Mars Surface and Curiosity Image Classification Project Report

Using a convolutional neural network

by Zachary Pruessner, for Springboard Final submission

Dataset DOI

10.5281/zenodo.1049137

Scientific Paper with additional details on the data

Kiri L. Wagstaff, You Lu, Alice Stanboli, Kevin Grimes, Thamme Gowda, and Jordan Padams. "Deep Mars: CNN Classification of Mars Imagery for the PDS Imaging Atlas." Proceedings of the Thirtieth Annual Conference on Innovative Applications of Artificial Intelligence, 2018.

Problem Statement

A large dataset of images from the martian rover Curiosity are unlabeled and difficult to search through. This will cause problems for those who wish to interact with the data. To solve this problem a machine learning model can be applied to sort through the data and label it.

Objective

The objective of this project is to build a convolutional neural network capable of classifying images from mars. The images in this data are from the rover Curiosity. There are over 39000 images available in this dataset. With an accurate neural network, one could implement it into a software meant to search through a large dataset of images from mars. This is useful in that it removes the difficulty of accessing images of specific types in a large dataset.

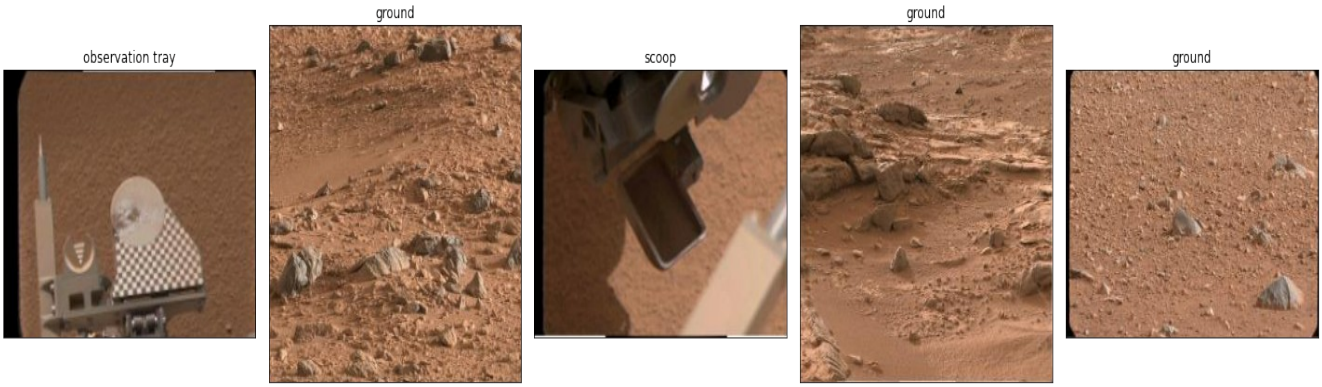
Data Wrangling

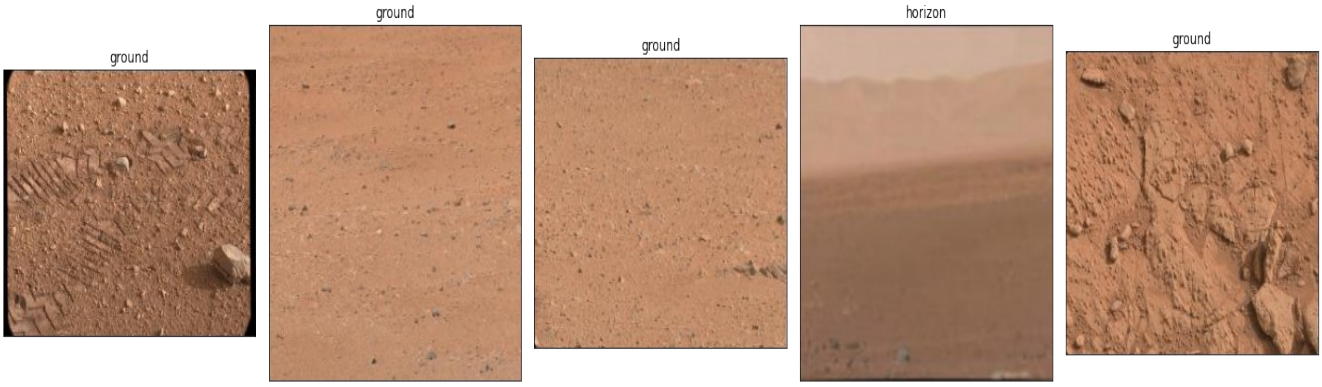
Initially, the data was setup for another computer system and needed to have the pathing corrected. This was fairly simple and only required a single helper function.

From there, the three separate csv files were merged and saved for further processing. This phase was quite simple as the data consisted of two folders that contained the images and three csv files that controlled the pathing to the images and their respective labels.

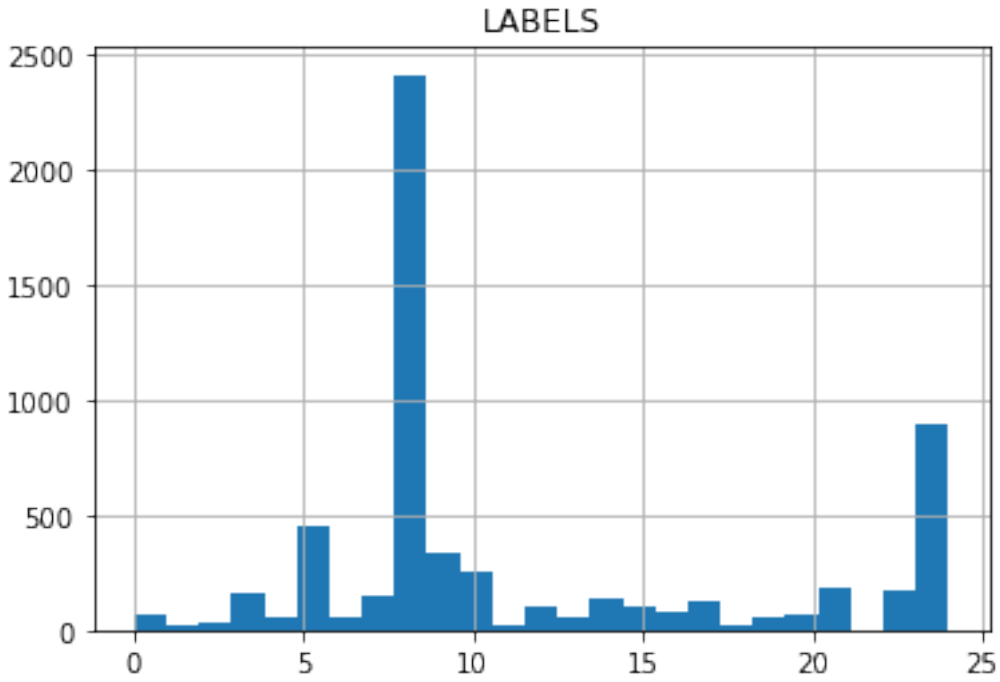
Exploratory Data Analysis

The analysis of this data was quite simple, as the data was fairly straight forward and organized. Although, it was found that the dataset was slightly imbalanced. This came in the form of numerous images of the ground. Significantly more than any other category. Another issue that will be addressed in the next section is that of image size. Some of the images are 256x256 pixels in size, while about half of them are not. Therefore it will be necessary to transform all of the images to a consistent size of 256x256 pixels.





The distribution of the labels is represented in the histogram below.



Pre-Processing

For pre-processing, the images were split into a train set and a test set. Then they were resized to 256x256 using the cv2 module, and normalized using the to_categorical function provided by the Keras module. This resulted in a label dataset with 25 columns with two possible values, 1 or 0. The normalization of that data resulted in all of the values falling between 1 and 0. Finally, all of the images were made to be the same size.

Modeling

This process went through many iterations before determining the final model. The intial start point was with a network very similar to the AlexNet. The issue with this approach was that the training time was quite long. Although it did appeare to produce an accurate model. Ultimately, I decided to approach this with a build-from-scratch standpoint.

I tampered with the batch_size quite a bit, as well as the overall structure of the model. My goal with the model is to come up with the simplest, most accurate model I can manage. This approach is challenging as it is my first attempt at creating a CNN from scratch.

Model 1

```
Model_1: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 126, 126, 32)      896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 63, 63, 32)        0
_________________________________________________________________
dropout (Dropout)            (None, 63, 63, 32)        0
_________________________________________________________________
flatten (Flatten)            (None, 127008)            0
_________________________________________________________________
dense (Dense)                (None, 64)                8128576
_________________________________________________________________
dense_1 (Dense)              (None, 25)                1625
=================================================================
Total params: 8,131,097
Trainable params: 8,131,097
Non-trainable params: 0
_________________________________________________________________
```

Model 2

```
Model_2: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 126, 126, 32)      896
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 63, 63, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 61, 61, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 30, 30, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 30, 30, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 57600)             0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               7372928
_________________________________________________________________
dense_3 (Dense)              (None, 25)                3225
=================================================================
Total params: 7,395,545
Trainable params: 7,395,545
Non-trainable params: 0
```

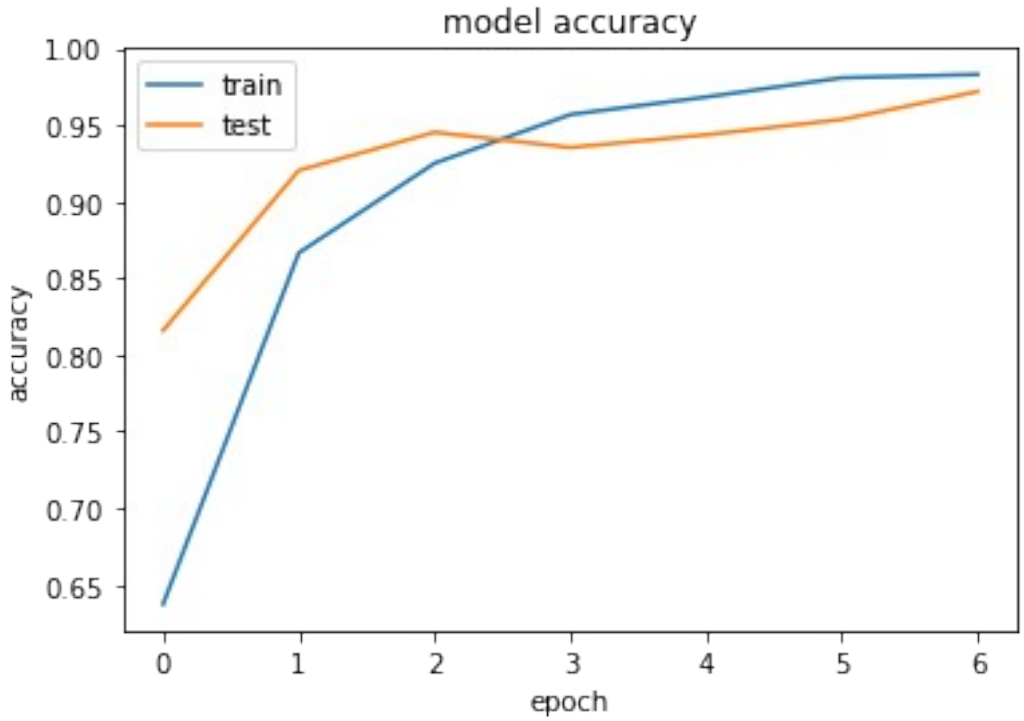
Model 3

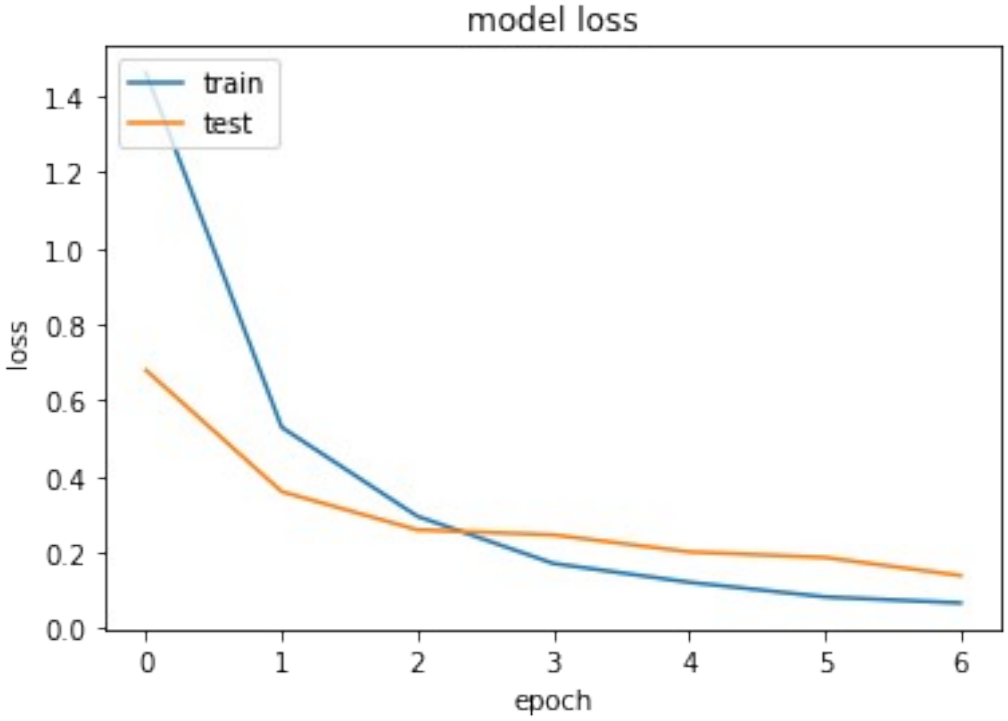
```
Model_3: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 126, 126, 32)      896
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 63, 63, 32)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 61, 61, 64)        18496
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 30, 30, 64)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 28, 28, 128)       73856
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 14, 14, 128)       0
_________________________________________________________________
dropout_2 (Dropout)          (None, 14, 14, 128)       0
_________________________________________________________________
flatten_2 (Flatten)          (None, 25088)             0
_________________________________________________________________
dense_4 (Dense)              (None, 256)               6422784
_________________________________________________________________
dense_5 (Dense)              (None, 25)                6425
=================================================================
Total params: 6,522,457
Trainable params: 6,522,457
Non-trainable params: 0
_________________________________________________________________
```

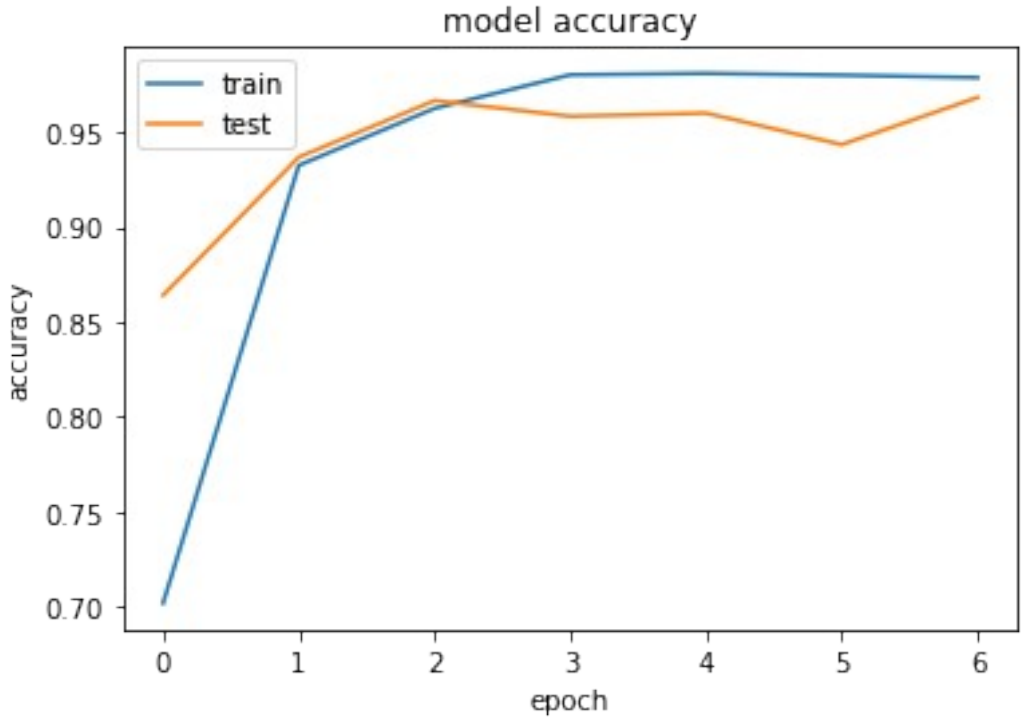
For the loss function I will use Categorical Crossentropy. The optimizer will be 'adam', which comes standard from the TensorFlow Libray.

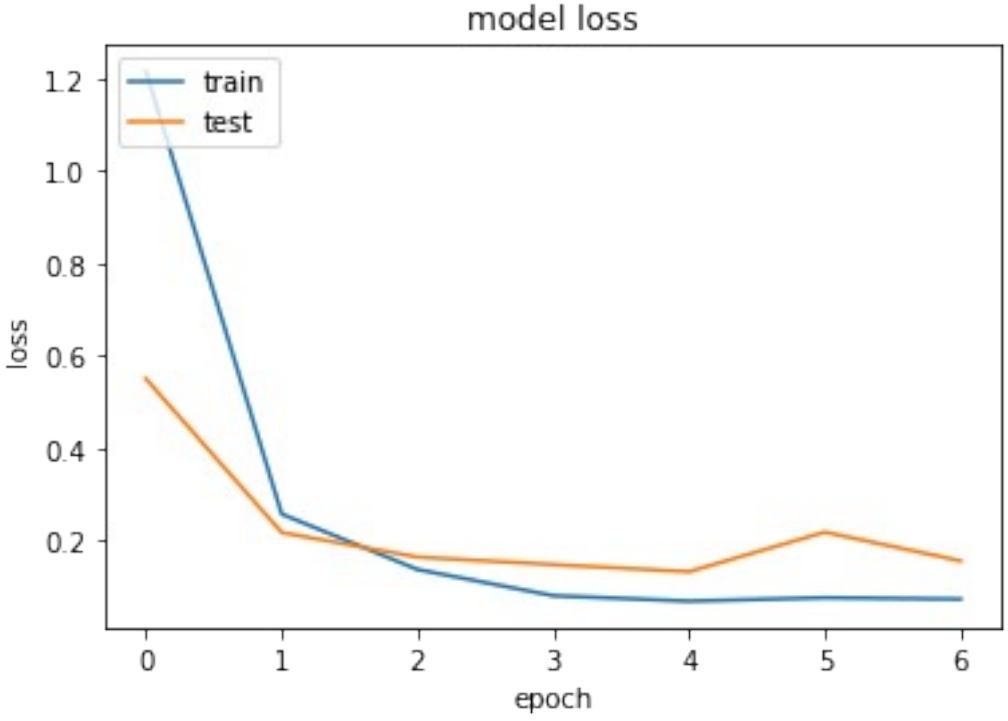
When training the three models, 7 epochs were used, with a batch size of 32, and a validation split of 10%.

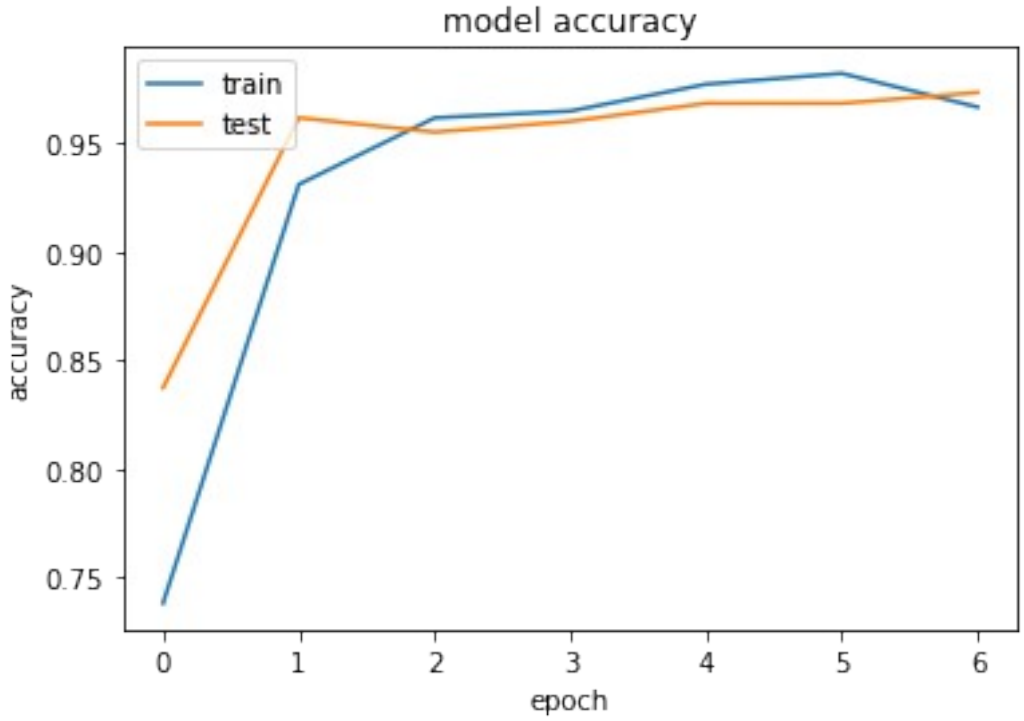
Performance

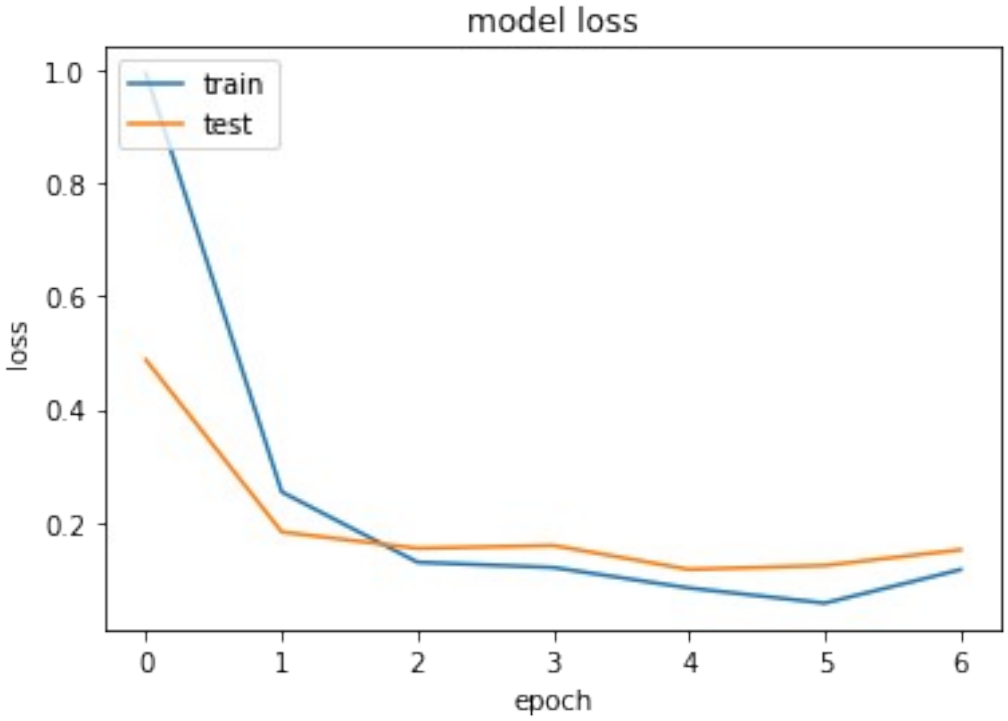





model accuracy
train
test
accuracy
0.95
0.90
0.85
0.80
0.75
0.70
0
1
2
3
4
5
6
epoch


model loss
train
test
loss
1.2
1.0
0.8
0.6
0.4
0.2
0
1
2
3
4
5
6
epoch





Model 1 loss: 0.0661 - accuracy: 0.9830 - mae: 0.0034 - val_loss: 0.1383 - val_accuracy: 0.9718 - val_mae: 0.0046

Model 2 loss: 0.0729 - accuracy: 0.9790 - mae: 0.0029 - val_loss: 0.1557 - val_accuracy: 0.9685 - val_mae: 0.0034

Model 3 loss: 0.1166 - accuracy: 0.9668 - mae: 0.0042 - val_loss: 0.1517 - val_accuracy: 0.9735 - val_mae: 0.0030

These are great numbers to start out with. Based on the above numbers, it appears as though Model 3 might be the ideal model. I propose this due to its performance on the validation set, rather than the training set.

The predictions of the individual models are as follows:

Model 1: 0.8507462686567164

Model 2: 1.0

Model 3: 1.0

Ideas for Further Research

Additional research might take the form of different model structures and their respective performance.