Lab Exercise 3

CS ELEC 2C

Alessandro Andrei Araza   Joshua Kyle Entrata

April 29, 2024

Abstract

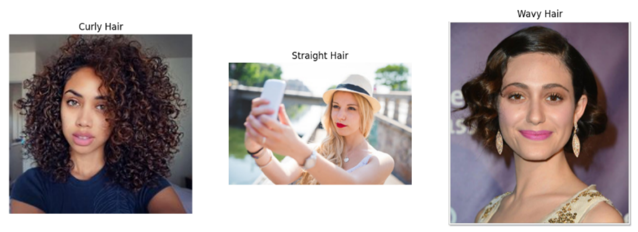
I INTRODUCTION

Convolutional Neural Networks (CNN) is one of the most if not the most popular neural network models for computer vision. They utilize a kernel for recognizing patterns within images. CNNs make use of the locality aspect of objects within images because of the kernel; instead of considering each and every pixel in the image, the kernel is able to take only a subset of that image and share weights or network parameters at many locations. Because of this, CNNs lead the way for image recognition in deep learning.

In this lab exercise, CNNs must be utilized to detect hair types. There are three main types to classify: curly, straight, and wavy hair.

1.1 Dataset

Figure 1 shows an example for each hair type in the dataset. It can be seen that much of these pictures are not in a controlled environment; these pictures have different lighting, backgrounds, etc. to introduce noise to the data. We'll see if different in preprocessing, modelling, and other aspects of the model can influence how it better learns given all this noise in the dataset.



```
<Figure size 640x480 with 0 Axes>
```

Figure 1: Hair Types Example

II METHODOLOGY

The methodology of this lab exercise follows the usual process for machine learning problems/researches

data → preprocessing → modeling → evaluation

The preprocessing section mainly consists of splitting the dataset into training and testing data, checking the validity of data, as well as many others. The Convolutional Neural Network, like other neural networks, can be customized especially in the number of convolutional layers, different pooling methods, different kernels, etc. Unlike previous models, there really isn't much customization that could be done to them other than the hyperparameters to alter slightly how it works. But when it comes to CNNs, there's a lot more room for varying techniques.

Later in experimentation, we will see how customizing the CNN can affect performance for detecting these hair types. For evaluation, the accuracy metric is used to measure the model's performance.

2.1 Data Acquisition and Preparation

2.1.1 Loading of Images

A folder named `hair_types` was manually imported in the repository containing 1000 images. Inside that folder has subdirectories for each hair type. Python Imaging Library (PIL) library was used to create a custom function of validating the images that was imported. It ensures that none of these images are corrupted and adhere to specific formats, such as PNG, JPG, and BMP. We excluded files that are not accepted by Tensorflow and does not match the specific formats, such as .WebP. After loading these images, 14 images were deleted and one image with a file type of .gif was also detected, but we didn't remove it because it is still valid for this experiment. This step is important to minimize potential issues during model training.

2.1.2 Image Categorization

This step involves segrating the images into three groups based on hair type labels found in their file paths. This will help in the structured training of the model by defining the classes of each images.

2.1.3 Manual Inspection

To ensure the integiry of our dataset, we manually inspected each image via the file explorer. During the inspection, we identified irrelevant image of a microphone logo found in the subdirectory folder of `Straight_Hair`. Removing this image was crucial to prevent the model from leaning incorrect and unnecessary information.

2.1.4 Image Display

Using matplotlib, we displayed a sample from each category. This step is important to ensure the integrity and appropriateness of the labels and the image themselves.

2.2 Preprocessing

2.2.1 Data Augmentation

Initaially, we tried applying several data augmentation technicques, such as random horizontal and vertical flips, random rotations, and random zooms. However, we didn't apply data augmentation in the final model, which will be discussed on the Experiments section.

2.2.2 Splitting Dataset

To ensure generalizability of this model, we utilized Tensorflow's `image_dataset_from_directory` method to split the image data into training and validation datasets. 20 % of the data was allocated for validation. Additionally, we resize all images to a uniform 256 $\times$ 256 pixels for consistency and batch them in sizes of 64. Splitting the data this way helps prevent overfitting by ensuring that the model does not memorize the training data and allows us to fine-tune the model parameters based on the performance metrics gathered from validation set.

2.2.3 Sobel Edge Detection

A custom preprocessing function was created to detect Sobel edges in the images. This technique highlights the edges within each image. This technique will be applied for both training sets and validation sets to ensure consistency in how images are presented to the model.

We generated 9 samples from the dataset to display sobel edges applied images.

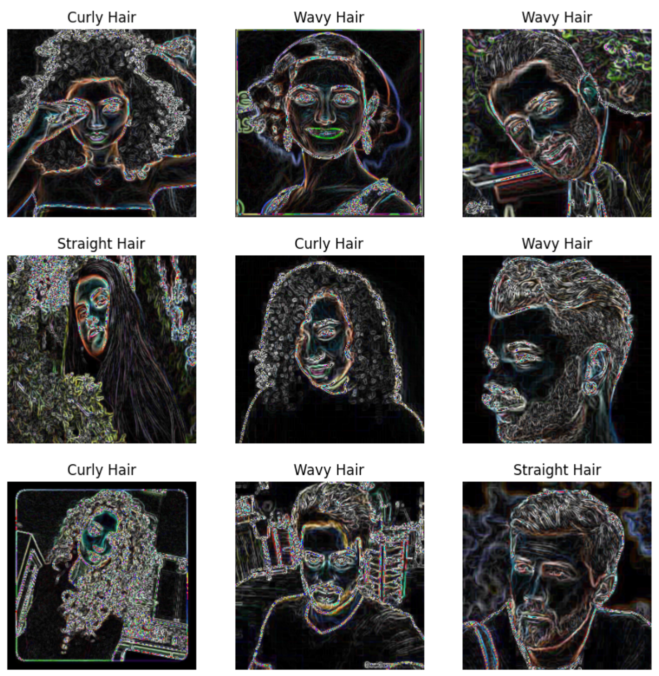


Figure 2: Hair Types Example with Sobel Edge

2.3 Modeling

2.3.1 Convolutional Neural Network Architecture

In this step, we constructed a convolutional neural network (CNN) model using the Keras library's 'Sequential' API. The network begins with a rescaling layer that normalizes the pixel values of the images from a range of 0 to 255 to a range of 0 to 1, enhancing the training efficiency.

The architecture of our convolutional neural network (CNN) begins with the 'Sequential' model framework from Keras, which allows layers to be added sequentially.

1. Input Layer: This layer specifies the shape of the input data, which is 256x256, and adds three color channgels (RGB), making it $256, 256, 3$.

2. Rescaling Layer: After the input layer, a `Rescaling` layer npormalizes the pixel values of the images from a range of 0 to 255 to a range of 0 to 1. This is important because it helps the model converge faster during the training stage.

3. First Convolution Layer: This layer uses 16 filters with a kernel size of 16 and a stride of 2. We also did not added padding in the input (`valid padding`). The dilation rate is set to 1, which keeps the kernel compact.

4. Activation Function: Every after convolution layer, a Rectified Linear Unit (ReLU) activation function is applied. This is used to introduce non-linearity to the model and solves the vanishing gradients issue.

5. Batch Normalization: This normalization technique is applied to help standardizing the data by normalizing the activation of the convolution layer.

6. Pooling Layer: After that, `MaxPooling2D` layer is applied to reduce the spatial dimentions of the output. It summarizes the features in a 2x2 window with the maximum value.

7. Dropout Layer: A dropout layer is introduced with a rate of 0.25 to help prevent overfitting by randomly setting a fraction of input unit to 0 during training.

8. Other Convolution Layers: The process above is repeated three more times with different filter and kernel sizes. The filters increase while the kernel size decrease as the process continue. For the last two convolution layers, max pooling and dropout were excluded.

9. Global Average Pooling: After the last convolutional layers, a `GlobalAveragePooling2D` layer is applied, which calculates the average value of each feature map and outputs a tensor that is smaller in size. This can help reduce the dimensionality of the feature maps and prevent overfitting.

10. Flattening: After that, a `Flatten` layer then converts these features into a single vector. This helps maintain all the information from the feature maps and connect the convolutional layers to the fully connected layers.

11. Dense and Output Layers: The feature vector coming from the Flattening layer is fed into a dense layer of 32 neurons, followed by a dropout layer with a rate of 0.5, and then an activation function of ReLU. The final output layer will consists of 3 neurons (one for each hair type) with a `softmax` activation function.

2.3.2 Training the Model

The model was compiled with the Adam optimizer, using a learning rate of $1e^{-4}$ and categorical cross-entropy loss function because the dataset consist of multiple classes. The training was conducted over 50 epochs, using both training and validation datasets.

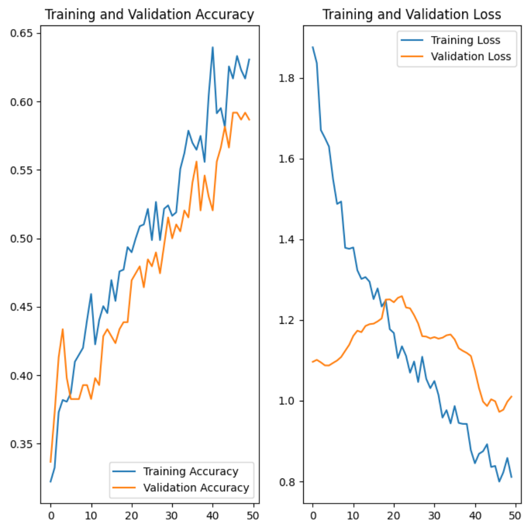


Figure 3: Training and Validation Results

2.4 Evaluation

To evaluate the performance of the convolutional neural network, we utilized different techniques that visually and quantitavely assess its effectiveness:

1. Accuracy and Loss Plots: We generated line plots to compare the training and valiation accuracy, as well as the training and validation loss over the epochs. These plots are important for us to understand the model's learning progress and see if it is overfitting or not.

2. Confusion Matrix: We created a confusion matrix from `sklearn.metrics` to visually inspect the performance of the model across different classes. It can also help us identify misclassifications in the model.

3. Classification Metrics Report: We utilized the `classification_report` from `sklearn.metrics` to see the results of the precision, recall, and F1-score for each class.

4. Image Prediction: Finally, to demonstrate the effectiveness of the model in predicting the hair type of the image, we used the `predict` method to see the percentage of each classes based on a single image.

III EXPERIMENTS

The objective of this experiment is to classify images into three hair types. To achieve this, we tested different hyperparameters, explored various preprocessing tecniques and experimented in creating the optimal CNN architecture. The primary goals in this experiment were to optimize evaluation metrics and minimize overfitting, ensuring robust model performance.

3.1 Adjustment of Image Size and Batch Size

The initial image size was set to 64×64 pixels. While this smaller dimension allowed us to train the model faster, it compromised the quality of the training images, potentially eliminating important information for accurate classification. Conversely, increasing the image size to 512×512 pixels significantly increased the training time to almost 30 minutes, which was impractical for our computational resources. Our experiments was conducted on a laptop with specifications of a GTX 1660Ti graphics card, a Ryzen 7 4800H processor, and 32GB of RAM.

Thambawita (2021) discussed in their paper that the performance of the classification model is dependent upon the resolution of images, but it is important to consider the trade-off between image size and the time needed for training the model. Considering these factors, we opted to use an image size of 256×256 pixels. This resolution provides an optimal balance between image quality and manageable training times.

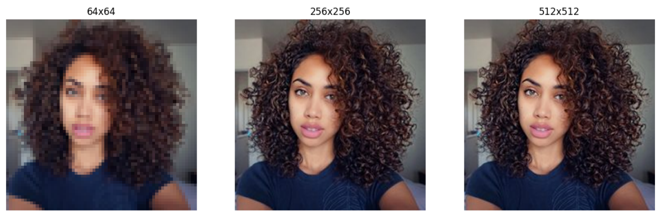


Figure 4: Different Image Resolution

Regarding the batch size, we experimented with sizes that are divisible by our image size, such as 32, 64, and 128. The batch size of 64 was chosen as it offers an efficient balance between minimizing the memory usage during training while providing a reliable estimate of the gradient.

3.2 Preprocessing

During the experimental phase, we utilized two preprocessing techniques in machine learning for image classification: Sobel Edge Detection and Data Augmentation. These methods were chosen based on their potential to improve model accuracy and minimize overfitting.

The models were evaluated under four different preprocessing scenarios to assess their impact:

- No preprocessing
- Sobel Edge Detection only
- Data Augmentation only
- Both Sobel Edge Detection and Data Augmentation

Table 1: Comparative Results of Different Preprocessing Techniques: none, Sobel Edge Detection only, data augmentation only, and both

	T Acc	Val Acc	T Loss	V Loss
None	0.639200	0.543100	0.771500	0.998300
Sobel only	0.614500	0.586700	0.839800	1.010100
Data Aug only	0.431100	0.543100	1.088000	0.983300
Both	0.404100	0.411200	1.173300	1.111400

The results from each scenario are shown in the Table 1. Alongside these numerical results, we also provided plots of accuracy and loss to better illustrate the models' tendencies to overfit or underfit.

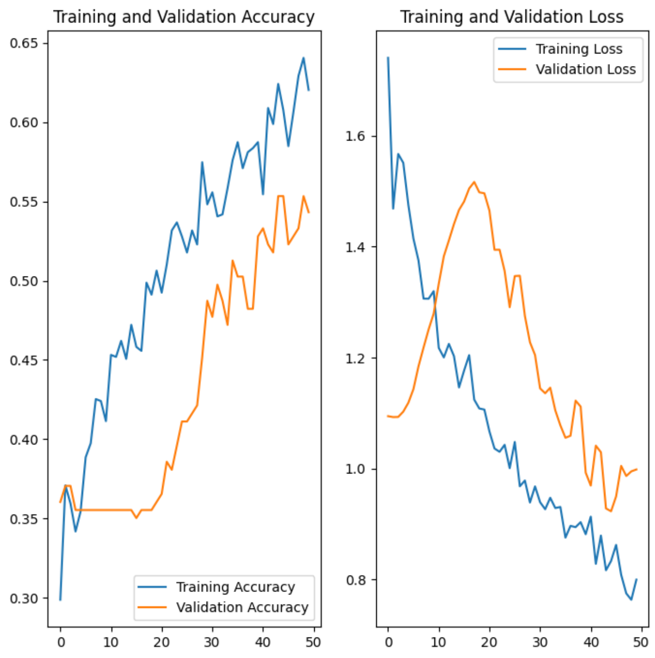


Figure 5: No Preprocessing

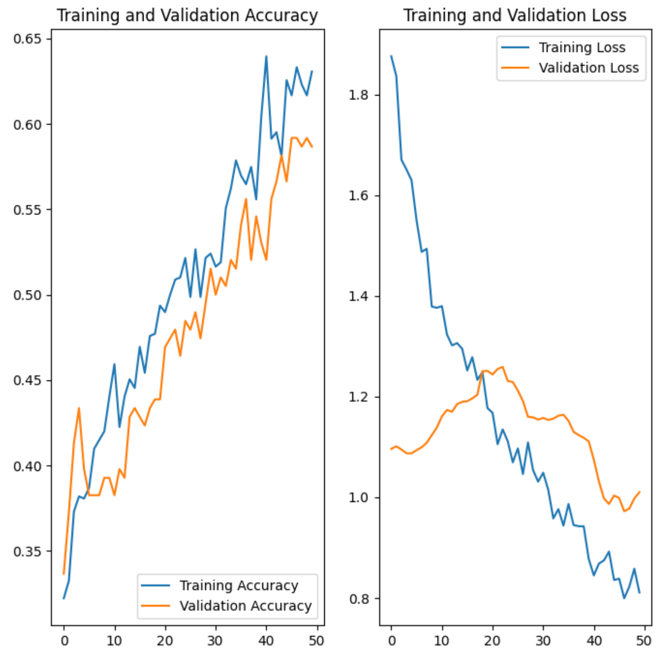


Figure 6: Sobel Edge Detection only

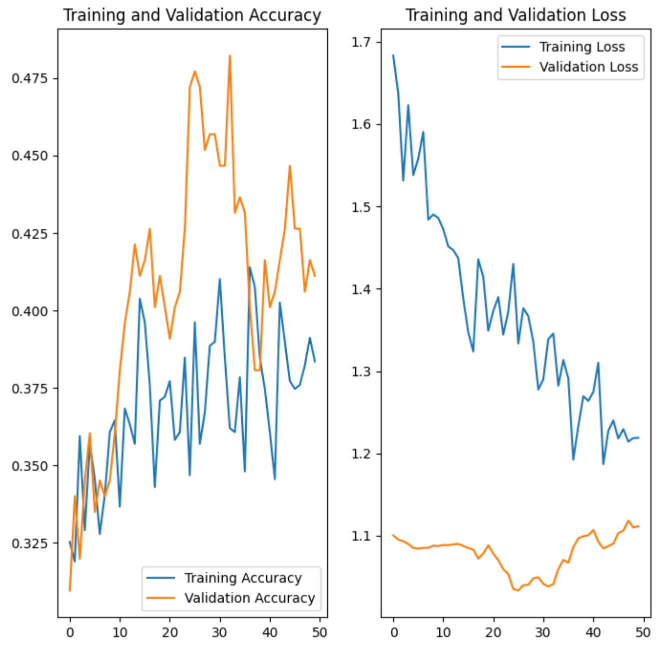


Figure 8: Both Sobel Edge Detection and Data Augmentation

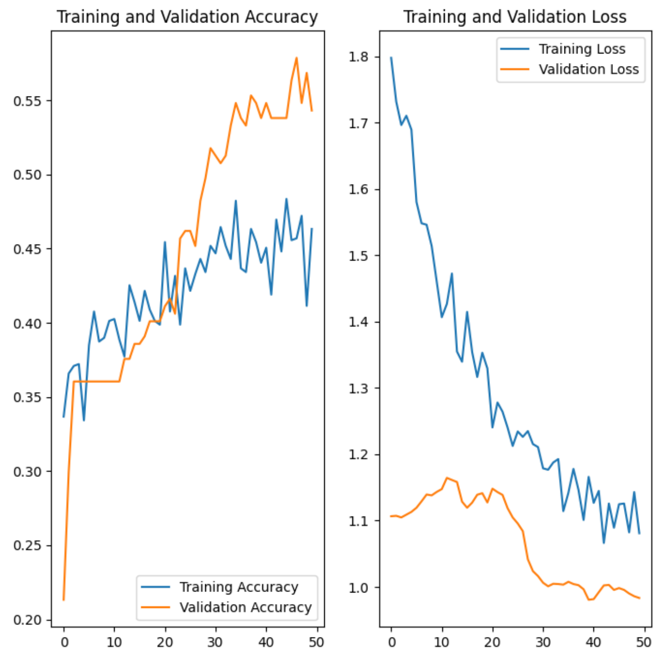


Figure 7: Data Augmentation only

The use of sobel edge detection helped improve object detection and segmentation, notably enhancing hair boundary delineation in the images, as seen in Figure 6. Given that our dataset consists of only 987 valid images, data augmentation was important in addressing the limited data situation. Despite its benefits, the best model performance was obseved with Sobel Edge Detection only, leading us to exlude data augmentation in next phases.

3.3 Modeling

In developing an effective CNN for our classification task, our goal was to strike a balance between a sufficient number of convolutional layers to capture relevant features and maintaining the model's complexity, which is good for generalizability.

3.3.1 Create a Simple Model

Our initial model was designed to be simple, serving as our starting point for exploration and providing insights about the straightforward application of convolutional layers. It is structured as follows:

1. Convolutional layers with filter sizes of 16, 8, and 4, and corresponding filter counts of 8, 16, and 32, each followed by ReLU activation.
2. A Global Average Pooling layer for dimensionality reduction.
3. A dense layer with 32 units for pattern learning.
4. A softmax output layer with 3 units for class prediction.

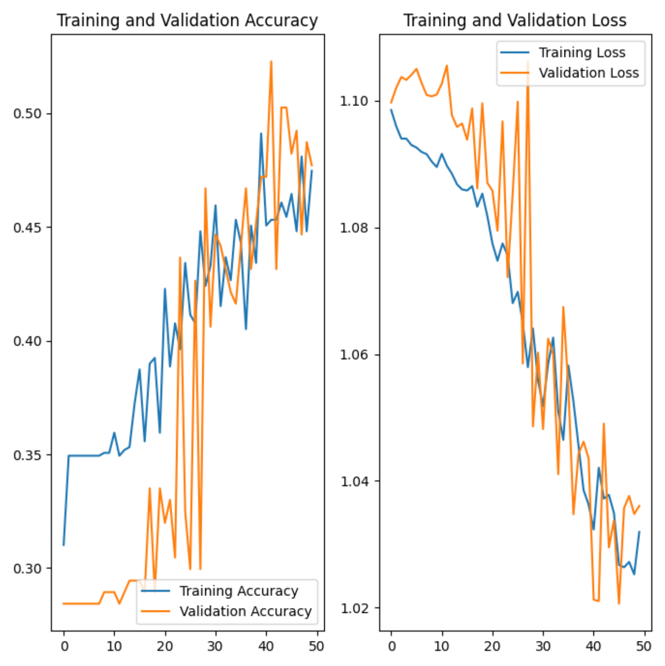


Figure 9: Accuracy and Loss Plots of the Simple Model

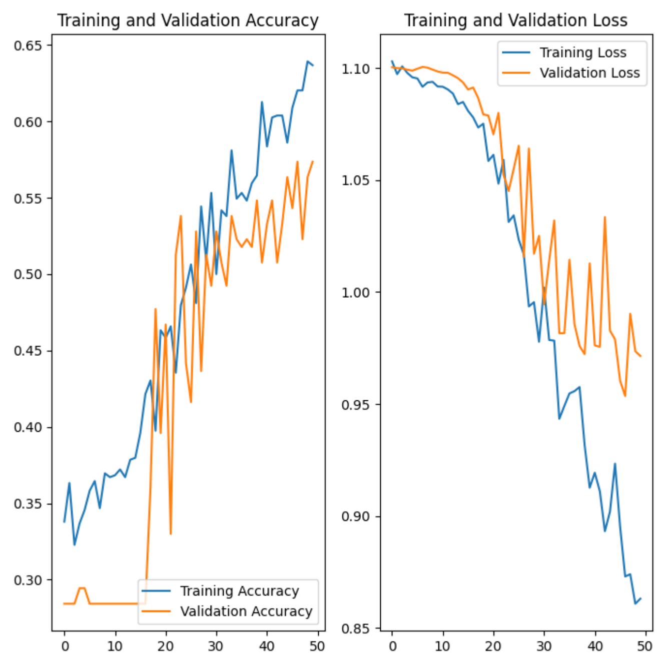


Figure 10: Accuracy and Loss Plots of the More Convolutional Layers Model

After training the simple CNN model, we observed in Figure 9 that the accuracy are fluctuating, which could suggest the need for normalization and additional layers to prevent it in overfitting. Both the training and validation losses decrease over time, which indicates that the model is learning. However, there is still a tendency to underfit, suggesting that the model is still too simple. Thus, this needs more experimentation with additional layers to capture more complex patterns.

3.3.2 Add More Layers

With insights gained from the simple model, we advanced our experimentation by increasing the model's complexity. We hypothesized that a greater number of convolutional layers would enable the capture of more patterns, which could be beneficial for the classification task, given the differences between hair types.

Following the initial experiment we made, we gained insights that helped us create a better model to meet our objectives. We hypothesized that adding more convolutional layers would help the model capture more important patterns. Increasing its complexity could be beneficial for our classification task, given that the dataset has multiple classes. The new model is structured as follows:

1. Convolutional blocks with 3×3 filters, ReLU activations, and 2×2 max pooling, increasing filter counts in a doubling pattern.

2. Global Average Pooling layer, a fully connected dense layer, and a softmax output layer, similar with the simple model architecture.

This model's training and validation metrics, as depicted in Figure 10, demonstrate a noticeable improvement in performance compared to the simple model. The training accuracy shows a steady increase with less fluctuation, while the validation accuracy stabilizes. Similarly, the training and validation loss curves show a more consistent decrease, suggesting a better fit to the data.

It can be seen that there is a slight divergence of training and validation loss towards the later epochs, which may suggest the start of overfitting. We hypothesize that this could be resolved by adding dropout layers or introducing `BatchNormalization`.

3.3.3 Flatten and Dropout

We also wanted to evaluate the effects of using a `Flatten` layer in place of `GlobalAveragePooling2D`. This help us to retain all the information from the feature map. However, we are aware that using this can potentially cause overfitting, which is why we also implemented dropout layers. This modified architecture with `Flatten` and dropout layers is structured as follows:

1. A series of convolutional layers with ReLU activations and batch normalization.

2. Dropout layers set at 0.25 following the first two convolutional layers.

3. A Flatten layer for vectorizing the feature maps before classification.

4. An expanded dense layer with 512 units for complex pattern learning.

5. A subsequent dropout layer with a rate of 0.5 following the dense layer.

6. A softmax output layer for probabilistic class predictions.

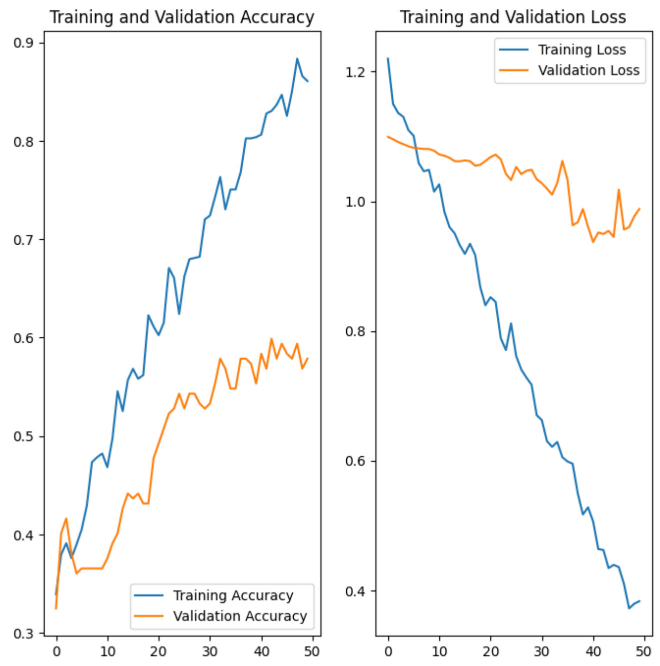


Figure 11: Training and Validation Accuracy and Loss with Flatten and Dropout Layers

In the training and validation plots (Figure 11), we can see that the training accuracy significantly increased, approaching a high of nearly 0.90. However, the model still exhibits signs of overfitting, despite the application of dropout layers to mitigate this issue. This is evidenced by the widening gap between the training and validation plots.

3.3.4 Leaky ReLU

In an effort to further refine our model, we implemented `Leaky ReLU` activation in our convolutional layers. `Leaky ReLU` is designed to overcome the limitations of the `ReLU function`. The modified architecture with Leaky ReLU is structured as follows:

1. Convolutional layers with Leaky ReLU activations set with an alpha of 0.1.

2. Batch normalization layers following each convolutional operation.

3. Dropout layers after sets of convolutional layers with a 0.25 rate.

4. A sequence of a Global Average Pooling layer followed by a Flatten layer.

5. A dense layer with a following dropout stage, leading to a softmax output layer.

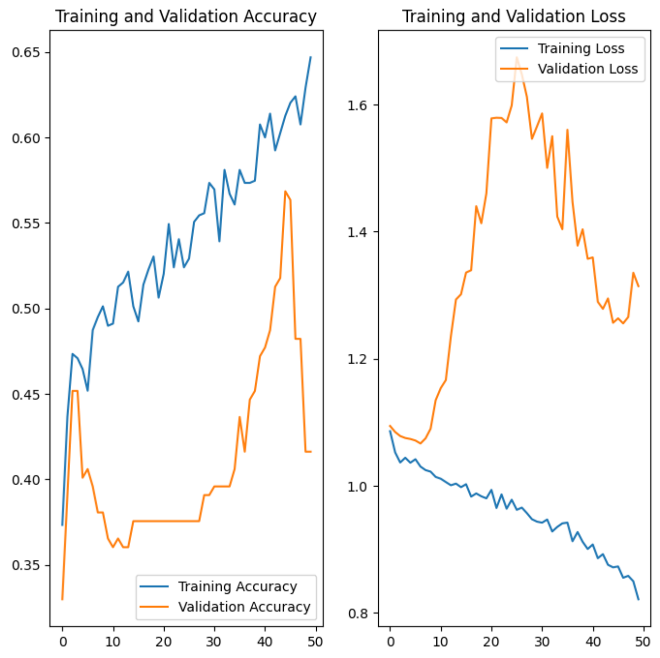


Figure 12: Training and Validation Accuracy and Loss with Leaky ReLU

As illustrated in the training and validation plots (Figure 12), there is a wide gap between the training and validation curves. This model performed worse compared to the other models we developed earlier.

3.3.5 Final Model

Leveraging the insights gained from our initial explorations, we have developed a final model aimed at surpassing the performance of our previous experiments. This final model represents a combination of techniques and layer structures that we hypothesize to be optimal for maximizin classification accuracy while avoiding overfitting. The final architecture is structured as follows:

1. A 16×16 filter convolutional layer with ReLU activation and batch normalization.

2. Additional convolutional layers with increased filter counts and reduced kernel sizes, each paired with ReLU activation, batch normalization, and max pooling.

3. Dropout layers implemented after pooling layers with a 0.25 dropout rate for the first two sets of convolutional layers.

4. A sequence of a Global Average Pooling layer followed directly by a Flatten layer.

5. A dense layer with 32 units, accompanied by a 0.5 dropout rate, culminating in a 3-unit softmax output layer.

The final model's performance, as showin in Figure 6, reflects the enhancements made during our experimentation phase. The accuracy plot reveals the most favorable outcomes compared to the prior models. It demonstrated the effectiveness of the modified CNN structure catered to our dataset. The integration of significant elements from prvious models, combined with additional fine-tuning, gave us an outcome that displays minimial overfitting compared to earlier versions. With this advanced model, we expected better results compared to others, which will be discussed in the next chapter.

3.3.6 Optimizer Configuration

In optimizing our CNN model, we chose the Adam optimizer for its adaptability and efficiency with sparse gradients. We initially set the learning rate of 1×10^{-3} to test our experiments, but observed a tendency to overfit. According to Vishwakarma (2024) Vishwakarma (2024), the level recommended for balancing fast convergence and training stability is 1×10^{-4}, which is why we used it throughout the recorded experiments. Additionally, the use of `categorical_crossentropy` is suitable for our model because we have a multi-class dataset.

3.4 U-Net model

We have also tried to dabble with other architectures that have been proposed in the past for Convolutional Neural Networks. One architectur that we implemented is the U-Net architecture by Ronneberger et al. (2015). By following their simple architecture, without using any dropout layers, using different activation functions, etc. this model is actually able to achieve a good performance. As seen in figure 13, the model is capable in its base form.

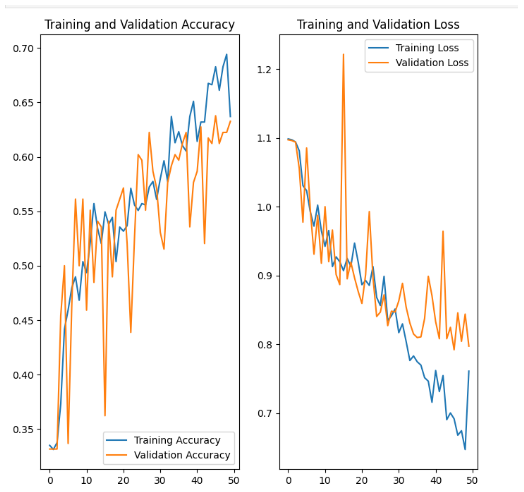


Figure 13: U-Net Performance History

IV DISCUSSION

4.1 Sobel Edge Detection and Highlighting hair Edges

Training a neural network can be a very tricky task; a lot of interconnecting parts can easily influence the other. The influence of one simple can be seen from the experimentation in each part of the process. For example, in previous lab exercises, adding or removing just one preprocessing technique can often be seen as futile or irrelevant. Oftentimes, in order to truly leverage preprocessing in the old machine learning models, these techniques need to be combined together. However there is a limit to how much preprocessing one should apply to the data, as introducing more interpolation, variance, or overall noise can be detrimental to the performance of the machine learning model.

Using Table 1 as an example, using Sobel Edge Detection (SED), it reduces overfitting when using the model without any preprocessing methods as the baseline for metrics. While Sobel Edge Detection lowered the training accuracy by around 0.02, overfitting reduced by increasing the accuracy for the validation set by 0.04. Looking at the baseline model, the difference or distance between the accuracy of training and validation sets is much larger than the model that applied SED.

Sobel Edge Detection allows the CNN to easily capture the features better. Due to SED highlighting the edges in the images, the CNN takes advantage of this modification as it can better look at how the hair flows for each sample. Looking at Figure FIGURE, hairs are no exception to the SED, strands (especially those in the edge of the person's hair) are highlighted and emphasized in the image usually as a white color or something that is in complete contrast with the color assigned to non-edges. Kernels can find these more easily therefore train more easily ultimately leading to a better performance.

4.2 Data Augmentation and Noise

Oftentimes data augmentation can be of help with machine learning models to better their performance in the task they were meant to do. But sometimes data augmentation can actually be harmful. Data augmentation can actually increase noise by variating data. Such data augmentations can be good to train a network similar to Kim et al. (2021)'s work, however there needs to be a good foundation of having a good network. For a network to be robust to noise, it must be well modelled to be capable of being trained for robustness in the first place. Without a well built model, introducing noise can be detrimental as seen in the experimentations. Table 1 shows the effects of data augmentation to the model, in both cases where it is isolated and combined with Sobel Edge Detection, the model had performed worse.

4.3 Overfitting

Overfitting is a lot easier when dealing with neural networks. Because of the thousandas or millions of little parts that influence each other, mismanagement of a feature can easily lead to overfitting. Looking at the experiments, flattening and using dropout layers, and changing the activation function to LeakyReLU can lead to overfitting without proper "compensation" techniques. In a way to reduce overfitting but maintain the use of flattening and dropout layers, Global Average Pooling can be done first to reduce the chances of overfitting the model.

More layers doesn't always mean more performance; there is the optimal amount of layers that must be identified. Batch Normalizaiton helps in reducing overfitting by applying a transformation that converts inputs to conform in the range $[0, 1]$. According to the tensorflow API, Batch Normalization works by applying the following equation:

$$\gamma \cdot \left(\frac{\text{batch} - mean(\text{batch})}{\sqrt{var(\text{batch}) + \epsilon} + \beta} \right)$$

where the ϵ, γ, β are hyperparameters tunable by its users.

As for LeakyReLU, the dichotomy between training and validation accuracy are as big as ever. Figure 12 shows this severe case. Due to this experimentation, we have decided to continue with ReLU instead as the activation function.

4.4 Other Models

Another model was also experimented on to see how it will compare in performance. The U-Net model has done well, reaching as high as 0.70 in accuracy but still converge with the accuracy score of the validation set. However, the convergence of the training and validation scores are only because of chance; if it were any other epoch that were about 10 epochs apart to the current one, the model would've been greatly off. In other words, it lacks consistency, training this with proper regularization or overfitting measures would surely improve performance. However, there is still the need to design our own model from scratch, but as a reference point, this is how a vanilla or plain U-Net would perform in this dataset; adding things such as other activation functions, dropout layers, and such would surely increase performance but that will be outside of the scope for now.

4.5 Kernels

Convolutinal Neural Networks often have multiple kernels and multiple sequences of kernels, We can see how these kernels eventually converge into selecting the most important feature to use for the dense neural network layer. Looking at figure 14, the first convolution of the model were a bit more inconsistent, but looking at the 11th layer, as shown in figure 15, this convolution layer has 64 output kernels but looking at only three of them, we can see that the kernels weights are more consistent; they all look for the same features or at the very least have resemblance with the other filters.

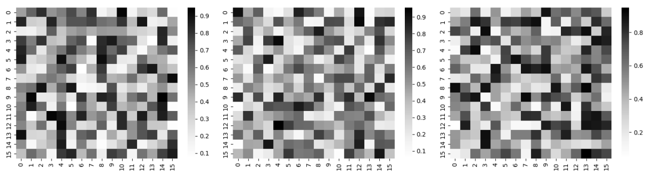


Figure 14: Filters in the First Convolution Layer

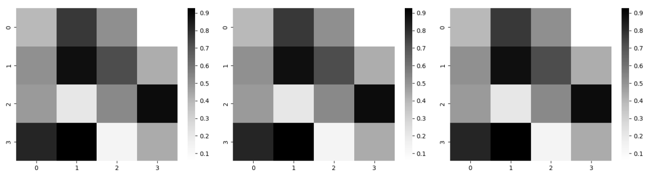


Figure 15: Filters in the Third Convolution Layer

V CONCLUSION

Creating a Convolutional Neural Network through this lab exercise have provided valuable insights into what it takes to design a machine learning for deep learning and image processing. By repetitive and meticulous experimentation, we were able to figure out what was most effective in terms of architectural configurations, activation functions, optimization techniques, etc. to achieve optimal performance for detecting hair types.

We have witnessed firsthand how small adjustments can lead to significant improvements or detriments in performance, truly encapsulating the ideal of balance between complexity and generalization to avoid overfitting and possess the ability for generalization.

Additionally, this lab exercise provided insights for CNN itself especially how convolution layers, pooling operations, and the hierarchical feature extraction process works. This gives us the edge on the skills for designing image processing models and how to really improve them.

In the future, the knowledge and skills acquired from this lab exercise is a solid foundation for further exploration in deep learning. As seen in one experimentation, models from previous studies (i.e. the U-Net architecture) are not only understood but also implemented. As we continue this journey, we will learn more and more about the field of machine learning and use it to deal with real-world challenges with ingenuity, creativity, and efficiency.

References

Kim, E., Kim, J., Lee, H., & Kim, S. (2021). Adaptive data augmentation to achieve noise robustness and overcome data deficiency for deep learning. *Applied Sciences*, 11(12), 5586.

Ronneberger, O., Fischer, P., & Brox, T. (2015). U-net: convolutional networks for biomedical image segmentation. In N. Navab, J. Hornegger, W. M. Wells, & A. F. Frangi (Eds.), *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015* (pp. 234–241). Cham: Springer International Publishing.

Thambawita, V., e. a. (2021). Impact of image resolution on deep learning performance in endoscopy image classification: An experimental study using a large dataset of endoscopic images. *Journal of AI Research*, 11, 2183.

Vishwakarma, N. (2024). What is adam optimizer? https://www.analyticsvidhya.com/blog/2023/09/what-is-adam-optimizer/.