**IMPLEMENTATION**

**MODULES:**

* Data Collection
* Dataset
* Data Preparation
* Feature Extraction
* Splitting the dataset
* Model Selection
* **Training the Model**
* Analyze and Prediction
* Accuracy on test set
* Saving the Trained Model
* Prediction Module
* Model Evaluation Module

**MODULES DESCSRIPTION:**

**Data Collection:**

* In the first module of the blood group detection using blood samples, we make the data collection process. This is the first real step towards the real development of a deep learning model, collecting data. This is a critical step that will cascade in how good the model will be, the more and better data that we get; the better our model will perform.
* There are several techniques to collect the data, like web scraping, manual interventions. The dataset is located in the model folder. The dataset is referred from the popular dataset repository called kaggle. The following is the link of the dataset:
* Kaggle Dataset Link: <https://www.kaggle.com/datasets/jayaprakashpondy/blood-dataset>

**Dataset:**

* Create directories for each blood type: A-, A+, AB-, AB+, B-, B+, O-, O+.
* Move Place each image in the directory corresponding to its blood type. For example:
* If an image is labeled as A-, move it to the A- directory.
* If an image is labeled as O+, move it to the O+ directory.
* Similarly, sort images for all the other blood types (A+, AB-, AB+, B-, B+, O-).
* By organizing the dataset in this structure, it becomes straightforward to load the data into your deep learning framework. This is because you can easily specify the directory path for each class when loading the data, ensuring that the correct images are loaded for each class and total dataset size is 750

**Data Preparation:**

* During the data preparation stage, it is crucial to preprocess the data to ensure it is suitable for training. This involves tasks such as resizing images to a standard size, normalizing pixel values, and encoding labels if necessary. To achieve this, the ImageDataGenerator from Keras can be utilized. For instance, to resize images to a standard size of 240x240 pixels, the Target\_size parameter can be set to (img\_height, img\_width) = (240, 240).
* Additionally, pixel values can be normalized by setting the rescale parameter to 1./255, Furthermore, data augmentation techniques such as random shear and zoom can be applied to enhance the training data. By leveraging these techniques, the data can be effectively preprocessed to improve the performance of the deep learning model.

**Feature Extraction:**

* For models like MobileNetV2, which come pre-trained with feature extraction layers, explicit feature extraction may not always be necessary. By setting trainable = False, we freeze these pre-trained layers, allowing them to retain their learned representations while preventing further updates during training.
* In the context of MobileNetV2, setting trainable = False ensures that the weights of the feature extraction layers remain fixed during training. This approach is commonly used in transfer learning scenarios, where the pre-trained model is fine-tuned on a new dataset for a specific task, such as image classification or object detection.
* By adopting this strategy, we strike a balance between leveraging powerful pre-trained representations and adapting the model to our specific dataset, ultimately improving both training efficiency and model performance.

**Splitting the dataset:**

* In this module, the image dataset will be divided into training and testing sets. Split the dataset into Train and Test. 80% train data and 20% test data. This will be done to train the model on a subset of the data, validate the model's performance, and test the model on unseen data to evaluate its accuracy. Split the dataset into train and test. 80% train data and 20% test data.
* Divide your dataset into training and validation to evaluate your model's performance. Typically, you might use an 80-20 split, but this can vary based on your dataset size and specific requirements.

**Model Selection:**

* The training module is responsible for training the deep learning models using the preprocessed data. It implements one popular architectures: MobileNetV2

***MobileNet:***

* MobileNetV2 is an efficient deep learning model designed for mobile and resource-constrained devices. The architecture is built around multiple layers of depthwise separable convolutions, which significantly reduce computational complexity and model size. It begins with a standard convolution layer, followed by a series of depthwise separable convolutions, each consisting of a depthwise convolution for spatial filtering and a pointwise convolution for combining features. These layers are typically accompanied by batch normalization and ReLU activation functions to stabilize training and improve non-linearity. This modular design enables MobileNetV2 to extract rich features efficiently, making it well-suited for applications requiring a low computational footprint.
* In addition to its depthwise separable convolutions, MobileNetV2 introduces a key innovation known as the inverted residual block with linear bottlenecks. These blocks allow the model to maintain high performance while reducing computational cost even further. Instead of expanding the number of channels as seen in traditional residual networks, the inverted residual block reduces the number of channels at the input and expands them at the output. This ensures that the bulk of computations happen in a lower-dimensional space, greatly improving efficiency.
* Moreover, MobileNetV2 incorporates skip connections between the bottleneck layers, which help preserve spatial information across layers, enabling the network to better capture fine-grained features. The final layers consist of a global average pooling layer followed by a fully connected layer, often used for classification tasks.
* Overall, MobileNetV2 strikes an optimal balance between accuracy and efficiency, making it a popular choice for tasks like object detection, image classification, and segmentation, especially in mobile and embedded devices where computational resources are limited.

**Training the Model:**

* To train the models, the training module first loads the pre-trained weights for MobileNetV2 from the ImageNet dataset. It then adds a global average pooling layer and a fully connected layer with the appropriate number of classes for the specific task. The base layers of the pre-trained models are frozen by setting their trainable flag to False, allowing only the added layers to be trained.
* The training process involves optimizing the model parameters using a suitable optimization algorithm, such as Adam or SGD, and a loss function appropriate for the task (e.g., categorical cross-entropy for classification). The training data is fed to the model in batches, and the gradients are computed and used to update the model weights. The training process continues for a specified number of epochs or until a certain performance metric is achieved on a validation set.
* After training, the module saves the trained models for future use in the prediction and evaluation modules. The saved models can be loaded and used for inference on new data or fine-tuned on additional datasets if needed.

**Analyze and Prediction:**

* Once training is complete, analyze the training process (e.g., loss curves) and make predictions on your validation set to assess the model's performance.

**Accuracy on test set:**

* After training and evaluating the model on the validation set, the accuracy of the model will be assessed on the test set. The accuracy on the test set will be an important metric for evaluating the model's performance.
* Once the model is trained, it needs to be evaluated for its performance. This module involves splitting the dataset into training and testing subsets and assessing the model's accuracy, precision, recall, and F1-score.
* The MobileNetV2 architecture attains a Training accuracy of 100%. and validation accuracy of 100%

**Saving the Trained Model:**

* Once you’re confident enough to take your trained and tested model into the production-ready environment, the first step is to save it into an .h5 or .pkl file using a library like pickle.
* Make sure you have pickle installed in your environment.
* Next, let’s import the module and dump the model into .pkl file.

**Prediction Module:**

* Develop a prediction module to make predictions using the trained MobileNet model. This module should take input images, preprocess them as necessary, and output predictions

**Model Evaluation Module**

* This module evaluates the performance of the trained models using the testing dataset. It calculates accuracy metrics and other performance indicators to assess model effectiveness.
* Evaluate model accuracy, precision, recall, and F1-score.
* Generate confusion matrix for the models.
* Compare the performance of the MobileNet test and train accuracies.
* Accuracy, precision, recall, and F1-score are used to evaluate model performance.