Home Assignment 3
*

Yash Arvindkumar Patel
School of Engineering and Applied Science
Ahmedabad University
Ahmedabad, India
AU1841141

Bhavesh Oza
School of Enginnering and Applied Science
Ahmedabad University
Ahmedabad, India
AU1949003

***Abstract*—In this Assignment we need to create a Convolution Neural Network for citrus leaves classification according to the classes given in dataset. Also we need to consider the hyper-parameters as our experiment focus and try to carry out intuition of how those parameters works.**

***Index Terms*—CNN(Convolution Neural Networks), Hyper-parameters, Learning Rate, Citrus-leaves, Classification**

I. Introduction/ Motivation and Background

In this assignment we need to work on a Convolutional Neural network and classify in which class does the given leaf image falls. There are five classes of leaves. Also we need to work with different hyperparameters of the model and compare how the loss and accuracy varies for different value of hyperparameters. In this particular assignment we are going to experiment with learning rates, dropout percentage and activation function.

II. Detailed Mathematical Analysis

A. Analysis of Activation function

Here we have used *ReLU* activation for all the layers except output layer, and for the output layer we have experimented with two different activation functions i.e., *sigmoid* and *softmax.*

- ReLU stands for rectified Linear Activation function and it switches neurons off whenever it came accross negative values. Here it is advised to use ReLU as activation function because we get rid of any negative values came from summation of product of weights and previous neurons.

$$\begin{cases} 0 & x \leq 0 \\ x & x > 0 \end{cases}$$

- Sigmoid activation function maps any values from $-\infty$ to ∞, to 0-1 and gives the predictive analysis of classification.

$$f(x) = \frac{1}{1 + e^{-mx+b}}$$

  In neural networks $m =$ weights, $x =$ value of neurons, $b =$bias
- Softmax activation is also famous for its mapping of large values to range of $0 - 1$ as it is a probabilistic function which gives proportion of similarity in probabilistic manner to different class and hence it is used at the output layer.

$$f(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{k} e^{x_i}}$$

  Here, $input = \sum w * x + b$ and output is probability of given classes.

B. Analysis of Dimensions

When we talk about dimensions of input and output, some of the parameters came into picture that can affect it. Strides, padding, kernel size are major factors that can affect the output shape.
In our model we had used $strides = 1$, $kernel = 5 * 5$ and padding as same. This mean that padding will make the output image same as the size of input. Our input shape is of $256 * 256$.

$$out = \frac{in - ksize + 2 * padd}{strides}$$

$$padd = \frac{(out * strides) + ksize - in}{2} = \frac{256 * 1 - 256 + 5}{2} = 1$$

That means padding of 1 pixel should be there by model itself.

C. Analysis of Learning Rate and Loss

Learning shows the amount by which model should learn. If we consider $J(\theta)$ as total loss then gradient descent can be shown as:

$$\theta_{n+1} = \theta_n - \alpha * \frac{\partial J(\theta)}{\partial \theta}$$

Here alpha is the amount of derivative i.e. gradient of loss we want to subtract from total loss and we can also put it as the amount of learning for parameter and hence we call it learning parameter.

III. Experiments and Results

In this assignment we have experimented with some model parameters including hyperparameters. We have compared different values of learning rates, different amount of dropout percentage and we have also experimented with the activation function. We will go through the results one by one.

A. *Experiment with Learning rates*

We have carry out a list of learning rates, starting with $10^{-}3$ and increment by multiple of 10 till 1. We have got five different values. Mapping of alpha values with index: $[1 : 0.0001, 2 : 0.001, 3 : 0.01, 4 : 0.1, 5 : 1]$

Fig. 1. Training Accuracy

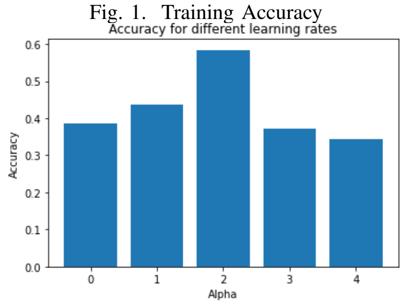
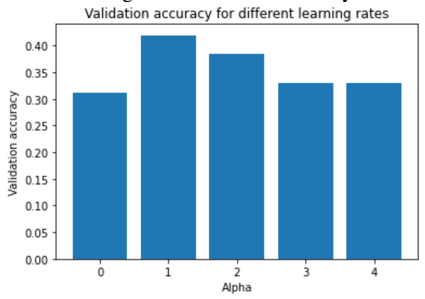


Fig. 2. Validation Accuracy



There are no major difference when we talk about validation accuracy this shows that no learning rate are perfect for given model. Some of the observations are:

- For very low learning rates such as (0.0001, 0.001) accuracy is increasing but in very slow manner and hence it needs more epochs to be more accurate.
- Moderate learning rates (0.01) gives satisfying results then other as they are not so slow and not encountering the minima.
- Higher learning rates (0.1,1) are not so satisfying as their training rates are not so good and that can be due to encountering the global minima.

B. *Experiments on Dropout*

When we talk about accuracy plots between training and validation accuracy, there can be overfitting and underfitting where we don't want to stuck. In first scenerio where we can see that training accuracy is quite higher than validation and also the rate of increment in validation accuracy is not so encouraging and thus we can say that there might be problem of overfitting.

To get out of it we have experimented with 30 percent dropout and 50 percent dropout and we can see the difference.

- For both the dropout weightage we got quite good training accuracy but the model struggles with lower dropout in validation accuracy.

Fig. 3. Accuracy plot with 50 percent Dropout

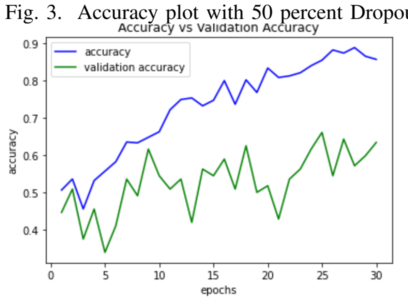
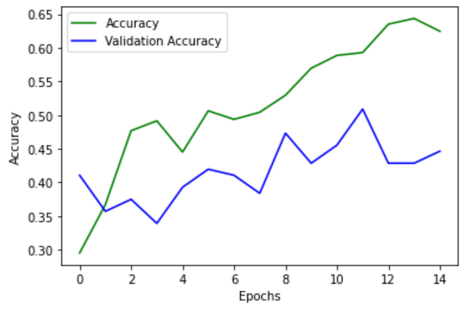


Fig. 4. Accuracy plot with 50 percent Dropout

```
<matplotlib.legend.Legend at 0x7fccb23e4a50>
```



- With higher dropout we can see a minor change and that is there is less frequency of ups and downs in validation accuracy.

C. *Experiments on Activation function*

As mentioned we have trained model with two activation functions at the output layer and we got some interesting results. With softmax activation we can see there is good slope in training loss and for validation loss it is very noisy and also there is no decrement in it.
With sigmoid activation both the loss are quite smoother than softmax and also validation loss slowly decreasing

Fig. 5. Loss plot with Softmax activation

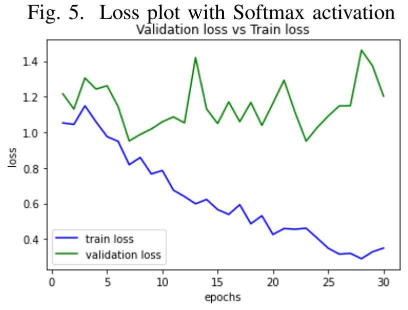
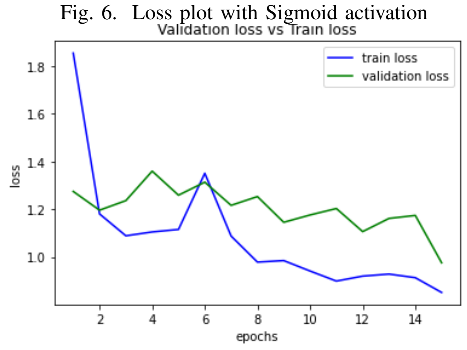


Fig. 6. Loss plot with Sigmoid activation



IV. Discussions and Conclusion

- Although we are not getting good enough accuracy in the model, but we can see the minor changes that are happening because of change of the parameters.
- Also there has been less number of images of leaves in each class and we are taking the resolution of $256 * 256$ pixels and on combining they re quite large amount of features and model may suffer from overfitting due to lesser data and larger set of features.
- Another conclusion we can draw here is it should be good idea to test first in less number of epochs and then if we found that there is a smooth and continuos decrement in loss then we can try with higher number of epochs.

References

[1] https://keras.io/api/preprocessing/image/

[2] https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html

[3] https://vijayabhaskar96.medium.com/tutorial-image-classification-with-keras-flow-from-directory-and-generators-95f75ebe5720

[4] https://www.geeksforgeeks.org/python-image-classification-using-keras/