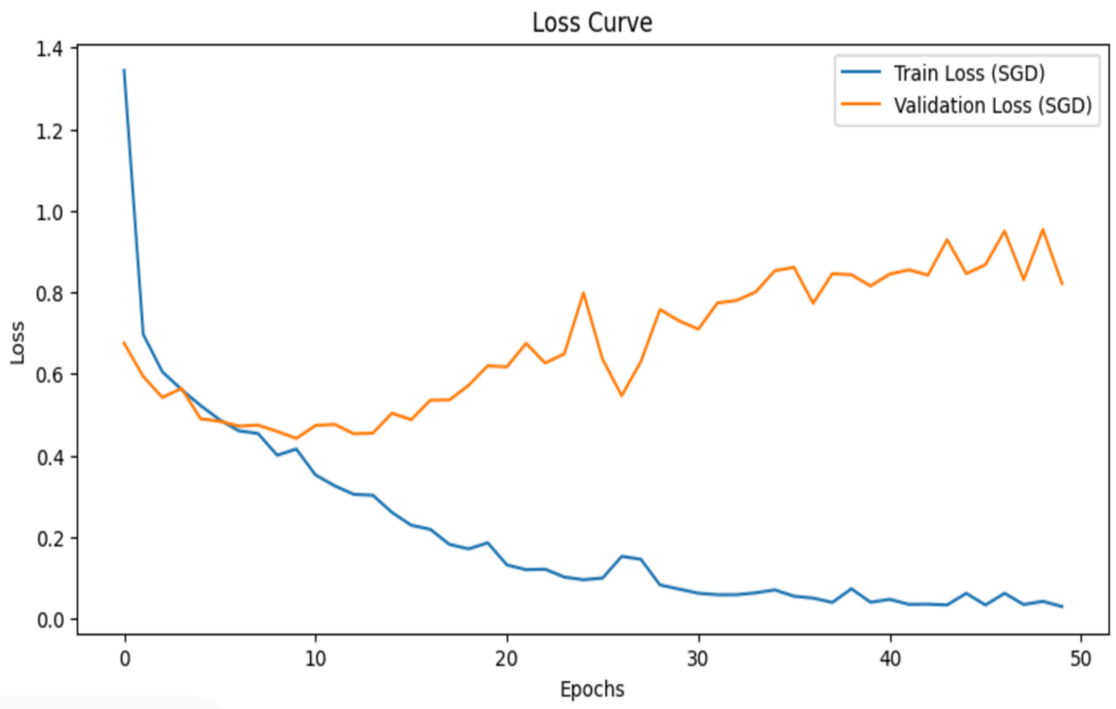
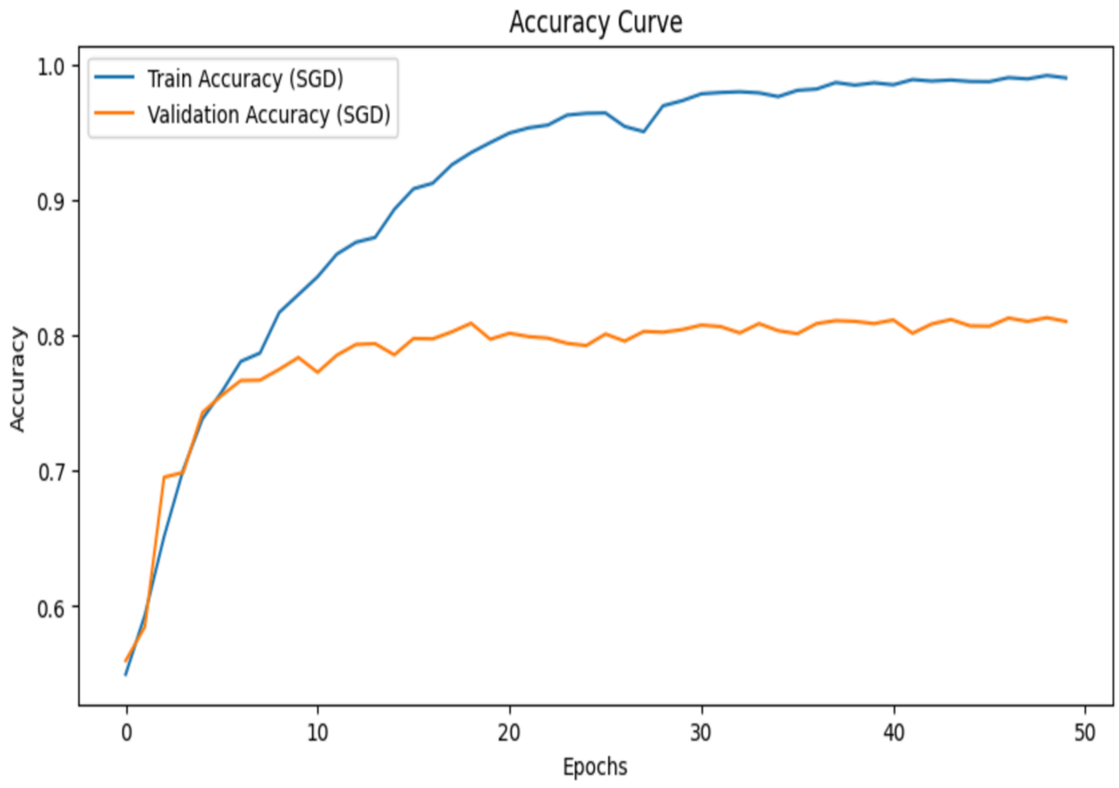
Results

SGD;

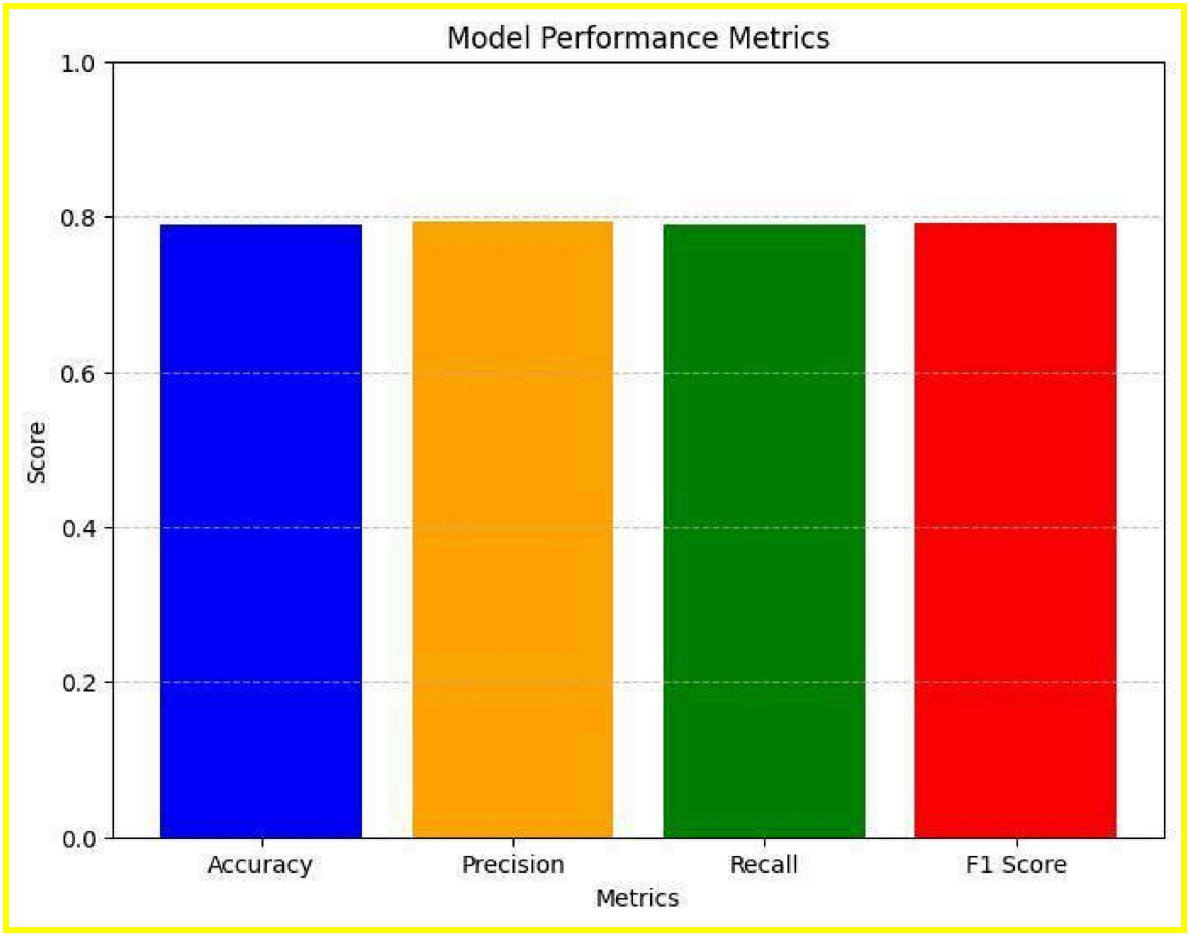


Loss Curve
Train Loss (SGD)
Validation Loss (SGD)
Loss
Epochs



Accuracy Curve
Train Accuracy (SGD)
Validation Accuracy (SGD)
Accuracy
Epochs

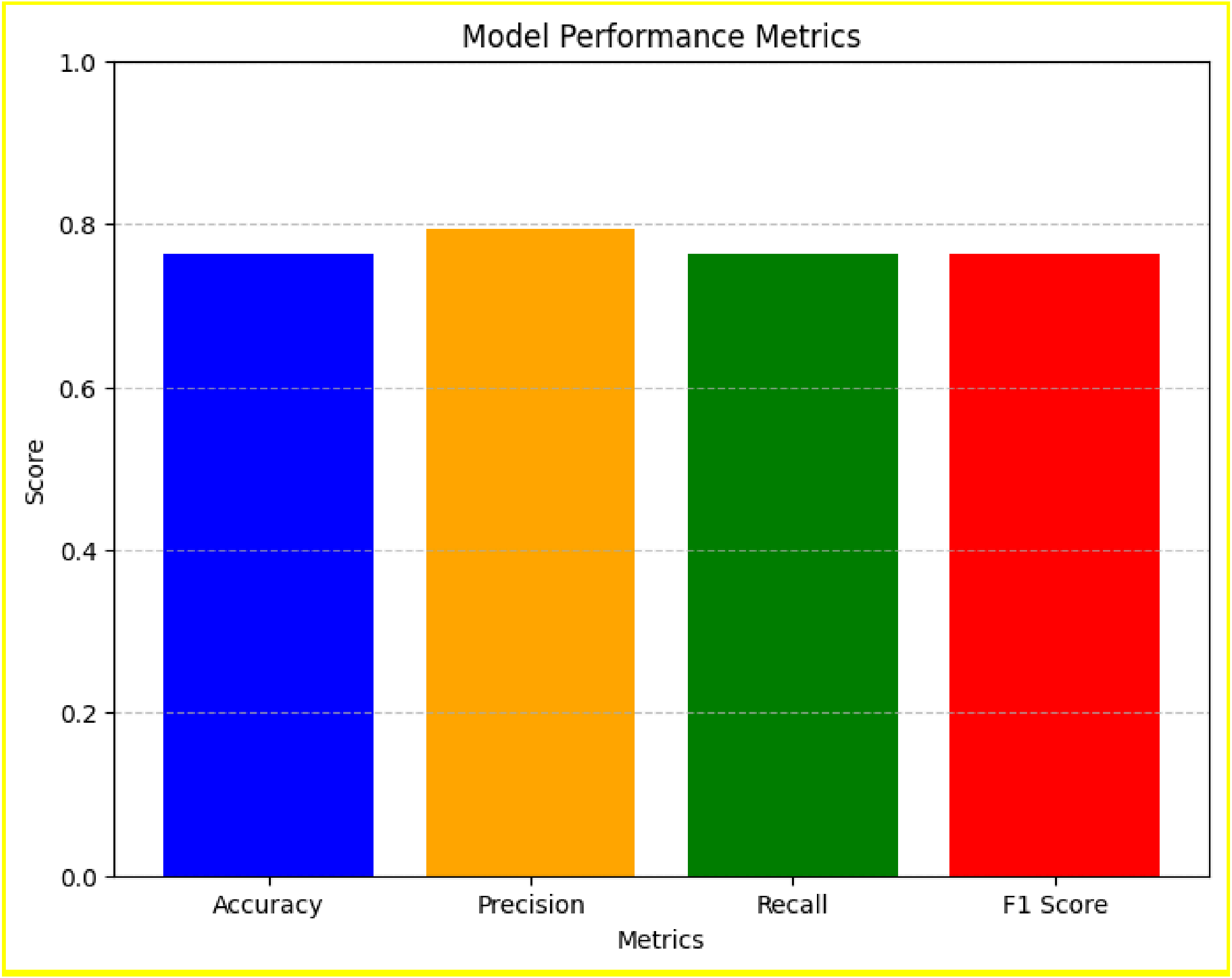
We also used other precision, recall, and F1-score to plot the performance of the different iterations of the model.



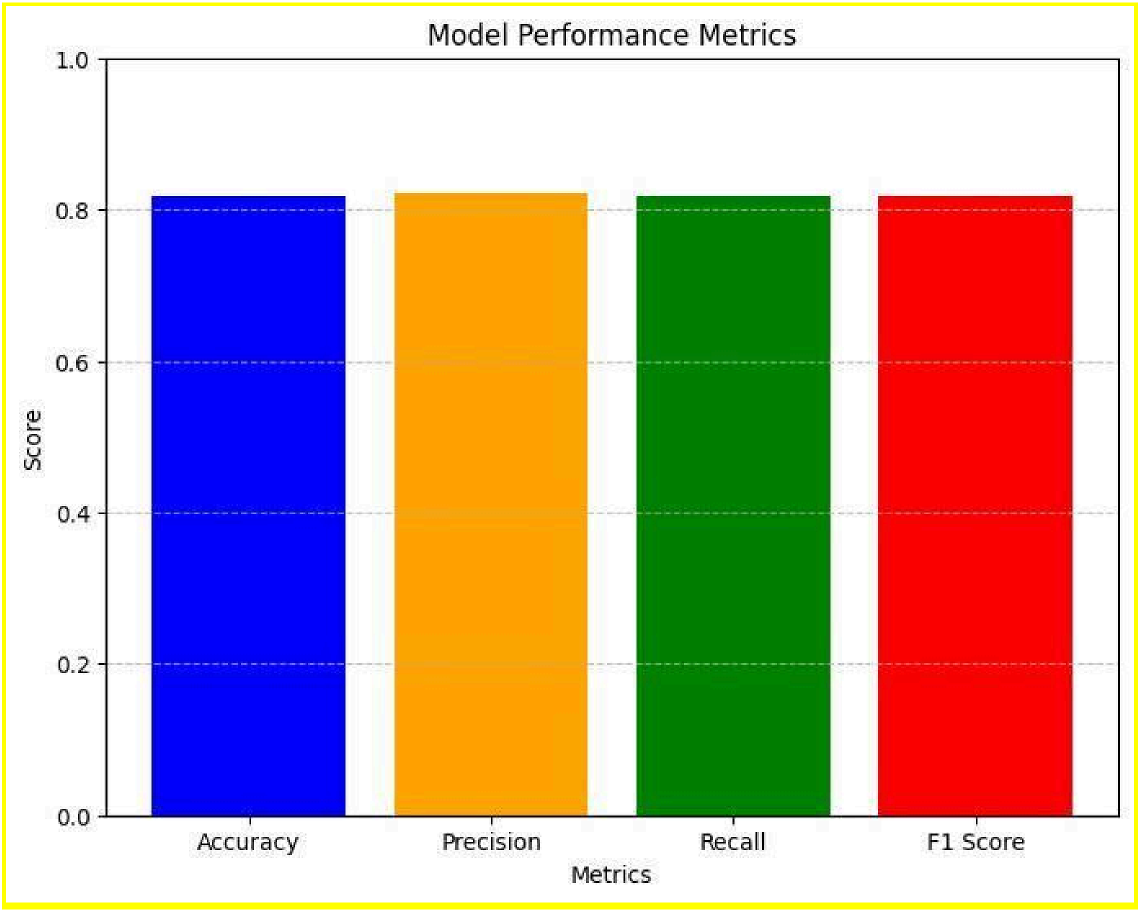
Probable overfitting.

Further augmentation was implemented to improve the model’s performance under SGD optimizer.

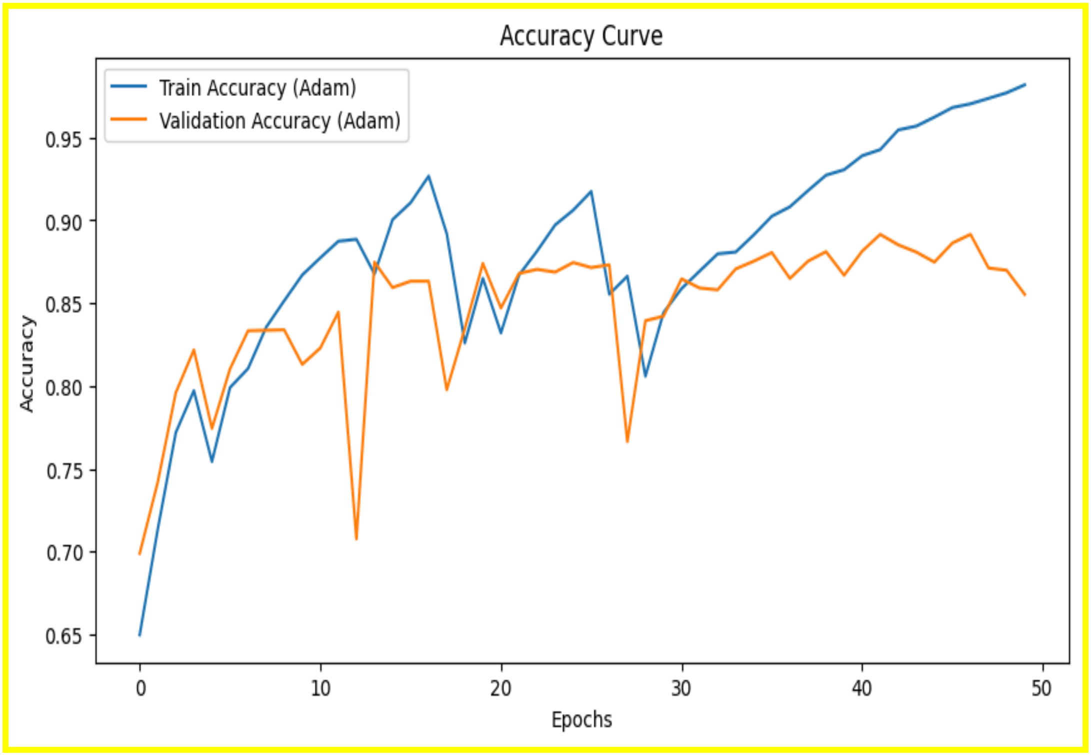
SGD with additional augmentations:

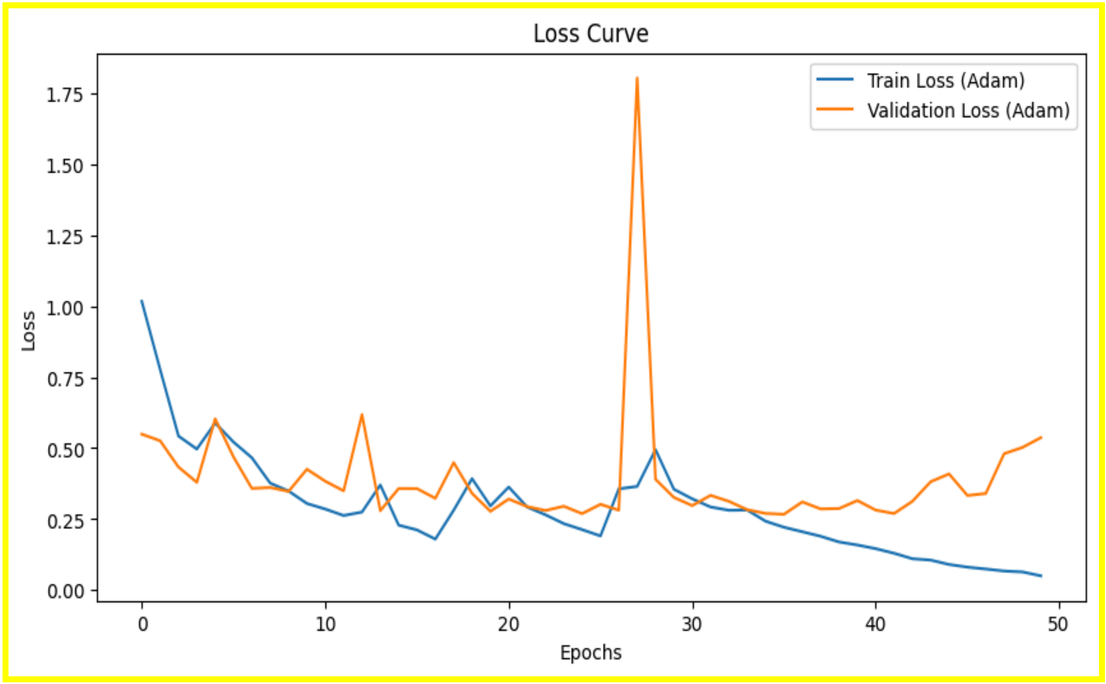


SGD with Focal Loss and augmentation:



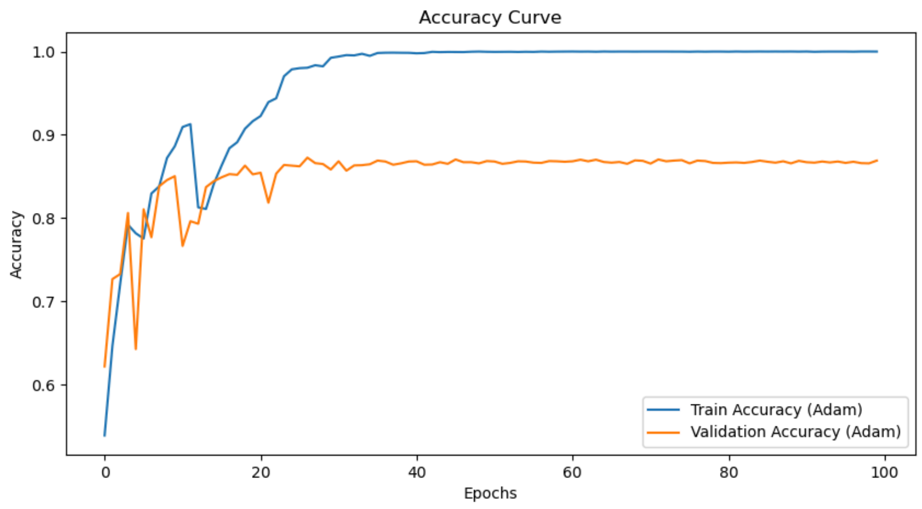
<u>Adam:</u>

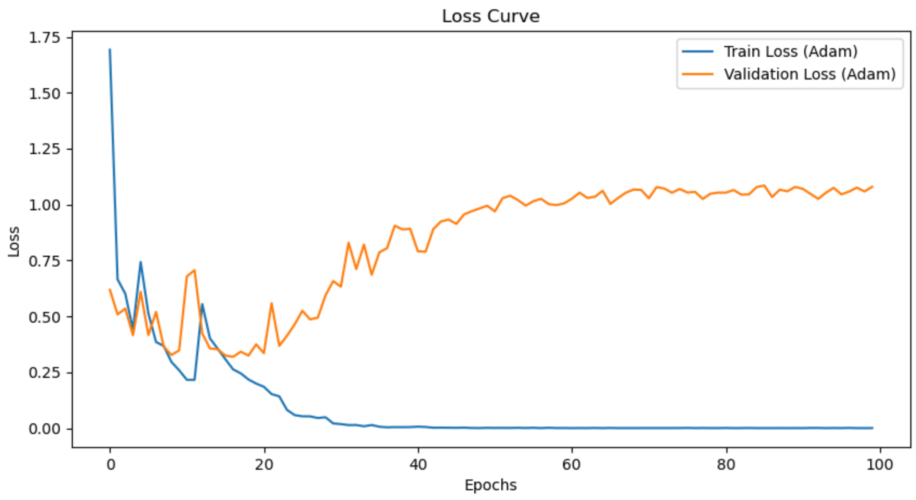




It seems that even Adam, after the 30th epoch fails to achieve any better result and starts overfitting. Curiously, just before the validation loss begins rising against the decrease in train loss, we have an outlier around the epoch of interest.

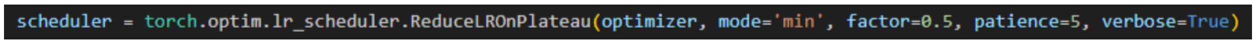
Alternative Results:
Batchsize = 128 , **Optimizer** = Adam, **Initial Learning Rate** = 0.001, **Epoch** = 100.





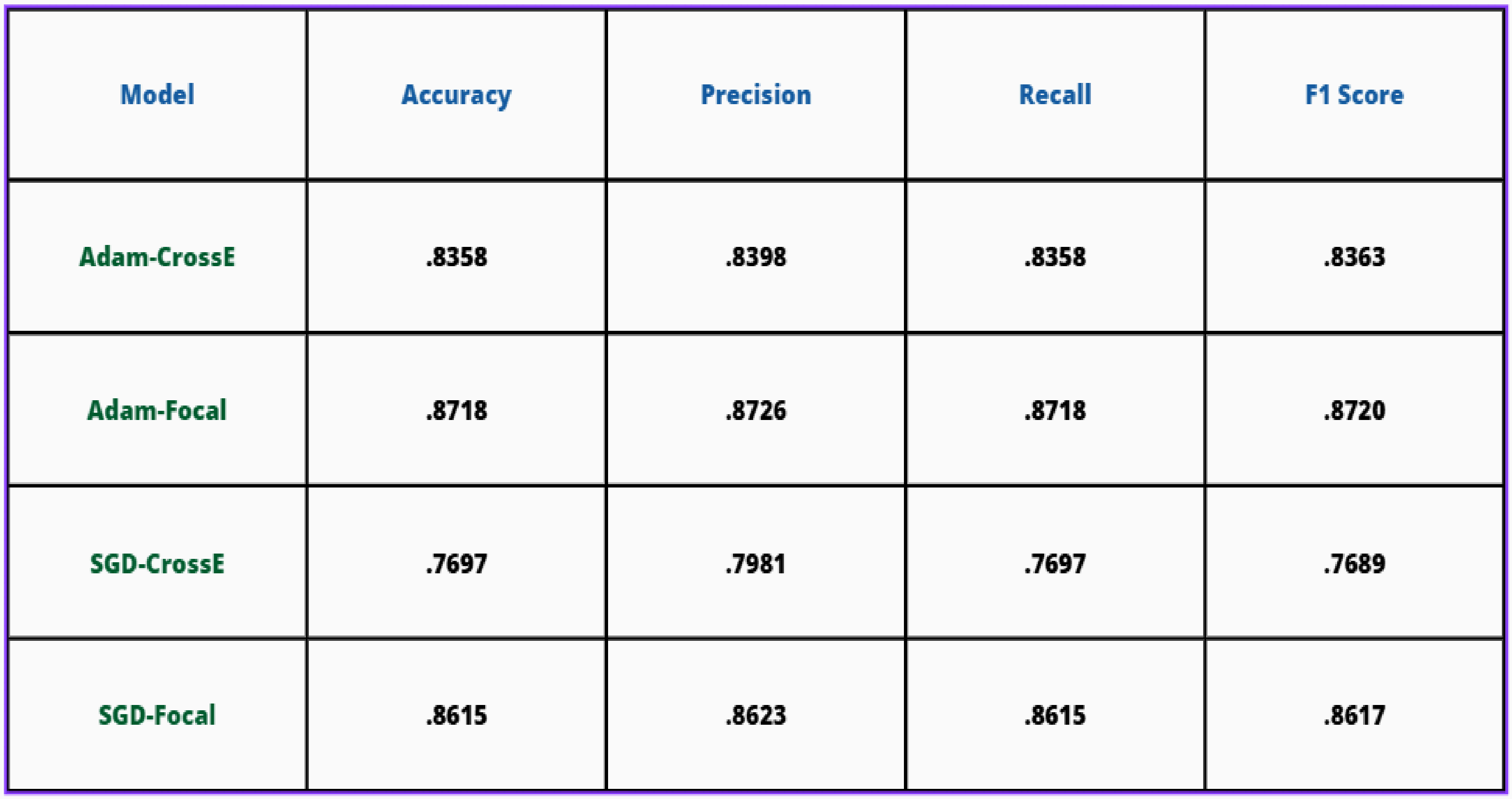
In these tweaked settings, training accuracy rapidly improves and manages to reach close to 99% around 25 epochs. The validation accuracy stabilizes around 87% after around 20 epochs and remains consistent after the training.

The scheduler using ***ReduceLROnPlateau*** somehow helped maintain training efficiency by reducing the learning rate when the validation loss plateaued, the model avoided drastic updates thus maintaining stable convergence. Validation accuracy of around 87% suggests good generalization, although **further improvement might be possible** with techniques like data augmentation or fine-tuning.



```
scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer, mode='min', factor=0.5, patience=5, verbose=True)
```

Overall Score Results:



Model	Accuracy	Precision	Recall	F1 Score
Adam-CrossE	.8358	.8398	.8358	.8363
Adam-Focal	.8718	.8726	.8718	.8720
SGD-CrossE	.7697	.7981	.7697	.7689
SGD-Focal	.8615	.8623	.8615	.8617

Use and Testing

As mentioned at point 4 of ***Implementation Plan***, we used a readily preprocessed dataset of images and split it into train, test and evaluation sets:

https://www.kaggle.com/datasets/belkhirnacim/textiledefectdetection/data

Made of grayscale images formatted into csv and h5 (HDF5) files.

To verify the effectiveness of the model on images of textiles we used 3 different sets of pictures:

- Fabric in an industrial context, TILDA400: https://www.kaggle.com/datasets/angelolmg/tilda-400-64x64-patches?resource=download
- 2500 open-source fabric-fault images ZJU leaper: https://github.com/nico-zck/ZJU-Leaper-Dataset
- Few low-resolution pictures of personal clothing taken with our phones.

E.g.

Unsurprisingly, the model is not precise in determining which samples are good textile when the pictures provided have subpar lighting and resolution, and all clothes were categorized as 1 (“Bad Textile”). In order to classify correctly, HD images taken at close range and with evenly distributed lighting are necessary.

When tested on additional pre-labeled datasets the model is still unable to accurately predict the correct class.