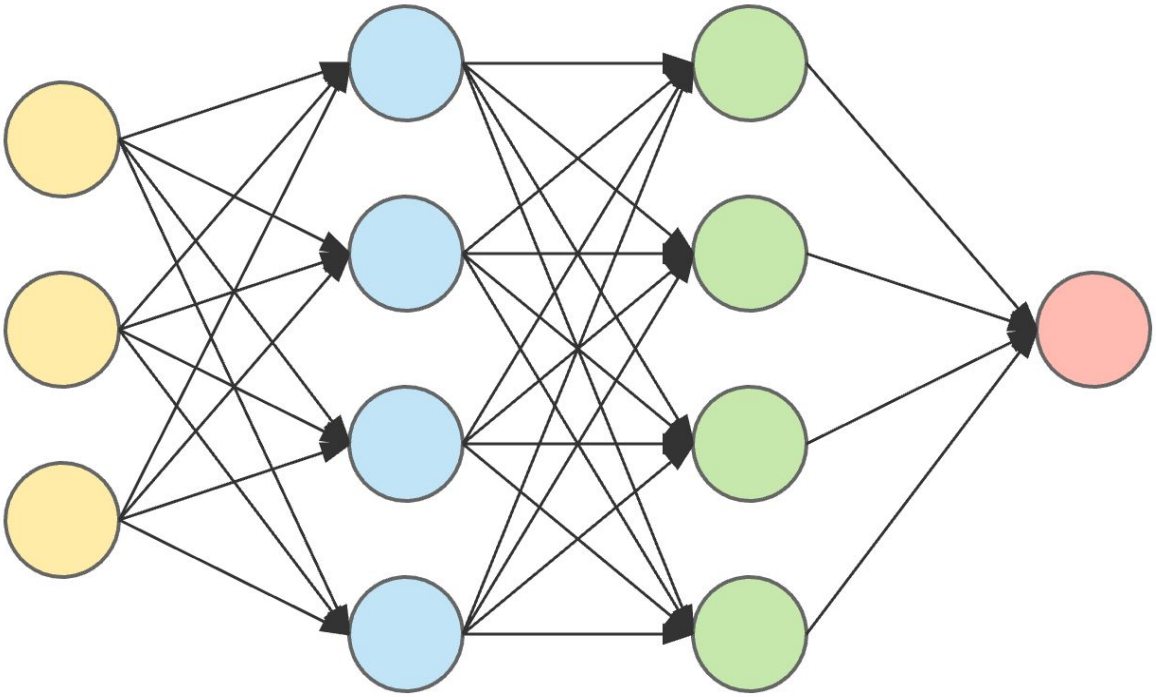
MUSK & NON-MUSK Chemical Compounds Classification Report

Credicxo Task



Meetkumar Ranoliya

GitHub

APPROACH

First of all, I took a look at the data given in .csv format to get an insight into it. It is a high dimensional data having 170 different columns including the target column. So as to start building a classifier for this, I went on the **"Google Colab"** and created a Notebook. Then I mounted it with my Google Drive to import the data in a DataFrame.

To build a good model it is important to have **clean data**. I found out that all the given compounds are unique using the "conformation-name" provided with each compound. As mentioned in the task statement we are supposed to split the data in **80:20 ratio** for training and testing. So, I split the data between train and test set. As a part of pre-processing, I have only done splitting of the data.

Now getting towards the model building, I thought of going with an **ANN(Artificial Neural Network)** created from scratch because with this small amount of data a small ANN can perform good and could be trained in less time. I am not using CNNs because CNNs are mostly used for image classification and here we are not dealing with images.

I started an ANN with only **one hidden layer** having **32 hidden units** but after training for **100 epochs** both training and validation accuracies were around ***0.99*** but it was not increasing that much so as to improve the model performance I thought of increasing the model complexity. For this, I increased the hidden units from 32 to 64. The results I got after **100 epochs** are: ***loss: 1.7720e-06 - accuracy: 1.0000 - val_loss: 0.0068 - val_accuracy: 0.9985***. The model is performing really very good at training data. Now, to increase performance on validation data or to make the model more generalized I thought of adding a **"Batch Normalization"** layer after the hidden layer. This will also help in model training as it will take less time for training and the model will learn fast. As expected it was able to achieve these results: ***loss: 0.0024 - accuracy: 0.9994 - val_loss: 0.0038 - val_accuracy: 0.9992*** in only **50 epochs**. To improve the model I trained it for other extra 50 epochs but I was not able to improve to the desired performance. To improve the training accuracy I increased the model complexity. For this, I increased the number of hidden units to 70 but after training for 100 epochs I was getting the same results as before. So instead of increasing the hidden units, I kept the 64 hidden units in the first hidden layers and added a new hidden layer with 64 hidden units. This will make the training easy and fast for the second hidden layer as it will get normalized values and at the same time, it will also increase the complexity of the model so that it can perform well on the training set.

After training the model for **100 epochs** the results I got are: ***loss: 0.0116 - accuracy: 0.9951 - val_loss: 0.0046 - val_accuracy: 0.9992***

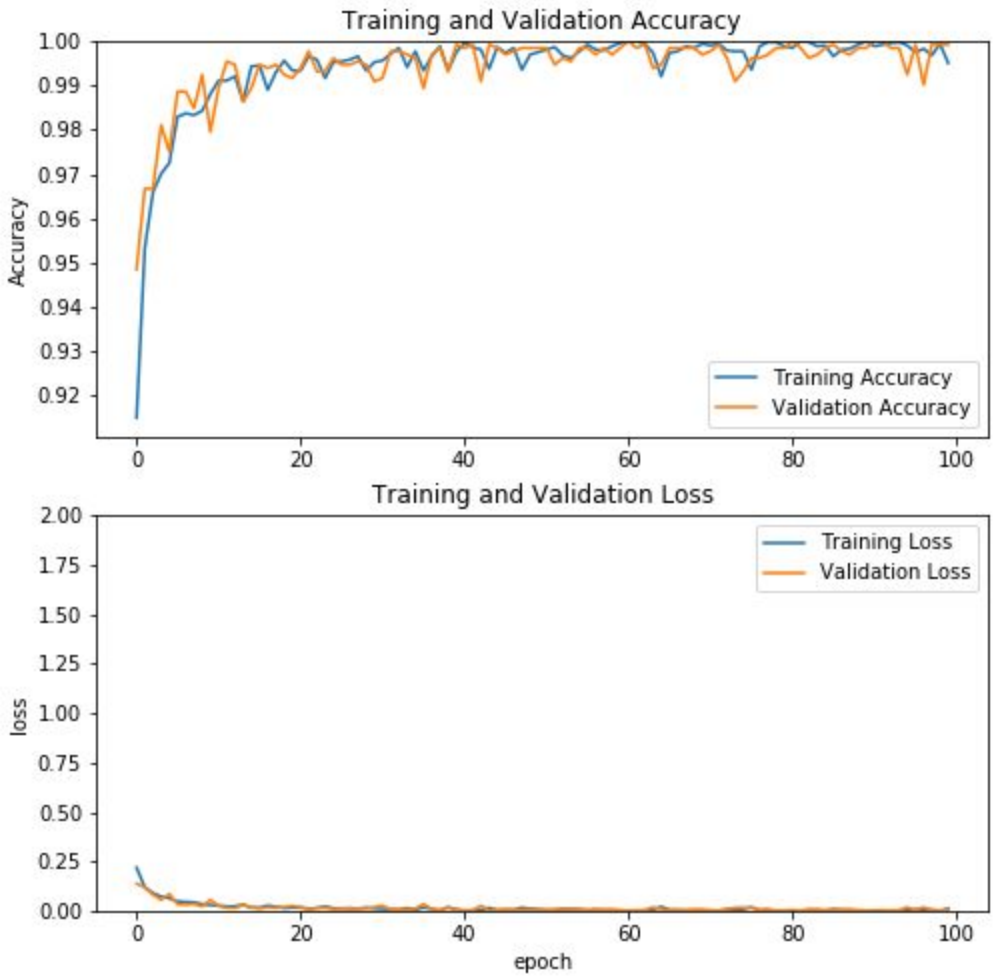


Fig 1. 1st 100 epochs (before finetuning)

For the finetuning, I trained it for more **50 epochs** with a callback to decrease the learning rate if it does not improve in 3 continuous epochs. Out of expectations, I got ***loss: 1.6283e-04 - accuracy: 1.0000 - val_loss: 0.0011 - val_accuracy: 1.0000*** in results.

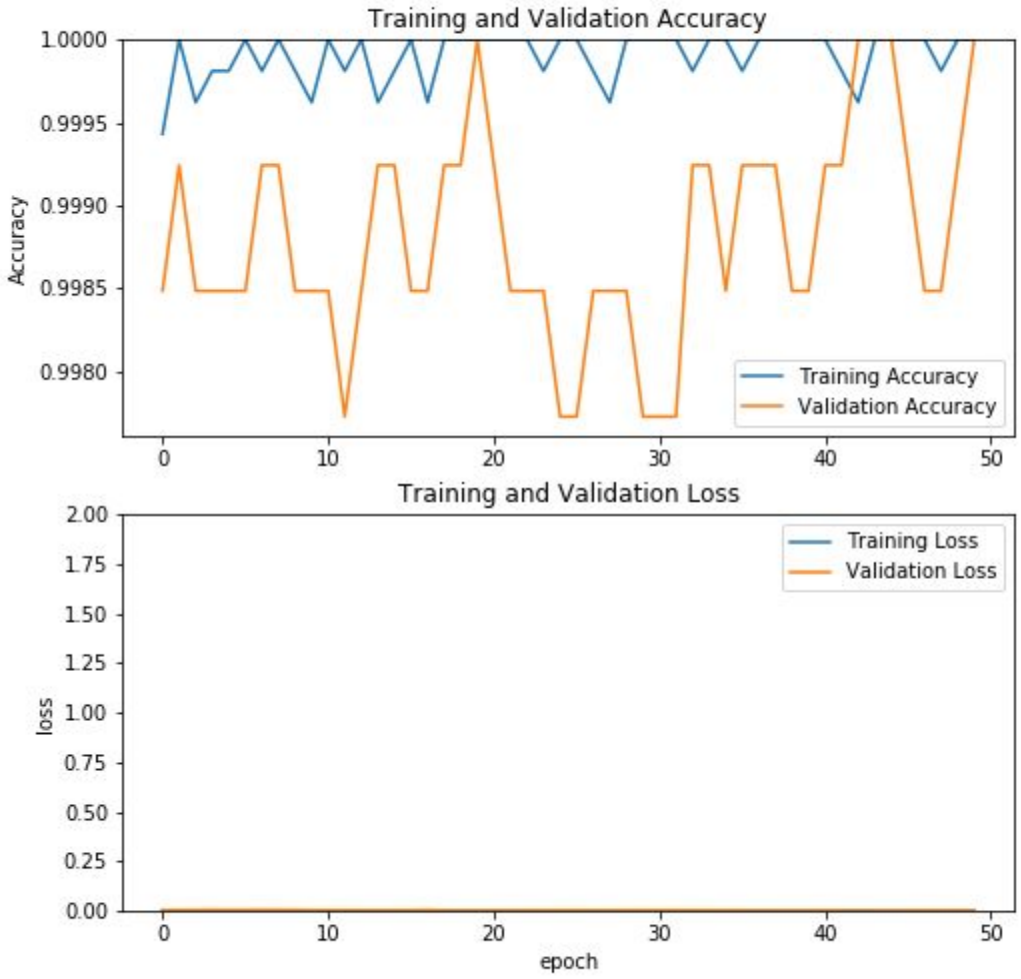


Fig 2. After Finetuning

MODEL LINK

[Model]

FINAL PERFORMANCE MEASURES

1. **Validation Loss: 0.0011**
2. **Validation Accuracy: 1.0000**
3. **Precision: 1.000000**
4. **Recall: 1.000000**
5. **F1 score: 1.000000**