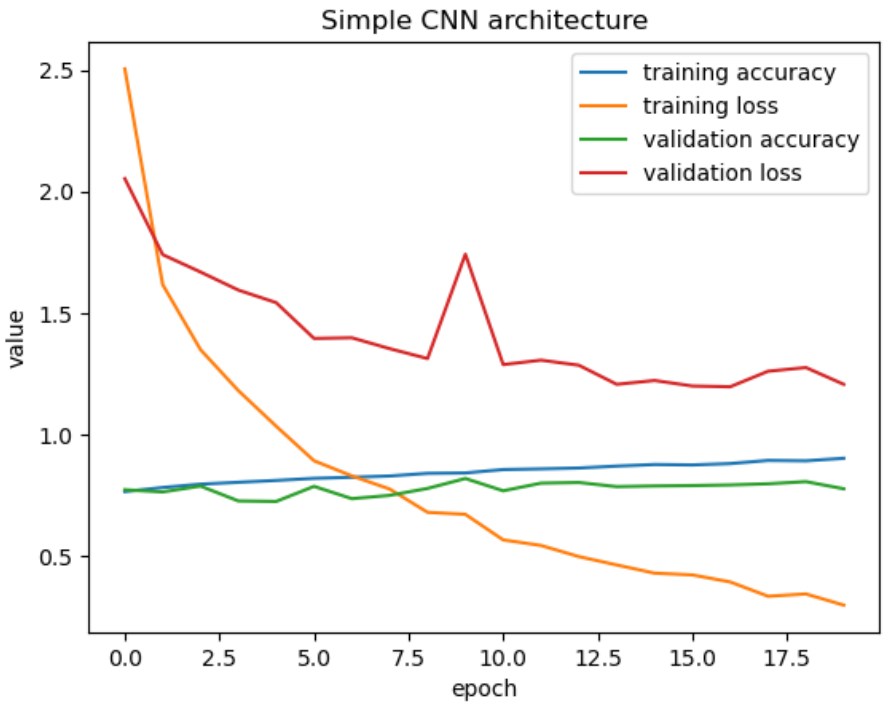
1396	328
158	118



No data augmentation, random train-validation dataset split.

- Learning rate = 1e-3, Accuracy = 77.6%, Confusion matrix:

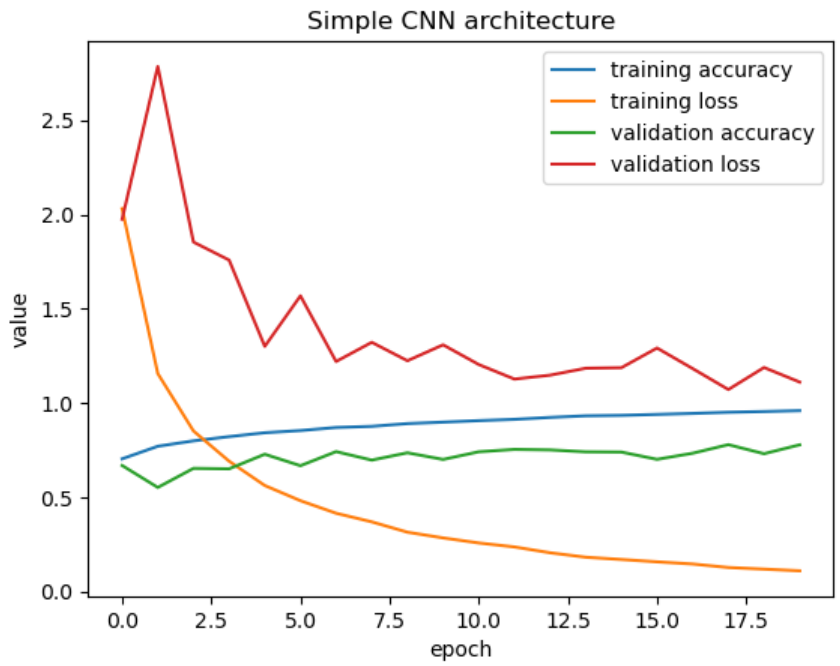
2462	401
358	179



Data augmentation, random train-validation dataset split.

- Learning rate = 1e-4, Accuracy = 77.8%, Confusion matrix:

2517	381
372	130



The result in the competition was not satisfactory, even though the model seemed to work quite well, based on the provided metrics and graphs.

3.5.2. The second architecture

I thought the architecture might not be complex enough for this classification problem, so I built the following model:

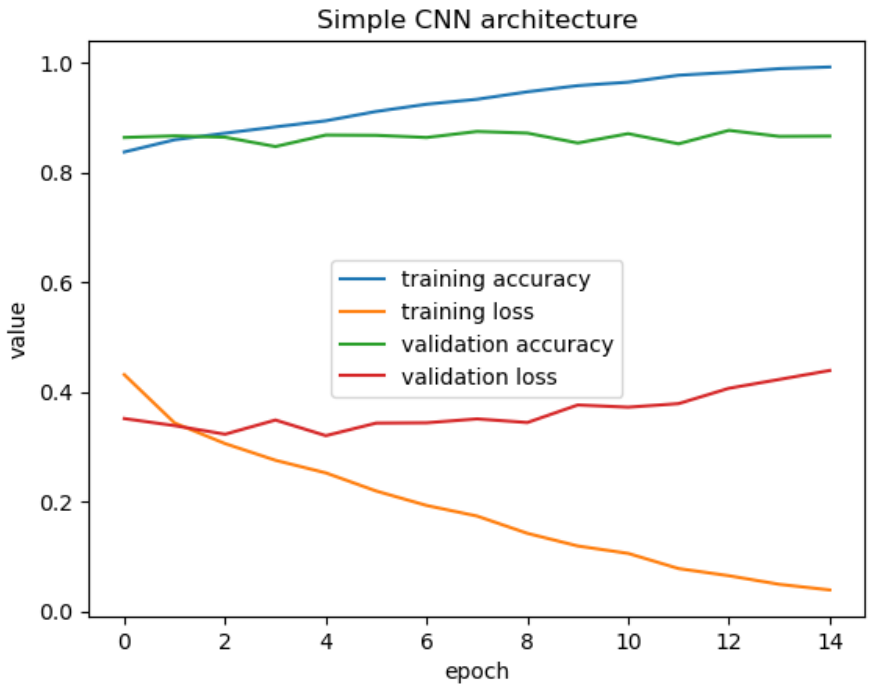
- 2 convolutional layers of 32 filters each of size 3, with padding, using the *ReLU* activation function, followed by a max pooling layer
- 2 convolutional layers of 64 filters each of size 3, with padding, using the *ReLU* activation function, followed by a max pooling layer
- 2 convolutional layers of 128 filters each of size 3, with padding, using the *ReLU* activation function, followed by a max pooling layer
- A flattening layer
- 2 fully connected layers, one with 256 neurons and the *ReLU* activation function, and the output layer, which used the *softmax* activation function

I decided to use the *ADAM* optimizer for this model as well.

No data augmentation, default train-validation dataset split.

- Learning rate = 1e-5, Accuracy = 86.65%, Confusion matrix:

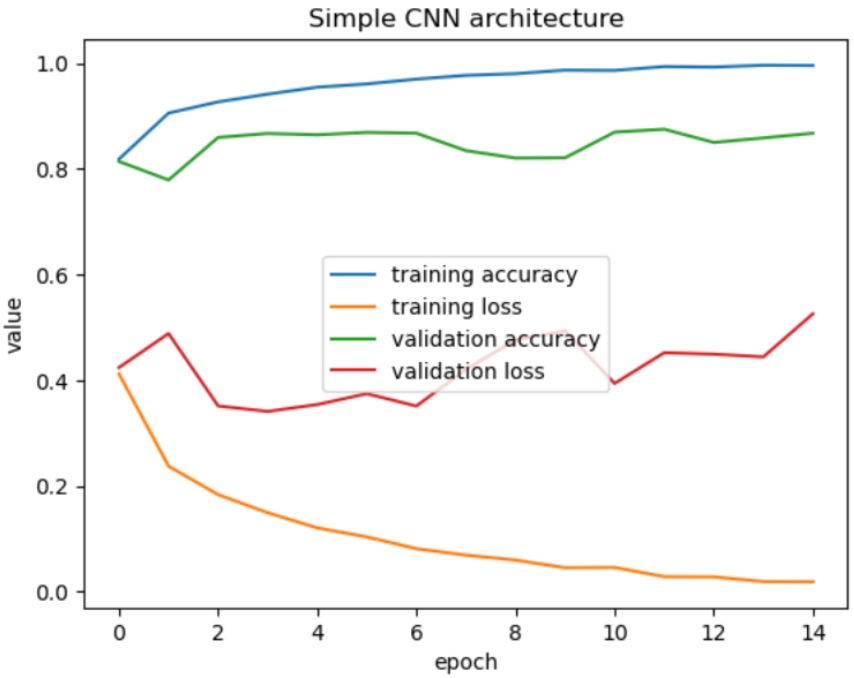
1613	111
156	120



Data augmentation, default train-validation dataset split.

- Learning rate = 1e-5, Accuracy = 86.7%, Confusion matrix:

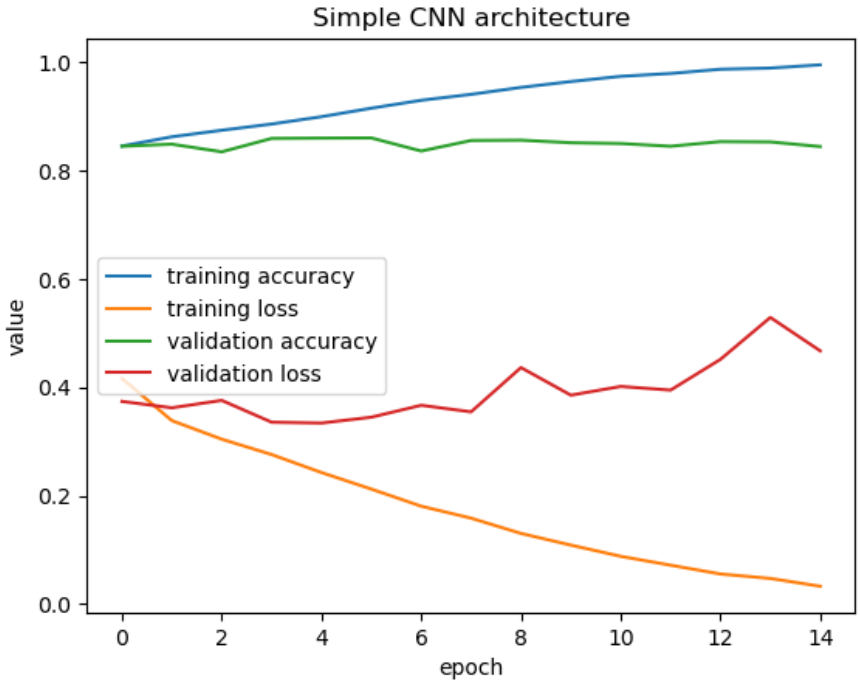
1630	94
171	105



No data augmentation, random train-validation dataset split.

- Learning rate = 1e-5, Accuracy = 84.4%, Confusion matrix:

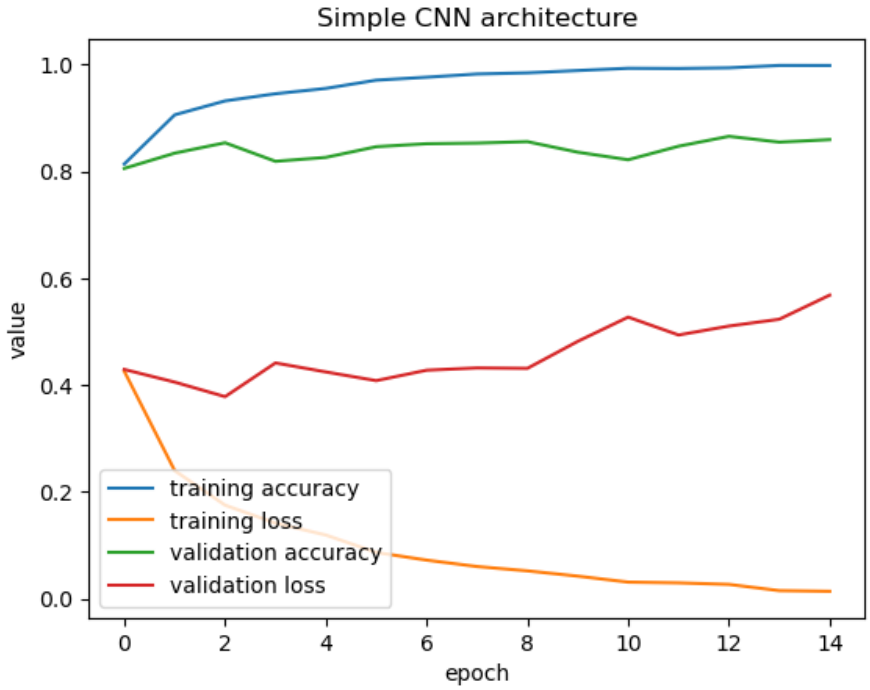
2656	207
323	214



Data augmentation, random train-validation dataset split.

- Learning rate = 1e-5 Accuracy = 85.9%, Confusion matrix:

2681	217
262	240



Surprisingly, even though the graphs depicting the accuracies and the loss functions didn't look as promising as it were in the case of the first architecture, I achieved a better F1 score in practice with this model than with the previous one.

- Learning rate = 1e-5, Accuracy = 86.7%, Precision (normal class) = 0.9, Recall (normal class) = 0.94, Precision (abnormal class) = 0.52, Recall (abnormal class) = 0.38, Confusion matrix:

1630	94
171	105

4. Conclusions

After carefully analyzing the submissions of the models presented so far, I chose the predictions generated by the more complex *CNN architecture* and the *Naive Bayes* model in the competition.