Assignment - Image Analysis and Deep Learning

Member 1: Jacopo Bulgarelli, jacopo.bulgarelli@gsom.polimi.it
Member 2: Gianmarco Fistani, gianmarco.fistani@gsom.polimi.it
Member 3: Thomas Colangelo, thomas.colangelo@gsom.polimi.it

1. Introduction

This project aims to leverage Convolutional Neural Networks (CNNs), a class of deep learning models particularly well-suited for image analysis tasks, to classify microscope images of cells. Accurate classification of cell types is critical for numerous biomedical applications, including disease diagnosis, research, and personalized medicine. The primary objective is to develop a robust CNN model capable of accurately classifying electron microscope images of cells into one of eight predefined classes: basophils, eosinophils, erythroblasts, immature granulocytes, lymphocytes, neutrophils, monocytes, and platelets. The dataset consists of 15,092 images, each with a resolution of 48x48 pixels and three color channels (RGB). The project follows this structured approach:

1. **EDA and Data Preprocessing**: Ensuring the dataset is properly formatted and normalized for optimal model training, plus visualizations to better understand it.
2. **Models**: Developing both a hand-crafted CNN architecture, tailored to the characteristics of the dataset, and a classical CNN (deep model).
3. **Training and Validation**: Training the CNN models on the dataset while monitoring its performance on a validation set to prevent overfitting.
4. **Evaluation**: Assessing the model's performance on a separate test set to determine its generalization capability.

2. Data Analysis

The primary goal of this section was to gain insights by performing some Exploratory Data Analysis (EDA), in order to understand the structure, detect anomalies, test underlying assumptions, and uncover patterns and relationships to inform subsequent modeling steps. Following the EDA, we conducted several preprocessing operations.

2.1 EDA

2.1.1 Label Distribution

A count plot visualized the number of samples in each class, identifying class imbalances. Classes 1, 3, and 6 are more represented, class 7 is moderately represented, and the remaining classes are less frequent [1].

2.1.2 Feature Extraction

Mean color values (red, green, blue) were extracted to understand the color distribution and potential patterns among cell types. Images were converted to grayscale to analyze intensity distribution, calculating mean grayscale values along with mean values of black and white pixels using a threshold.

2.1.3 Scatter Plots

Several scatter plots [2] were created to visualize relationships between extracted features. Positive correlations were observed between most variables, except for the negative correlation between mean black and white values. Distinct clusters indicated that some labels tend to have higher mean blue values for similar levels of red and green values. Label 7 had lighter colors, suggesting smaller cells, while label 1 had darker colors, indicating larger cells.

2.2 Data Preprocessing

This section focuses on preparing and structuring the data before it is input into our neural network for training, a critical step as data quality and format significantly affect model performance. Initially, we normalized the image pixel values from the range [0, 255] to [-1, 1] to ensure uniformity across all inputs, enhancing the stability and speed of the training process. Subsequently, we employed one-hot encoding for categorical data, such as class labels. In this technique, each of the eight class labels is represented by a binary vector equal in length to the number of classes, where the position corresponding to a class is marked as 1, and all others as 0.

3. Experiments

In order to build our deep-model we started with the classical implementation of LeNet [3] which gave us an accuracy of 91%. To gain a different perspective, we experimented with a widely recognized convolutional neural network known as AlexNet, which is particularly effective for image processing. However, given that AlexNet typically handles larger images, we adapted it into a custom version tailored for smaller image sizes, specifically with 48x48 pixels [5]. Despite these implementations, we ultimately discontinued its use due to bad performance (Accuracy: 80%). Ultimately, we chose to implement a customized version of LeNet, detailed further in paragraph 5, as the basis for our final deep learning model.

Regarding the *test_model* function, we tried to run it with a test dataset, but it didn't work, so we used the original dataset "data.npz" and we got an accuracy of 98%.

4. Hand-Crafted Model

Our hand-crafted classification model combines various feature extraction techniques with a Random Forest classifier to classify cell images obtained from an electron microscope. The feature extraction process includes color histograms, Local Binary Patterns (LBP), average RGB values, and cell perimeter measurement. These features capture both structural and color information from the images, providing a comprehensive representation of the cell characteristics.

The Random Forest classifier, an ensemble learning algorithm, is trained on the extracted features to predict the cell types. With 100 decision trees and careful hyperparameter tuning, the classifier demonstrates strong performance in classifying cell images. The model achieves an accuracy of 0.87 on the validation dataset, indicating its ability to generalize well to unseen data.

Overall, the model's feature extraction process effectively captures relevant information from the cell images, and the Random Forest classifier demonstrates robust classification performance. However, further experimentation with different feature extraction techniques and model architectures could potentially enhance the model's performance and broaden its applicability in various cell classification tasks.

5. Deep Model

For the deep model, we explored several approaches before ultimately selecting LeNet [3]. Although it was not the optimal network for this task, it delivered the best results among those we tested. We employed a modified version of the LeNet architecture [4], adapted to align with contemporary deep learning practices. These are the modifications:

1. Activation Functions: Uses the ReLU activation function for the convolutional and first dense layer, which is common in modern neural networks due to its ability to reduce the likelihood of vanishing gradients and generally faster training. It retains tanh in the second dense layer.
2. Kernel Sizes and Filters: Uses 3×3 kernels for both convolutional layers, which is smaller than the original. The number of filters has been increased to 32 and 64 in the first and second convolutional layers, respectively, allowing the network to learn more complex features.
3. Padding Strategy: Uses "same" padding, which keeps the dimensions of the output feature maps the same as the input by padding the input with zeros on the borders. This is helpful for maintaining more spatial information throughout the network.
4. Pooling: Uses max pooling, which tends to capture more pronounced features in the input feature maps and is more common in modern CNN architectures.
5. Number of Neurons in Dense Layers: The first dense layer has 128 neurons, and the second maintains 84 neurons as in the original.
6. Output Layer and Loss Function: Uses softmax activation in the output layer for multi-class classification and categorical cross-entropy as the loss function, which is standard for multi-class problems today.
7. Optimizer: Uses the Adam optimizer [6], which is an adaptive learning rate method and is known for being robust and efficient in a wide range of problems.

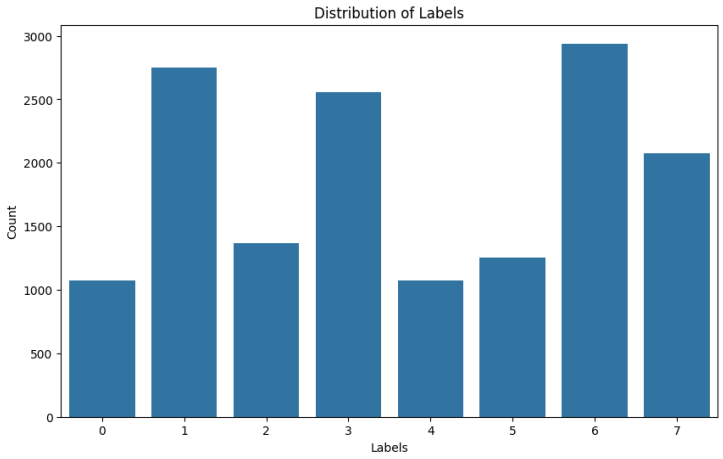
This custom-built LeNet model effectively captured spatial hierarchies and performed classification tasks, leveraging ReLU activations and the Adam optimizer for enhanced performance. All of this gave us an accuracy of **0.9397**.

6. Conclusion

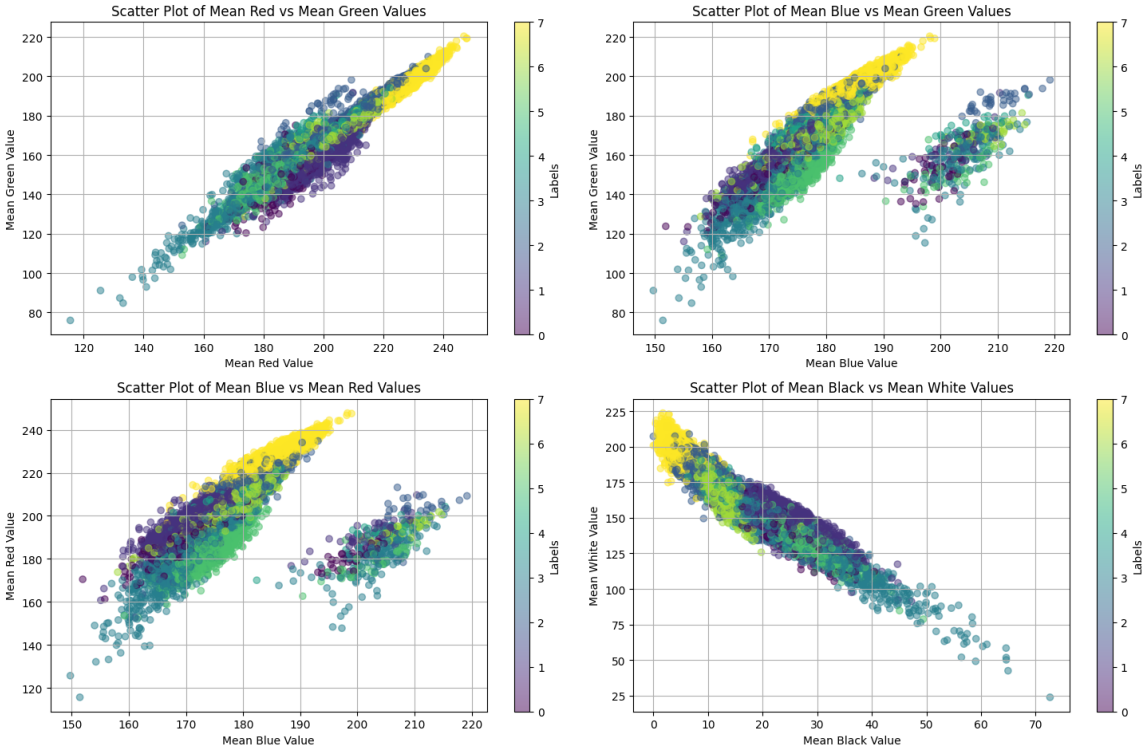
This project demonstrates the application of CNNs for classifying electron microscope images of cells into eight classes. By developing both a hand-crafted CNN architecture and an adapted LeNet model, we achieved robust performance in our classification task. The project highlights the power of CNNs in biomedical image classification, achieving high accuracy and providing valuable insights into the dataset's characteristics, paving the way for future advancements in automated cell classification.

References

1. Image N.1 - Labels distribution



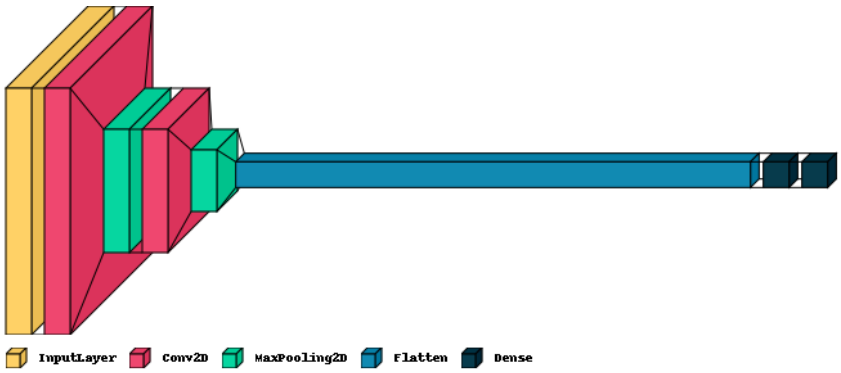
2. Image N.2 - Scatterplots



3. LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. "Deep learning." nature 521.7553 (2015): 436-444.

```python
def build_LeNet(input_shape, num_classes):
    model = models.Sequential(name='Original_LeNet')

    model.add(layers.Conv2D(6, kernel_size=(5, 5), activation='tanh', input_shape=input_shape, padding='valid', name='C1'))

    model.add(layers.MaxPooling2D(pool_size=(2, 2), name='S2'))

    model.add(layers.Conv2D(16, (5, 5), activation='tanh', padding='valid', name='C3'))

    model.add(layers.MaxPooling2D(pool_size=(2, 2), name='S4'))

    model.add(layers.Flatten(name='Flatten'))

    model.add(layers.Dense(120, activation='tanh', name='C5'))

    model.add(layers.Dense(84, activation='tanh', name='F6'))

    model.add(layers.Dense(num_classes, activation='softmax', name='Output'))

    model.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

    return model
```

4. Representation of a custom LeNet model



5. Custom AlexNet implementation for 48x48 image size

```python
alexnet_model2 = Sequential([
    Conv2D(96, (3, 3), activation='relu', input_shape=(48, 48, 3), padding='same'),
    BatchNormalization(),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(256, (3, 3), activation='relu', padding='same'),
    BatchNormalization(),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(384, (3, 3), activation='relu', padding='same'),
    Conv2D(384, (3, 3), activation='relu', padding='same'),
    Conv2D(256, (3, 3), activation='relu', padding='same'),
    MaxPooling2D(pool_size=(2, 2)),
    Flatten(),
    Dense(1024, activation='relu'),
    Dropout(0.5),
    Dense(1024, activation='relu'),
    Dropout(0.5),
    Dense(y_train.shape[1], activation='softmax')
])
```

6. Kingma, Diederik P., and Jimmy Ba. "Adam: A method for stochastic optimization." arXiv preprint arXiv:1412.6980 (2014).