**1. AIMS & OBJECTIVES:**

The aim of this project is to develop two Convolutional Network models, a breast cancer prediction model and a blood cell classifier that can identify 8 distinct types of blood cells, for medical diagnosis. I was required to use BloodMNIST and BreastMNIST datasets publicly available at [https://medmnist.com](https://medmnist.com/), to train multiple Convolutional Neural Networks (CNNs), using a combination of self-designed CNN architectures and pretrained networks like VGG16 (used in blood cells classifier model). To address class imbalance issues, I proposed a solution in our report in the implementation of Breast Cancer Model using synthetic data.

The end goal is to create a web interface that allows users to upload medical data, choose the among the two classifiers (breast cancer or blood cell classifier) and receive a prediction on whether they have breast cancer or identify the type of blood cell based on the model selected. To ensure the reliability and generalization of our models, I have extensively evaluated our model’s performance using classification metrics such as Precision, Recall, F1-Score and Weighted accuracies.

**2. OVERVIEW OF THE DATASETS:**

**2.1. BREASTMNIST:**

The BreastMNIST data set is publicly available at [https://medmnist.com](https://medmnist.com/), has two classes. The data is splitted into Train/Test/Validation splits of 546/156/78 image data respectively. Each image is a grey scale image of 28x28 pixels. The distribution of classes across all sets is as follows:

* Class 0: 27% of data
* Class 1: 73% of data

**2.2. BLOODMNIST:**

The Blood MNIST dataset, available at [https://medmnist.com](https://medmnist.com/), comprises 8 classes, and is divided into Train/Test/Validation splits of 11959/3421/1712 image data respectively. Each image in the dataset has a pixel size of 28x28 and 3 colour channels (R, G, and B). The distribution of classes across all sets is as follows:

* Class 0: 7% of data
* Class 1: 18% of data
* Class 2: 9% of data
* Class 3:17% of data
* Class 4: 7% of data
* Class 5: 8% of data
* Class 6: 19% of data
* Class 7: 14% of data

**3. MODEL WISE ANALYSIS OF CONVOLUTIONAL NEURAL NETWORKS IMPLEMENTED:**

I will now discuss our model development process. It is divided into two broader categories named after each model; Breast Cancer Model and Blood Cell Classifier Model.

In each of these two categories I will discuss, Data Pre-processing steps, Model Architecture, Model Evaluation, and then finally the best model. Let us now begin with Breast Cancer Model in section 3.1 and then I will discuss Blood Cell Classifier in section 3.2.

**3.1. BREAST CANCER MODEL**

**3.1.1. DATA PREPROCESSING**

Pre-processing is critical for training any machine learning model. In our case, I took three pre-processing steps for optimal model performance. Note that each step is designed for specific approaches that I have used to generate three different Convolutional Neural Networks. Step 1 is common for all 3 cases, while the remaining two pre-processing steps are for other two models.

In first step, I added a dimension to the image for grayscale channel. Since our model expects a 3D input, I added an extra dimension to represent the channel. I then normalized our image data. Normalizing the image data reduced the scale from 0-255 to 0-1, ensuring that the model assigned the correct weights. This technique is widely followed and has proven impact on model’s performance. The resultant image array generated from this step was fed into our one of our models, and further pre-processing.

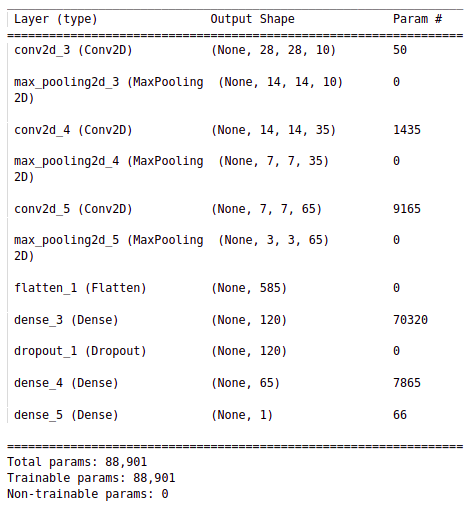
In second step, I further pre-processed the data produced from step 1, and induced variations in it using an image generator. I induced 11% random horizontal shifts (11% of total width), random vertical shifts to images (18% of total height), and random horizontally flipped images. I introduced these variations to see if they improve the model's performance by making it more robust to variations in the input.

In the last step, I generated synthetic data from the validation set’s minority class (class 0) by adding random noise to it. I then used this slightly modified validation data, containing only the minority class, and added it to training data. This was done to improve the class imbalance issue in the training set. This technique improved the occurrence of the minority class in training dataset from 27% to 30%. In the section below I will discuss how this slight improvement in class ratio resulted in significant increase in performance of our target class.

In summary, I used three different techniques to pre-process data for the CNN model. First, Normalized the data and added a channel for grayscale, Second augmented the dataset with variations, and Third addressed the class imbalance issue by generating synthetic data.

**3.1.2. MODEL ARCHITECTURE**

Three different approaches were used for the Breast Cancer Model, all with the same architecture. Only modification I proposed was in data pre-processing steps. It is important to add that I tried multiple models with each of approach however the model architecture I finalised proved to be best among all other architectures.

The proposed model architecture looks like:

*Fig. 3.1: Self Designed CNN architecture*

The architecture consists of an input layer for grayscale images of size 28x28, three sets of Convolutional and MaxPooling layers with increasing filter sizes of 10, 35, and 65, respectively, a Flatten layer, two Dense layers with ReLU activation functions, a Dropout function with a rate of 0.4, and an output layer with a single neuron and a sigmoid activation function for binary classification.

This CNN architecture is designed to extract important features from the input image through the Convolutional layers and retain these features, whilst reducing dimension, using MaxPooling. The Flatten layer transforms the output from the Convolutional layers to a 1D vector, which is then passed through two Dense layers with ReLU activation functions. The Dropout function is used to prevent overfitting and increase the model's generalization. Finally, the output from the second Dense layer is passed through a single neuron with a sigmoid activation function to predict the binary classification of the input image.

To train and evaluate the model, I used the Binary Cross entropy loss function, Adam optimizer, and accuracy metric.

In summary, the CNN architecture I implemented consists of several layers and methods that work together to extract important features (using filters) from the input images and then predict the binary classification accurately.

**3.2.3. MODELS EVALUATION RESULTS**

As discussed above in section 3.1.1, I applied three different approaches to obtain the model with optimal performance. I named these approaches as:

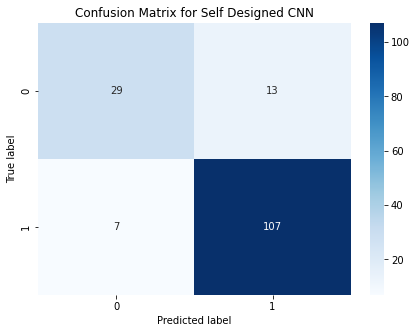
* *Self-Designed Convolutional Neural Network*
* *Self-Designed Convolutional Neural Network with Augmented Data*
* *Self-Designed Convolutional Neural Network with Synthetic Data*

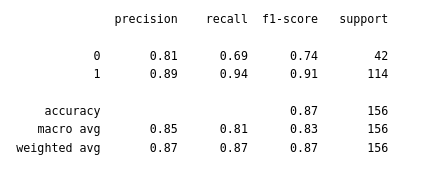
Let us now discuss evaluation results of each of these model.

Before proceeding, let's provide an overview of the classification metrics I used to evaluate our model's performance. I began with simple accuracy to obtain a basic idea of the model's performance. However, for more comprehensive evaluation, I used the confusion matrix, a valuable metric that allowed us to thoroughly analyze the classification results. Additionally, I employed the classification report, which includes precision, recall, F1 score, and balanced accuracy. Together, these metrics provide a holistic view of the model's performance.

**3.1.3.1. SELF DESIGNED CONVOLUTIONAL NEURAL NETWORK**

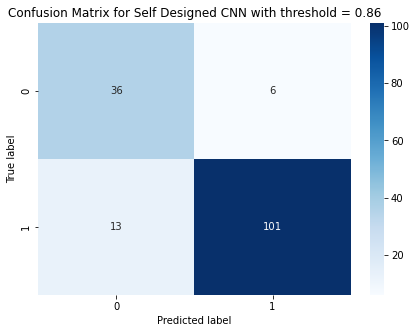
The data used for our model is simple pre-processed data in step 1 discussed in section 3.1.1. The results obtained are shown below.

*Fig. 3.2: Confusion Matrix for self-designed CNN*

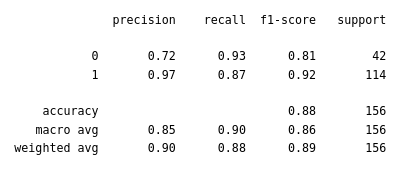
*Fig. 3.3: Classification Report for self-designed CNN*

The results are not as satisfactory. Especially when it comes to recall of target class (Class 0). Meaning that our model is only able to catch 69% of Cancer patients. One way to improve the precision score would be to vary our decision threshold but that would come at the expense of precision of target class and recall on non-target class. However, given the nature of our case, it is better to prioritize recall over precision to avoid missing potential cancer patients.

By trying out different values for threshold I was able to achieve optimal model performance at threshold 0.86, meaning that our model will predict that patient has breast cancer even if there is 14% chance.

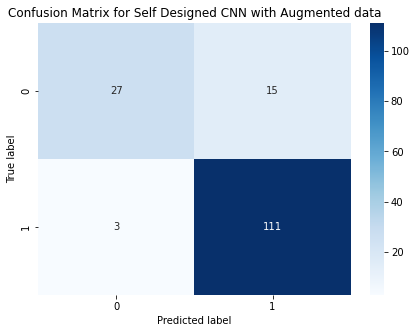


*Fig. 3.4: Confusion Matrix for self-designed CNN with threshold = 0.86*

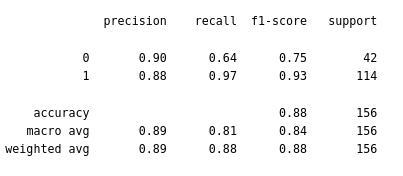
*Fig. 3.5: Classification Report for self-designed CNN with threshold = 0.86*

**3.1.3.2. SELF DESIGNED CONVOLUTIONAL NEURAL NETWORK WITH AUGMENTED DATA**

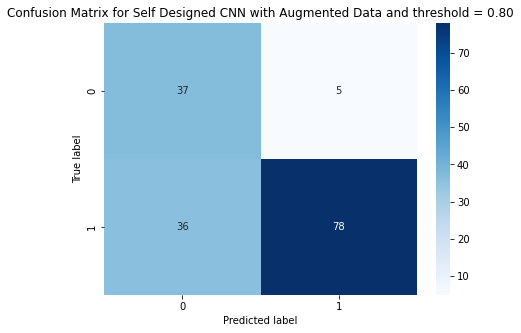
In the second approach I tried to generalize the model and make it more robust by adding variations in our training data by incorporating “Image augmentation” techniques. These variations in existing image are induced by randomly changing height, width and other parameters of the image. These changes results in creating variations in training dataset which in turn results in better model generalization.

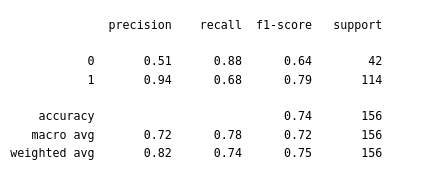


*Fig. 3.6: Confusion Matrix for self-designed CNN with augmented training data*

*Fig. 3.7: Classification Report for self-designed CNN with augmented training data*

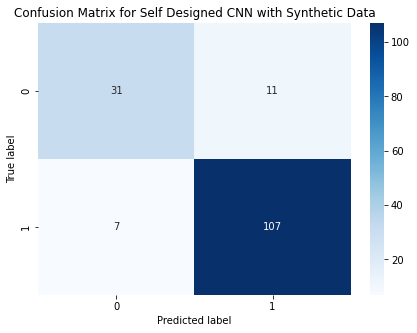
The results from augmented data performed slightly worse than the previously discussed model with no augmentation. One possible explanation would be that our class is imbalanced and majority class is already performing quite good. In fact, variations induced uncertain patterns in minority class, hence resulting in even more misclassified minority class. I then varied the decision threshold to achieve optimal model performance with respect to our target class “Class 0”. The optimal model performance was achieved at threshold = 0.8.

*Fig. 3.8: Confusion Matrix for self-designed CNN with augmented training data and threshold = 0.8*

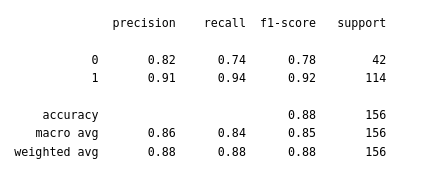
*Fig. 3.9: Classification Report for self-designed CNN with augmented training data and threshold = 0.8*

**3.1.3.3. SELF DESIGNED CONVOLUTIONAL NEURAL NETWORK USING SYNTHETIC DATA**

Class imbalance problem was improved by implementing the approach discussed in section 3.1.1, which involved data pre-processing techniques. Here I will present the classification results and discuss them.

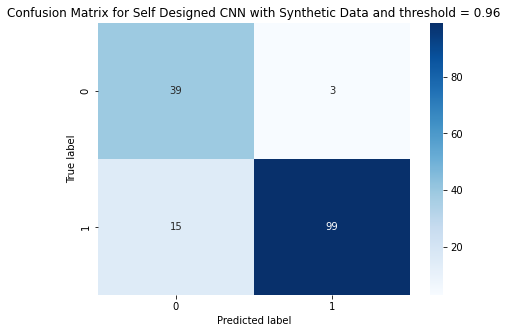


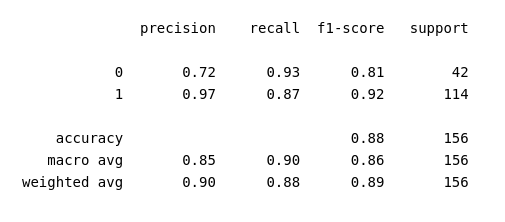
*Fig. 3.10: Confusion Matrix for self-designed CNN using synthetic data*

*Fig. 3.11: Classification Report for self-designed CNN using synthetic data*

If I analyse the evaluation results of our model I can see that recall has improved from 0.69, in the case of self-designed CNN, to 0.74 which is significant. Let us now improve the results further by varying the decision threshold.

Based on analysis the value of threshold for optimal model performance came out to be 0.96.

*Fig. 3.12: Confusion Matrix for self-designed CNN using synthetic data with threshold = 0.96*

*Fig. 3.13: Classification Report for self-designed CNN using synthetic data with threshold = 0.96*

**3.1.3.3. FINAL MODEL FOR BREAST CANCER CLASSIFIER**

Based on performance evaluation using classification metrics, I finalized the *“SELF DESIGNED CNN WITH SYNTHETIC DATA”* with decision threshold at 0.96 as the model that will go in production. The model performed reasonably well especially in terms of catching the positives of target class.

**3.2. BLOOD CELL CLASSIFIER MODEL**

**3.2.1. DATA PREPROCESSING**

For blood classifier, I pre-processed our image data in three different ways to achieve optimal model performance. Note that each approach is specific to type of model I have deployed. Approach 1, is common for first 2 models, while a different pre-processing technique was implemented for our last model where I incorporated Transfer Learning to obtain our final model. Important to add that I had to encode our labels too (Target Variable) as it is a multiclass problem. Encoding prevents our model to assume that the existing categories have some natural ordering which in turn could lead to wrong predictions.

In Approach 1, which was common for first two models, I normalized our image data. The normalized image data from this step was fed to our first model, and also pre-processed further for second model.

In Approach 2, which was for the second model, I introduced variations in the normalized data, using an image generator. I induced 11% random horizontal shifts (11% of total width), random vertical shifts to images (13% of total height), and random horizontally flipped images. These variations were added to see if they improve the model's performance by making it more robust to variations in the input.

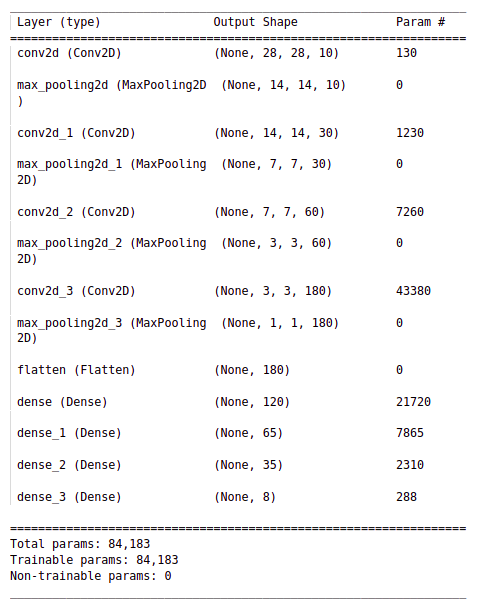
In our third approach, I incorporated transfer learning using the VGG16 model. I needed to pre-process our image data to match the required input shape of (224x224x3), which was different from our original input shape of (28x28x3). For that I had to resize our images from (28x28x3) to (224x224x3). Due to computational constraints, I was unable to normalize our resized images, so I had to use them as is. I also encountered issues during training as the larger images causing kernel crashes on Google Colab. To resolve this, I had to reduce our dataset size by cutting it in half, resulting in 5000/2500/1500 train/test/validation splits.

To summarize, three different data pre-processing approaches were used for all three models. In first two approaches I normalized the image data, while the second approach also introduced variations using image augmentation. The third approach involved transfer learning using the VGG16 model and required resizing the images as needed by input layer of VGG16. However, in third approach normalization was not possible due to computational constraints, and the dataset size also had to be reduced to overcome kernel crashes. Label encoding for target variable was done for all models.

**3.2.2 MODEL ARCHITECTURE**

**3.2.1.1. SELF DESIGNED CONVOLUTIONAL NEURAL NETWORK**

I have use the same model architecture stated below for two different approaches discussed in 3.2.1. Please note that I tried multiple architectures however this architecture performed best among all.

*Fig. 3.14: Self Designed CNN architecture*

The model architecture consists of 4 convolutional layers, each followed by max pooling layers, and 4 fully connected (Dense) layers. The Conv2D layers uses filters that generate feature maps from the input image, and the MaxPool2D layers down sample the feature maps to reduce their size.

The model starts with a Conv2D layer with 10 filters of size 2x2, followed by a max pooling layer with a pooling window of 2x2 and stride of 2x2. This is then followed by two more Conv2D and max pooling layers with 30 and 60 filters respectively, each followed by a 2x2 max pooling layer. The final Conv2D layer has 180 filters and is also followed by a 2x2 max pooling layer.

The Flatten layer converts the output of the last max pooling layer into a 1D vector, which is then fed into several fully connected layers. The Dense layers use the ReLU activation function and gradually reduce the number of neurons from 120 to 65 to 35. Finally, the output layer has 8 neurons with a SoftMax activation function, which produces a probability distribution over the 8 possible classes of the input image.

The model is compiled using the categorical\_crossentropy loss function, the Adam optimizer with a learning rate of 0.001, and the accuracy metric.

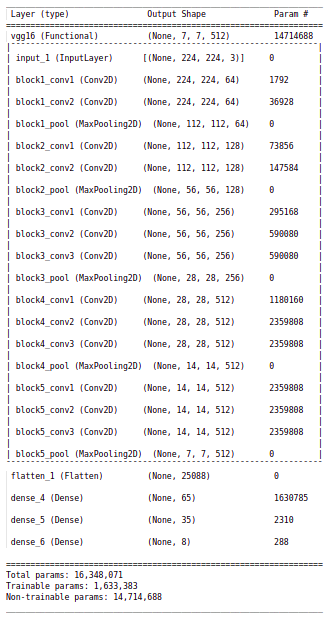
**3.2.1.2. TRANSFER LEARNING (PRE TRAINED NETWORK)**

Transfer learning involves using a pre-trained neural network and fine-tuning it on a new dataset. I will be using VGG16 which is a popular architecture used in transfer learning, and it is quite suitable for feature extraction from image-based datasets. The VGG16 model achieved a test accuracy of 92.7% in ImageNet (a dataset containing more than 14 million hand labelled training images across 1000 object classes).[1]

I have used three fully connected layers with the VGG16. The first layer which takes on the flattened feature array of the VGG16 convolutional network contains 65 neurons, followed by 35 and finally 8 neurons for the prediction layer. I have used ReLU activation on the top 2 and SoftMax (for multiclass prediction) on the prediction layer. The model is compiled using Categorical Cross Entropy loss and is trained on 5000 images out of total 11,959 training images, due to computational constraints.

In this architecture, it's worth noting that I fine-tuned the pre-trained VGG16 model by only updating a small portion of its weights. Specifically, I only trained the model on 1.6 million out of the total 16 million parameters, which means that only 10% of weights were updated at each epoch. This approach is commonly used in transfer learning to leverage both the existing architecture and the significant features captured by the pre-trained convolutional network. By fine-tuning a portion of the parameters, you can improve the model's performance on the new dataset without completely overwriting the previously learned features.

The summarized architecture I have used is shown below.

 *Fig. 3.15: Transfer Learning model architecture*

**3.2.3. MODEL EVALUATION RESULTS**

As discussed above in section 3.2.1, I applied three different approaches to obtain the model with optimal performance. I named these approaches as:

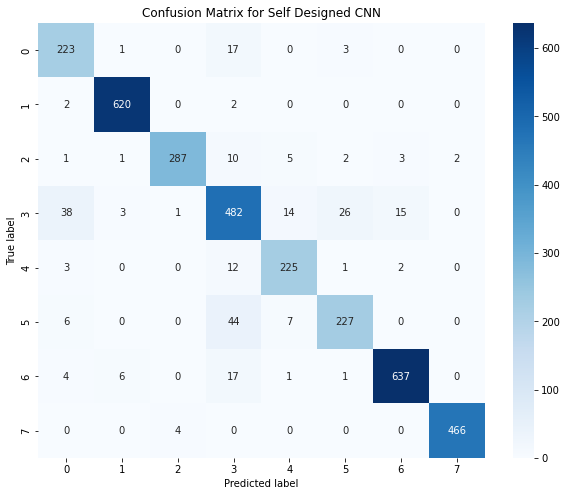
* *Self-Designed Convolutional Neural Network*
* *Self-Designed Convolutional Neural Network with Augmented Data*
* *Transfer Learning (Using existing DNN)*

The model architecture of first two approaches is same, while for the transfer learning I have used the convolutional layers of VGG16 model and used our fully connected layers with it to obtain our multiclass prediction.

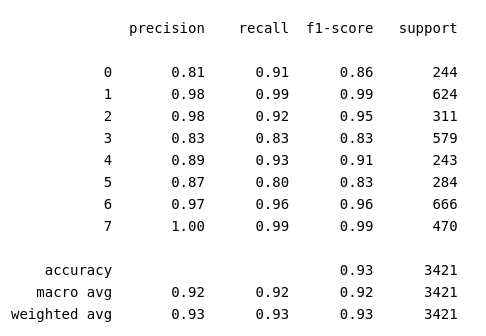
Let us now evaluate each model one by one to obtain the best model for production.

**3.2.3.1. EVALUATION OF SELF DESIGNED CNN:**

Following classification results were obtained from the self-designed CNN model.



*Fig. 3.16: Confusion Matrix for self-designed CNN*

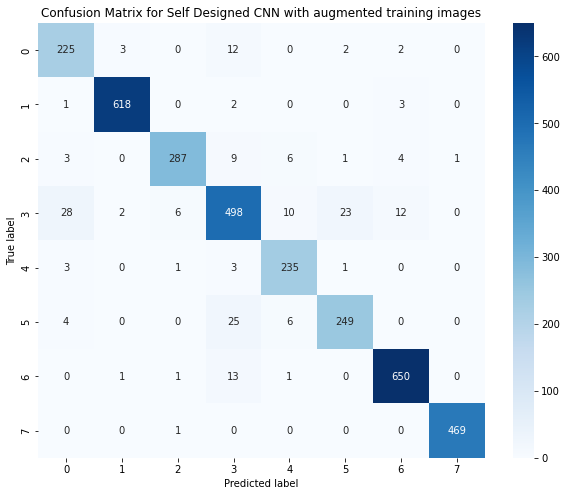


*Fig 3.17: Classification Report for self-designed CNN*

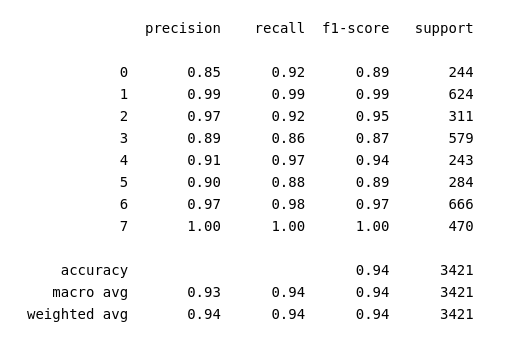
From the results above, the overall performance of our model seems quite satisfactory.

**3.2.3.2. EVALUATION OF SELF DESIGNED CNN WITH AUGMENTED DATA:**

Following classification results were obtained from the self-designed CNN model with augmented training data.



*Fig. 3.18: Confusion Matrix for self-designed CNN with augmented training Data*

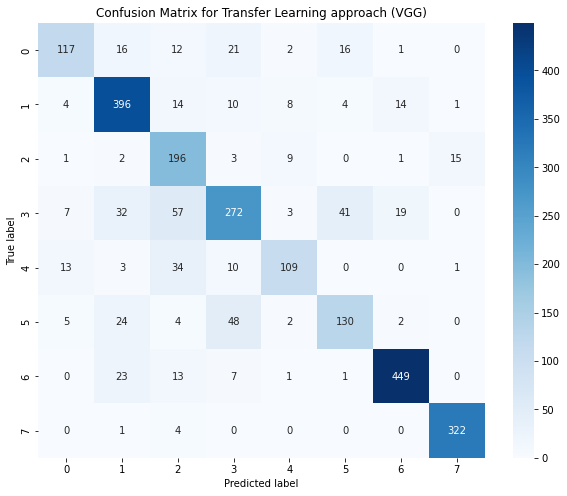


*Fig. 3.19: Classification Report for self-designed CNN with augmented data*

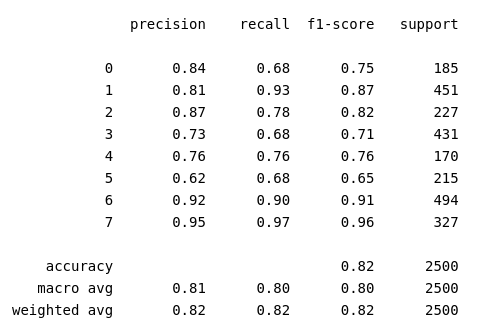
It is evident that the evaluation result of model with augmented training set is even better than the one with plain training images.

**3.2.3.3. EVALUATION OF TRANSFER LEARNING MODEL:**

I expected evaluation results of transfer learning to be below average as I had to sacrifice on a lot of data pre-processing steps, also discussed completely in section 3.2.1. in order to demonstrate the transfer learning approach. However, it performed better despite of all hurdles. The results obtained are shown below.



*Fig. 3.20 : Confusion Matrix for Transfer Learning method*



*Fig. 3.21: Classification Report for Transfer Learning method*

As expected, sub-optimal results were obtained through transfer learning due to poor data pre-processing. This also highlights the significance of data pre-processing in any Machine Learning or Deep Learning problem.

**3.2.3.4. BEST MODEL FOR BLOOD CELL CLASSIFIER**

Based on performance evaluation using classification metrics, I finalized the *“SELF DESIGNED CNN WITH AUGMENTED DATA”* as the model that will go in production. The model performed reasonably well across all classes with a balanced accuracy of 94%.