Experimenting with Convolutional Neural Network (CNN) Components for Hair Type Classification

Machine Learning (CSELEC2C) – Laboratory Activity 3

I. INTRODUCTION

Artificial intelligence has indeed revolutionized various aspects of our lives, offering solutions to optimize everyday tasks and improve efficiency. From virtual assistants like Siri and Google Assistant to object detection in vehicles, AI has become integral in modern living. One significant application of AI lies in automating classification problems, such as distinguishing between spam and non-spam emails. Among these classification tasks, image classification stands out, offering numerous applications from recognizing handwritten characters to diagnosing diseases through X-ray scans. Whether for simple tasks or complex medical diagnoses, image classification aids in streamlining processes and enhancing productivity.

Convolutional Neural Networks (CNNs) serve as a powerful tool in image classification, leveraging principles from linear algebra to identify patterns and features within images. CNNs consist of three main components: the convolutional layer, pooling layer, and fully connected layer. The convolutional layer extracts features from the input image through filters, while the pooling layer reduces dimensionality, enhancing efficiency and mitigating overfitting. Finally, the fully connected layers integrate these features to perform classification. For instance, in a hair type classification problem, CNNs can analyze images to classify individuals' hair as curly, straight, or wavy, showcasing the versatility and effectiveness of CNNs in image classification tasks.

In addressing the specific problem of classifying hair types, CNNs offer a scalable approach, capable of learning intricate patterns and nuances within images. By leveraging convolutional layers to extract relevant features and fully connected layers for classification, CNNs can accurately identify different hair types. Whether for personal grooming apps or salon recommendation systems, CNNs provide a robust solution for automating hair type classification, demonstrating the broad applicability of AI in enhancing various aspects of our lives through image classification technologies.

1.1 Scope and Limitations

This study contains a Convolutional Neural Network (CNN) for an image classification problem for identifying hair types.

Furthermore, the hardware infrastructure utilized for conducting this study includes an 11th Gen Intel Core i5-11400 processor and NVIDIA GeForce RTX 3060 Laptop GPU. The experiments were compiled and run using Jupyter Notebook.

II. METHODOLOGY

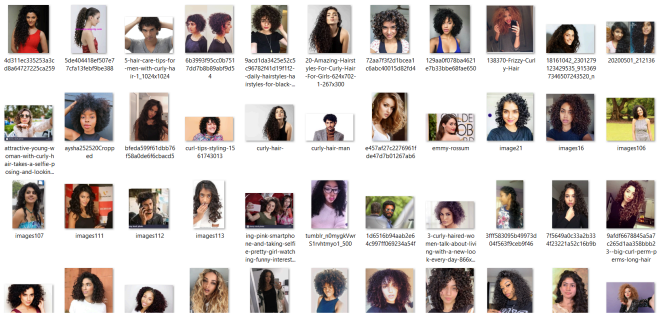


Figure 1. The initial data set

Our investigation commences with a thorough examination of a rich dataset (refer to Figure 1) comprising 994 images representing a diverse range of hair types. Within this dataset, we meticulously analyze 339 instances of curly hair, 324 instances of straight hair, and 331 instances of wavy hair. These images encapsulate individuals of varying ages, both genders, and diverse ethnicities, ensuring a comprehensive representation of the population under study.

2.1 Data Cleaning

After filtering the dataset, images in formats other than PNG, JPG, BMP, GIF, and JPEG have been discarded due to their ineligibility for processing. Primarily, images in the WEBP format were excluded. Following this filtration process, the dataset now comprises 985 images, ensuring compatibility with the processing pipeline.

2.2 Image Formatting

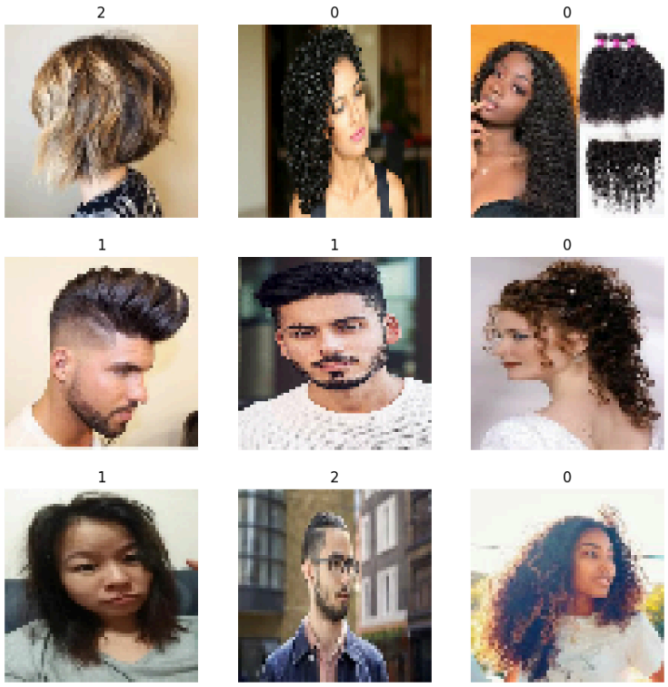


Figure 2. 3x3 Matrix of Normalized Images

In order for the neural network to be able to use the remaining input data with ease, it must delve into the crucial process of preparing image data for neural network consumption. To analyze the impact of different image sizes on the performance of our convolutional neural network (CNN) model, we conducted a series of experiments and employed statistical analysis to interpret the results.

a. Image Resizing

Table 1. Image for Configuring the Resizing of Images

	Image Size
Baseline Model	64x64
Experiment 1	128x128
Experiment 2	32x32
Experiment 3	256x256

This table outlines the image sizes employed for configuring image resizing in the experiments, illustrating the significance of this preprocessing step. Image resizing plays a crucial role in optimizing model performance by adjusting the input image dimensions to suit the requirements of the convolutional neural network (CNN) architecture. Different image sizes allow researchers to explore how variations in resolution impact the model's ability to extract features and classify images accurately. The baseline model, with images resized to 64x64 pixels, is a reference point for comparing the effects of different resizing strategies. Experimentation with larger and smaller image sizes provides insights into the trade-offs between model complexity, computational efficiency, and classification accuracy.

b. Batch Size

Aside from this, we attempted to change the Batch size of the baseline study if there would be changes in the Loss and Accuracy.

Table 1.1 Experiments for Configuring the Batch Size

	Batch Size
Baseline Model	32
Experiment 1	100

This table presents the experiments conducted to investigate the effect of batch size on model training. Batch size is crucial in determining the efficiency and effectiveness of the training process in deep learning models. The baseline model utilizes a batch size of 32 as a reference point. Experiment 1 explores the impact of increasing the batch size to 100. By varying the batch size, researchers can assess its influence on convergence speed, training stability, and generalization performance. Understanding the optimal batch size is essential for optimizing training efficiency and achieving better model performance.

2.3 Data Split, Test, and Validation

The dataset is divided into batches, each containing 32 images, resulting in a total of 30 batches (rounded down). Utilizing batches instead of individual data enhances efficiency as it allows the neural network to train on fewer samples, leading to faster processing due to reduced workload. These batches are further categorized into three distinct groups: training, validation, and testing.

As their names suggest, the training group is utilized to train the model, while the validation group serves as a means to compare the model's performance during the training process. Once the model is trained, it undergoes evaluation using the test group to determine its final performance score. The allocation of images into each group follows a distribution of 70%, 20%, and 10%, respectively, resulting in 21, 6, and 3 batches allocated to training and validation groups.

This distribution aligns with the general standard of splitting data into training, validation, and test groups, where approximately 60-80% of the data is reserved for training, and 10-20% is allocated to both the validation and test groups.

2.4 Beyond Basic Preprocessing: Exploring Different Image Preprocessing Techniques

Prior to feeding our data into the neural network, we employ a crucial preprocessing step: rescaling our images from the original range of 0-255 to a normalized range of 0-1. This normalization facilitates smoother processing within the neural network architecture and enhances the efficiency of subsequent functions, such as softmax. By rescaling the pixel values, we ensure that the neural network can effectively learn and extract meaningful features from the images while optimizing computational resources for improved performance.

This study investigates the impact of various pre-processing techniques on CNN performance. We utilize a baseline model trained with standard image normalization (0-1). Subsequently, we perform the following experiments:

Normalization Removal: We train a model without any image normalization to assess the baseline performance and understand the influence of normalization on feature extraction.

Random Cropping: We randomly crop sections of the training images, exposing the model to various portions of the data and potentially enhancing its ability to generalize to unseen variations.

Rotation: We rotate the training images by random angles, aiding the model in becoming invariant to object orientation within the images.

Grayscale Conversion: We convert the training images to grayscale, simplifying the data and reducing the number of channels the model needs to process, which could potentially improve efficiency for specific tasks.

Grayscale w/ Gaussian Blur: We convert the training images to grayscale and apply a Gaussian blur filter. This combined approach investigates if focusing the model on essential features through both simplification and noise reduction can improve its performance.

Brightness and Contrast Adjustments: We introduce variations in illumination by randomly adjusting brightness and contrast in the training images. This aims to make the model more robust to changes in lighting conditions encountered in real-world scenarios.

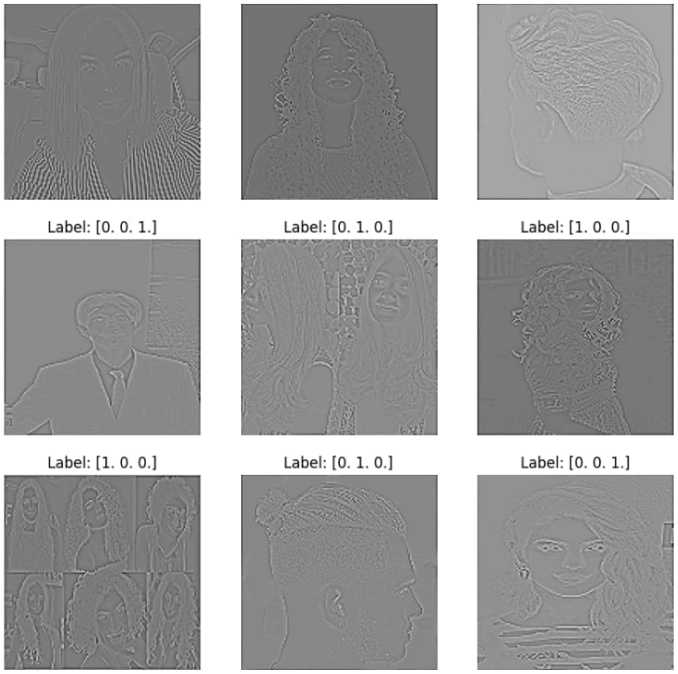


Figure 3. Laplacian Image with Grayscale

Laplacian Sharpening with Grayscale: Finally, we apply Laplacian sharpening to the training images, aiming to enhance edge details and increase the contrast between features. This technique enhances the training process of the Convolutional Neural Network (CNN) by accentuating important visual cues, which may lead to better feature extraction and improved classification accuracy. **Further into proposing our own model, we will introduce laplacian with RGB and showcase its performance.**

Table 2. Experiments for Image Normalization Techniques

	Image Normalization Techniques
Baseline Model	Rescaling pixel val [0,1]
Experiment 1	No Normalization

Experiment 2	Cropping
Experiment 3	Rotation
Experiment 4	Grayscale
Experiment 5	Graysc + Gaussian Blur
Experiment 6	Gaussian Blur only
Experiment 7	Brightness and Contrast
Experiment 8	Median blur, kernel size = 2
Experiment 9	Laplacian Sharp; 25 sc factor

2.5. Experiments and Exploration for different Pooling methods

In this subsection, we detail the experiments conducted to explore the impact of different types of pooling techniques on the performance of the Convolutional Neural Network (CNN) for hair type classification. Pooling layers play a crucial role in CNN architectures by reducing spatial dimensions, capturing dominant features, and aiding in translation invariance.

Table 3. Experiments for different types of Pooling.

	Type of Pooling
Baseline Model	Global Average Pooling 2d
Experiment 1	Max Pooling after each convolutional layer (2x2) with a dropout layer for regularization (w/Laplacian Sharpening)
Experiment 2	Max Pooling after each convolutional layer (2x2) with a dropout layer for regularization
Experiment 3	Max pooling without laplacian and applying REGULARIZATION/DROPOUT LAYER for every conv layer followed by max pooling
Experiment 4	Global Max Pooling (w/Laplacian Normalization)
Experiment 5	Omitted ANY type of Pooling and replaced with Flatten (w/ Laplacian Normalization
Experiment 6	Added average pooling at the end
Experiment 7	Max Pooling at each while adding GlobalAveragePooling at the last layer
Experiment 8	Max Pooling with pool size of 3x3
Experiment 9	Max Pooling with pool size of 2x2
Experiment 10	Max Pooling with pool size of 2x2 and 2x2 stride

This table presents an overview of the experiments designed to evaluate various pooling strategies applied in the CNN model. Each row corresponds to a specific experiment, where different configurations of pooling layers are tested to assess their effect on model performance metrics such as accuracy, overfitting, and computational efficiency. The table provides a concise summary of the pooling techniques employed in each experiment, facilitating comparison and analysis of their respective outcomes.

Baseline: Global Average Pooling 2D
The average of each feature map across all spatial locations was computed. In 2D Global Average Pooling, the average for each feature map was independently computed.

Experiment 1: Max Pooling after each convolutional layer (2x2) with a dropout layer for regularization (w/Laplacian Normalization)
After each convolutional layer, we apply a 2x2 Max Pooling layer to extract the maximum value from each subregion of the input feature map. To prevent overfitting, we add a dropout layer that randomly sets a fraction of input units to zero during training. Additionally, we may use Laplacian Sharpening to enhance model performance by normalizing the feature maps after pooling.

Experiment 2: Max Pooling after each convolutional layer (2x2) with a dropout layer for regularization
Similar to Experiment 1, but without Laplacian Normalization, we apply Max Pooling after each convolutional layer and add a dropout layer for regularization.

Experiment 3: Max pooling without Laplacian and applying Regularization/Dropout Layer for every Convolutional Layer followed by max pooling
We apply a dropout layer for regularization after every convolutional layer, followed by Max Pooling after each layer.

Experiment 4: Global Max Pooling (w/Laplacian Normalization)

Global Max Pooling computes the maximum value of each feature map across all spatial locations.

Experiment 5: Omitted ANY type of Pooling and replaced with Flatten (w/ Laplacian Normalization)

We replace any type of pooling with a Flatten operation, which transforms the input into a 1D array suitable for feeding into a fully connected layer. Laplacian Normalization may be applied before or after flattening.

Experiment 6: Added average pooling at the end

At the end of the network architecture, we apply Average Pooling to compute the average value of each feature map across all spatial locations.

Experiment 7: Max Pooling at each while adding GlobalAveragePooling at the last layer

We apply Max Pooling after each convolutional layer and add Global Average Pooling at the last layer. This aggregates information from all feature maps.

Experiment 8: Max Pooling with pool size of 3x3

We extract the maximum value from each subregion of the input feature map using a 3x3 pooling window.

Experiment 9: Max Pooling with pool size of 2x2

Same with 3x3 but just changed to 2x2.

Experiment 10: Max Pooling with pool size of 2x2 and 2x2 stride

We extract the maximum value from each subregion of the input feature map using a 2x2 pooling window with a stride of 2.

2.6 Experimentation for Filters

In this subsection, we delve into the experimentation conducted to explore the impact of different filter configurations on the performance of our Convolutional Neural Network (CNN) for hair type classification. Filters, also known as kernels, are essential components of CNNs, used to extract features from input data through convolution operations.

Table 4. Experimentation with filter configurations in CNN Architecture

	Filters
Baseline Model	4, 8, 16
Experiment 1	Filters = 1 for all layers
Experiment 2	Filters = 5 for all layers
Experiment 3	Filters = 10 for all layers
Experiment 4	2, 4, 6
Experiment 5	4, 4, 4
Experiment 6	16, 32, 64
Experiment 7	16, 32, 64 (with batch normalization and dropout and kernel regularizer)
Experiment 8	8, 16, 32
Experiment 9	8, 16, 32 with 75 epochs
Experiment 10	DECREASING for deeper layers Filters: 16, 8, 4
Experiment 11	32, 16, 8

Experiment 1: Filters = 1 for all layers

In this experiment, we set the number of filters to 1 for all layers in the CNN architecture to observe the impact of minimal feature extraction.

Experiment 2: Filters = 5 for all layers

We increase the number of filters to 5 for all layers to evaluate the effect of a moderate amount of feature extraction.

Experiment 3: Filters = 10 for all layers

Here, we further increase the number of filters to 10 for all layers to assess the influence of a higher number of feature extraction.

Experiment 4: Filters = 2, 4, 6

We vary the number of filters across layers, with 2 filters for the first layer, 4 for the second, and 6 for the third, to examine the impact of different filter sizes throughout the network.

Experiment 5: Filters = 4, 4, 4

All layers are configured with an equal number of filters (4) to investigate the effects of uniformity in filter quantity.

Experiment 6: Filters = 16, 32, 64
In this experiment, we progressively increase the number of filters across layers to observe the effects of a higher number of filters in deeper layers.

Experiment 7: Filters = 16, 32, 64 (with batch normalization and dropout and kernel regularizer)
Similar to Experiment 6, but with the inclusion of batch normalization, dropout, and kernel regularization techniques to further enhance model performance and stability.

Experiment 8: Filters = 8, 16, 32
We adjust the number of filters across layers to 8, 16, and 32, respectively, to explore the effects of a different filter configuration.

Experiment 9: Filters = 8, 16, 32 with 75 epochs and 100 epochs variation
This experiment involves varying the number of training epochs (75 and 100) while maintaining the filter configuration to observe any differences in model performance.

Experiment 10: DECREASING for deeper layers: Filters = 16, 8, 4
We reverse the trend of increasing filter numbers in deeper layers by decreasing the number of filters across layers to evaluate its impact.

Experiment 11: Filters = 32, 16, 8
In this experiment, we decrease the number of filters across layers to explore the effects of a decreasing filter configuration.

2.7 Experimenting with Kernel Size
In this subsection, we delve into the experimentation conducted to explore the effects of varying kernel sizes on the performance of our Convolutional Neural Network (CNN) for hair type classification. Kernel size refers to the dimensions of the convolutional filter used to extract features from the input data. Throughout these experiments, the Laplacian preprocessing method remains constant, ensuring consistency in the preprocessing steps, while the CNN model architecture remains unchanged. We investigate how different kernel sizes impact the model's ability to learn and extract meaningful features from the input images, ultimately aiming to identify the optimal kernel size configuration for our CNN architecture.

Table 5. Kernel Size configurations for the Experimentations

	Kernel Size
Baseline Model	16, 8, 4
Experiment 1	4, 4, 4
Experiment 2	8, 8, 8
Experiment 3	16, 16, 16
Experiment 4	4, 8, 16 (increasing)
Experiment 5	5, 3, 3
Experiment 6	18, 9, 5 - With batch normalization and dropout

In Experiment 1, we deviate from the baseline model by employing a uniform filter configuration of 4 filters for each layer. Experiment 2 continues this trend of uniformity, with all layers configured with 8 filters each. Experiment 3 explores the effects of further increasing filter counts uniformly across layers, with 16 filters for each layer. Experiment 4 introduces an increasing trend in filter counts from the first to the last layer, with configurations of 4, 8, and 16 filters, respectively, aiming to assess the impact of such variation. Experiment 5 takes a novel approach by utilizing non-traditional filter sizes of 5, 3, and 3 for the respective layers, potentially offering unique feature extraction capabilities. Lastly, Experiment 6 employs larger filter sizes of 18, 9, and 5, respectively, while incorporating batch normalization and dropout techniques to enhance model stability and performance. Through these experiments, we aim to discern the influence of different filter configurations on the CNN's ability to extract relevant features for hair type classification tasks.

2.8. Experimentations for Padding

In these experiments, we systematically explore the effects of different padding strategies, specifically "same" and "valid," on the performance of our Convolutional Neural Network (CNN) for hair type classification. The padding strategy determines how the input data is processed at the edges of the feature maps during convolution operations.

Table 6. Padding Configurations

	Padding
Baseline Model	3 Conv layers & 'valid, valid, valid', respectively
Experiment 1	All 'same'
Experiment 2	Valid, valid, same
Experiment 3	Valid, same, valid
Experiment 4	Valid, same, same
Experiment 5	Same, valid, valid
Experiment 6	same, valid, same
Experiment 7	Same, same valid

Baseline: Valid, valid, valid
In the baseline configuration, denoted as "baseline," we employ the "valid" padding strategy consistently across all convolutional layers in the Convolutional Neural Network (CNN) architecture for hair type classification. This choice of padding ensures that the convolutional operations are applied only to valid input data, without any additional zero-padding at the edges.

Experiment 1: All 'same'
In this experiment, we apply the "same" padding strategy for all convolutional layers. This ensures that the output feature maps have the same spatial dimensions as the input, which may facilitate better preservation of spatial information.

Experiment 2: Valid, valid, 'same'
Here, we apply the "valid" padding strategy for the first two convolutional layers and "same" padding for the last layer. This configuration allows us to investigate the impact of using different padding strategies across layers.

Experiment 3: Valid, 'same', valid
Similar to Experiment 2, but with the padding strategies arranged differently. The first and last layers use "valid" padding, while the middle layer uses "same" padding. This variation aims to assess how the distribution of padding strategies affects model performance.

Experiment 4: Valid, 'same', 'same'
In this experiment, we employ a combination of "valid" and "same" padding, with the first layer using "valid" padding and the subsequent layers using "same" padding. This configuration aims to strike a balance between preserving spatial information and reducing computational overhead.

Experiment 5: 'Same', valid, 'same'
Here, we start with "same" padding, followed by a layer with "valid" padding, and conclude with "same" padding. This arrangement explores the impact of different padding strategies at different stages of the network.

Experiment 6: 'Same', 'same' valid
Finally, we explore a configuration where the first two layers use "same" padding, while the last layer uses "valid" padding. This setup aims to assess the effects of gradually reducing spatial dimensions through the network.

2.9. Experiments for Strides
In this subsection, we conduct experiments to investigate the effects of varying stride values on the performance of our Convolutional Neural Network (CNN) for hair type classification. Stride values determine the step size of the convolutional filter as it moves across the input data. By experimenting with different stride values, we aim to analyze how the spatial resolution of feature maps and the overall model performance are affected.

Table 7. Stride Configurations

	Stride
Baseline Model	3 Conv layers & '1, 1, 1' stride, respectively

Experiment 1	2 for all
Experiment 2	1, 2, 2
Experiment 3	1,1,2
Experiment 4	1,2,1
Experiment 5	2,1,1
Experiment 6	2,2,1
Experiment 6	2,1,2

Baseline:
In our baseline model, we employ three convolutional layers with a consistent stride value of 1 for each layer. This configuration ensures that the convolutional filters move across the input data with a step size of 1 pixel, preserving spatial resolution and enabling detailed feature extraction. By maintaining uniformity in the stride values across all convolutional layers, we establish a reference point for comparison with other experimental configurations.

Experiment 1: 2 for all
In this experiment, we set the stride value to 2 for all convolutional layers. This configuration effectively downsamples the spatial dimensions of the feature maps by a factor of 2, potentially reducing computational complexity and memory requirements.

Experiment 2: 1, 2, 2
Here, we apply different stride values for each convolutional layer, with strides of 1, 2, and 2, respectively. By varying the stride values across layers, we aim to explore their individual and cumulative effects on feature extraction and model performance.

Experiment 3: 1, 1, 2
Similarly, we configure the stride values as 1, 1, and 2 for the respective layers in this experiment. This setup allows us to assess how different combinations of stride values impact the spatial resolution of feature maps and the ability of the model to capture relevant information.

Experiment 4: 1, 2, 1
In this experiment, we vary the stride values across layers, with strides of 1, 2, and 1, respectively. By exploring different arrangements of stride values, we aim to discern their effects on feature map generation and the model's capacity to extract discriminative features.

Experiment 5: 2, 1, 1
Here, we reverse the order of stride values compared to Experiment 4, with strides of 2, 1, and 1 for the respective layers. This configuration allows us to investigate the impact of changing the order of stride values on feature extraction and model performance.

Experiment 6: 2, 2, 1
Finally, we explore a configuration where the first two layers have stride values of 2, while the last layer has a stride value of 1. This setup evaluates the effects of gradually decreasing the stride value across layers on feature map generation and model performance.

2.10. Experimentations for configuring the Dilation
In this section, we conduct experiments to explore the effects of configuring different dilation rates ni our CNN architecture for hairtype classification. Dilation determine the spacing between the kernel elements during the convolution, which allows the network to capture features at different scales and resolutions.

Table 8. Different Configurations for Dilation Rate

	Dilation Rate
Baseline Model	3 Conv layers & ‘1, 1, 1’dilation rate, respectively
Experiment 1	2 for all
Experiment 2	3 for all
Experiment 3	3, 2, 1
Experiment 4	2, 1, 1
Experiment 5	2,2,1
Experiment 6	1,1,2
Experiment 6	1,2,3

Table 8 presents the experiments conducted to investigate the impact of different dilation rate configurations on the performance of our Convolutional Neural Network (CNN) architecture for hair type classification. The baseline model consists of three convolutional layers, each with a dilation rate of '1' to maintain spatial resolution during feature extraction. Subsequent experiments explore variations in dilation rates across layers, aiming to assess their effects on feature representation and model performance. Experiment 1 investigates the effects of setting the dilation rate to '2' for all layers, while Experiment 2 explores the impact of a dilation rate of '3' for all layers. Experiment 3 introduces a configuration of '3, 2, 1' for dilation rates across the three layers, followed by Experiment 4 with '2, 1, 1' and Experiment 5 with '2, 2, 1'. Additionally, Experiment 6 explores configurations of '1, 1, 2' and '1, 2, 3', providing a comprehensive analysis of the effects of varying dilation rates on model performance. Through these experiments, we aim to identify optimal dilation rate configurations that enhance the CNN's ability to capture relevant features for hair type classification tasks.

2.11. Experimentations for using various Optimizers

In this section, we explore the impact of using different optimization algorithms, known as optimizers, on the training process and performance of our CNN model. By experimenting with various optimizers such as Adam, SGD, and RMSprop, we aim to identify the most suitable optimization algorithm for enhancing convergence speed and model accuracy.

Table 7. Use of Different Optimizers

	Optimizer
Baseline Model	Adam Optimizer (1e-3)
Experiment 1	SGD (learning_rate=1e-3, momentum=0.9)
Experiment 2	RMSprop (learning_rate=1e-3)
Experiment 3	Adagrad (learning_rate=1e-3)
Experiment 4	Adadelta (learning_rate=1e-3)
Experiment 5	Adadelta (learning_rate=1.0)
Experiment 6	Adamax (learning_rate=1e-3)

This table outlines the outcomes of experiments conducted to assess the effects of employing different optimization algorithms on the performance of our Convolutional Neural Network (CNN) architecture for hair type classification. The baseline model utilizes the Adam optimizer with a learning rate of 1e-3 as a reference point. Experiment 1 explores the use of Stochastic Gradient Descent (SGD) with a learning rate of 1e-3 and momentum of 0.9. Experiment 2 investigates the RMSprop optimizer with a learning rate of 1e-3. Experiment 3 examines the performance of the Adagrad optimizer with a learning rate of 1e-3. Experiment 4 evaluates the Adadelta optimizer with a learning rate of 1e-3. Experiment 5 further explores the Adadelta optimizer with a higher learning rate of 1.0. Lastly, Experiment 6 introduces the Adamax optimizer with a learning rate of 1e-3. Through these experiments, we aim to determine the most effective optimization algorithm for improving model convergence and accuracy in hair type classification tasks.

2.12. Exploration of using different Activation Functions

Table 8. Experiment using Different Activation Functions

	Activation Function
Baseline Model	ReLU
Experiment 1	Leaky ReLU for all layers except for the last dense layer (softmax activation)
Experiment 2	Leaky ReLU for all layers including dense layers
Experiment 3	Exponential Linear Unit (ELU) for all layers except for the last dense layer (softmax activation)
Experiment 4	SELU for all layers
Experiment 5	GELU for all layers
Experiment 6	Swish for all layers (softmax activation for the last dense layer)

Experiment 7	Sigmoid for all layers (softmax activation for last dense layer)
Experiment 8	Hyperbolic tangent (tanh) for all layers
Experiment 9	Hard Swish for all layers
Experiment 10	Mish for all layers

Experiment 1: Leaky ReLU for all layers except for the last dense layer (softmax activation)
In this experiment, we utilize the Leaky ReLU activation function for all convolutional layers, introducing a slight negative slope for negative input values to prevent dead neurons. However, for the last dense layer, we retain the softmax activation function to ensure proper probability distribution for multiclass classification tasks.

Experiment 2: Leaky ReLU for all layers, including dense layers
Here, we extend the use of the Leaky ReLU activation function to include all layers, including the dense layers. This configuration aims to maintain consistency in the activation function throughout the network, potentially enhancing feature extraction and model expressiveness.

Experiment 3: Exponential Linear Unit (ELU) for all layers except for the last dense layer (softmax activation)
Similarly, in this experiment, we apply the ELU activation function for all convolutional layers, allowing for improved learning dynamics by reducing the vanishing gradient problem. However, the last dense layer retains the softmax activation function for appropriate output probabilities in multiclass classification tasks.

Experiment 4: SELU for all layers
In this experiment, we employ the Scaled Exponential Linear Unit (SELU) activation function for all layers. SELU introduces self-normalizing properties, potentially leading to faster convergence and improved performance, particularly in deeper networks.

Experiment 5: GELU for all layers
Here, we explore the Gaussian Error Linear Unit (GELU) activation function for all layers. GELU introduces non-linearity with smooth gradients, potentially enhancing the model's ability to capture complex patterns in the data.

Experiment 6: Swish for all layers (softmax activation for last dense layer)
In this experiment, we utilize the Swish activation function for all layers, introducing a non-linear activation function that may offer improved performance compared to traditional ReLU-based activations. The softmax activation function is retained for the last dense layer for proper probability distribution in multiclass classification tasks.

Experiment 7: Sigmoid for all layers (softmax activation for last dense layer)
Here, we apply the Sigmoid activation function for all layers, which is suitable for binary classification tasks. However, the softmax activation function is retained for the last dense layer to accommodate multiclass classification scenarios.

Experiment 8: Hyperbolic tangent (tanh) for all layers
In this experiment, we utilize the hyperbolic tangent (tanh) activation function for all layers. Tanh introduces non-linearity and boundedness to the activation outputs, potentially aiding in feature representation and model convergence.

Experiment 9: Hard Swish for all layers
Here, we explore the Hard Swish activation function for all layers. Hard Swish offers a compromise between computational efficiency and performance, potentially improving model efficiency while maintaining expressive power.

Experiment 10: Mish for all layers
Finally, we investigate the Mish activation function for all layers. Mish introduces a smooth non-linearity with beneficial properties for model training and convergence, potentially enhancing the model's ability to learn complex patterns in the data.

2.13. Exploration of different Learning Rates for Adam Optimizer

Table 9. Exploring Different Learning Rates Using Adam Optimizer

	Adam Optimizer Learning Rate
Baseline Model	Adam Optimizer (1e-3)
Experiment 1	Adam Optimizer (1e-4)
Experiment 2	Adam Optimizer (1e-2)
Experiment 3	Adam Optimizer (1e-1)
Experiment 4	Adam Optimizer (0.005)

Baseline: Adam Optimizer, but the learning rate is 1e-3

In the baseline configuration, the Adam optimizer is utilized with a learning rate set to 1e-3. The Adam optimizer is a popular choice for optimizing neural networks due to its adaptive learning rate and momentum. A learning rate of 1e-3 strikes a balance between convergence speed and stability, allowing for effective training of the model while mitigating the risk of overshooting or diverging from optimal parameter values.

Experiment 1: Learning Rate of 1e-4 (using Adam optimizer)

In this experiment, we set the learning rate to 1e-4 while utilizing the Adam optimizer. A lower learning rate may lead to slower but more stable convergence during training, allowing the model to finely adjust its parameters over time.

Experiment 2: Learning Rate of 1e-2 (using Adam optimizer)

Here, we explore a higher learning rate of 1e-2 with the Adam optimizer. A higher learning rate can accelerate convergence during training but may also introduce instability and risk of overshooting optimal parameter values.

Experiment 3: Learning Rate of 1e-1 (using Adam optimizer)

In this experiment, we set the learning rate to 1e-1 while utilizing the Adam optimizer. A significantly higher learning rate may lead to rapid convergence but also increases the risk of oscillation or divergence during training.

Experiment 4: Learning Rate of 0.005 (using Adam optimizer)

Finally, we investigate a moderate learning rate of 0.005 with the Adam optimizer. This learning rate strikes a balance between convergence speed and stability, potentially leading to efficient and effective training of the model.

2.14 Further Exploration: Investigating Combinations of CNN Components

We delve into further exploration by investigating various combinations of components within the Convolutional Neural Network (CNN) architecture. Building upon the insights gained from individual experiments involving different CNN components such as pooling, filters, activation functions, and normalization techniques, we aim to explore how these components interact and influence overall model performance when combined in different configurations. By systematically varying and combining these components, we seek to uncover synergistic effects and optimal configurations that enhance model accuracy, reduce overfitting, and improve training efficiency. Through this comprehensive exploration, we aim to gain deeper insights into the intricate dynamics of CNN architectures and inform the design of more robust and effective models for image classification tasks.

2.14.1 The First Proposed Model

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 113, 113, 4)	1,028
activation (Activation)	(None, 113, 113, 4)	0
conv2d_1 (Conv2D)	(None, 106, 106, 8)	2,056
activation_1 (Activation)	(None, 106, 106, 8)	0
conv2d_2 (Conv2D)	(None, 103, 103, 16)	2,064
activation_2 (Activation)	(None, 103, 103, 16)	0
global_average_pooling2d (GlobalAveragePooling2D)	(None, 16)	0
dense (Dense)	(None, 64)	1,088
dense_1 (Dense)	(None, 3)	195
flatten (Flatten)	(None, 3)	0
flatten_1 (Flatten)	(None, 3)	0
flatten_2 (Flatten)	(None, 3)	0
flatten_3 (Flatten)	(None, 3)	0
flatten_4 (Flatten)	(None, 3)	0
flatten_5 (Flatten)	(None, 3)	0
flatten_6 (Flatten)	(None, 3)	0

The first proposed CNN model for hair type classification employs a 128x128 image resolution to capture detailed information crucial for distinguishing between subtle characteristics in hair types. Laplacian sharpening is utilized for preprocessing, enhancing both edges and texture details essential for classification. Global average pooling is chosen over max pooling to preserve spatial information effectively. The model incorporates varying numbers of filters and kernel sizes across layers to capture features at different scales, with "valid" padding applied to maintain spatial dimensions and reduce padding artifacts. A stride of 1 for all convolutional layers prioritizes the preservation of spatial information, while a dilation rate of 1 ensures fine-grained spatial relationships are accurately captured. The optimizer chosen is Adam with a learning rate of 1e-3, leveraging its adaptive learning rate and momentum features for efficient optimization. Swish activation function and batch normalization combined with dropout at a rate of 0.25 are used for regularization. These design choices collectively aim to optimize performance, ensure efficient training, and enhance generalization capability for accurate hair type classification.

2.14.2 The Second Proposed Model

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 126, 126, 4)	112
activation (Activation)	(None, 126, 126, 4)	0
conv2d_1 (Conv2D)	(None, 124, 124, 8)	296
activation_1 (Activation)	(None, 124, 124, 8)	0
conv2d_2 (Conv2D)	(None, 122, 122, 16)	1,168
activation_2 (Activation)	(None, 122, 122, 16)	0
conv2d_3 (Conv2D)	(None, 120, 120, 32)	4,640
activation_3 (Activation)	(None, 120, 120, 32)	0
conv2d_4 (Conv2D)	(None, 118, 118, 64)	18,496
activation_4 (Activation)	(None, 118, 118, 64)	0
global_average_pooling2d (GlobalAveragePooling2D)	(None, 64)	0
dense (Dense)	(None, 128)	8,320
dense_1 (Dense)	(None, 3)	387
flatten (Flatten)	(None, 3)	0
flatten_1 (Flatten)	(None, 3)	0
flatten_2 (Flatten)	(None, 3)	0
flatten_3 (Flatten)	(None, 3)	0
flatten_4 (Flatten)	(None, 3)	0
flatten_5 (Flatten)	(None, 3)	0
flatten_6 (Flatten)	(None, 3)	0

The second proposed CNN model for hair type classification follows a refined architectural design to enhance feature learning and discrimination capability. The model starts with a smaller number of filters (4) and gradually increases them across layers (4, 8, 16, 32, 64), enabling the network to capture increasingly complex features as it progresses. This hierarchical feature learning aligns with the principles of CNNs, where lower layers detect simple features and higher layers combine them to form more abstract representations. Adopting a consistent kernel size of 3x3 for all convolutional layers follows the successful design of architectures like VGGNet, balancing effective capture of local spatial patterns with computational efficiency. Increasing the dense layer units to 128 allows for a more expressive transformation of learned features, enhancing the model's capacity to extract higher-level abstractions and relationships between features. These modifications collectively aim to improve the model's discriminative ability and accuracy in hair type classification tasks.

III. RESULTS AND ANALYSIS

3.1 Effects of Resizing the Image in Relation to its Performance and Training Time

Table 10. Results of the Experiments for Resizing the Image

	Image Size	Acc	Loss	Val_Acc	Val_Loss	Train Time (s)
Baseline Model	64x64	0.5320	0.9379	0.4467	1.0483	74.77
Experiment 1	128x128	0.6024	0.8456	0.5025	1.0514	285.93
Experiment 2	32x32	0.6353	0.7518	0.4873	1.1076	62.95

Experiment 3	256x256	0.4598	1.0385	0.3706	1.08990	982.78

Comparing the results of Experiment 1 (128x128 pixels) with the baseline, we observed an increase in accuracy and a decrease in loss, indicating improved performance. Experiment 2 (32x32 pixels) demonstrated further enhancements in accuracy and reduction in loss compared to both the baseline and Experiment 1, suggesting that lower resolution images can still yield effective classification results. However, Experiment 3 (256x256 pixels) showed unexpected outcomes, with a decrease in accuracy and an increase in loss compared to other experiments.

Insights gained:

- Accuracy and Loss: Experiment 2 (32x32) achieved the highest accuracy of 0.6353, while Experiment 3 (256x256) had the lowest accuracy of 0.4598. Interestingly, the baseline model (64x64) and Experiment 1 (128x128) fell in between with accuracies of 0.5320 and 0.6024, respectively. Similarly, Experiment 2 (32x32) had the lowest loss of 0.7518, indicating better convergence compared to the other experiments.
- Validation Performance: Experiment 2 (32x32) also exhibited the highest validation accuracy of 0.4873, indicating superior generalization performance. In contrast, Experiment 3 (256x256) had the lowest validation accuracy of 0.3706, suggesting poor generalization capabilities.
- Training Time: Notably, Experiment 3 (256x256) required significantly more training time (982.78 seconds) compared to the other experiments. In contrast, Experiment 2 (32x32) achieved comparable or even better performance with a much shorter training time of 62.95 seconds. **This highlights the trade-off between model complexity (due to image resolution) and computational efficiency, with lower image resolutions generally resulting in faster training times**.

3.2 Effects of Batch Size in Relation to its Performance

Table 11. results of the Experiment for Configuring the Batch Size

	Batch Size	Acc	Loss	Val_Acc	Val_Loss
Baseline Model	32	0.5320	0.9379	0.4467	1.0483
Experiment 1	100	0.3488	1.0863	0.2944	1.1087

The provided data on Table 11 highlights a notable decrease in validation accuracy when increasing the batch size from 32 to 100. Specifically, the validation accuracy decreased from 0.4467 to 0.2944. While this decrease may not appear definitive at first glance, it raises important implications regarding the impact of batch size on model training.

Insights gained:

- With larger batch sizes, such as 100, gradient updates occur less frequently compared to smaller batch sizes like 32. This decrease in frequency can potentially hinder the model's ability to converge to the optimal solution efficiently. The infrequent updates may lead to slower convergence or even stagnation in the optimization process.
- Larger batches introduce more variance into the gradient updates due to the increased diversity of samples included in each batch. This variance poses challenges for the optimizer in determining the optimal direction for weight updates during training. The increased variance makes it more difficult for the optimizer to navigate the loss landscape effectively and find the optimal solution.

3.3 Effects of Different Preprocessing Techniques in Relation to Loss and Validation Performance

Table 12. Results of the Experiment on Different Normalization Techniques

	Img Norm	Acc	Loss	Val_Acc	Val_Loss
Baseline Model	Rescaling pixel val [0,1]	0.5320	0.9379	0.4467	1.0483
Experiment	No	0.3479	1.0905	0.2893	1.1020

1	Normali-zation				
Experiment 2	Cropping	0.5063	0.9665	0.4670	1.0242
Experiment 3	Rotation	0.3493	1.0980	0.2843	1.1028
Experiment 4	Grayscale	0.6044	0.8623	0.4619	1.0669
Experiment 5	Graysc + Gaussian Blur	0.5522	0.9374	0.3909	1.0978
Experiment 6	Gaussian Blur only	0.6159	0.8623	0.4619	1.0882
Experiment 7	Brightness and Contrast	0.5735	0.8996	0.4822	0.9698
Experiment 8	Median blur, kernel size = 2	0.5047	0.9643	0.4365	1.0635
Experiment 9	Laplacian Sharp; 25 sc factor	0.7017	0.6800	0.6548	0.7803

Insights gained:

- **Preprocessing Techniques**: The choice of preprocessing technique significantly impacts model performance. Experiments 4 (Grayscale) and 9 (Laplacian Sharpening) achieved the highest accuracies of 0.6044 and 0.7017, respectively, indicating the effectiveness of these techniques in enhancing model performance. In contrast, Experiment 1 (No Normalization) had the lowest accuracy of 0.3479, suggesting that omitting normalization adversely affects model learning and generalization capabilities.
- **Effect on Loss**: Experiments 4 (Grayscale) and 9 (Laplacian Sharpening) also exhibited relatively low losses compared to other experiments, indicating better convergence during training. Conversely, Experiment 1 (No Normalization) and Experiment 3 (Rotation) had the highest losses, suggesting instability or inadequate learning due to the absence of proper normalization.
- **Validation Performance**: While Experiment 9 (Laplacian Sharpening) achieved the highest accuracy, it also demonstrated the highest validation accuracy of 0.6548, indicating strong generalization capabilities. Experiment 7 (Brightness and Contrast) exhibited the lowest validation accuracy of 0.4822, suggesting that this preprocessing technique may not effectively enhance model generalization.

Results highlight the importance of selecting appropriate preprocessing techniques to enhance model performance and stability during training. Grayscale conversion and Laplacian sharpening emerge as promising techniques for improving classification accuracy, while proper normalization is crucial for ensuring model stability and generalization.

3.4 Effects of Pooling Configuration in Relation to its Performance

Table 13. Results of Configuring the Pooling of the CNN Model in Relation to its Overall Performance

Experiment Num	Type of Pooling	Accuracy	Loss	Validation Accuracy	Validation Loss
Baseline Model	Global Average Pooling 2D	0.5320	0.9379	0.4467	1.0483
Experiment 1	Max Pooling after each convolutional layer (2×2) with dropout layer for regularization (w/Laplacian Normalization)	0.9318	0.1906	0.6041	1.2577
Experiment 2	Max Pooling after each convolutional layer (2×2) with dropout layer for regularization	0.9611	0.1082	0.5635	1.9578
Experiment 3	Max pooling without Laplacian and applying regularization/dropout layer for every conv layer followed by max pooling	0.8969	0.2612	0.5533	1.8424
Experiment 4	Global Max Pooling (w/Laplacian Normalization)	0.8890	0.2990	0.3959	2.7196
Experiment 5	Omitted ANY type of Pooling and replaced with Flatten (w/ Laplacian Normalization)	0.9968	0.0036	0.5431	1.3330
Experiment 6	Added average pooling at the end	0.9772	0.0973	0.5787	1.6475
Experiment 7	Max Pooling at each while adding GlobalAveragePooling at the last layer	0.8432	0.4206	0.5888	0.9587
Experiment 8	Max Pooling with pool size of 3×3	0.9995	0.0026	0.5787	1.5549
Experiment 9	Max Pooling with pool size of 2×2	0.9968	0.0036	0.5431	1.3330
Experiment 10	Max Pooling with pool size of 2×2 and 2×2 stride	0.9985	0.0131	0.5736	1.2826

Insights Gained:

- While max pooling effectively enhances accuracy, it **often leads to overfitting**, especially when combined with dropout regularization. This indicates the importance of

carefully tuning hyperparameters and monitoring model performance to mitigate overfitting risks.

- **Max pooling retains only the most significant activation within each region**, which may lead to a **loss of detailed spatial information**. In contrast, global average pooling computes the average activation across the entire feature map, which can help preserve more nuanced spatial information.
- Max pooling focuses on selecting the most prominent features within each region, which can be beneficial for tasks where precise localization of features is crucial. However, it may discard less prominent but still relevant features. **Global average pooling, by considering the average activation, provides a more comprehensive representation of the entire feature map, potentially capturing a broader range of features**.

3.5 Effects on Configuring the Number of Filters in Relation to Performance and Training Time

Table 13. Results of Configuring the Number of Filters in Relation to Performance and Training Time

Experiment Num	Num of Filters	Accuracy	Loss	Validation Accuracy	Validation Loss	Training Time (s)
Baseline Model	4, 8, 16	0.5320	0.9379	0.4467	1.0483	74.77
Experiment 1	1 for all	0.4768	0.9996	0.4467	1.0088	114.67
Experiment 2	5 for all	0.6216	0.7844	0.5482	0.9313	-
Experiment 3	10 for all	0.8277	0.4446	0.6701	0.8616	264.39
Experiment 4	2, 4, 6	0.6517	0.7452	0.4822	1.0369	132.86
Experiment 5	4, 4, 4	0.6238	0.8120	0.6091	0.9021	-
Experiment 6	16, 32, 64	0.8780	0.2932	0.6142	1.2794	-
Experiment 7	16, 32, 64	0.8036	0.5839	0.6345	0.9950	-
Experiment 8	8, 16, 32	0.6401	0.7226	0.6142	0.8906	-
Experiment 9	16, 32	0.7389	0.5850	0.6142	0.8791	-
Experiment 10	16, 8, 4	0.7801	0.5305	0.6650	0.8199	-
Experiment 11	32, 16, 8	0.9372	0.1765	0.5431	1.4376	-

The experiments delving into the impact of diverse filter configurations within the CNN architecture unveiled a spectrum of performance outcomes. In the baseline model, where filter configurations of 4, 8, and 16 were employed, an accuracy of 53.20% was attained, albeit accompanied by conspicuous signs of overfitting, as evidenced by the discernible gap between training and validation metrics. Experiment 1, utilizing a singular filter for all layers, resulted in diminished accuracy compared to the baseline, highlighting the necessity of filter diversity in capturing varied features. Conversely, Experiment 2, with an increased filter count of 5 for all layers, yielded improved accuracy, albeit with a slight uptick in loss, suggesting a potential enhancement in feature extraction capabilities with augmented filter diversity. Experiment 3, employing 10 filters across all layers, witnessed a notable surge in accuracy and loss reduction. Nonetheless, this configuration succumbed to overfitting, evident from the discordance between training and validation metrics. Experiment 4, varying the filter counts (2, 4, 6) across layers, exhibited moderate accuracy but echoed the overfitting tendencies akin to the baseline model. Experiment 5 maintained an equilibrium with an equal number of filters (4) across layers, mirroring the baseline's accuracy, accentuating the significance of filter balance in optimizing model performance. Experiment 6's substantial augmentation in filter count (16, 32, 64) corresponded with a significant surge in accuracy. However, this surge was marred by pronounced overfitting, highlighted by a substantial gap between training and validation metrics. Experiment 7, fortified with batch normalization, dropout, and kernel regularization, mitigated overfitting to an extent while maintaining commendable accuracy. Experiment 8, with a reduction in filter count for deeper layers (8, 16, 32), mirrored moderate accuracy but still bore signs of overfitting akin to other configurations. Experiment 9, employing 16 and 32 filters, achieved moderate accuracy but showcased overfitting tendencies. Experiment 10's strategic decrementing of filter count for deeper layers (16, 8, 4) ameliorated overfitting compared to Experiment 6, underscoring the pivotal role of filter distribution. Lastly, Experiment 11, utilizing 32, 16, and 8 filters, yielded high accuracy but suffered from severe overfitting, underscoring the imperative need for regularization techniques to mitigate overfitting risks.

Key Insights:

- Increasing the number of filters generally improves accuracy but may lead to overfitting if not appropriately regulated.
- Varying filter configurations across layers can influence model performance and overfitting tendencies.
- Regularization techniques such as dropout, batch normalization, and kernel regularization play a

crucial role in mitigating overfitting risks associated with increased filter diversity.

- Balancing filter diversity and model complexity is essential to optimize model performance while minimizing overfitting.
- A notable observation was the trade-off between filter count and training time. Experiments with higher filter counts, such as Experiment 6, incurred significantly longer training times, highlighting the computational overhead associated with increased model complexity. **More filters = slower training time**.

3.6. Effects of Configuring the Kernel Size in Relation to the Performance and Training Time

Table 14. Results of Configuring the Kernel Size

Experiment Num	Kernel Size	Accuracy	Loss	Validation Accuracy	Validation Loss	Training Time (s)
Baseline Model	16, 8, 4	0.5320	0.9379	0.4467	1.0483	74.77
Experiment 1	4, 4, 4	0.5879	0.8865	0.5076	1.0143	87.46
Experiment 2	8, 8, 8	0.6187	0.7839	0.4518	1.0667	124.79
Experiment 3	16, 16, 16	0.8838	0.3200	0.6294	1.2354	577.87
Experiment 4	4, 8, 16	0.6666	0.7373	0.5482	0.9742	-
Experiment 5	5, 3, 3	0.5377	0.9280	0.4365	1.0866	-
Experiment 6	18, 9, 5	0.7900	0.4751	0.6701	0.8950	-

The experimentation with varying kernel size configurations in CNN architecture provided valuable insights into the trade-offs between model complexity, computational efficiency, and performance metrics. In the baseline model, employing kernel sizes of 16x16x16, 8x8x8, and 4x4x4 yielded moderate accuracy but exhibited signs of overfitting, underscoring the importance of finding a balance between feature extraction and model generalization. Experiment 1 demonstrated that reducing the kernel size to 4x4x4 slightly improved accuracy and reduced loss, indicating enhanced feature extraction capabilities. Conversely, Experiment 2 showed that increasing the kernel size to 8x8x8 resulted in higher accuracy but also higher loss and longer training time, suggesting a potential trade-off between accuracy and computational efficiency. Experiment 3 highlighted the diminishing returns associated with larger kernel sizes (16x16x16), with significantly higher accuracy but at the cost of substantially longer training time. Experiment 4's progressive increase in kernel size (4, 8, 16) demonstrated moderate accuracy improvements but exhibited signs of overfitting, similar to the baseline model. Experiment 5, using non-uniform kernel sizes (5, 3, 3), led to lower accuracy and higher loss, suggesting limitations in capturing relevant features effectively. Finally, Experiment 6's incorporation of batch normalization and dropout with larger kernel sizes (18, 9, 5) resulted in improved accuracy and validation metrics, highlighting the potential benefits of regularization techniques in enhancing model performance. These findings underscore the importance of carefully selecting kernel size configurations to optimize model performance while managing computational complexity and overfitting risks.

Gathered Insights:

- Larger kernel sizes tend to improve accuracy but may also increase computational complexity and training time.
- Smaller kernel sizes can enhance feature extraction capabilities and computational efficiency but may be limited in capturing complex patterns.
- **The gradual increase of kernel size allows the model to progressively capture features at different scales, starting from smaller, more detailed patterns and gradually transitioning to larger, more global features**. This approach ensures that the network learns a hierarchy of representations, enabling it to extract both fine-grained and high-level features effectively. By incorporating a range of kernel sizes, the model can adapt to varying complexities in the input data while balancing computational efficiency with feature extraction capabilities.
- Regularization techniques such as batch normalization and dropout can mitigate overfitting risks and improve model generalization, particularly when using larger kernel sizes.
- **Smaller kernel size, faster training time**.

3.7. Effects of Configuring the Padding in Relation to the Performance and Training Time

Table 15. Results Comparing the Effects of Different Padding

Experiment Num	Padding Configuration	Accuracy	Loss	Validation Accuracy	Validation Loss	Training Time (s)
Baseline Model	valid, valid, valid	0.5320	0.9379	0.4467	1.0483	74.77
Experiment 1	same, same, same	0.5211	0.9438	0.4721	1.0047	163.14
Experiment 2	valid, valid, same	0.7125	0.6840	0.6599	0.8745	150.72
Experiment 3	valid, same, valid	0.6230	0.8118	0.4873	1.1284	145.52
Experiment 4	valid, same, same	0.6153	0.8037	0.4619	1.1337	148.59
Experiment 5	same, valid, valid	0.6595	0.7716	0.5279	1.0034	196.21
Experiment 6	same, valid, same	0.6232	0.7870	0.5228	1.0002	192.87
Experiment 7	same, same, valid	0.6129	0.8128	0.5076	1.0224	198.40

The experiments investigating the effects of different padding techniques on CNN performance showcased intriguing insights. The baseline model, employing 'valid' padding across all convolutional layers, achieved a moderate accuracy of 53.20%. However, overfitting was evident, as indicated by the notable gap between training and validation metrics. Experiment 1, utilizing 'same' padding for all layers, yielded slightly lower accuracy compared to the baseline, suggesting potential impacts on feature extraction and model performance. Conversely, Experiment 2, employing 'valid' padding for the first two layers and 'same' padding for the last layer, exhibited significant accuracy improvement and validation metrics enhancement, highlighting the importance of adaptive padding strategies for different layers. However, Experiment 3, applying 'valid' padding for the first and last layers and 'same' padding for the middle layer, demonstrated moderate accuracy with signs of overfitting, akin to the baseline model. Experiment 4, using 'valid' padding for the first layer and 'same' padding for the remaining layers, showed slightly lower accuracy and higher loss, suggesting a significant impact of padding arrangement on model performance. Reversing the padding configuration in Experiment 5 led to moderate accuracy improvements but exhibited overfitting tendencies similar to Experiment 3. Experiment 6, employing 'same' padding for the first and last layers and 'valid' padding for the middle layer, yielded comparable accuracy to the baseline but showed signs of overfitting. Finally, Experiment 7, utilizing 'same' padding for all layers, resulted in slightly lower accuracy compared to Experiment 2, indicating that uniform padding might not be optimal for all layers. Overall, the choice of padding technique significantly influences model performance, with adaptive padding strategies showing promise in enhancing accuracy and validation metrics while mitigating overfitting risks.

Insights:

- Experiment 2, which utilized 'valid' padding for the first two layers and 'same' padding for the last layer, demonstrated the highest accuracy among all experiments, indicating the importance of adaptive padding strategies tailored to different layers. *This result aligns with the concept of receptive field size in convolutional neural networks (CNNs). The use of 'valid' padding for the initial layers allows the network to extract features without adding any artificial borders, thereby preserving the spatial dimensions of the input and enabling the network to learn more discriminative features. Meanwhile, employing 'same' padding for the final layer ensures that the spatial dimensions of the feature maps are maintained, allowing for more comprehensive feature extraction and better coverage of the input space*.
- 'valid' padding applies no padding to the input image, ensuring that no additional values are added around the borders. This approach helps minimize distortions or border effects and preserves the spatial integrity of the feature maps.
- using 'valid' padding eliminates the need for computations associated with padding, resulting in faster training times. Additionally, applying 'valid' padding in the initial layers allows the network to focus solely on extracting features from the input images without introducing unnecessary padding. This is particularly beneficial for capturing fine details and local patterns.

3.8. Effects of Configuring the Strides in Relation to the Performance and Training Time

Table 16. Results Comparing the Effects of Configuring Different Strides

Experiment Num	Stride	Accuracy	Loss	Validation Accuracy	Validation Loss	Training Time (s)
Baseline Model	1, 1, 1	0.5320	0.9379	0.4467	1.0483	74.77
Experiment 1	2 for all	0.9958	0.0699	0.4873	4.7643	77.31
Experiment 2	1, 2, 2	0.8259	0.3941	0.5838	1.1925	131.00
Experiment 3	1, 1, 2	0.7357	0.5441	0.5635	1.1224	131.83
Experiment 4	1, 2, 1	0.7482	0.6557	0.6142	0.8800	132.80
Experiment 5	2, 1, 1	0.7191	0.6308	0.5888	0.9473	79.83
Experiment 6	2, 2, 1	0.8663	0.3349	0.5431	1.6936	78.13
Experiment 7	2, 1, 2	0.9461	0.2135	0.5431	2.6137	79.98

The experiments investigating the effects

Insights:

- Larger stride values, as seen in Experiment 1 where stride was set to 2 for all layers, expedite feature extraction by reducing the spatial dimensions of feature maps. This concept aligns with the notion that larger strides lead to more aggressive down-sampling of feature maps, effectively capturing high-level features in fewer operations (He et al., 2015). However, this comes at the expense of spatial resolution and potentially finer-grained feature representation.
- Experiment 7 achieved high accuracy but showed signs of overfitting, indicating the need for regularization techniques. When using larger stride values, there is a risk of overfitting, particularly if the model becomes too aggressive in discarding spatial information. Regularization techniques like dropout and kernel regularization can help mitigate overfitting by introducing noise during training and imposing constraints on model parameters, respectively.
- **A stride of 1 ensures that the spatial dimensions of the feature maps remain unchanged after convolution, preserving spatial information**. This preservation of spatial details allows the network to capture fine-grained features present in the input images. Without downsampling due to larger strides, the model can maintain a higher resolution representation of the input, potentially leading to more precise feature extraction.

3.9. Effects of Configuring the Dilation in Relation to the Performance and Training Time

Table 17. Results Comparing the Effects of Configuring Different Dilation

Experiment num	Stride	Accuracy	Loss	Validation Accuracy	Validation Loss	Training Time (s)
Baseline Model	1, 1, 1	0.5320	0.9379	0.4467	1.0483	74.77
Experiment 1	2	0.6169	0.8388	0.5482	0.9311	137.01
Experiment 2	3	0.5558	0.9359	0.5025	0.9634	63.47
Experiment 3	3, 2, 1	0.5911	0.8522	0.5482	0.9669	
Experiment 4	2, 1, 1	0.6685	0.7844	0.5127	1.0303	148.82
Experiment 5	2, 2, 1	0.5609	0.9485	0.5888	0.9403	106.66
Experiment 6	1, 1, 2	0.7207	0.6712	0.6599	0.8325	221.23
Experiment 7	1, 2, 3	0.6923	0.6776	0.6193	0.8982	179.65

The experimentation with different dilation rates in CNN architecture unveiled crucial insights into their impact on model performance and training efficiency. While the baseline model with a stride of 1 for all convolutional layers served as a reference, experiments altering the dilation rates demonstrated varying effects. Increasing the stride to 2 improved accuracy but at the cost of increased training time, highlighting a trade-off between accuracy and computational efficiency. Further increasing the stride to 3 led to decreased accuracy, indicating potential limitations of larger strides on feature extraction. Varying dilation rates introduced complexity, with certain configurations yielding competitive accuracy but requiring longer training times. Asymmetric dilation patterns showed promise, with configurations like (1, 1, 2) and (1, 2, 3) achieving higher accuracy and validation metrics, suggesting their potential in capturing spatial dependencies effectively. However, it's essential to balance these benefits with the associated increase in training time.

Insights:

- Experimentations or explorations with different Dilation revealed that they determine the spatial resolution of feature maps, with larger strides reducing the spatial dimensions. This reduction can lead to loss of fine-grained spatial information but may also increase computational efficiency by reducing the number of computations required per layer.
- Configurations such as **(1, 1, 2)** and (1, 2, 3) demonstrated competitive accuracy, suggesting their effectiveness in capturing spatial dependencies across different scales. In CNNs, capturing spatial dependencies is crucial for recognizing patterns and objects within images.

Asymmetric dilation patterns allow the network to focus on specific spatial scales, enhancing its ability to detect objects of varying sizes and complexities. **This potential almost made us consider using the 1, 1, 2 dilation because of its performance**.

3.10. Effects of Changing the Different Optimizers in Relation to the Performance and Training Time

Table 18. Results Comparing the Effects of Configuring Different Optimizers

Experiment num	Optimizer	Accuracy	Loss	Validation Accuracy	Validation Loss	Training Time (s)
Baseline Model	Adam (1e-3)	0.5320	0.9379	0.4467	1.0483	74.77
Experiment 1	SGD (lr=1e-3, momentum=0.9)	0.4687	1.0272	0.4518	1.0303	86.43
Experiment 2	RMSprop (lr=1e-3)	0.6171	0.8294	0.6396	0.8390	149.11
Experiment 3	Adagrad (lr=1e-3)	0.4869	1.0231	0.4112	1.0532	145.21
Experiment 4	Adadelta (lr=1e-3)	0.3533	1.0848	0.2843	1.1021	-
Experiment 5	Adadelta (lr=1.0)	0.5351	0.9399	0.3959	1.0518	-
Experiment 6	Adamax (lr=1e-3)	0.5347	0.9304	0.5178	1.0418	150.33

The experiments with different optimizers showcased varying effects on model performance. Stochastic Gradient Descent (SGD) in Experiment 1, with a learning rate of 1e-3 and momentum of 0.9, exhibited lower accuracy and higher loss compared to the baseline model, struggling to converge effectively within a relatively short training time. In contrast, Experiment 2 using RMSprop with the same learning rate achieved higher accuracy and lower loss, demonstrating faster convergence and superior generalization capabilities, as evidenced by the higher validation accuracy. However, Experiment 3 employing Adagrad at a learning rate of 1e-3 yielded moderate accuracy but struggled with generalization, evident from the lower validation accuracy. Meanwhile, Experiments 4 and 5 with Adadelta at different learning rates displayed varying degrees of performance; Experiment 4 resulted in low accuracy and poor convergence, whereas Experiment 5 showed improved accuracy but suffered from slower convergence and higher validation loss. Lastly, Experiment 6 utilizing Adamax with a learning rate of 1e-3 achieved accuracy comparable to the baseline model but demonstrated faster convergence and better generalization, indicated by the higher validation accuracy.

Insights:

- The choice of optimizer plays a crucial role in determining how effectively the model converges to an optimal solution during training. Adam optimizer, as the baseline, incorporates adaptive learning rates and momentum, enabling it to adjust parameter updates based on gradient magnitudes and maintain directionality in updates, respectively. This adaptability allows Adam to navigate the optimization landscape efficiently, facilitating faster convergence. More advanced optimizers like RMSprop and Adamax, which also incorporate adaptive learning rate mechanisms, may further enhance convergence and performance due to their potentially more effective management of momentum.
- **Adam optimizer incorporates adaptive learning rates for each parameter in the network**. This adaptability allows it to adjust the step size of parameter updates based on the magnitude of gradients, ensuring that parameters with large gradients undergo smaller updates and vice versa. In CNNs, where the optimization landscape can be highly non-linear and vary across different layers, adaptive learning rates help navigate the landscape more efficiently, facilitating faster convergence.

3.11. Effects of Changing the Activation Function in Relation to the Performance and Training Time

Table 19. Results Comparing the Effects of Configuring Different Activation Functions

Experiment	Activation Function	Accuracy	Loss	Validation Accuracy	Validation Loss	Training Time (s)
Baseline	ReLU	0.5320	0.9379	0.4467	1.0483	74.77
1	Leaky ReLU	0.6846	0.7237	0.5736	0.9734	159.39
2	Leaky ReLU	0.6935	0.7090	0.5990	0.9821	-
3	ELU	0.7741	0.5332	0.6802	0.8621	144.54
4	SELU	0.7545	0.5938	0.6548	0.8281	150.11
5	GELU	0.7505	0.5902	0.6548	0.8033	202.43
6	Swish	0.7586	0.5408	0.6853	0.7411	168.58
7	Sigmoid	0.5353	0.9501	0.4416	1.0168	150.50
8	Hyperbolic tangent	0.5068	0.9754	0.4112	1.0426	87.78
9	Hard Swish	0.7515	0.5316	0.6954	0.7530	187.70
10	Mish	0.7519	0.5278	0.6954	0.8369	-

The experiments comparing various activation functions revealed intriguing insights into their impact on model performance. Leaky ReLU and ELU activations exhibited notable improvements in accuracy and validation metrics compared to the baseline ReLU activation. Particularly, ELU demonstrated the highest accuracy and validation accuracy, highlighting its effectiveness in capturing complex patterns in the data. SELU and GELU activations also showed competitive performance, with SELU achieving high accuracy despite slightly lower validation accuracy compared to ELU. Swish activation showcased promising results, indicating its effectiveness in capturing intricate data patterns, as evidenced by its high accuracy and validation accuracy. Conversely, activations like Sigmoid and Hyperbolic tangent exhibited lower accuracy and validation metrics, suggesting their limitations in learning complex representations. However, Hard Swish and Mish activations performed comparably well to Swish, demonstrating their potential as effective alternatives to traditional activation functions like ReLU.

Insights

- Swish activation exhibited high accuracy and validation accuracy, indicating its effectiveness in capturing complex patterns in the data. Swish introduces non-linearity through a smooth function, enabling better gradient flow during training, which helps mitigate the vanishing gradient problem often encountered with traditional activation functions like ReLU.
- Hard Swish, a modified version of Swish that introduces a non-linear component, demonstrated comparable performance to Swish. Hard Swish simplifies computation by replacing the expensive exponential operation in Swish with a cheaper linear function, making it more efficient for deployment in resource-constrained environments such as mobile devices. This efficiency makes Hard Swish a compelling alternative activation function for deep neural networks, offering a balance between performance and computational cost.
- **On the matter of 'Exploding Gradients', Swish has been observed to lead to smoother loss landscapes, reducing the likelihood of exploding gradients during training, unlike ReLU where the gradients become too large.**

3.12. Effects of Using the Adam Optimizer and Changing its Learning Rate in Relation to the Performance and Training Time

Table 20. Results Comparing the Effects of Configuring Different Learning Rates for Adam Optimizer

Experiment	Learning Rate	Accuracy	Loss	Validation Accuracy	Validation Loss	Training Time (s)
Baseline	1e-3	0.5320	0.9379	0.4467	1.0483	74.77
Experiment 1	1e-4	0.4960	0.9817	0.4518	1.0253	189.78
Experiment 2	1e-2	0.8965	0.2596	0.5431	1.6980	185.16
Experiment 3	1e-1	0.5234	0.9933	0.4518	1.0300	-
Experiment 4	0.005	0.8365	0.4004	0.6345	1.0912	-

The experiments comparing different learning rates for the Adam optimizer yielded valuable insights into the optimization dynamics of the model. Experiment 1, with a low learning rate of 1e-4, exhibited slow convergence and subpar accuracy, indicating that the rate was insufficient for effective learning. Conversely, Experiment 2, utilizing a higher learning rate of 1e-2, demonstrated significantly improved accuracy and faster convergence, suggesting that this rate was better suited for the dataset and model architecture. Experiment 3, with a learning rate of 1e-1, performed poorly, likely due to the rate being too high, causing instability in the optimization process. Experiment 4, employing a moderate learning rate of 0.005, achieved competitive performance, striking a balance between convergence speed and stability. Overall, these findings highlight the importance of selecting an appropriate learning rate to facilitate effective optimization in CNNs. Furthermore, the baseline Adam optimizer with a learning rate of 1e-3

emerged as the most suitable option, striking a balance between convergence speed and stability, as evidenced by its competitive performance across experiments.

Insights:

- The balance between convergence speed and stability is a fundamental consideration in deep learning. While a higher learning rate may accelerate convergence, it can also introduce erratic behavior, hindering the model's ability to converge to an optimal solution. Conversely, lower learning rates may ensure stability but at the cost of slower convergence. A moderate learning rate, such as 1e-3, strikes a balance between these competing objectives, allowing for relatively fast convergence while maintaining stability throughout the optimization process.
- The experimental results presented support the superiority of the Adam optimizer with a learning rate of 1e-3. Across various experiments varying the learning rate, Adam consistently demonstrated competitive performance in terms of accuracy, loss, and convergence speed. This empirical evidence aligns with theoretical expectations and reinforces the importance of careful optimization hyperparameter selection in deep learning tasks.
- While a higher learning rate may accelerate convergence, it can also introduce erratic behavior, hindering the model's ability to converge to an optimal solution. Conversely, lower learning rates may ensure stability but at the cost of slower convergence. **A moderate learning rate, such as 1e-3, strikes a balance between these competing objectives, allowing for relatively fast convergence while maintaining stability throughout the optimization process**.

3.13. Our Proposed Models

3.13.1 First proposed model

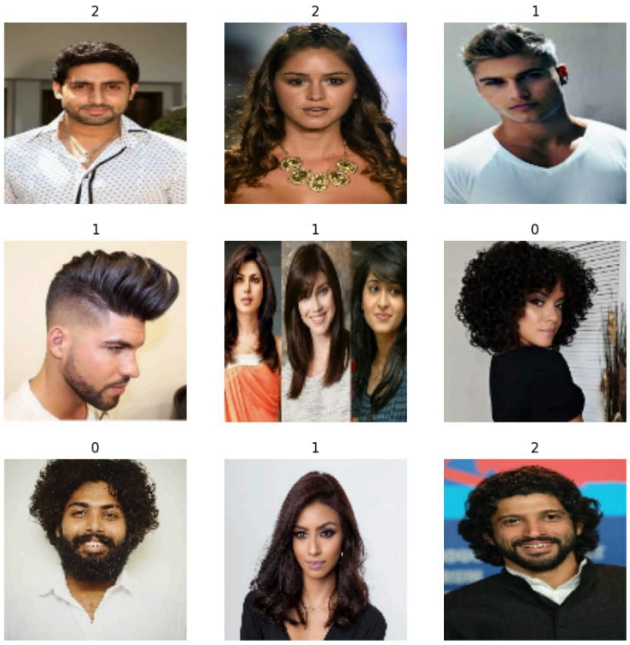


Image Resizing: **128x128**. We chose a higher resolution of 128x128 pixels to capture more detailed information from the images, which is crucial for distinguishing subtle characteristics in hair types. Despite the increased detail, training with 128x128 images resulted in faster training times compared to the larger 256x256 resolution, making it a practical choice.

Pre Processing Technique: Laplacian Sharpening. In our experimentation, we observed that Laplacian sharpening significantly improved model performance by enhancing edges and increasing contrast in the images. While Sobel edge detection was also employed, Laplacian sharpening proved to be more effective for hair classification tasks. This superiority stems from Laplacian's ability to not only highlight edges but also capture texture details, which are crucial for distinguishing between different hair types. Unlike Sobel, which primarily emphasizes edges, Laplacian's capability to enhance both edges and textures makes it better suited for our hair classification task.

Pooling: Global Average Pooling. We retained global average pooling over max pooling due to its ability to preserve more spatial information. Max pooling, while effective in retaining salient features, may increase the risk of overfitting by retaining only the most prominent features. Global average pooling provides a summary of the entire feature map, including both important and less relevant information, making it more suitable for our classification task.

Number of Filters and Kernel Size: 16, 8, 4 and 4, 8, 16 respectively. We retained varying numbers of filters

and kernel sizes across layers to capture features at different scales. Smaller kernel sizes are effective at capturing fine details and local patterns, while larger kernel sizes are better at capturing broader, global features. By varying the kernel size across layers, the network can learn hierarchical representations that start with local details and gradually transition to more global features. An experiment where a large kernel size was used for initial layer led to overfitting but using regularization and dropout layer, it helped but still have a sign of overfitting.

Padding: 'Valid' for all layers. In our CNN model, we opted for "valid" padding for all layers due to its advantages in preserving spatial dimensions and reducing the likelihood of padding artifacts. "Valid" padding applies no padding to the input image, ensuring that no additional values are added around the borders, thereby minimizing distortions or border effects. This approach maintains the spatial integrity of the feature maps, as no padding is introduced. Additionally, using "valid" padding eliminates the need for computations associated with padding, resulting in faster training times.

The 'valid, valid, same' also showed a comparable performance. By applying "valid" padding in the initial layers, we allow the network to focus solely on extracting features from the input images without introducing unnecessary padding. This is particularly beneficial for capturing fine details and local patterns. Using "same" padding in the final convolutional layer ensures that the spatial dimensions of the feature maps are preserved before passing them to subsequent layers or the output layer, maintaining consistency throughout the network architecture.

Stride: '1,1,1' for all layers.

In our proposed CNN model, we chose a stride of 1 for all convolutional layers to preserve spatial information and enable a detailed analysis of the input data. By maintaining the same spatial dimensions as the input, this configuration ensures that the network can capture fine-grained details and local features effectively. However, we also considered a stride configuration of 1, 1, 2 due to its comparable performance. This combination allows the network to balance between capturing both local and global context within the input data. The initial layers with strides of 1 focus on extracting intricate details and local patterns, while the final layer with a stride of 2 captures broader, global features by downsampling the feature maps. Ultimately, we retained the stride of 1 for all layers to prioritize the preservation of spatial information and detailed analysis, which aligns well with the objectives of our hair type classification task.

Dilation: **'1, 1, 1' dilation rate**. In our proposed CNN model for hair type classification, we opted for a dilation rate of 1 for all convolutional layers to maintain contiguous receptive fields and effectively capture fine-grained spatial relationships within the input data. This configuration ensures that adjacent elements in the receptive field are closely connected, allowing the network to extract intricate details and local features accurately. However, we also considered a dilation rate configuration of 1, 2, and 3 due to its comparable performance. This alternative introduces increasing gaps between receptive field elements, enabling the network to capture spatial context across larger regions. While this configuration may provide a broader perspective of the input data, it also comes with some spacing between the elements, potentially sacrificing the level of detail captured by the model. Ultimately, we prioritized the preservation of fine-grained spatial information by retaining a dilation rate of 1 for all layers, aligning with the specific requirements of our hair type classification task.

Optimizer and Learning Rate: Adam Optimizer (1-e3). The selection of the Adam optimizer with a learning rate of 1e-3 for our proposed CNN model in hair type classification is grounded in its effectiveness in navigating the complex optimization landscape inherent in deep neural networks. Adam optimizer combines the benefits of adaptive learning rates and momentum, which are vital for efficient optimization in CNNs. The adaptive learning rate mechanism dynamically adjusts the step size of parameter updates based on the magnitude of gradients, allowing for faster convergence and improved generalization. Additionally, the inclusion of momentum helps dampen oscillations and smooth out the optimization trajectory, facilitating robust convergence towards the global minimum. In the context of CNNs, where the optimization process involves high-dimensional and non-convex spaces, the adaptive learning rate and momentum features of Adam optimizer play a crucial role in effectively navigating and optimizing the network parameters. By striking a balance between exploration and exploitation, Adam optimizer with a learning rate of 1e-3 enables the model to learn meaningful representations from the hair image

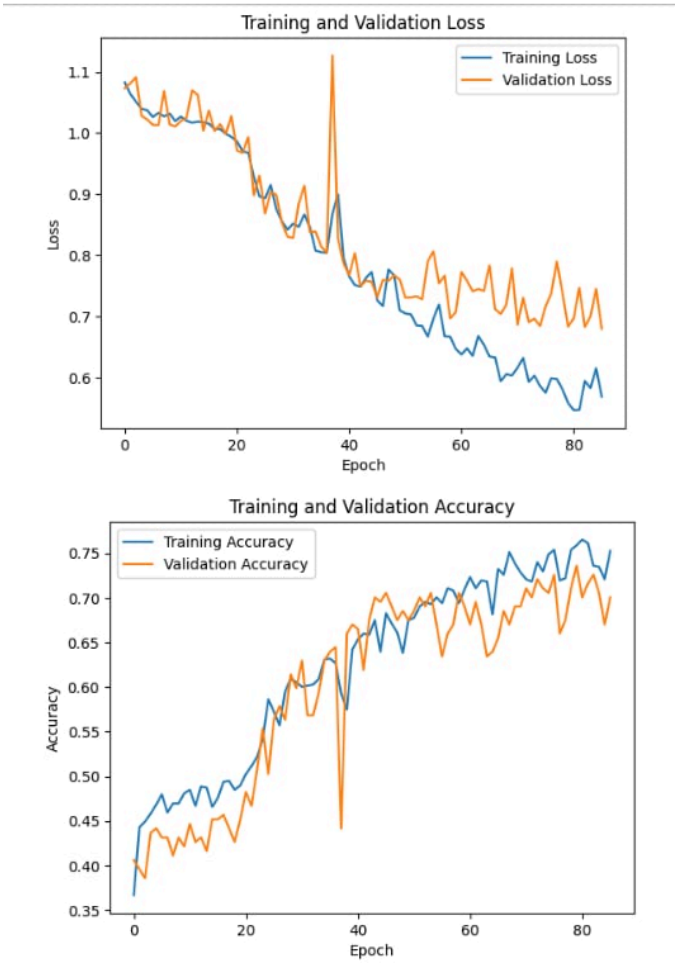
data efficiently, ultimately leading to superior performance in hair type classification.

Activation Function: **Swish**. Swish is a smooth and continuous activation function, which can help alleviate the issue of "dying ReLU" where neurons can become inactive during training. The smoothness of Swish allows gradients to flow more freely during backpropagation, facilitating training and potentially leading to better convergence.

Regularization: **Batch normalization combined with dropout at a rate of 0.25**. Batch normalization is a regularization technique commonly used in CNNs to improve training stability and accelerate convergence. By normalizing the activations of each layer, batch normalization reduces internal covariate shift, making the optimization process more efficient. Additionally, batch normalization introduces a degree of noise during training, which acts as a form of regularization, helping prevent overfitting. Dropout, on the other hand, randomly deactivates a fraction of neurons during training, forcing the network to learn more robust and generalizable features. By combining batch normalization with dropout at a rate of 0.25, the model gains the benefits of both techniques: improved training stability, faster convergence, and enhanced generalization capability. This regularization strategy ensures that the model learns meaningful and discriminative features from the data while mitigating the risk of overfitting, ultimately leading to better performance and higher accuracy.

Final Performance of Model 1:

Metric	Value
Accuracy	0.7956
Loss	0.4938
Validation Accuracy	0.7005
Validation Loss	0.6857
Training Time	310.18s
Precision	0.7143
Recall	0.6091
Categorical Accuracy	0.7005
Categorical Crossentropy	0.6857
AUC	0.8704
F1-Score (Class 1)	0.7544
F1-Score (Class 2)	0.6963
F1-Score (Class 3)	0.6621



The precision, recall and F1-Score metrics demonstrate a well-rounded performance across various classes, indicating balanced classification accuracy. Moreover, the AUC metric indicates strong discriminative ability, highlighting the model's effectiveness in distinguishing between different classes.

3.13.2 Second proposed model

The second model is similar to the first model, however, there are certain modifications that were made:

Filters: Starting with a smaller number of filters (4) and gradually increasing them (**4, 8, 16, 32, 64**) allows the network to capture increasingly complex features as it progresses through the layers. This approach aligns with the concept of hierarchical feature learning in CNNs, where lower layers detect simple features and higher layers combine them to form more abstract representations, enhancing the network's ability to discriminate between classes.

Kernel size: Adopting a consistent kernel size of **3x3** for all convolutional layers follows the design principle of architectures like VGGNet, which has shown strong performance in various tasks. A 3x3 kernel size strikes a

balance between capturing local spatial patterns effectively and being computationally efficient. By using smaller kernel sizes, the network can learn more detailed and diverse features while keeping the number of parameters manageable, facilitating deeper architectures without significantly increasing computational complexity.

Increased dense layer units (128): Increasing the number of units in the dense layer allows for a more expressive transformation of the features learned in earlier layers. This modification enables the model to capture higher-level abstractions and relationships between features, potentially improving its capacity to discriminate between different classes. A denser representation in the final stage of feature extraction enhances the model's ability to extract meaningful insights from the learned features and make more accurate predictions.

Final Performance of the 2nd proposed model:

Metric	Value
Accuracy	0.7488
Loss	0.5638
Validation Accuracy	0.7005
Validation Loss	0.6794
Training Time	1000.86 s
Precision	0.7359
Recall	0.6650
Categorical Accuracy	0.7005
Categorical Crossentropy	0.6794
AUC	0.8749
F1 Score (Class 0)	0.7611
F1 Score (Class 1)	0.6897
F1 Score (Class 2)	0.6618

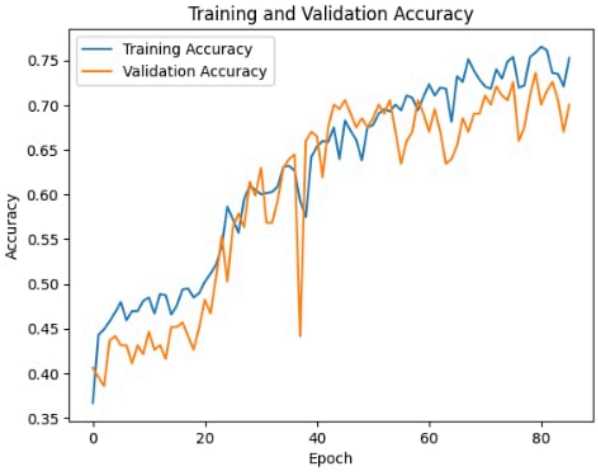


Figure 7. Tested image on its prediction classifying whether the subject has curly, straight, or wavy hair - For Proposed Model 2

IV. CONCLUSION AND RECOMMENDATIONS

4.1 Conclusion

In conclusion, our experimentation with CNNs has provided valuable insights into the factors influencing model performance for hair types classification. Our first proposed model achieved a notable validation accuracy of 0.7005 and a validation loss of 0.6857, representing a significant improvement over the base model. This enhancement in performance can be attributed to several key factors.

Firstly, the application of preprocessing techniques such as Laplacian sharpening, along with increased image resolution, played a crucial role in enhancing the model's ability to extract relevant features from the input images. These preprocessing steps helped to enhance image clarity and emphasize important details, resulting in more discriminative representations.

Additionally, the choice of activation function, particularly the utilization of Swish activation

throughout the model, contributed to improved performance.

Furthermore, our second proposed model, featuring a deeper architecture, demonstrated the potential for further performance gains. By increasing the depth of the network, we provided the model with a greater capacity to learn hierarchical representations of the input data, leading to more sophisticated feature extraction and classification capabilities. This model achieved a training accuracy of 74.88% and a training loss of 0.5638, while on the validation set, it attained an accuracy of 70.05% with a loss of 0.6794. Despite the longer training time of 1000.86 seconds, the deeper architecture showcased improvements in precision (73.60%), recall (66.50%), and AUC (87.49%), suggesting enhanced classification capabilities, especially in distinguishing between different classes. This underscores the significance of exploring deeper architectures to unlock the full potential of convolutional neural networks for hair type classification.

Throughout our experimentation, we have gained valuable insights into the principles governing CNNs and the impact of various architectural choices on model performance.

4.2 Recommendations

Based on the findings, we recommend the following for future researchers:

- To effectively classify hair types within the dataset, which consists of typical images containing people, including facial features, it is advisable to perform hair segmentation first. This initial step will help narrow the focus to the relevant area of interest, enabling more accurate classification of hair types.
- If hair segmentation proves too complex to implement, we suggest incorporating attention mechanisms to emphasize critical regions for classification.
- We also recommend the utilization of data augmentation techniques to further enhance model performance.
- Exploring deeper architectures is also recommended as it has the potential to uncover more intricate patterns.

V. REFERENCES

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep Residual Learning for Image Recognition. Computer Vision and Pattern Recognition, https://doi.org/10.48550/arXiv.1512.03385