Report on Detection of Vegetation Stress from multispectral remote sensing images using Neural Network

Problem Statement:

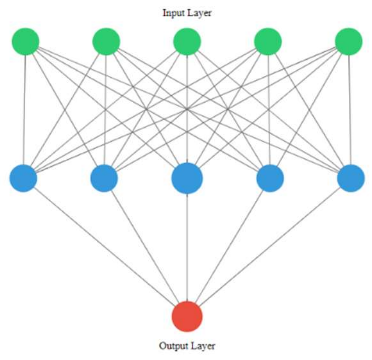
The type of the problem we are dealing with here is Classifica tion problem, where we are implementing a neural network model (using Python programming language) to classify each region on a remote sening image as 1 or 0 based on whether or not it contain stressed vegetation. The multispectral remote sensing images data is given in csv format, which comprises 574 number of samples of regions and 6 columns of variables (5 input variables and 1 output variable).

Data analysis:

Pandas is a library for data analysis and manipulation, using this we visualized our data and intepreted for further analysis. Since the input varibles values are in different range we normalized those values in the range of 0 and 1 using sklearn library for giving equal weightage to all the input features for classifing the regions. And then converting our data which is in pandas dataframe format into array for inputting these features into our neural netowrk model. Now, slicing down our data into 5 variables as input varibles (X) and 1 variable as output varibles(y) arrays of size 574 x 5 and 574 x 1 respectively. And splitting the whole data randomly into 70% training set (and 20% validation set) and 10% test set using train_test_split module of sklearn library.

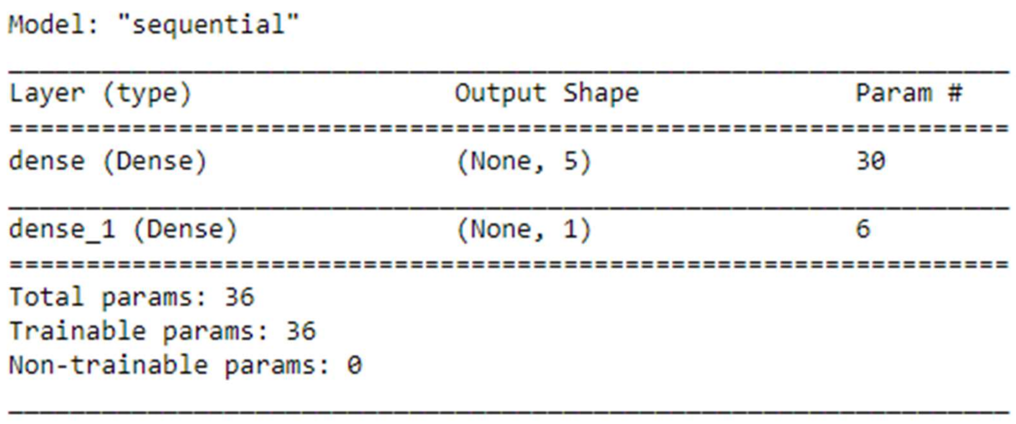
Modelling:

We modelled and trained our neural network using Keras library which wraps the numerical computation library Tensorflow. Using Sequential model & Dense class of the Keras, we designed the architecture of the fully-connected 3-layer neural network. We assigned 5 neurons to the input layer since we are dealing with 5 features. Since most of the problems does not require more than one hidden layer and to avoid overfitting we are restricted to one hidden layer. On iteration from 2 to 6 neurons in hidden layer with same number of training dataset, one with 5 neurons performs better than other so we choose 5 neurons for the hidden neuron. As the problem we are dealing with is binary classification problem so we put one neuron in output layer. The activation fucntions



for input and hidden layers is ReLU activation fucntion because of its better performance than other activation functions, while Sigmoid function for output layer to ensure that our network outputs either class 0 or class 1 i.e., maps the real-valued number to a probability of either 0 or 1.

Now that our model is defined as:



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 5)                 30
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 6
=================================================================
Total params: 36
Trainable params: 36
Non-trainable params: 0
_________________________________________________________________
```

The loss function that we use to evaluate a set of weights is *binary_crossentropy* and to optimize the weights we use the efficient stochastic gradient descent algorithm called *adam* as the optimizer and finally we will collect and report the classification *accuracy*, to compile the model.

To tackle the problem of choosing the number of training iterations (more causes overfit and few causes underfit problem) we use *EarlyStopping* method for training which allows us to specify large number of training epochs and stop training once our model validation loss reaches a minimum and stops improving till a specified number of training epochs. And then saving our model with optimized set of weights as best model using *ModelCheckpoint* method of Keras.

Using the *fit* method of the Keras library we train our model on the training dataset and monitoring our model performance using the validation dataset. We set the number of training epochs as high as 10000 and we are stopping the training as we reach a minimum validation loss (and there is no improvement in validation loss upto a 500 training epochs) and saving our model to with this optimized set of weights to evaluate out test dataset.

Results:

The *fit* method that we used to train our model returns a history object that summarizes the loss and accuracy at the end of each epoch so we can plot learning curves for our train and validation datasets, which allows to monitor the performance of our model training. We can see the training stopped at somewehere around at after 3500 iterations we observe from learning curves fig.1 and fig.2 (exactly at 3775/10000).

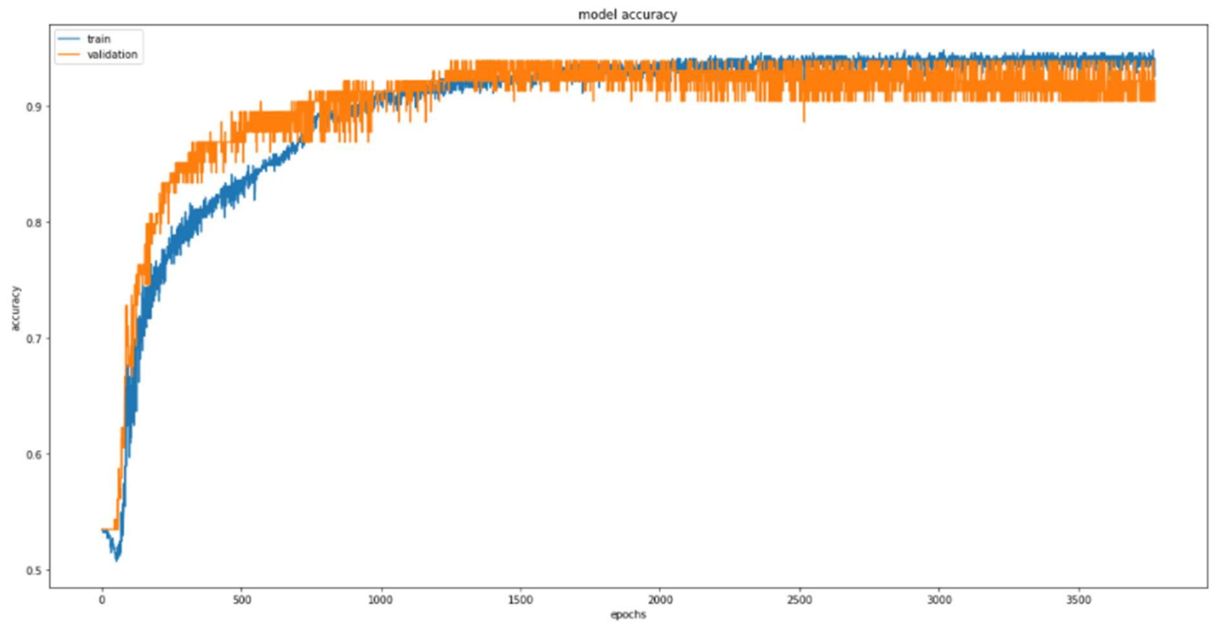


fig.1: Accuracy plots

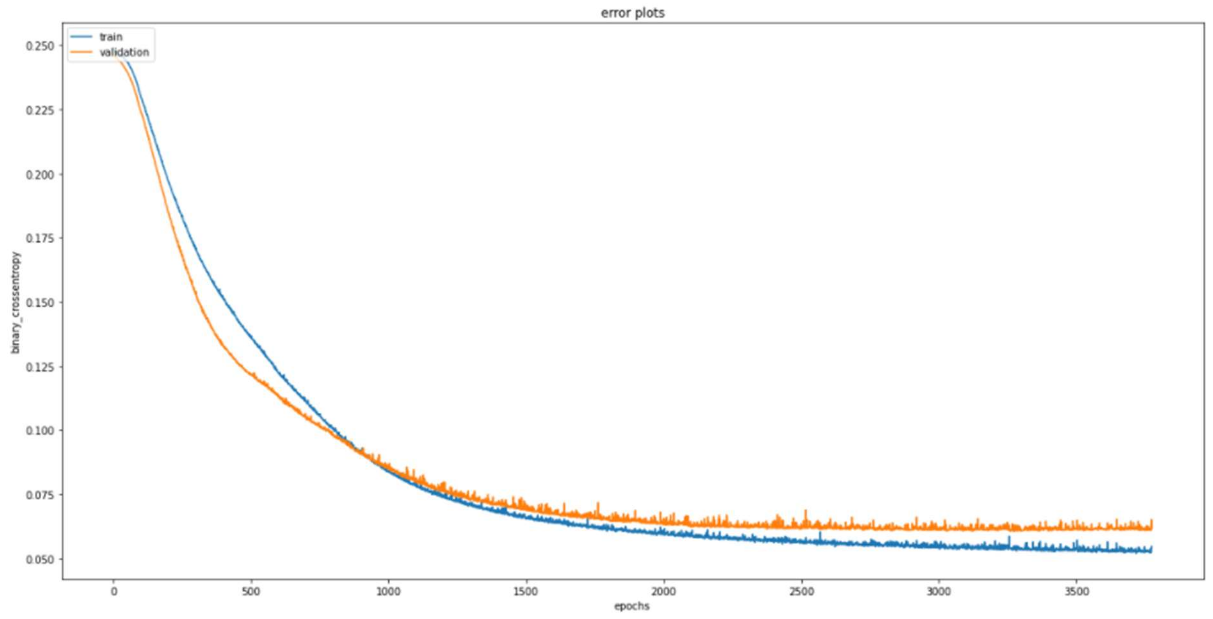


fig.2: Error plots

Now, we predict the classes of the test dataset regions using our saved model, best_model.h5. So we got the predicted classes and we have actual classes. Then we constructed the confusion matrix using predicted & actual classes we turns to be

```
[[35  1]
 [ 3 19]]
```

Then we evaulte the test dataset to obtain overall classification accuracy and other metrices.

Therefore, we achieved

an accuracy of 93.10% (number of samples correctly classified),

an error rate of 6.90% (number of samples misclassfied),

a sensitivity of 92.11% (model’s ability to predict non-stressed regions),

and a specificity of 95% (model’s ability to predict stressed regions).