Capstone Final Report: Blood Cell Classification via CNN

Problem Statement

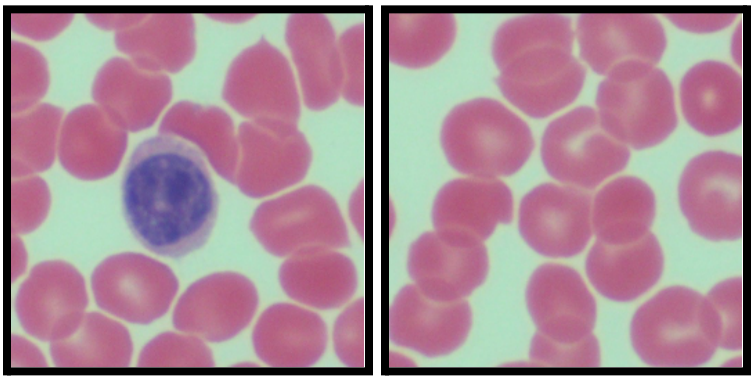
Currently, pathologists manually classify blood cells, which is time-consuming and prone to human error. Artificial Intelligence could speed up this process, leading to faster disease diagnosis. Blood cell classification is important because it plays a key role in medical diagnosis, treatment planning, and disease monitoring. Blood cell abnormalities are linked to serious conditions such as: Leukemia, Anemia, Infections, etc. Hospitals and labs handle thousands of samples daily. Automating this process could reduce operational costs through cutting down labor-intensive manual work. By developing a model for blood cell classification, we could improve medical efficiency, enhance patient care, and increase hospital/lab efficiency.

The focus of this business initiative is to automate blood cell classification using deep learning, specifically convolutional neural networks (CNN). This will improve diagnostic speed through reducing time required for manual classification. The model will be interpretable and prove medical AI trustworthiness. This AI-powered blood cell classification system will improve healthcare efficiency. By successfully developing and deploying this AI-powered blood cell classification system, this model can make a meaningful impact in the medical field. This initiative represents a step forward in integrating AI into healthcare, ultimately leading to faster, more accurate disease diagnosis and improved treatment outcomes.

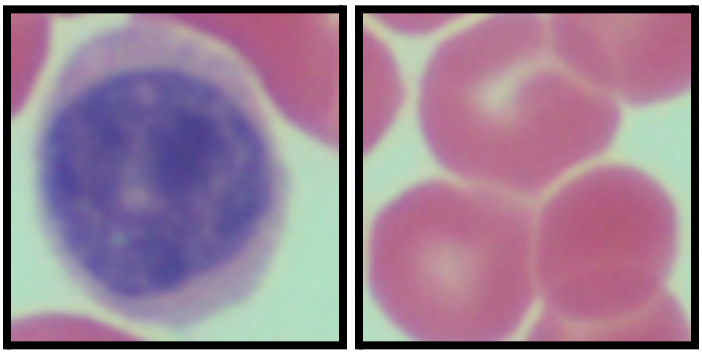
Data Wrangling

The dataset came from a project posted on Kaggle.com for blood cell classification via object detection. The CSV file “annotations.csv” came with labels, locations on the image of each cell for the grids in object detection. The dataset consisted of approximately 2400 rows and 7 columns. The images originally contained both white and red blood cells in each image which is not helpful for simple binary classification. These images were also repeated multiple times in the dataset for each cell in the image. The task of this project did not require a lot of the coordinate information therefore the images that were used were cropped to include only one type of blood cell, which helps the model differentiate between each cell. The final dataset contained a single iteration of each image with a cropped image of one cell in that image, as well as a label, either red or white blood cell (“rbc", “wbc”). The dataset that was used consisted of 100 rows and two columns, with 51 rbc images and 49 wbc images. Examples of the initial and cropped images can be viewed figures 1 - 4. Once the dataset was finalized we then loaded it to the notebook and edited the image column for each row to include the direct file path to the image. Now the images could be plotted and inspected with the plot_images() function that was created to plot the images. It was also verified that there was one value for each image, to ensure that no bias occurs later on in the modeling phase. After this initial data inspection, and since the images were cropped from the original image it was necessary to make sure that each image was of the same size before data augmentation. Therefore each image was resized to (224, 224), this specific size was chosen so that it would be

compatible with future pre-trained models. These steps in the data wrangling process created a clean and balanced dataset, allowing for effective model training and minimizing potential biases in classification.

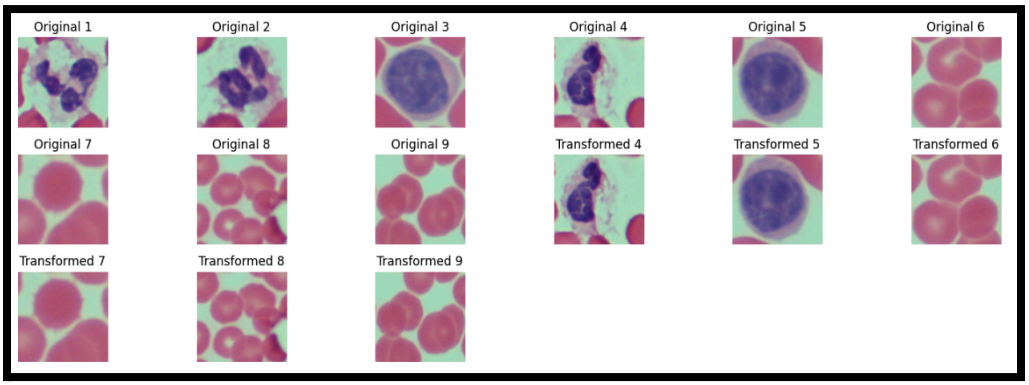


Figures 1-2: Examples of white and red blood cell images before cropping for binary classification



Figures 3-4: Examples of white and red blood cell images after cropping for binary classification

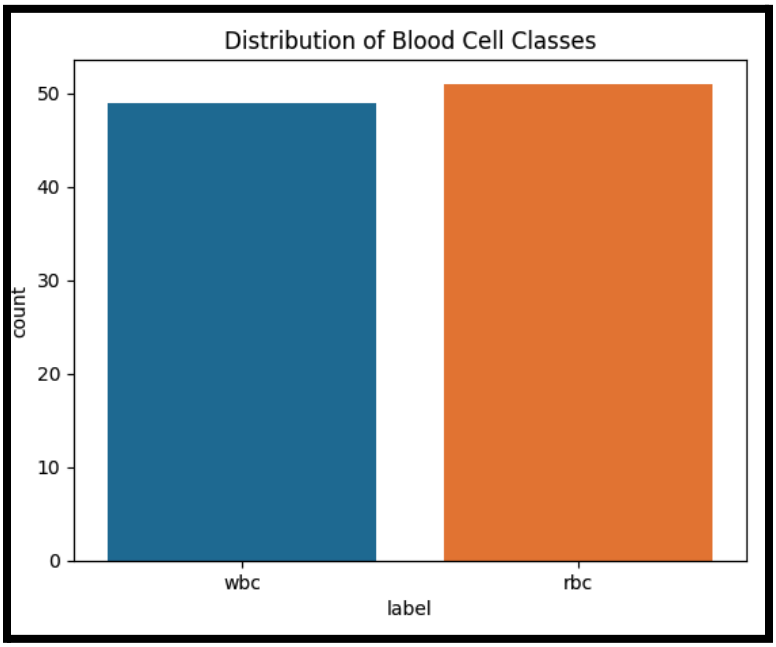
The last portion of the data wrangling process consisted of some preliminary data augmentation. An unsharp masking technique was applied to the images to enhance the edges for better feature extraction in the modeling portion of the images. These images were then saved into a new folder named transformed_images. This new folder would be used from now on for image extraction. Figure 5 conveys the before and after of the original and transformed images.



Figures 5: Examples of white and red blood cell images before and after unsharp masking technique

Exploratory Data Analysis

Since the model consisted of only images and their labels there was very little data analysis to be conducted. Using seaborn the distribution of blood cell classes was plotted, shown via figure 6. It was observed that there were slightly more red blood cell images than white blood cells, but overall there was a balanced dataset. Due to the little amount of images there will be a considerable amount of data augmentation performed in the preprocessing portion of the project. Lastly in the exploratory data analysis portion, the image size for the original and transformed images was inspected to make sure the images maintained the same aspect ratio and size. Fortunately, the images were still (224, 224, 3); the value of 3 stands for the 3 color channels of the image, RGB.



Figures 6: Seaborn count plot of the distribution of blood cell classes

The dataset had a small amount of blood cell images that needed data augmentation to increase robustness of the model. This will be dealt with in the preprocessing phase with brightening, cropping, etc. of all classes of images. The images contain red and white blood cells. Unsharp masking techniques were applied to sharpen the edges of the blood cells. These transformed images will be the ones used in the next steps of the project. Therefore, the file paths were corrected to include the transformed folder images and the updated CSV was saved as “annotations3.csv”. This was the file used for the next steps preprocessing and modeling.

Preprocessing

Using the updated dataset “annotations3.csv” and generating an updated function to plot the images in a grid, it was time to start the preprocessing portion of the project. This is where a heavy amount of data augmentation was performed to increase the robustness of the model. This included flipping, brightness adjustment, rotation, and cropping of all images. Each time a data augmentation

technique was performed the size of the dataset was doubled. First each image was flipped vertically and then appended to the end of the dataframe, transforming it from 49 “wbc” images and 51 “rbc” images to 98 “wbc” images and 102 “rbc” images. The second augmentation that was performed was applying random amounts of brightness to each image in the updated dataframe. Once these images were updated they were then appended to the dataframe creating a class balance of 196 “wbc” images and 204 “rbc” images. The third data augmentation technique that was applied was image rotation. Each image in the appended and updated data frame was rotated 90 degrees. Lastly as part of the data augmented process cropping of each image was performed. The final data frame contained 784 “wbc” images and 816 “rbc” images, with a final shape of (1600, 2). A few of these augmented images are conveyed in Figures 7 - 10. Now that the images have been flipped, cropped, rotated, and adjusted brightness for all classes to create a more robust dataset and allow for the model to be more robust by learning a diverse set of images. Before the modeling phase encoding had to be performed on the labels. Using the LabelENcoder() function the label column in the dataframe was transformed into 0 and 1 labels, 0 being red blood cells and 1 being white blood cells. Next, the custom model is built. After the custom model then its performance was compared to a pre-trained model.

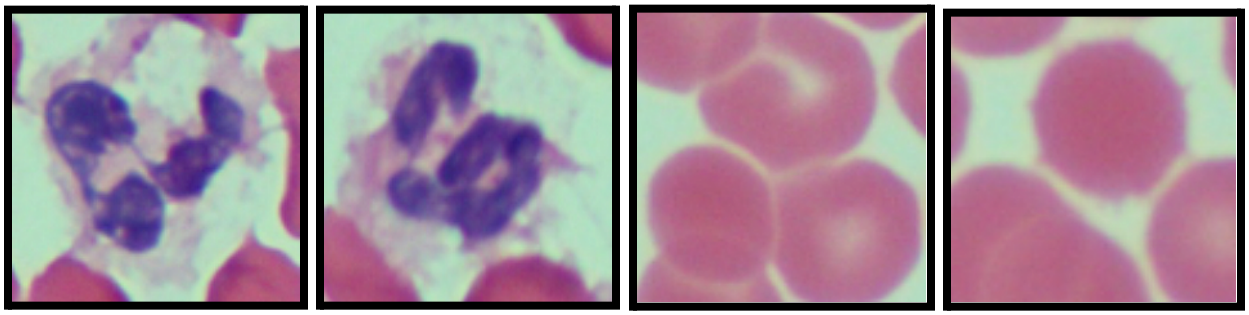


Figure 7: Flipped white and red blood cell images

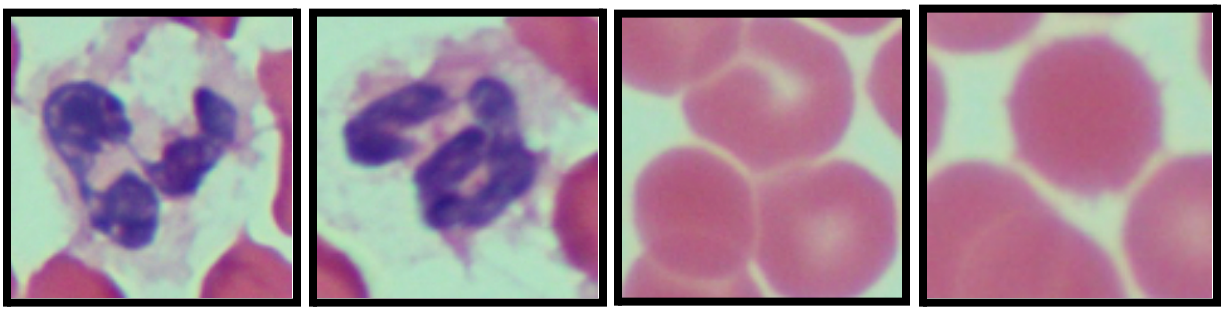


Figure 8: Brightness white and red blood cell images

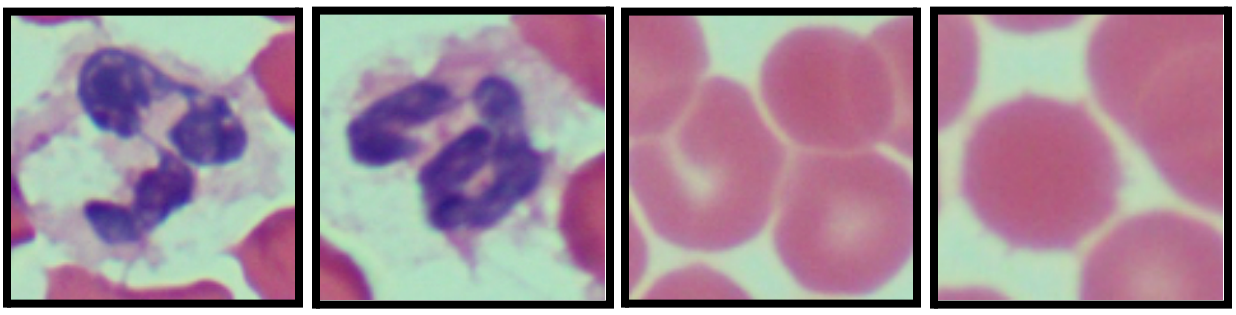


Figure 9: Rotated white and red blood cell images

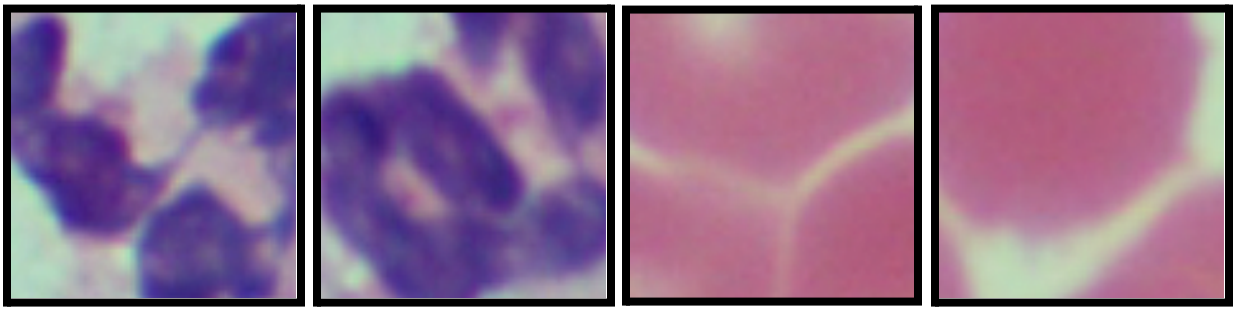


Figure 10: Cropped white and red blood cell images

Modeling

Before any architectural construction of the model could happen, the images needed to be normalized and a train test split needed to be conducted. To normalize the images, each image was individually selected and then each pixel value was divided by 255 which scales the values to the range of 0.0 to 1.0. These images then were used to create the X variable which took on the shape of (1600, 224, 224, 3). The target variable is the 0,1 labels that were encoded earlier. Once these variables were defined they were used in the train_test_split() function with a test size of 25%. 25% was chosen because the dataset is on the smaller side and a good amount of the data is needed for the training portion of the modeling. Then it is time to build the custom model.

Custom CNN Model

Originally there were many issues with the model only being able to classify one class, which was possibly due to the exploding or vanishing gradient problem, so many proactive measures were taken to prevent underfitting. That is why in the model the ELU activation function was used, which assists in improving the stability of the backpropagation process and avoiding vanishing gradients. This led to a smoother gradient and faster training convergence. There is also a gradual increase in the filter size in each convolutional layer to detect more complex patterns based on previous layers. Then for pooling the MaxPool2D was used to reduce spatial size and keep important features while lowering computation. Lastly in the model the sigmoid activation function was used to output a probability of 0 or 1 for binary classification. When it came to choosing the optimizer, Adam was considered but then it was realized that the loss landscape for the model is too complex for a simple optimizer. Therefore, the RMSprop was chosen which adjusts the learning rate individually for each parameter, leading to more stable convergence and overall more robustness. It also works well with ELU activation. Within the optimizer it was also chosen to start with a moderate to high learning rate, 0.008, which decreases gradually while monitoring the validation loss with the ReduceLROnPlateau() function. It was also decided to include momentum to speed up convergence and create more stable training, with a value of 0.9. Lastly in the optimizer the gradient clipping was included to further prevent the problem of exploding gradients by limiting the maximum gradient value, this value was chosen to be 0.7. Since the model originally was very unstable many steps were taken to prevent the instability and promote faster and smoother training.

Along with the decreasing learning rate in callbacks, early stopping was also implemented to prevent overfitting by stopping training when the model's performance on a validation set starts to degrade. Both of these functions were implemented into the fitting of the model, which consisted of 20 epochs, a small batch size of 32, which was discovered has a large effect on this model, if doubled then it takes much longer to converge, and lastly a validation split of 0.2. These were the optimal parameters that were found for this dataset and model.

Once the model was fit to the dataset, the model could then be evaluated on its performance. The model performed very well overall, converging to near perfect validation accuracy and loss at about epoch 8, with little improvement after. Figures 11 and 12 convey the performance of the model on the test and validation set. From the figures it is portrayed that the model was a bit unstable in the earlier portions of the epochs but then eventually converges towards little error/loss and near perfect accuracy. The final training loss approaches zero, suggesting the model is fitting the training data extremely well. The model was also evaluated using the classification report. This report conveyed that the model had nearly perfect precision, recall, and f-1 score for both classes. The f-1 score for both classes was 0.98. Finally the last steps of this data science process included comparing this custom model to a pre-trained model.

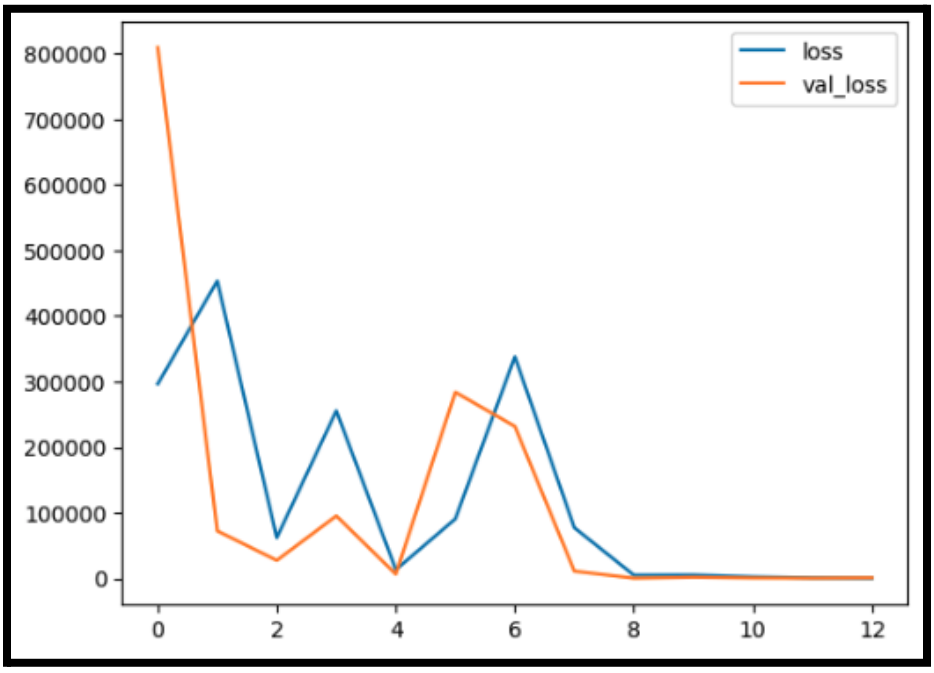


Figure 11: Custom model loss function values for test data and validation data

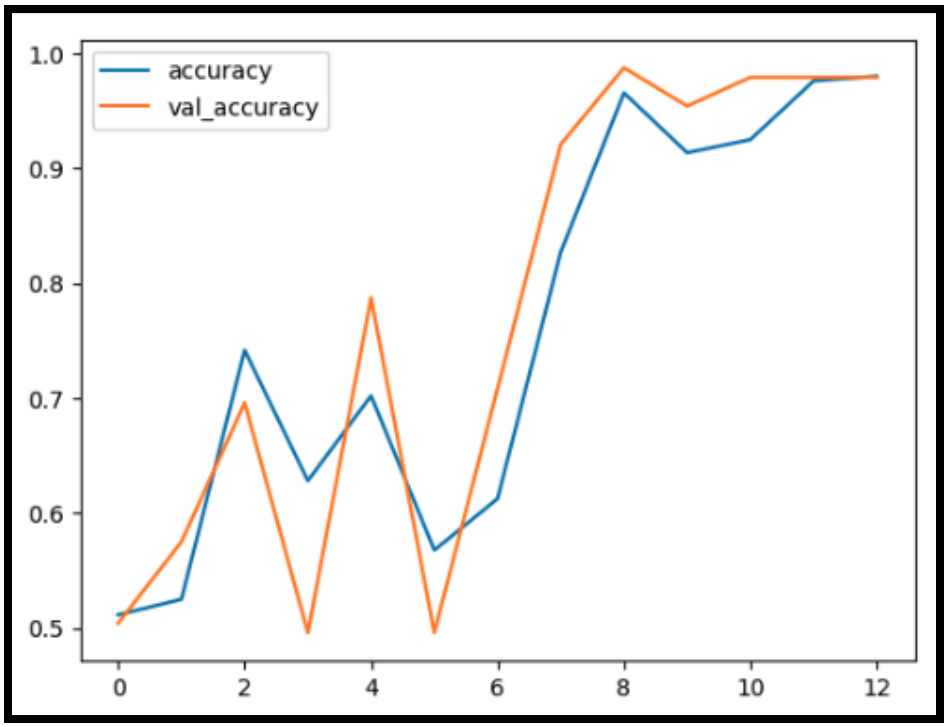


Figure 12: Custom model accuracy values for test data and validation data

Pre-trained CNN Model

Although the custom model performed very well, it was important for the scope of this project to compare the robustness and performance of the custom model to a pre-trained model. The pre-trained model that was chosen was VGG16. VGG16 has a relatively simple and straightforward architecture, making it easy to understand and implement. This pre-trained model generally achieves good performance on a variety of image classification tasks. VGG16 is a well-studied architecture, with extensive research and documentation available, which can be helpful for understanding its strengths and limitations. This section details the implementation and training of a binary classification model utilizing a pre-trained VGG16 architecture. The VGG16 model, pre-trained on ImageNet, was loaded without its top classification layer and configured with an input shape of (224, 224, 3). The convolutional layers of VGG16 were frozen to preserve pre-trained weights. A sequential model was constructed, appending a Flatten layer, a Dense layer with 128 neurons and ReLU activation, and a final Dense layer with sigmoid activation for binary classification. The model was compiled using the Adam optimizer, binary cross-entropy loss, and accuracy as the metric. Early stopping was implemented to monitor validation accuracy and prevent overfitting. Training was conducted for 10 epochs with a batch size of 32 and a 20% validation split. The model demonstrated rapid learning, achieving almost perfect 100% training accuracy and a validation accuracy by epoch 9. The validation loss decreased consistently, reaching near zero. These evaluation metrics were conveyed in figures 13 and 14. This indicates successful transfer learning and effective adaptation of the pre-trained VGG16 model to the binary classification task. Early stopping halted training after nine epochs due to no significant improvement in validation accuracy, demonstrating efficient training. The model exhibits strong performance and generalization capabilities. The classification report for this model conveyed even better performance than the previous model with an f-1

score of 0.99. The report indicates that the model is highly accurate and performs very well on the binary classification task.

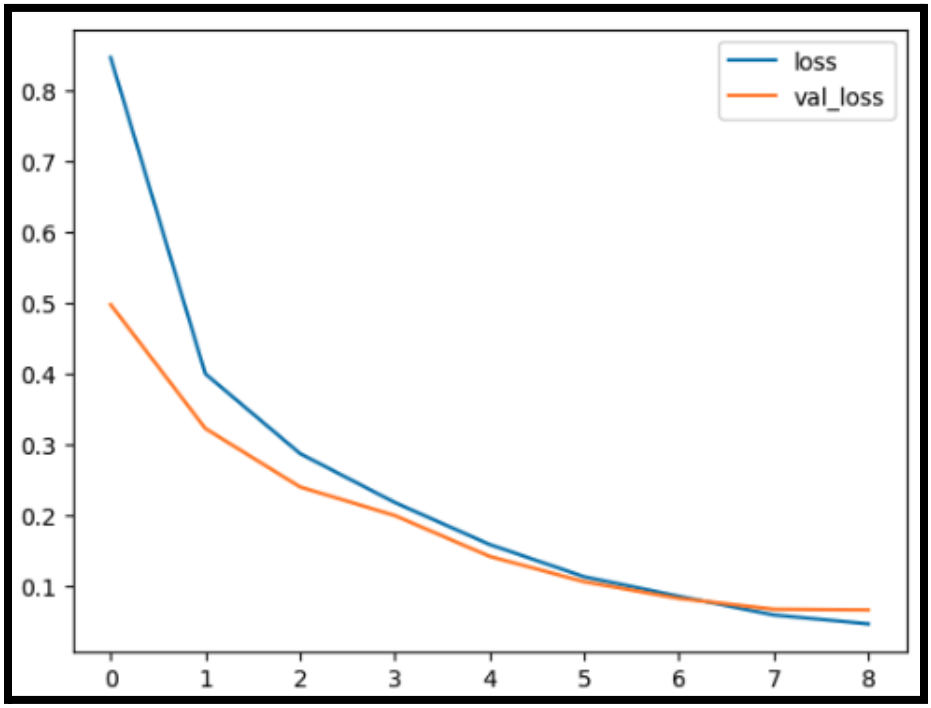


Figure 13: Pre-trained model error/loss values for test data and validation data

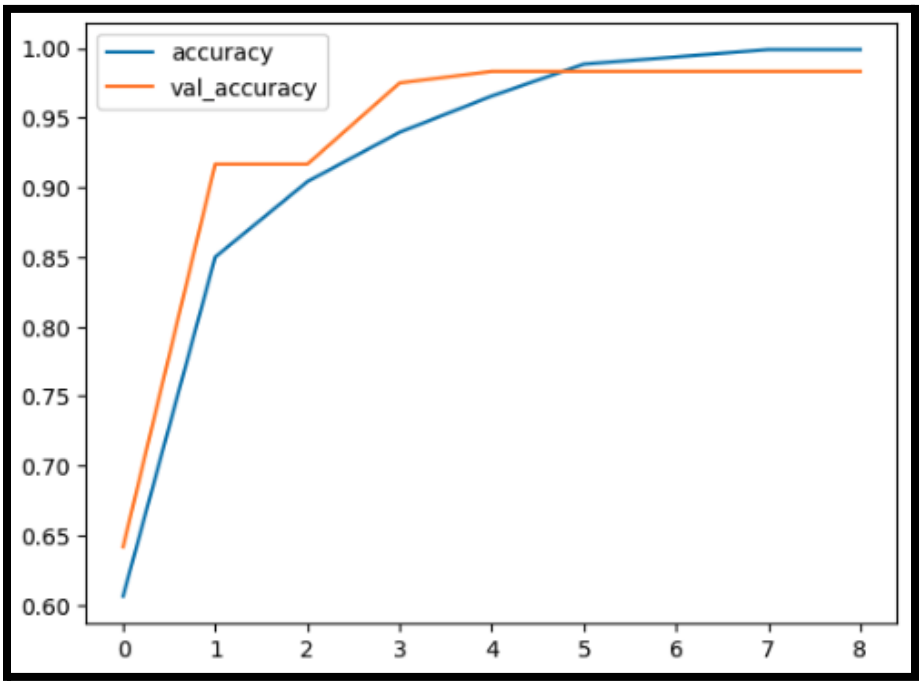


Figure 14: Pre-trained model accuracy values for test data and validation data

Conclusion

Both models are excellent and demonstrate exceptional performance. Both have strong generalization with high accuracy. Although our model is an excellent candidate, the pre-trained model significantly accelerates learning, improves generalization, and leads to better overall performance. Based on the provided metrics, the pre-trained model appears to be the superior choice in most scenarios. Its faster training, higher accuracy, and excellent generalization make it a strong performer. However, if there are specific requirements, such as medical imaging, for task specificity and control the custom model might be a viable alternative.

Overall, the high accuracy and efficiency of the models present significant business opportunities in the healthcare sector, with the potential to improve patient care, reduce costs, and drive innovation. The model can be integrated into blood analysis systems to automate the process of identifying and counting white blood cells and red blood cells. These models can also leverage automation by reducing the need for manual blood cell analysis, leading to lower labor costs for healthcare providers. The high accuracy of the model can minimize errors in blood cell analysis, leading to more reliable diagnoses and treatment decisions. Quicker and more accurate blood cell analysis can lead to faster diagnoses, enabling timely treatment and potentially improving patient outcomes. Faster results can then reduce patient anxiety associated with waiting for lab results. The model can be integrated into new blood analysis tools and devices, creating innovative products for healthcare providers and researchers. The technology can be expanded to new markets and applications, such as veterinary medicine or environmental monitoring. Offering efficient and accurate blood cell analysis solutions can attract new customers and increase market share. In conclusion, the use of this model has many benefits for the healthcare industry from reducing labor costs to even reducing patient anxiety.