UNDERSTANDING AND INTEGRATION OF BATCH NORMALIZATION TO U - NET WITH AN APPLICATION TO SCHEMANTIC SEGMENTATION OF AERIAL IMAGES

Sai Ram Ajay Krishna Gabbula
e-mail: s_gabbula@uncg.edu

Abstract – **Training a deep neural networks is difficult and time consuming. Obtaining a convergence within a reasonable amount of time can be tricky. Here I am integrating batch normalization, a popular and effective technique that consistently accelerates the convergence of deep neural networks [4]. Batch Normalization makes the optimization landscape significantly smoother. By this the behavior of the gradients is more predictive and stable allowing for faster training.[9]**

Semantic Segmentation is the most common approach which is used to assign a label to every pixel in the image. Semantic segmentation of the images is preferred in deep learning techniques. Huge datasets with images and labels are required to carry out deep learning techniques [11]. Gathering images may not be a tough task but labelling them will take lot of time and effort. The main aim here is to train a U - Net model with different configurations and compare the results. Integrating batch normalization technique [2] to the model to improve the overall training speed of the network. The results obtained are presented.

Keywords – **Keywords: Deep learning, Satellite Imagery, U-Net Models, Batch Normalization.**

I. INTRODUCTION

With the advancement of space technologies more and more satellite deployment is taking place. The earth observing satellites are increasing in number and so do the data produced by them. With the help of computer vision tools and machine leaning we can process this data and extract information from them.

Building footprints or classification of land cover can be done with the help of deep learning techniques. Large amount of training data are required to train the deep learning model like U-net. Use of these deep convolutional models have proven good accuracy results over large variety of computer vision techniques especially in above said tasks. The main problem in implementing these techniques is the availability of large labelled training data set which requires a lot of effort and time.

Here a basic U - Net model is considered for the schemantic segmentation tasks [5]. I am training the model with the dataset present in kaggle. The data set consists of satellite images of Dubai. The dataset consists of aerial imagery of Dubai obtained by MBRSC satellites and these images are annotated with pixel-wise semantic segmentation in 6 classes. The total volume of the dataset is 72 images grouped into 6 folders. The classes are as follows: Building (#3C1098), Land or unpaved area (#8429F6), Road (#6EC1E4), Vegetation (#FEDD3A), Water (#E2A929), Unlabeled (#9B9B9B) [3].

II. IMPLEMENTATION

I have implemented a U - Net model [1][5], I have defined a U - Net model with a total of 9 blocks where block1 to block5 comes under contraction path and block6 to block9 comes under expansion path. Each block in the contraction path have two convolution layers followed by a max-pooling layer. The stride of the max-pooling operation is selected as 2. This is done to reduce the size (halve) of the image while keeping same number of channels. The feature number is doubled in the next two convolution layers. So in the contraction path the size of the image gets smaller and smaller after each block while the feature channels get larger and larger.

The blocks fro 6 to 9 comes under expansion path, the operation carried out here is reverse to that of the contraction path. Here each block consists of a conv2d transpose with the size of 2 which doubles the size of the image. This layer is followed by a concatenation operation and then followed by two convolution layers.

A. Concatenation

The interesting property of the U-Net is that it combines layers from the contraction path with the expansion path by an operation of concatenation. The size of the layers should match to perform a concatenation operation. These types of connections between layers are called skip connections. The advantages of using these skip connections are that they allow gradients to more freely flow through the model, mitigating the issue of vanishing gradients. This operation allows features from the contraction path of the network to the expansion path of the network there by adding extra information that might be lost in the max-pooling on the contraction path of the network. This concatenation operation helps in enhancing the classification process.

B. Batch Normalization

Training deep neural networks is a challenging process and the complexity increases with increase in the layers. The layers in the network can be sensitive to the initial random weights and configuration of the learning algorithm. One of the possible reason is the distribution of the inputs to layers deep in the network may change after each mini-batch when the weights are updated. This is called as the “internal covariate shift.” [4]

Batch normalization is a technique used for training very deep neural networks. For every batch of data batch normalization standardizes the inputs. Batch normalization helps in stabilizing the learning process. Normalizing the inputs to the layer has an effect on the training of the model, and dramatically reducing the number of training epochs required to train deep networks. By this fast overall training of the network can be achieved. This technique helps higher learning rates. Batch normalization makes the process easy to work with any activation functions. Batch normalization helps in reducing generalization error much like the use of activation regularization. Batch normalization applies a transformation that maintains the mean output close to 0 and the output standard deviation close to 1.[2]

Dropout is a technique typically used to reduce overfitting. By using batch normalization the dropout can be removed or reduced. Overall better results can be achieved with faster network convergence using Batch normalization.[10]

Batch normalization reparametrizes the optimization problem to make it more stable and smooth. This means that the gradients that are used in training are well-behaved and more predictive. This results in faster and effective optimization. Batch normalization helps in achieving robustness to hyperparameter setting and avoiding gradient vanishing. [9]

1) Jaccard Coefficient: The Jaccard coefficient is a measure used to measure the similarities between the datasets. Jaccard coefficient is defined as the size of the intersection divided by the size of the union of sample sets. It is used to compare set of predicted labels for a sample to the corresponding set of labels. [13,14,15]

Convolutional Neural Networks (CNN's) with object detection applications, apply the Jaccard Index measurements as a way of conceptualizing accuracy of object detection. For example, if a computer vision algorithm is tasked with detecting faces from an image, the Jaccard index is a measure of the similarities between the results obtained to those of the training data.

Jaccard coefficient J(A,B) can be represented as :

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

$$J(A,B) = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

where A and B are two different sets, here A and B are labelled dataset and predicted label set respectively.[13]

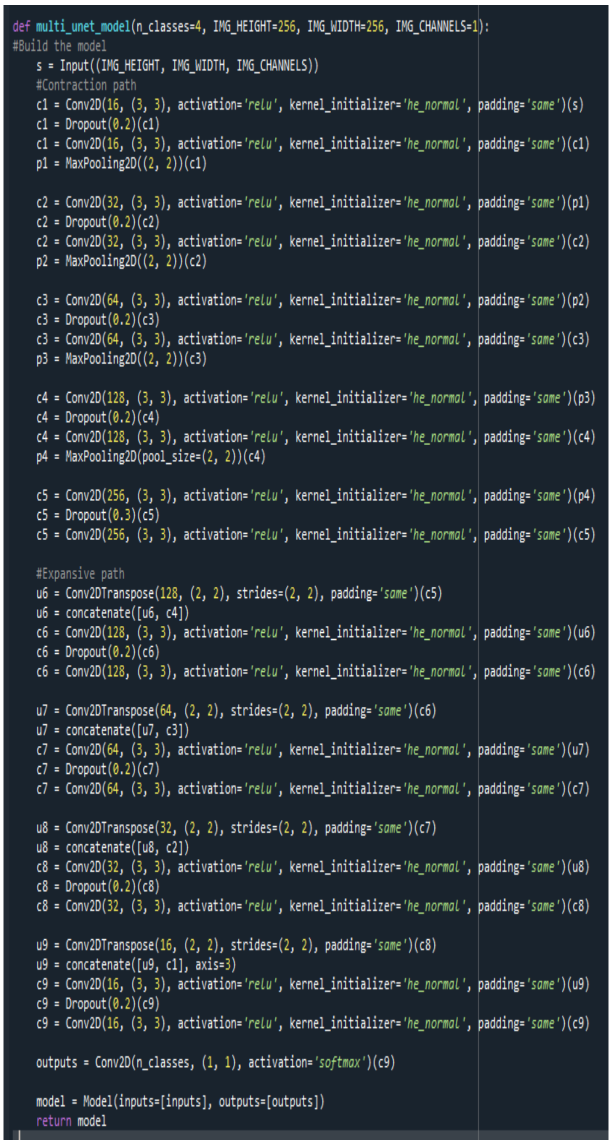
2) Early stopping: One of the challenge in training deep neural networks is to estimate the epochs to train the model. No one can really guess the training time for a model. It depends on a lot of factors. If we take less number of epochs then that model will underfit the training data and if we take more number of epochs then that model will overfit the training data. These two scenarios have poor performance on the test set resulting in low validation accuracy.[6,7]

Overfitting of the training dataset makes the generalization error to increase thereby making the model less useful at making predictions on new data. We need to train the network long enough so that it is capable of learning the mapping from inputs to outputs.

The best approach is to stop the training when the performance of the validation dataset doesn't improve significantly. This can be achieved by using early stopping. Early stopping helps to stop training when the metric specified to monitor doesn't improve. A patience number can be specified which helps to stop the training after the specified number of epochs with no improvement. .

III. MODEL

The U - Net model used is presented here. I am presenting the referred U - Net model and the proposed model.

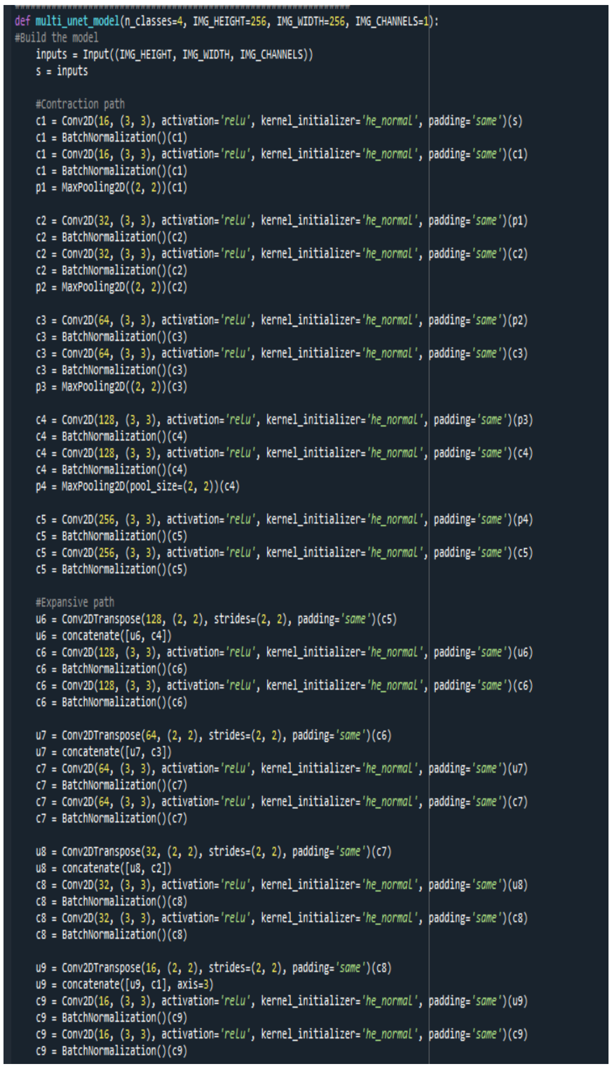


```python
def multi_unet_model(n_classes=4, IMG_HEIGHT=256, IMG_WIDTH=256, IMG_CHANNELS=1):
#Build the model
    s = Input((IMG_HEIGHT, IMG_WIDTH, IMG_CHANNELS))
    #Contraction path
    c1 = Conv2D(16, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(s)
    c1 = Dropout(0.2)(c1)
    c1 = Conv2D(16, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(c1)
    p1 = MaxPooling2D((2, 2))(c1)

    c2 = Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(p1)
    c2 = Dropout(0.2)(c2)
    c2 = Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(c2)
    p2 = MaxPooling2D((2, 2))(c2)

    c3 = Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(p2)
    c3 = Dropout(0.2)(c3)
    c3 = Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(c3)
    p3 = MaxPooling2D((2, 2))(c3)

    c4 = Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(p3)
    c4 = Dropout(0.2)(c4)
    c4 = Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(c4)
    p4 = MaxPooling2D(pool_size=(2, 2))(c4)

    c5 = Conv2D(256, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(p4)
    c5 = Dropout(0.3)(c5)
    c5 = Conv2D(256, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(c5)

    #Expansive path
    u6 = Conv2DTranspose(128, (2, 2), strides=(2, 2), padding='same')(c5)
    u6 = concatenate([u6, c4])
    c6 = Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(u6)
    c6 = Dropout(0.2)(c6)
    c6 = Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(c6)

    u7 = Conv2DTranspose(64, (2, 2), strides=(2, 2), padding='same')(c6)
    u7 = concatenate([u7, c3])
    c7 = Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(u7)
    c7 = Dropout(0.2)(c7)
    c7 = Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(c7)

    u8 = Conv2DTranspose(32, (2, 2), strides=(2, 2), padding='same')(c7)
    u8 = concatenate([u8, c2])
    c8 = Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(u8)
    c8 = Dropout(0.2)(c8)
    c8 = Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(c8)

    u9 = Conv2DTranspose(16, (2, 2), strides=(2, 2), padding='same')(c8)
    u9 = concatenate([u9, c1], axis=3)
    c9 = Conv2D(16, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(u9)
    c9 = Dropout(0.2)(c9)
    c9 = Conv2D(16, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(c9)

    outputs = Conv2D(n_classes, (1, 1), activation='softmax')(c9)

    model = Model(inputs=[inputs], outputs=[outputs])
    return model
```

Fig. 1. Referred U - Net model without batch normalization.

Fig1 shows the refered U - Net model without batch normalization. In this model drop out is used to overcome the overfitting problem. The input image height and width is taken as 256. [5]

Fig2 shows the proposed U - Net model with batch normalization. In this model drop out removed in each block and batch normalization is integrated after each convolution 2d layer in all the blocks of the model.By this the whole output coming out from the conv2d layer gets normalized before feeding it to the next layer of the model. All the remaining code is kept same for a better understanding of only batch normalization.



```
def multi_unet_model(n_classes=4, IMG_HEIGHT=256, IMG_WIDTH=256, IMG_CHANNELS=1):
#Build the model
    inputs = Input((IMG_HEIGHT, IMG_WIDTH, IMG_CHANNELS))
    s = inputs

    #Contraction path
    c1 = Conv2D(16, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(s)
    c1 = BatchNormalization()(c1)
    c1 = Conv2D(16, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(c1)
    c1 = BatchNormalization()(c1)
    p1 = MaxPooling2D((2, 2))(c1)

    c2 = Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(p1)
    c2 = BatchNormalization()(c2)
    c2 = Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(c2)
    c2 = BatchNormalization()(c2)
    p2 = MaxPooling2D((2, 2))(c2)

    c3 = Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(p2)
    c3 = BatchNormalization()(c3)
    c3 = Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(c3)
    c3 = BatchNormalization()(c3)
    p3 = MaxPooling2D((2, 2))(c3)

    c4 = Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(p3)
    c4 = BatchNormalization()(c4)
    c4 = Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(c4)
    c4 = BatchNormalization()(c4)
    p4 = MaxPooling2D(pool_size=(2, 2))(c4)

    c5 = Conv2D(256, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(p4)
    c5 = BatchNormalization()(c5)
    c5 = Conv2D(256, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(c5)
    c5 = BatchNormalization()(c5)

    #Expansive path
    u6 = Conv2DTranspose(128, (2, 2), strides=(2, 2), padding='same')(c5)
    u6 = concatenate([u6, c4])
    c6 = Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(u6)
    c6 = BatchNormalization()(c6)
    c6 = Conv2D(128, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(c6)
    c6 = BatchNormalization()(c6)

    u7 = Conv2DTranspose(64, (2, 2), strides=(2, 2), padding='same')(c6)
    u7 = concatenate([u7, c3])
    c7 = Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(u7)
    c7 = BatchNormalization()(c7)
    c7 = Conv2D(64, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(c7)
    c7 = BatchNormalization()(c7)

    u8 = Conv2DTranspose(32, (2, 2), strides=(2, 2), padding='same')(c7)
    u8 = concatenate([u8, c2])
    c8 = Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(u8)
    c8 = BatchNormalization()(c8)
    c8 = Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(c8)
    c8 = BatchNormalization()(c8)

    u9 = Conv2DTranspose(16, (2, 2), strides=(2, 2), padding='same')(c8)
    u9 = concatenate([u9, c1], axis=3)
    c9 = Conv2D(16, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(u9)
    c9 = BatchNormalization()(c9)
    c9 = Conv2D(16, (3, 3), activation='relu', kernel_initializer='he_normal', padding='same')(c9)
    c9 = BatchNormalization()(c9)
```

Fig. 2. U - Net model with Batch Normalization.

IV. RESULTS

The results obtained for different configurations of the model are presented in this section. I have used only 54 images out of 72 images (from tile 1 to tile 6) because of the limitations of the computational power of the computer used for the analysis.

A. Referred Model (without Batch Normalization)

Fig3 shows the plot of the training and validation loss for the epochs (taken 100 epochs). The loss is higher at initial stages and as the epochs increases the loss is decreasing. The Validation loss is higher than the Training loss, validation loss decreases even further if the dataset has more images.

Fig4 shows the plot of the training and validation IoU. IoU is the measure of the jaccard coefficient that is explained in the above section. IoU is the intersection over union or the similarity index or the jaccard coefficient.[5]

B. Proposed Model (with Batch Normalization)

Fig5 shows the plot of the training and validation loss with the Batch Normalization technique. The number of epochs are taken as 100 (same as above). The loss is quite less in this model and the convergence of the model is quick. Better results can be achieved with smaller number of epochs.

Fig6 shows the plot of the training and validation IoU with the Batch Normalization technique. I have trained the model for different number of epochs and presented the results in the table below.

C. Table with results

Results		
Model Configurations	Loss (%)	Accuracy (%)
Referred Model (100 epochs)	25.31	91.13
Proposed Model (100 epochs)	11.85	95.64
Proposed Model (30 epochs)	39.16	86.85
Proposed Model (50 epochs)	30.62	89.60

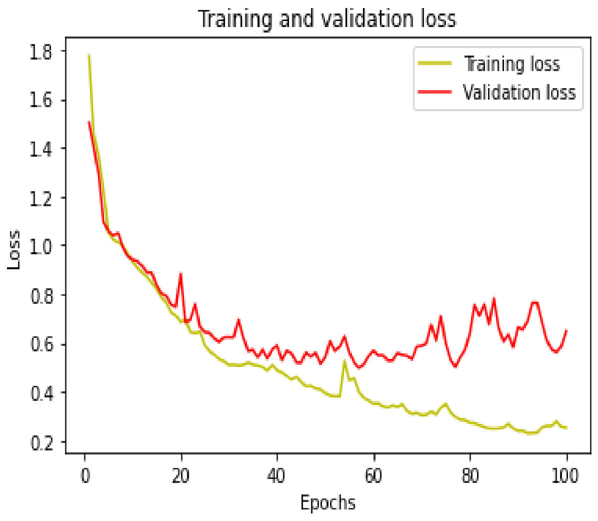


Fig. 3. Plot of Training and Validation loss of the model

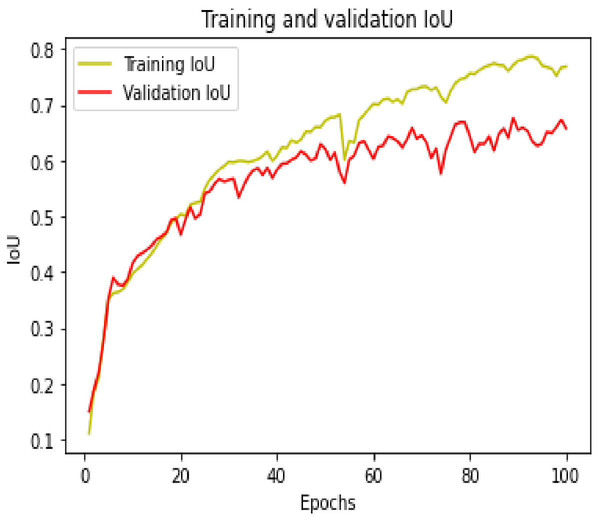


Fig. 4. Plot of Training and Validation IoU of the model

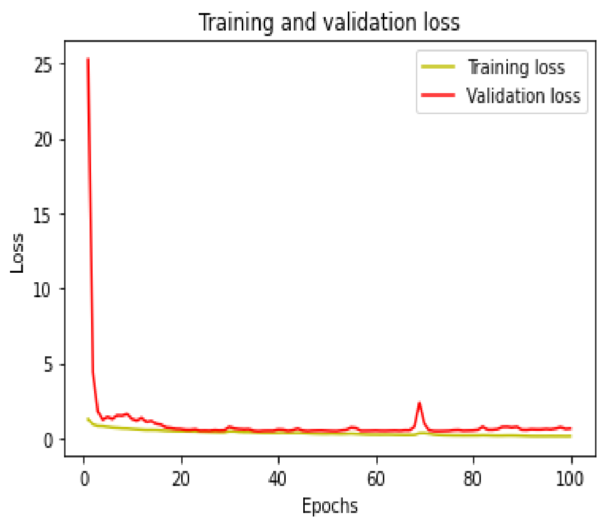


Fig. 5. Plot of Training and Validation loss of the model with Batch Normalization

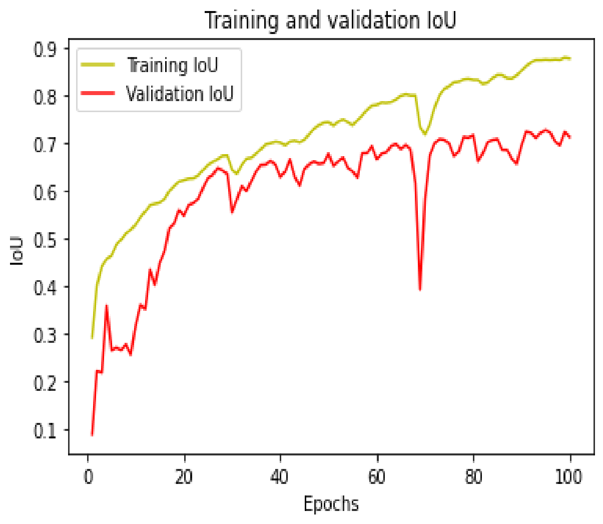


Fig. 6. Plot of Training and Validation IoU of the model with Batch Normalization

The results show how the loss and accuracy are after the end of training for the specified number of epochs. I have increased the number of epochs to 200 but the results were similar when compared with the 100 epochs for the referred model. I have used the early stopping technique to access the good number of epochs for the model where I have monitored loss and the training stops just after 100 epochs. So i am presenting the values up to 100 epochs.

V. CONCLUSION

From the table it is clear that the loss and accuracy are almost similar for the proposed model with 50 epochs and the referred model with 100 epochs. The training time for each epoch during the training is similar for both configurations thereby reducing the training time for the same amount of data used for training.

Fig 9,10,11 shows the predictions of the proposed model after the training. I have picked random images to check the performance of the model. These images are the results of the proposed model with just 30 epochs. The model is giving decent results for the images.

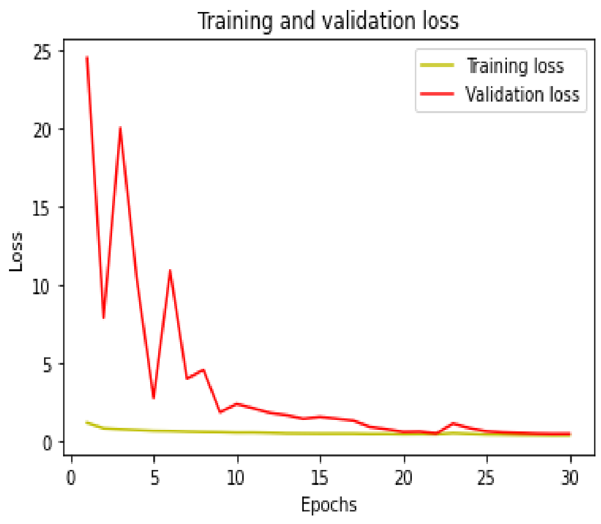


Fig. 7. Plot of Training and Validation loss of the model with Batch Normalization for 30 epochs

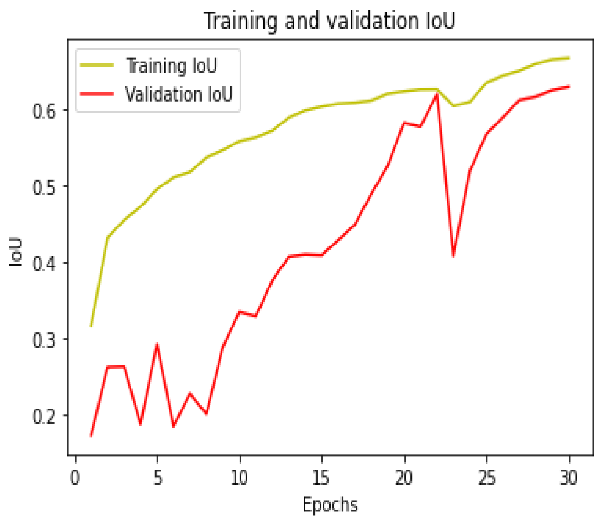


Fig. 8. Plot of Training and Validation IoU of the model with Batch Normalization for 30 epochs

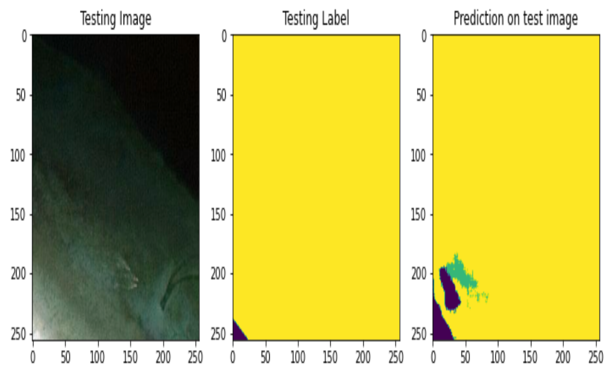


Fig. 9. showing the predictions of the model

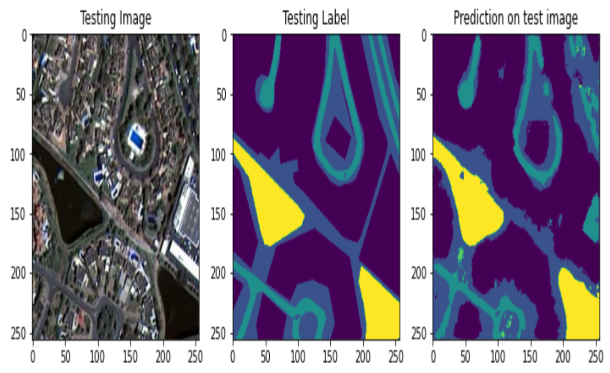


Fig. 10. showing the predictions of the model

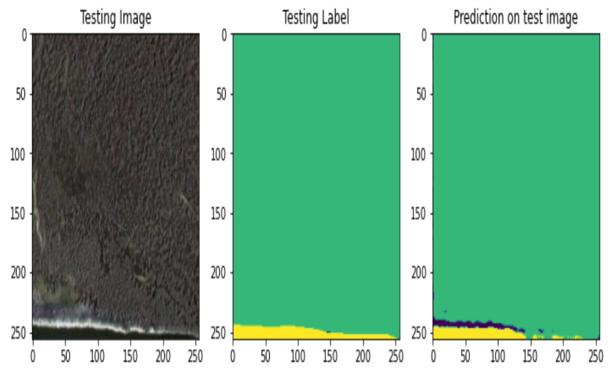


Fig. 11. Plot of Training and Validation IoU of the model with Batch Normalization

The results will definitely be better with more number of training images. Because of the computational limits I have on my computer I could not augment the images to increase the training set. The more deeper the model trains the more better are the results.

REFERENCES

[1] Olaf Ronneberger, Philipp Fischer, and Thomas Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," arXiv:1505.04597v1 [cs.CV]18 May 2015.

[2] TensorFlow - Batch Normalization.

[3] Semantic segmentation of aerial imagery dataset, Kaggle.

[4] Sergey Ioffe,Christian Szegedy,"Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," arXiv:1502.03167v3 [cs.LG] 2 Mar 2015.

[5] python for microscopists, 228 semantic segmentation of aerial imagery using unet.

[6] A Gentle Introduction to Early Stopping to Avoid Overtraining Neural Networks - Machine Learning Mastery.

[7] TensorFlow - EarlyStopping.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun "Deep Residual Learning for Image Recognition," arXiv:1512.03385v1 [cs.CV] 10 Dec 2015.

[9] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas and Aleksander Madry, "How Does Batch Normalization Help Optimization?," arXiv:1805.11604v5 [stat.ML] 15 Apr 2019.

[10] A Gentle Introduction to Batch Normalization for Deep Neural Networks - Machine Learning Mastery.

[11] PRIIT ULMAS and INNAR LIIV, "Segmentation of Satellite Imagery using U-Net Models for Land Cover Classification, " arXiv:2003.02899v1 [cs.CV] 5 Mar 2020.

[12] Shervin Minaee, Yuri Boykov, Fatih Porikli, Antonio Plaza, Nasser Kehtarnavaz, and Demetri Terzopoulos, "Image Segmentation Using Deep Learning: A Survey, " arXiv:2001.05566v5 [cs.CV] 15 Nov 2020

[13] Jaccard index - Wikipedia.

[14] Jaccard index - deepai.org.

[15] sklearn.metrics.jaccard$_{s}$*core*.