Blood Cell Image Classification

Objective

The main objective of this analysis is to build a deep learning model using the Blood Cell Images dataset for image classification. The focus of the analysis is on using Convolutional Neural Networks (CNNs) for the classification task. The benefits of this analysis are twofold: first, it provides an accurate model for classifying different types of blood cells, which can be crucial for medical diagnosis and research. Second, the use of deep learning models can automate and streamline the process of blood cell classification, saving time and effort for medical professionals and researchers.

Dataset Description:

The Blood Cell Images dataset contains 96x96-pixel images of histopathological lymph node scans with metastatic tissue. The dataset is divided into different folders, each containing images from different categories of blood cells, including eosinophils, lymphocytes, monocytes, and neutrophils. The dataset is labeled, which means each image is associated with its respective blood cell type. The objective of this analysis is to use this dataset to train a deep learning model to classify blood cells into their respective categories accurately.

Data Exploration and Preprocessing:

During data exploration, I checked for any missing values and visualized some sample images to understand the characteristics of the data. I performed data preprocessing steps, such as resizing the images to a uniform size (96x96 pixels) and applying normalization to scale the pixel values between 0 and 1. Additionally, I split the dataset into training and testing sets to evaluate the model's performance on unseen data accurately.

Train and Testing Data

In [2]:
```
train_folder = 'TRAINING'

train_dataset = torchvision.datasets.ImageFolder(root=train_folder,
                                                 transform=transforms.Compose([
        transforms.Resize((64, 64)),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ])
)
```

In [3]:
```
test_folder = 'TESTING'

# Load the train dataset
test_dataset = torchvision.datasets.ImageFolder(root=test_folder, transform=transforms.Compose([
        transforms.Resize((64, 64)),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ])
)
```

In [4]:
```
batch_size = 64
train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=batch_size, shuffle=False)
```

Summary of Deep Learning Models:

I experimented with three variations of the CNN model. The first model was a basic CNN with two convolutional layers and one fully connected layer. The second model had additional convolutional layers with increased depth and used dropout for regularization. The third model used transfer learning, where I loaded a pre-trained ResNet-18 model, replaced the final fully connected layer and fine-tuned the model for the blood cell classification task. To train the model i implemented a training loop to iterate through the entire training dataset for the specified number of epochs, updating the model's parameters to minimize the loss function, ultimately improving the model's performance over time.

```python
class BloodCellCNN(nn.Module):
    def __init__(self):
        super(BloodCellCNN, self).__init__()
        # Convolutional layers
        self.conv1 = nn.Conv2d(3, 32, 3, padding=1)
        self.conv2 = nn.Conv2d(32, 64, 3, padding=1)
        self.conv3 = nn.Conv2d(64, 128, 3, padding=1)
        self.conv4 = nn.Conv2d(128, 256, 3, padding=1)

        # Max pooling layers
        self.pool1 = nn.MaxPool2d(2, 2)
        self.pool2 = nn.MaxPool2d(2, 2)
        self.pool3 = nn.MaxPool2d(2, 2)
        self.pool4 = nn.MaxPool2d(2, 2)

        # Fully connected layers
        self.fc1 = nn.Linear(256 * 4 * 4, 512)
        self.fc2 = nn.Linear(512, 128)
        self.fc3 = nn.Linear(128, 4)

         # Dropout layers
        self.dropout1 = nn.Dropout(0.25)
        self.dropout2 = nn.Dropout(0.5)

    def forward(self, x):
        # Convolutional layers
        x = self.pool1(F.relu(self.conv1(x)))
        x = self.pool2(F.relu(self.conv2(x)))
        x = self.dropout1(x)
        x = self.pool3(F.relu(self.conv3(x)))
        x = self.pool4(F.relu(self.conv4(x)))
        x = self.dropout2(x)

        # Flatten the output for the fully connected layers
        x = x.view(-1, 256 * 4 * 4)

        # Fully connected layers
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)

        return x
```

```python
num_epochs = 10
losses = []
for epoch in range(num_epochs):
    model.train()
    for images, labels in train_loader:
        images = images.to(device)
        labels = labels.to(device)

        outputs = model(images)
        loss = criterion(outputs, labels)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    losses.append(loss.item())

    # Print average loss for the epoch
    print(f'Loss at epoch {epoch+1}: {loss.item():.4f}')
```

```
Loss at epoch 1: 1.1286
Loss at epoch 2: 0.4304
Loss at epoch 3: 0.5654
Loss at epoch 4: 0.2876
Loss at epoch 5: 0.3191
Loss at epoch 6: 0.2834
Loss at epoch 7: 0.2533
Loss at epoch 8: 0.0965
Loss at epoch 9: 0.2752
Loss at epoch 10: 0.1145
```

Recommended Final Model:

Among the three models, the pre-trained CNN model is recommended as the final model for this specific task. It demonstrates higher accuracy, F1 score, and AUC score, making it a better choice for classifying blood cell images into their respective categories. The reason that the CNN model performed better than transfer learning might have been due to limited data.

Metrics for CNN:

```
Test accuracy: 0.8082
Test F1 score: 0.8108
Test AUC score: 0.9630
```

Metrics for Transfer Learning:

```
Test accuracy: 0.4326
Test F1 score: 0.4351
Test AUC score: 0.7079
```

Key Findings and Insights:

The CNN-based models demonstrated the capability to classify blood cell images accurately. The models effectively learned distinctive features from the images to differentiate between different cell types. The CNN model achieved high accuracy, demonstrating the potential of models for medical image classification tasks with limited data.

Suggestions for Next Steps:

To further improve the model's performance, additional data augmentation techniques and fine-tuning of hyperparameters could be explored. Moreover, conducting a thorough analysis of misclassified images can provide insights into the challenges of the classification task and potentially guide the data collection and preprocessing efforts. Additionally, exploring other pre-trained models and ensembling multiple models could be beneficial to achieve even better accuracy.