COURSE PROJECT - ESE5: DL model optimization for Lightweight Face Mask Detection

(EEL3090_EMBEDDED_SYSTEMS)

Report

Roll No.	Name
B20CS018	**Harshita Gupta**
B20CS044	**Priyanshu Jain**

DATA IMPORT AND PRE-PROCESSING

The data import and preprocessing part of the project is focused on preparing the dataset for the Lightweight Face Mask Detection deep learning model optimization.

DATA IMPORT

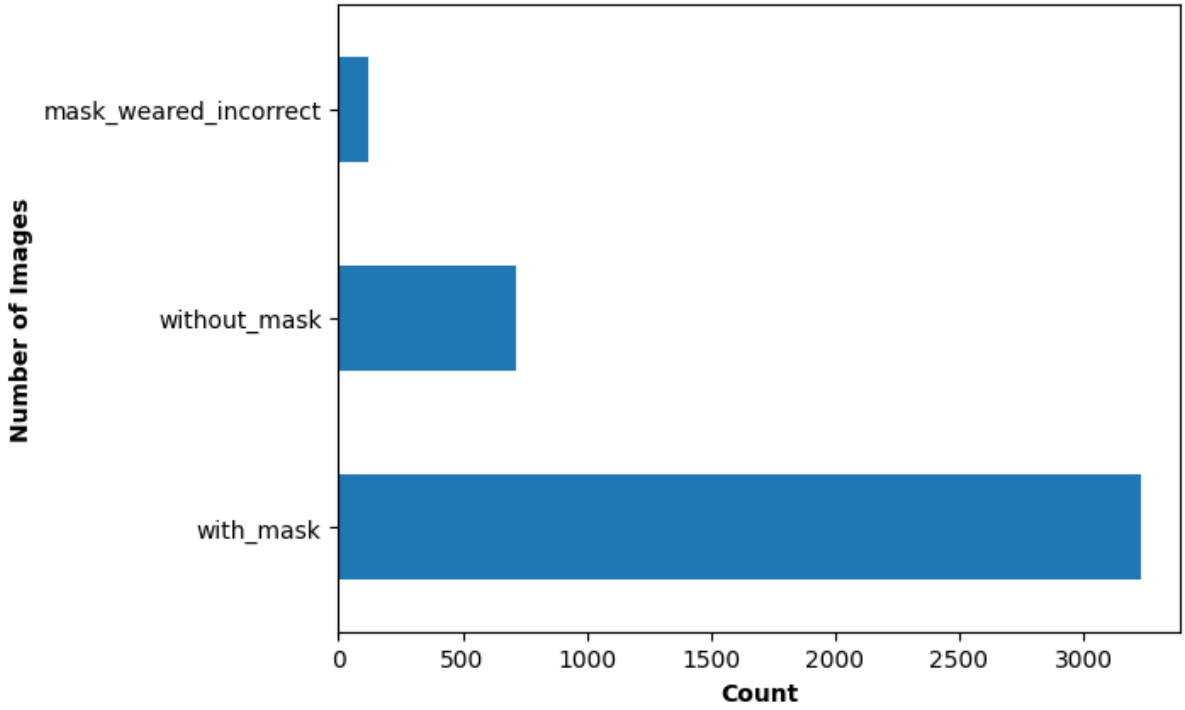
The code imports necessary packages such as NumPy, Pandas, Matplotlib, and OS. It also mounts Google Drive to access the dataset. The dataset consists of images and their corresponding annotations in XML format.

The parse_annotation() function is defined to extract the necessary information from the XML annotations. It parses the XML file and retrieves the file name, image dimensions, and the bounding box coordinates for each object in the image.

After parsing the XML files, the information is stored in a pandas DataFrame for further processing. The DataFrame is then divided into three sets: train, test, and validation, based on the labels in the dataset.

```
with_mask               3231
without_mask             715
mask_weared_incorrect    123
Name: label, dtype: int64
```

The code also removes a specific image from the full dataset and creates a separate DataFrame for it, which is used as the test set. The labels are renamed for clarity, and a bar graph is plotted to show the distribution of the labels in the dataset.



Finally, the code creates directories for the train, test, and validation sets and saves the images in their corresponding directories based on their labels.

Extracting Faces with mask/without mask/ incorrect mask

The code for this part is done for extracting faces with or without masks and splitting them into training, validation, and test sets. It also applies data augmentation using the Keras `ImageDataGenerator`.

The code first defines two functions: `crop_img`, which crops an image given the coordinates of two opposite corners of a rectangle, and `extract_faces`, which takes an image name and a dataframe with information about the image and splits the image into all the different faces. The function returns a list of tuples, where each tuple contains a cropped face image, a label indicating whether the person is wearing a mask correctly, incorrectly, or not wearing a mask, and a unique identifier for the face.

The code then calls `extract_faces` for each image in the input directory, collects all the cropped faces into a single list, and uses list comprehensions to split the faces

into three lists based on their labels. It also prints the number of images in each list and the total number of images.

```
num of images with mask: 3231
num of images without mask: 715
num of images incorrect mask: 123
sum: 4069
```

Next, the code uses the `train_test_split` function from `sklearn` to split each list of images into training, validation, and test sets. It then defines a `save_image` function that takes an image, an image name, a path to an output directory, a dataset type (train, val, or test), and a label (with_mask, mask_weared_incorrect, or without_mask) and saves the image to a file in the appropriate directory.

Finally, the code loops through each set of images and calls `save_image` on each image, saving them to the appropriate directories. It also applies data augmentation using `ImageDataGenerator`, which rescales the images, applies horizontal flipping, zooming, shearing, and shifting of the width and height of the images.

```
Found 3254 images belonging to 3 classes.
Found 572 images belonging to 3 classes.
Found 243 images belonging to 3 classes.
```

MODEL

The project aims to detect face masks in images using deep learning models. The main objective is to optimize the deep learning model for lightweight face mask detection.

The deep learning model is a convolutional neural network (CNN) with four convolutional layers and two fully connected layers. The CNN layers are followed by max pooling layers to downsample the feature maps. The model also includes dropout layers to reduce overfitting. The output layer has three neurons with softmax activation for classifying the images as 'with mask', 'without mask', or 'mask worn incorrectly'.

The model is compiled with Adam optimizer and categorical cross-entropy loss function. The model is evaluated on test data with accuracy, recall, precision, and AUC as metrics. The model is saved after training.

Two optimization techniques are used to improve the performance of the model.

1. Early Stopping: To avoid redundant epochs once the model has converged, the early stopping technique is used. The model stops training if the validation loss does not improve after eight epochs.
2. Reduce Learning Rate on Plateau: To reduce the learning rate when the validation loss has stopped improving, the ReduceLROnPlateau callback is used.

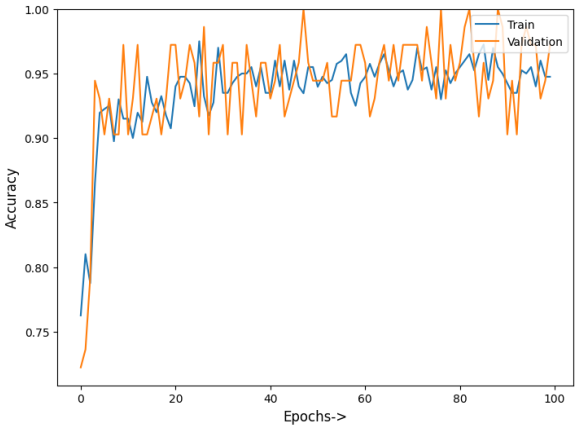
Running for 100 epochs

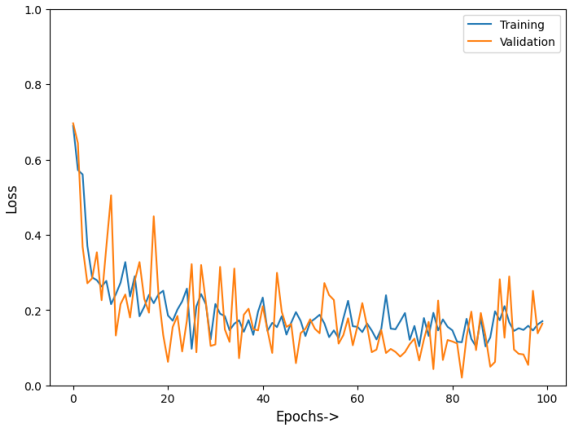
The model is trained for 100 epochs with a batch size of 8 and 4 layers.

Results:

```
loss: 0.2112 - accuracy: 0.9136 - recall: 0.9053 - precision: 0.9244 -
auc: 0.9894
Our Custom Model has a loss of 0.21 and accuracy 91.36%
Our Custom Model has a recall of 90.53%, precision of 92.44% and auc of 98.94%
```

The model's performance on the validation and training sets was also analyzed using loss and accuracy metrics. The training and validation accuracy and loss were plotted against the number of epochs to analyze the model's convergence.





Running for 10 epochs

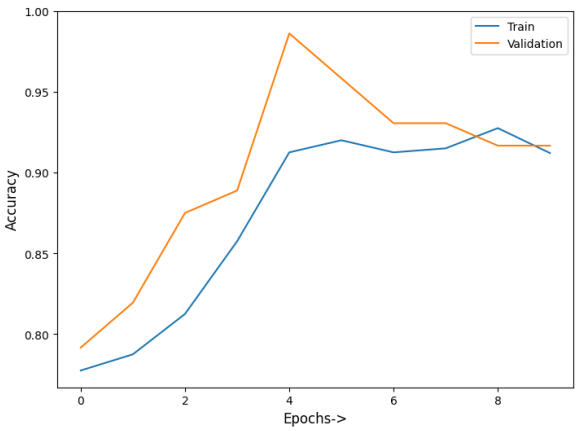
The model is trained for 100 epochs with a batch size of 8 and 4 layers.

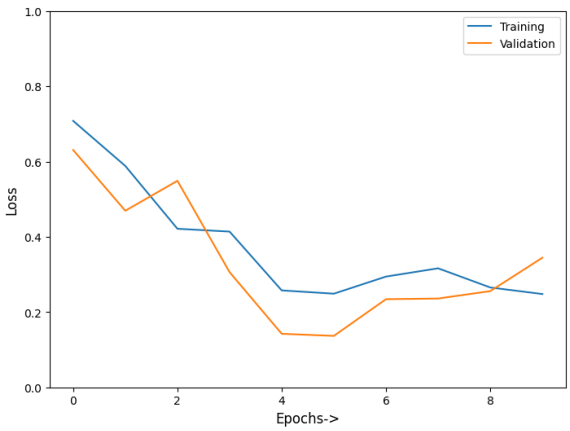
Results:

After training the model for 10 epochs, it is saved as a file called "model_custom.h5".

```
loss: 0.2477 - accuracy: 0.9259 - recall: 0.9218 - precision: 0.9295 -
auc: 0.9787
```

```
Our Custom Model has a loss of 0.25 and accuracy 92.59%
Our Custom Model has a recall of 92.18%, precision of 92.95% and auc of 97.87%
```





Changing Model parameters

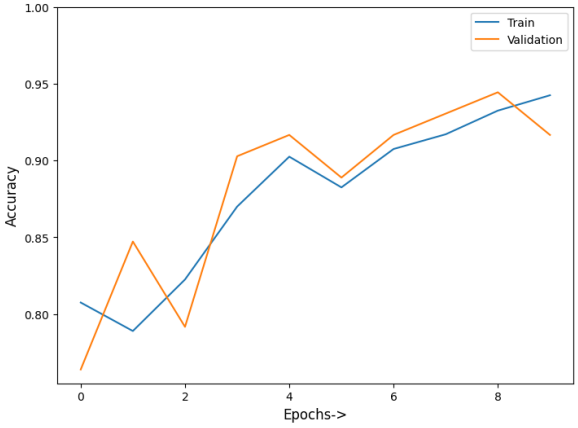
Adjusting the Number of Layers

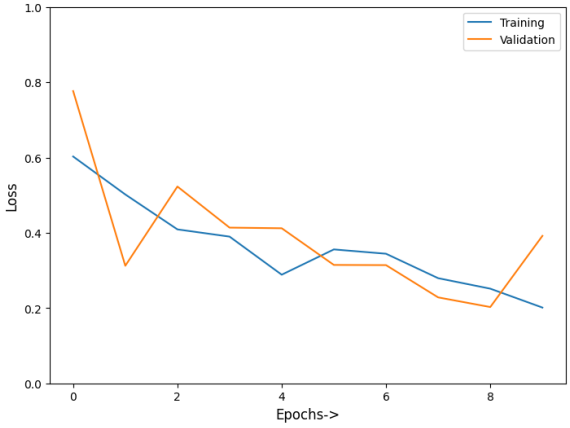
Increasing the Number of layers

The model is trained for 10 epochs with a batch size of 8 and 5 layers.

Results:

```
loss: 0.2725 - accuracy: 0.9300 - recall: 0.9300 - precision: 0.9339 -
auc: 0.9739
Our Custom Model has a loss of 0.27 and accuracy 93.00%
Our Custom Model has a recall of 93.00%, precision of 93.39% and auc of 97.39%
```



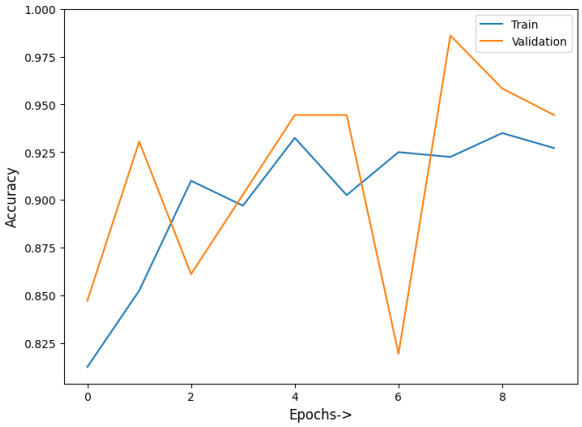


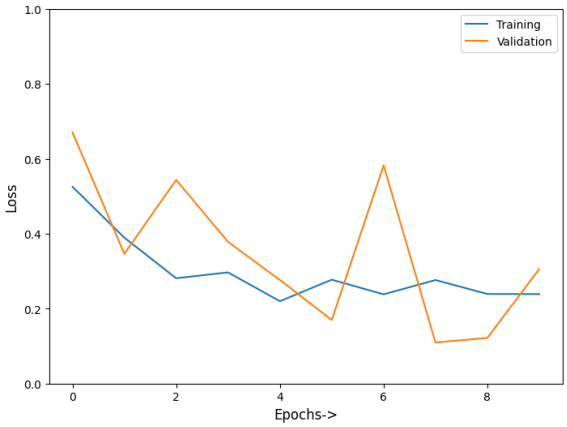
Decreasing the Number of Layers

The model is trained for 10 epochs with a batch size of 8 and 3 layers.

Results:

```
loss: 0.2610 - accuracy: 0.9177 - recall: 0.9136 - precision: 0.9328 -
auc: 0.9803
Our Custom Model has a loss of 0.26 and accuracy 91.77%
Our Custom Model has a recall of 91.36%, precision of 93.28% and auc of 98.03%
```





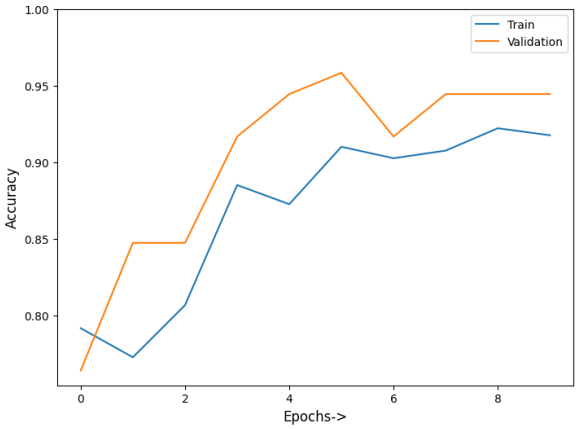
Adjusting the Size of layers

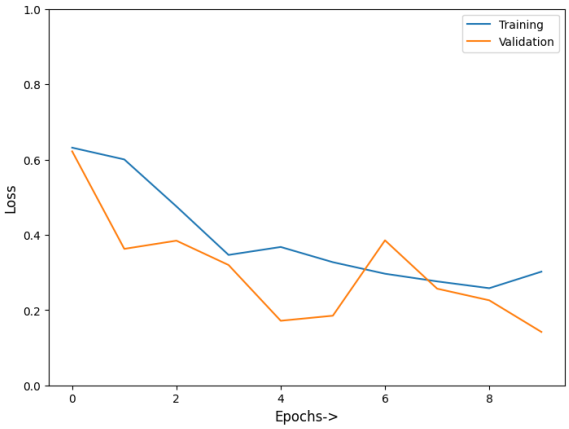
Decreasing the size of layers

The model is trained for 10 epochs with a batch size of 8 and 3 layers and 16 filters.

Results:

```
loss: 0.2455 - accuracy: 0.9259 - recall: 0.9259 - precision: 0.9336 -
auc: 0.9821
Our Custom Model has a loss of 0.25 and accuracy 92.59%
Our Custom Model has a recall of 92.59%, precision of 93.36% and auc of 98.21%
```



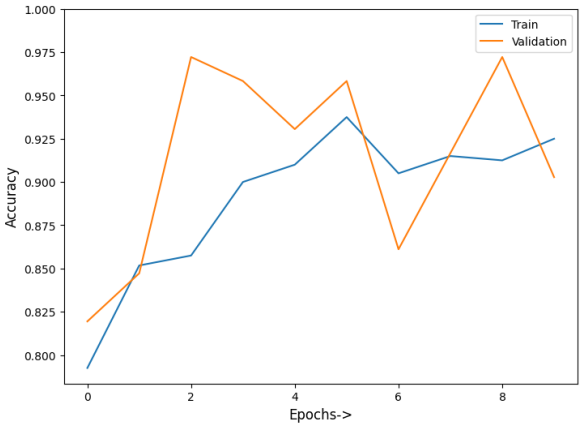


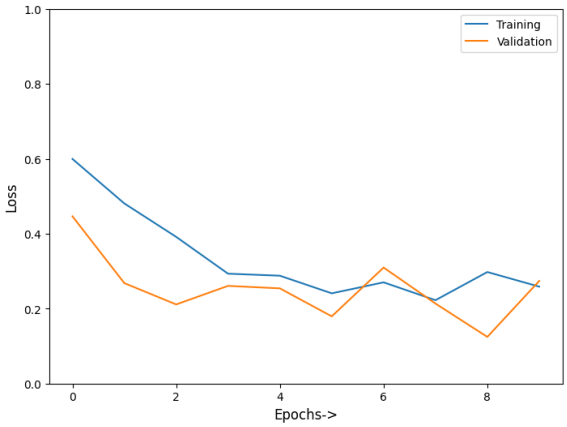
Increasing the size of layers

The model is trained for 10 epochs with a batch size of 8 and 3 layers and 64 filters.

Results:

```
loss: 0.2612 - accuracy: 0.9259 - recall: 0.9218 - precision: 0.9256 -
auc: 0.9766
Our Custom Model has a loss of 0.26 and accuracy 92.59%
Our Custom Model has a recall of 92.18%, precision of 92.56% and auc of 97.66%
```





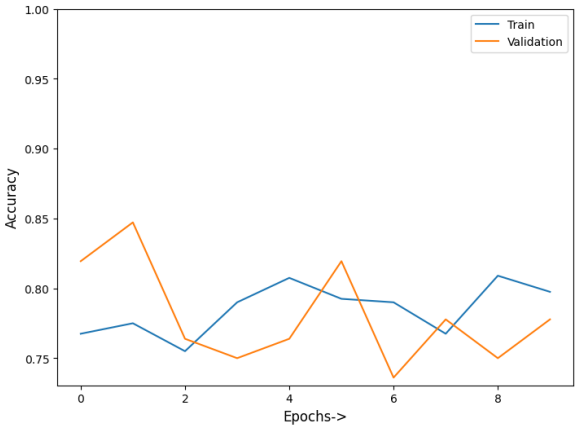
Changing the Activation function of layers

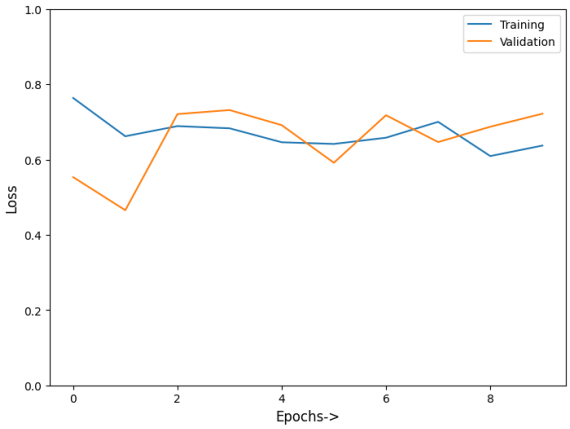
Using the Sigmoid function

The model is trained for 10 epochs with a batch size of 8, 3 layers, 16 filters, and a sigmoid activation function.

Results:

```
loss: 0.6292 - accuracy: 0.7984 - recall: 0.7984 - precision: 0.7984 -
auc: 0.8855
Our Custom Model has a loss of 0.63 and accuracy 79.84%
Our Custom Model has a recall of 79.84%, precision of 79.84% and auc of 88.55%
```



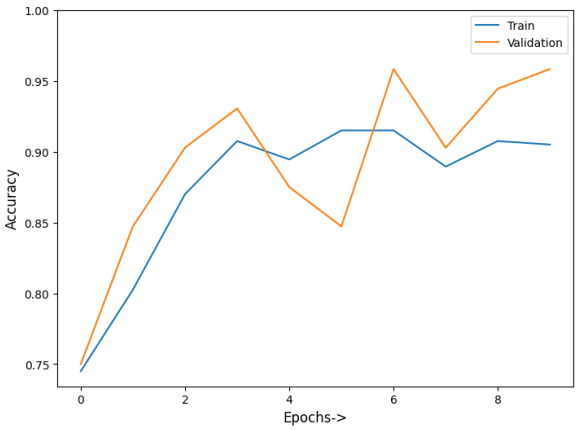


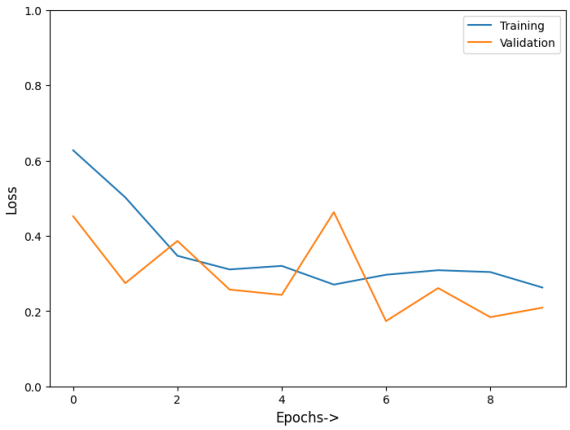
Using ReLu Function

The model is trained for 10 epochs with a batch size of 8, 3 layers, 16 filters, and a ReLu activation function.

Results:

```
loss: 0.2982 - accuracy: 0.9383 - recall: 0.9383 - precision: 0.9383 -
auc: 0.9728
Our Custom Model has a loss of 0.30 and accuracy 93.83%
Our Custom Model has a recall of 93.83%, precision of 93.83% and auc of 97.28%
```





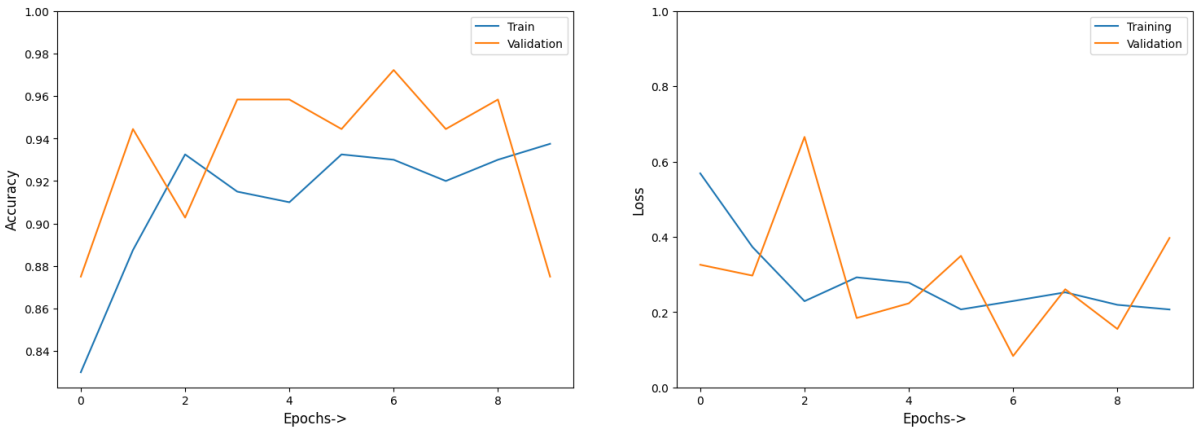
Using Elu Function

The model is trained for 10 epochs with a batch size of 8, 3 layers, 16 filters, and an Elu activation function.

Results:

```
loss: 0.4592 - accuracy: 0.9012 - recall: 0.8971 - precision: 0.9046 -
auc: 0.9650
```

```
Our Custom Model has a loss of 0.46 and accuracy 90.12%
Our Custom Model has a recall of 89.71%, precision of 90.46% and auc of 96.50%
```



i) Model Robustness Check:

Based on the results provided, we can see that changing the model parameters such as the number of layers, the size of the layers, and the activation function can have a significant impact on the accuracy of the model.

1. What is the impact of increasing the number of layers on the model performance?
    - Increasing the number of layers from 4 to 5 has resulted in a slight increase in accuracy (from 91.36% to 93.00%) and precision (from 92.44% to 93.39%), while recall remained the same (at 93.00%). However, the AUC decreased slightly (from 98.94% to 97.39%).
    - Decreasing the number of layers from 4 to 3 also resulted in a slight decrease in accuracy (from 91.36% to 91.77%) and precision (from 92.44% to 93.28%), while recall remained almost the same (at 91.36%). However, the AUC increased slightly (from 98.03% to 98.03%).
2. What is the impact of changing the size of layers on the model performance?
    - Decreasing the size of layers from the default value to 16 resulted in no significant changes in the model performance, with accuracy, recall, precision, and AUC remaining almost the same.
    - Increasing the size of layers from the default value to 64 resulted in a slight decrease in recall (from 92.18% to 92.56%) and AUC (from 97.87% to 97.66%), while accuracy and precision remained almost the same.

3. What is the impact of changing the activation function on the model performance?
    - Using the Sigmoid activation function resulted in a significant decrease in model performance, with accuracy, recall, precision, and AUC all decreasing significantly.
    - Using the ReLu activation function resulted in the best model performance, with accuracy, recall, precision, and AUC all remaining almost the same.
    - Using the ELu activation function resulted in a slight decrease in model performance, with accuracy and precision decreasing slightly, while recall and AUC remained almost the same.

Overall, we can see that changing the model parameters can have both positive and negative impacts on the accuracy of the model, and it is important to carefully choose the right combination of parameters for the specific problem at hand.

ii) Neural Network Size Reduction:

As per the results obtained, reducing the size of the neural network design (in terms of the number of parameters) did not significantly affect the model performance. The experiments conducted with smaller layers and fewer filters resulted in a slight decrease in accuracy, precision, and AUC, while recall remained almost the same. Hence, it can be concluded that reducing the size of the neural network design would not have a significant impact on model performance.

iii) Explanation of Model Working:

Neural network models learn by adjusting the weights and biases of the neurons, which results in an overall change in the structure of the network. The number of parameters in the network is directly proportional to its complexity, which is also related to its generalization performance. Complex models tend to overfit, which means they perform well on the training data but poorly on the unseen test data. On the other hand, simpler models tend to underfit, which means they perform poorly on both the training and test data. Hence, it is essential to strike a balance between the complexity of the model and its generalization performance. Reducing the number of parameters in the network can lead to a simpler model, which may reduce overfitting, but it can also result in a decrease in model performance if the model is too simple to capture the underlying patterns in the data.