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.

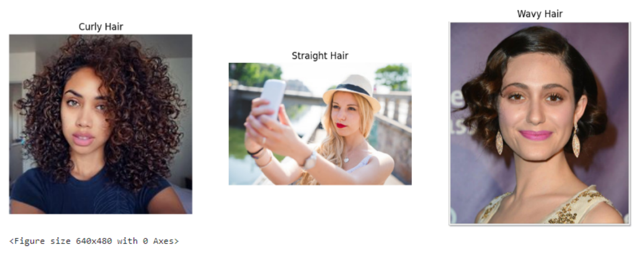

Figure 1: Hair Types Example

II METHODOLOGY

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

$$\text{data} \rightarrow \text{preprocessing} \rightarrow \text{modeling} \rightarrow \text{evaluation}$$

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 exclude 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 that no invalid images were overlooked, we manually inspected the images through file explorer. Through this inspection, it was detected that there is one image in `Straight_Hair` subdirectory that is just a microphone logo. Thus, excluding this is beneficial to prevent the model learning unnecessary and wrong 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 × 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.

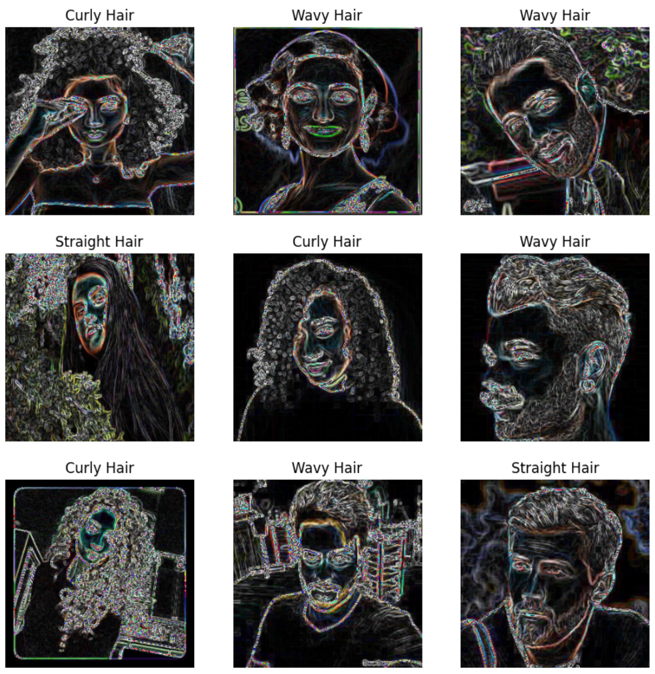

Figure 2: Hair Types Example with Sobel Edge

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.

10. 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.

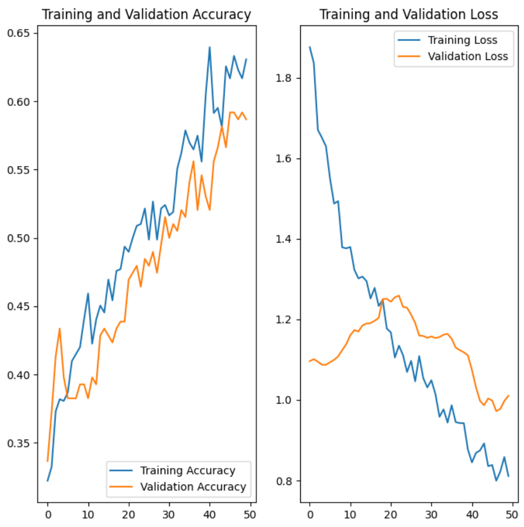

Figure 3: Training and Validation Results

2.4 Evaluation

To evaluate the performance of the convolutional neural network, we utilized different techniques that visually and quantitatively 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.

? 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.

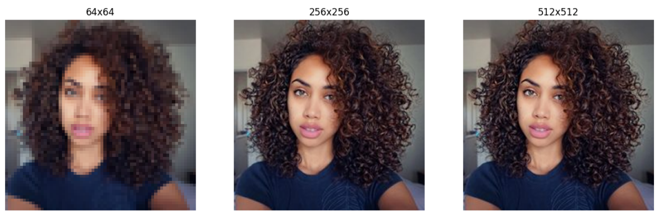

Figure 4: Different Image Resolution

Regarding the batch size, we experiments 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.

For the batch size, we used 64. We tested using batch size of 128 and 32 for this model since they are all divisible by 256 which is the image size, but we opted to continue using 64 for a better balance of requiring less memory in training and the accuracy of the 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
- With Sobel Edge Detection only
- With Data Augmentation only
- With 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.

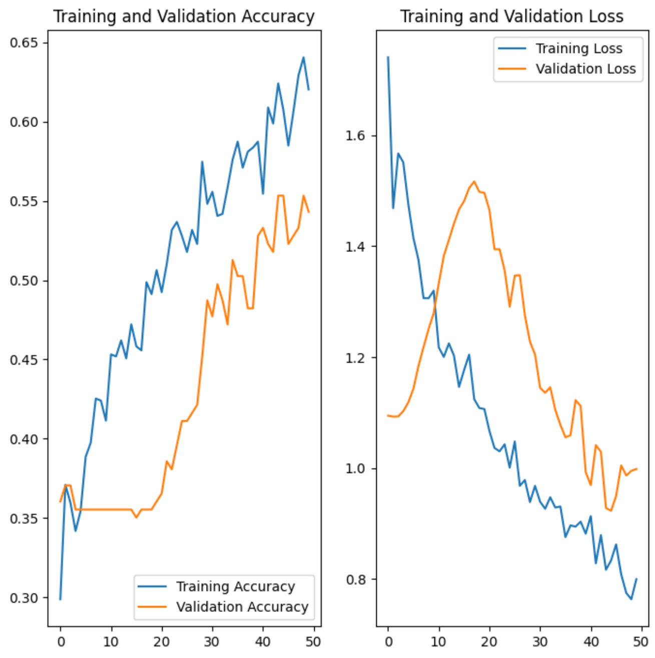

Figure 5: No Preprocessing

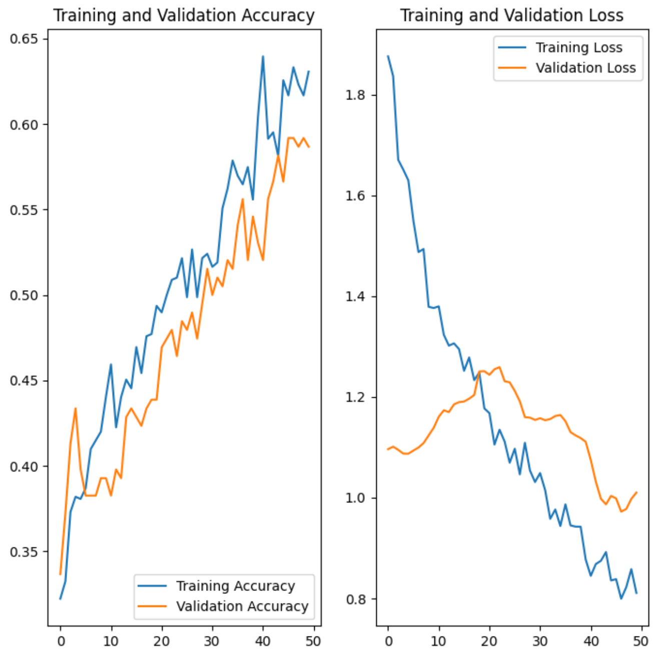

Figure 6: With Sobel Edge Detection only

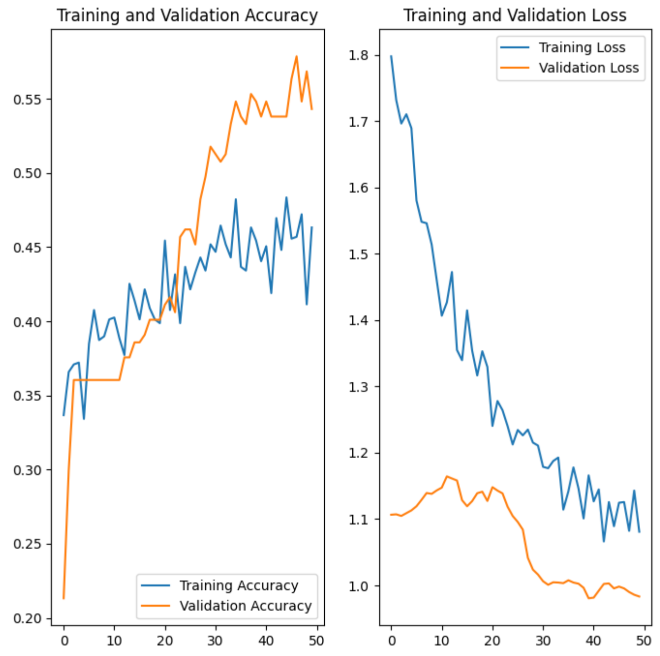

Figure 7: With Data Augmentation only

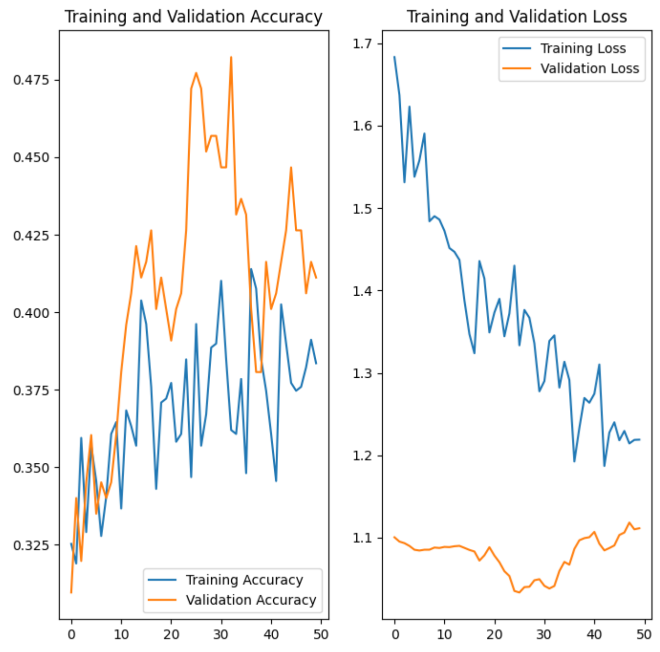

Figure 8: With both Sobel Edge Detection and Data Augmentation

The use of sobel edge detection help improved object detection and segmentation, notably improving hair boundary delineation in the images as seen in Figure 8. Given our dataset only consists of 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

3.3.1 Convolutional Layers

3.3.2 Pooling Layers

3.3.3 Batch Normalization

3.3.4 Dropout Layers

3.3.5 GlobalAveragePooling2d and Flattening

3.3.6 Training the Model

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

V CONCLUSION

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

?
this is a reference (**?**) vs the regular reference **?**.