COMP1804

APPLIED MACHINE LEARNING

Joshua Roses-Agoro

000992718

| MSc: Data Science |

***Abstract*—This report contains an investigation into facial recognition machine learning algorithms, more specifically, a multi-output convolutional neural network created by usage of Keras and has been implemented in python. The purpose of this investigation is to demonstrate thorough understanding of the full pipeline associated with annotating a dataset, pre-processing said data, defining model constructs, training a model, and fine-tuning it. Before it can be evaluated and fully understood.**

Keywords—Machine Learning, facial recognition, SGD, RMSProps, Keras, image augmentation, Convolutional Neural Network (CNN).

I. Introduction

Machine Learning is a crucial part of modern-day society, playing a key role in our development and further advancements into a more technologically enabled era. The focus of this coursework is drawn toward a subset of Machine Learning known as facial recognition and the full pipeline associated with it which is a complex subject with many potential avenues to discuss, but the one being taken in this instance is to do with the entire process from first obtaining the images that need to be classified, annotating them, creating visualizations of said data to understand distributions, image pre-processing, normalization, image augmentation, building the model, fitting the model, and finally, graphically visualizing the results.

The first and most important step in this process was annotating the dataset that had been provided as part of the coursework. Initially all 2000 images were labelled, but this was achieved by usage of a third-party tool provided by makesense.ai, but the way in which this tool output the labels created numerous difficulties which resulted in a failed model. Final attempts were made using the provided annotation tool that was recommended by the module leader and the model achieved success!

Particular attention needs to be drawn toward the goals and outcomes of this coursework, the pipeline otherwise known as the steps required from annotating and preparing a dataset, all the way to fine-tuning the finished product are what the core focus is drawn toward, for the best possible comprehension. And although a minimum accuracy of 50% must be achieved on the model, the aforementioned pipeline being implemented correctly is of utmost importance, before even taking the accuracy of the model into consideration.

II. Related Work

On the subject of Deep Learning there are various techniques which provide several limitations, some are best suited to facial recognition systems and those will be discussed here.

Joint line fine tuning is one of these techniques and has been implemented through usage of Deep Neural Networks (DNN). The methodology of said technique incorporates two levels of deep networks. The first of which is the ability to capture temporary features by utilization of the deep network, and these features are related to the appearances of the images based on their sequence. The second DNN captures geometrical shapes which are unique to each individual and can serve the purpose of identification.

With the combination of these neural networks, a new integration method for facial recognition was developed which yielded better results than the traditional models which had been implemented in the past, with an overall accuracy resulting in 97.25%.

When breaking down the two DNN's used for this methodology. The first of which to explain is the deep temporal appearance network (DTAN), and the second being the deep temporal geometry network (DTGN).

As explained briefly before, the DTAN is based on a Convolutional Neural Network (CNN) which deals with the temporal features of the relevant images used in the dataset for facial recognition. And the DTGN which comprises of a fully connected Deep Neural Network, which captures and extracts facial landmark points.

During this methodology image sequences are used without weight sharing along the time axis. This results in the different features playing a different role depending on the time in which it is accessed.

By means of capturing the facial recognition landmark points, the various trajectories are considered as one-dimensional signals. The gathered trajectories, otherwise, can be considered as coordinates, and are not an acceptable input for a deep neural network, and this is where the process of normalization fixes this issue and prepares it for the model.

In conclusion, the two deep network models that have been presented in this related work propose an interesting and technical approach to facial recognition. The way in which two networks are being concatenated is an unusual approach but proved successful with an achieved accuracy of 97.25%, as mentioned previously. This would make the joint fine-tuning model more successful than any previously created integration method.

III. Dataset Preparation

For this coursework 2000 images were provided as a dataset to base the facial recognition algorithm on. As briefly described in the introduction, the first attempts at labelling the dataset were performed with a tool provided by makesense.ai which allows you to import the images from the dataset, create a custom list, consisting of the necessary labels, and manually labelling each image before exporting it in a CSV file.

Once all 2000 images had been labelled, the way in which it had been output by makesense.ai created several difficulties in terms of compatibility within the algorithm itself, and the results from the final trained model were horribly biased due to an incompetent dataset, and unfortunately this resulted in all 2000 images needing to be scrapped and relabeled.

The next approach was to use the annotation tool provided by the module leader and this resulted in a labelled dataset that was perfectly compatible with the algorithm and image processing techniques which needed to be implemented. Due to the large number of biases existing within the dataset, it was determined that a similar result could be achieved with less images, and so 500 were annotated and prepared for the algorithm.

There were 5 classifications in which the images needed to be divided between, that being wrinkles, freckles, glasses, hair colour, and hair top, and within those many sub-classes, defined by integers.

Once the images were successfully annotated, they could be imported along with their corresponding CSV file, and from this the relevant distributions of each class from the entire dataset were visualized. The separate distributions for each individual class can be found in the appendix at the end of the report.

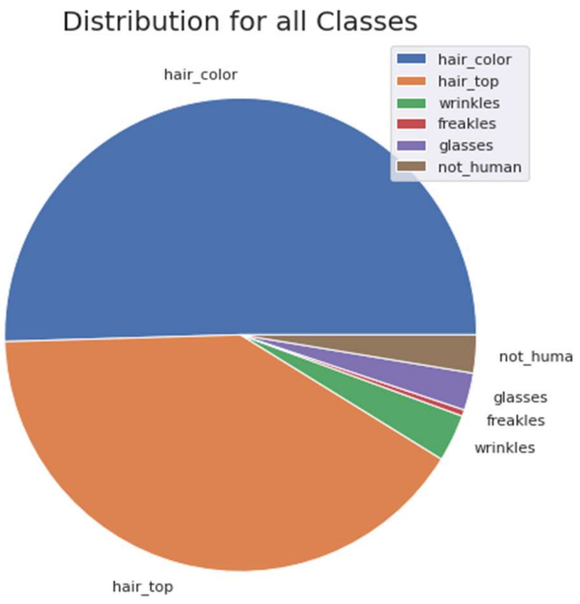

Figure 1 - Distribution for all classes

The dataset itself is stored locally within the computer, and each image is stored using the JPEG extension. Several of the images have different dimensions and this can pose potential problems when being fed into a neural network, and so one of the first steps regarding image pre-processing is identifying the images which have a dimension which is not uniform with the rest, and resizing it using the CV2 (Open Computer Vision) library to 200,200. Another benefit of ensuring all of the images have the same dimension is that it reduces the computational power needed to process them.

The images were loaded in batches to reduce computational power needed, thus ensuring no errors occur due to there being an insufficient amount of memory available. This also means the entire dataset does not need to be loaded at once. As they are loaded in, the corresponding CSV file containing their unique annotations is loaded simultaneously as a means of identification and is attached to the images. Finally, the images are loaded into a Numpy array and are prepared for the next pre-processing technique, which is Gaussian Blur.

The Image Augmentation Library, otherwise known as IAA by its import was used to perform Gaussian Blur, a crucial tool in image pre-processing which smooths edges and reduces the overall noise of the image, therefore simplifying the task of interpretation for the algorithm.

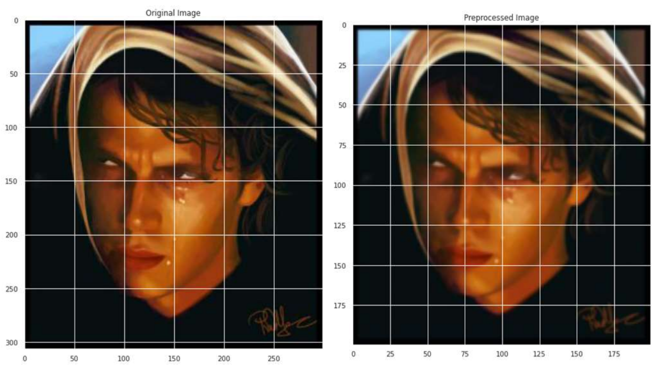

Figure 2 - Gaussian Blur

Image Augmentation was the next implemented phase of pre-processing. It is a technique which artificially expands the size of a given training dataset by augmenting the images and creating modified versions which can be used during the training process.

There are unique parameters in which an image can be augmented by, such as orientation, blur, brightness, pan, zoom, etc. And many of these features have been incorporated on the dataset used in this coursework in order to enhance the diversity of it and ensure a more accurate model.

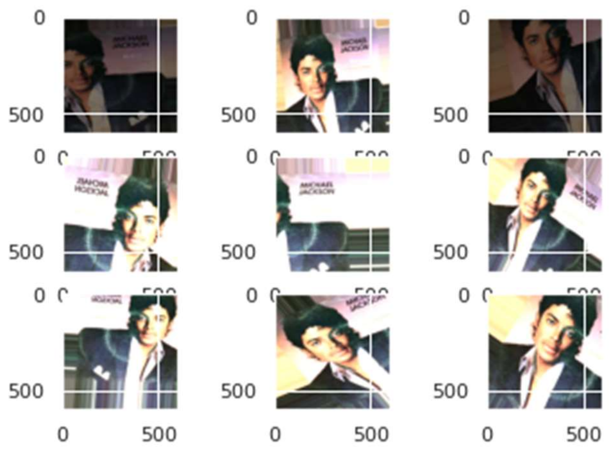

Figure 3 - Image Augmentation

The above image contains the output of the ImageDataGenerator which is a Keras function, used to augment images, by inputting the parameters by which a user would like to augment according to, and executing the function. The result are images which can be used to enhance the currently existing dataset by artificially expanding it, thus providing more samples for the model to train against and potentially increasing the model accuracy.

A random scale set has been applied to each parameter of augmentation, which corresponds to the likelihood of that augmentation being applied, and this will enable the model to generalize on images it has not been exposed to.

IV. Machine Learning Method

During this phase, the multi-output Keras model needed to be defined, and this model was composed of 5 major branches. This is so because each feature of classification needs its own branch for identification when it comes to training, seeing as this is a multi-output model after all.

A Convo2D layer with ReLU activation is the default structure for convolutional layers, and this is followed by BatchNormalization layers, MaxPooling, and finally a Dropout layer, the final missing parameter is a Dense layer which needs to be applied, and then this step can and must be repeated for each output that is trying to be predicted, because this is a multi-output model.

Then the class responsible for creating the multi-output model needs to be defined and this is done on the hidden default layers and will be reused over again for each branch of the model. The final result is a model structure comprising of an input layer which retrieves images from the dataset in batches, based on the previously defined batch generator, and then decomposes into separate branches, defined in the CNN, which each have their own subset of convolutional layers, which finally end in their dense layers, before the model can be trained.

Now that the model architecture has been successfully defined, it needs to be compiled and that will lead to the first phase of training able to commence. Hyperparameters were defined in their own separate variables so that trial and error training would be a simplified process whilst fine-tuning the model.

Within these hyperparameters several variables were defined. Thus being, batch size, epochs, activation layer, activation output, optimizer, and loss. Batch size determines how many images are being passed at a time during training, the chosen value was 32, to prevent memory issues and the network trying to learn too many features at once. Next is the number of epochs, seeing as only 500 images were used, training began at 50 epochs, but this was beyond the point of diminishing returns and the model was negatively impacted because of overfitting. Which was a result of the model not actually learning the data, and instead memorizing it, and therefore it is crucial to find the accuracy of validation data for each epoch.

The final number chosen for how many epochs should be used was 10, whilst simultaneously incorporating the batch generator, set at 128. And when you multiply the steps per epoch with the number of images in the batch generator, this results in 1280 images per epoch (because the dataset had been artificially enhanced by image augmentation).

Next the learning rate was applied, and this determines at what rate the model will learn new features. Three separate methods of optimization were tested, thus being RMSProp, Adam, and SGD (Stochastic Gradient Descent). Whilst testing the separating methods of optimization at different learning rates, SGD resulted in the best option because it allowed the trained model to achieve the best results.

The way in which SGD works is by randomly picking a single data point from the entirety of the dataset at each iteration to reduce the computations by a significantly large amount.

$$for\ i\ in\ range(m):$$

$$w_j := w_j - \alpha \frac{\partial J_i}{\partial w_j}$$

Figure 4 - Update step for SGD

Above is the equation which represents how Stochastic Gradient Descent performs its incremental updates. And the reason it works is because not every training example needs to be checked in order to determine the direction of a decreasing slope.

By the selection of a single data point and careful analysis in following its slope, it is possible to reach a point that is extremely close to the actual minimum.

The final hyperparameter worth mentioning is the categorical cross entropy method used to calculate loss.

$$\text{Loss} = -\sum_{i=1}^{\substack{\text{output}\\\text{size}}} y_i \cdot \log \hat{y}_i$$

Figure 5 - Loss = Categorical Crossentropy equation

This method utilizes the scalar value in the model output, the corresponding target value, and the output size which is the number of scalar values in the model output.

When trying to distinguish discrete probability distributions from each other, this loss function is a good measure in doing so. The reason being that classification tasks can belong to a particular category with a probability of 1, and to other categories with a probability of zero.

V. Evaluation

"A good fit is identified by a training and validation loss that decreases to a point of stability with a minimal gap between the two final loss values.

The loss of the model will almost always be lower on the training dataset than the validation dataset. This means that we should expect some gap between the train and validation loss learning curves. This gap is referred to as the generalization gap." - (Brownlee, 2019)

The overall results attained for the multi-output model are relatively satisfying and moderate success has been achieved. The reason why total success has not been achieved is because of the dataset used for the implementation.

It is extremely biased and does not present an equal contrast of samples in terms of how everything is distributed between the various classifications and this resulted in a bias that no amount of image augmentation was able to fix.

The predominant bias which occurred after training was to do with hair colour. The vast majority of the samples all had brown and black hair, and this hindered the algorithm's ability to generalize in accordance with the other hair colour classifications. All of the graphical results can be found within the appendix, the only graphical result being displayed in text is to do with the overall loss from all 5 separate models.

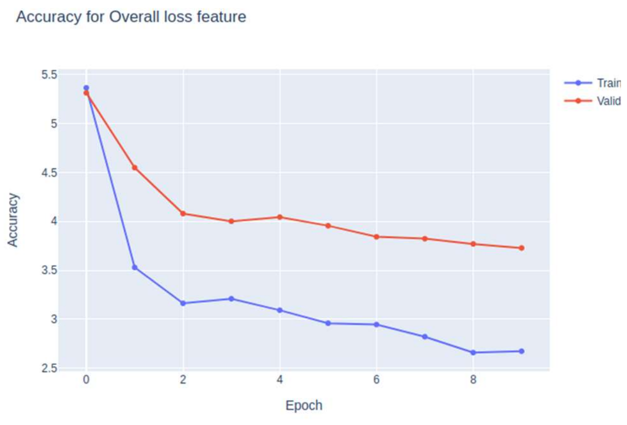

Figure 6 - Accuracy for Overall Loss

As seen in the above diagram is the accuracy for overall loss, with a train accuracy of 2.67% and a validation accuracy of 3.72%, which is well within the accepted ratios.

Classification:	**Training:**	**Validation:**
Overall Loss	2.67%	3.72%
Freckles	98%	96%
Wrinkles	87%	86%
Glasses	92%	95%
Hair Colour	51%	48%
Hair Top	69%	66%

Above, the results for all of the various classifications can be found, and on average the training process was a success, and the model was able to generalize enough to produce acceptable results.

The major exception with these results being hair colour, and as explained before there was not enough diversity amongst the samples to achieve a higher accuracy than 50% at most. Whether or not 2000 images were used or 500, the result was exactly the same regarding hair colour.

There are several pros of the model and they are listed as follows: Several layers were added in the model constructs before training in order to correctly establish the weights. An SGD optimizer with a very slow learning rate was also added to ensure small increments would take place whilst training occurs as to not overwrite previously learnt steps, and finally, the perfect number of epochs were used in the training process with a sizeable number of samples set aside for validation. Thus, ensuring that a model which seemingly appears to be a good fit, can successfully be validated.

When testing between SGD and RMSProp as an optimizer to use for training, the reason RMSProp was not chosen in the final implementation is because it is an adaptive learning optimizer which has a large magnitude in the sizes of increments during training, which were likely to eradicate important features learnt during training, and this was the case because the results in validation were horrific.

One point that needed specific attention is that a low learning rate without sufficient number of epochs tend to underfit the data, so the low learning rate of SGD is not always appropriate, and that is why it was essential to find a balance between too few epochs and too many epochs.

Although SGD was a significantly better choice of optimizer over RMSProp, it comes with its cons too. SGD is fast but the convergence path is noisier than that of original gradient descent. The reason for this being that no calculation is performed for the actual gradient descent, instead an approximation is made. And the result of that is fluctuations in the cost.

VI. Future Work

There are several avenues which can be explored in order to extend the work accomplished in this coursework. But there are two which should primarily be focused on.

The first of which is improving the actual pipeline from cleaning data, to pre-processing, to defining model constructs, to training the model, etc. By fine-tuning all of these steps a more efficient pipeline can be achieved which may ultimately lead to better results. Another point which coincides with this one would be different methodologies or approaches to the problem. A Convolutional Neural Network is only one of many potential avenues and exploration with different model types is the only way to figure out which is the most efficient.

I would also like to improve my ability to fine tune a model because this phase of the algorithm's development is extremely crucial and although I did a fine job in this instance, it is definitely not the most optimal way and further explorations into extreme fine-tuning may yield better results with the final trained model's accuracy and ability to learn instead of memorize.

The second point worth consideration would be to hand pick images to form my own dataset with equal distributions in hopes of preventing the major bias's which occurred in this dataset. An imbalanced dataset can significantly impair a model's ability to generalize, and this was the result in this case with one of the classifications.

VII. References

[1] Bressan, R., 2020. Building a multi-output Convolutional Neural Network with Keras. [online] Medium. Available at: <https://towardsdatascience.com/building-a-multi-output-convolutional-neural-network-with-keras-ed24c7bc1178> [Accessed 10 April 2021].

[2] Brownlee, J., 2019. How To Use Learning Curves To Diagnose Machine Learning Model Performance. [online] Machine Learning Mastery. Available at: <https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/> [Accessed 11 April 2021].

[3] Jung, H., 2015. [online] Cv-foundation.org. Available at: <https://www.cv-foundation.org/openaccess/content_iccv_2015/papers/Jung_Joint_Fine-Tuning_in_ICCV_2015_paper.pdf> [Accessed 11 April 2021].

[4] Team, K., n.d. Keras Documentation: Optimizers. [online] Keras.io. Available at: <https://keras.io/api/optimizers/> [Accessed 12 April 2021].

[5] Roses-Agoro, J (2020). 'COMP1801 Machine Learning Coursework' – University of Greenwich [Accessed 05 April 2021].

[6] Dutta, S., 2019. Why Stochastic Gradient Descent Works?. [online] Medium. Available at: <https://towardsdatascience.com/https-towardsdatascience-com-why-stochastic-gradient-descent-works-9af5b9de09b8> [Accessed 15 April 2021].

[7] Peltarion.com. 2021. Categorical crossentropy loss function | Peltarion Platform. [online] Available at: <https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/loss-functions/categorical-crossentropy> [Accessed 16 April 2021].

[8] Nooney, K., 2018. Deep dive into multi-label classification..! (With detailed Case Study). [online] Medium. Available at: <https://towardsdatascience.com/journey-to-the-center-of-multi-label-classification-384c40229bff> [Accessed 9 April 2021].

VIII. Appendix

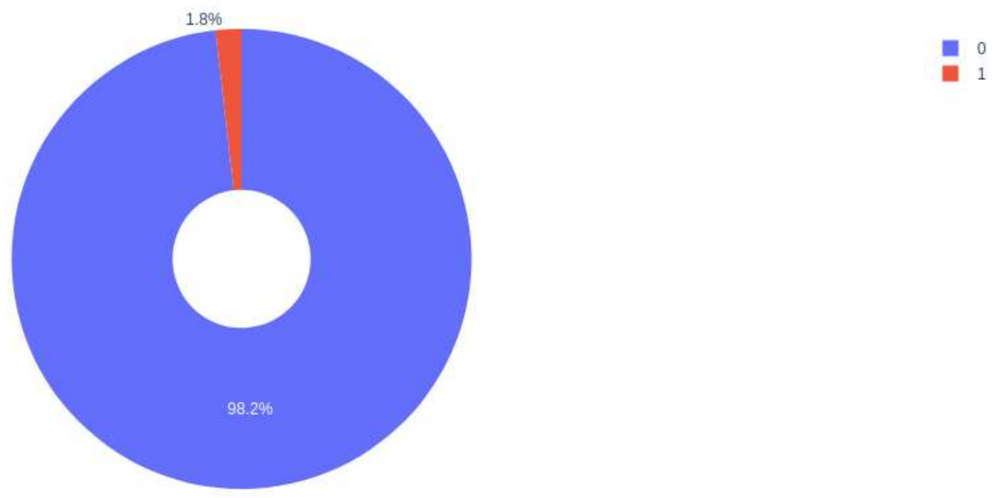

Figure 7 - Distribution for freckles

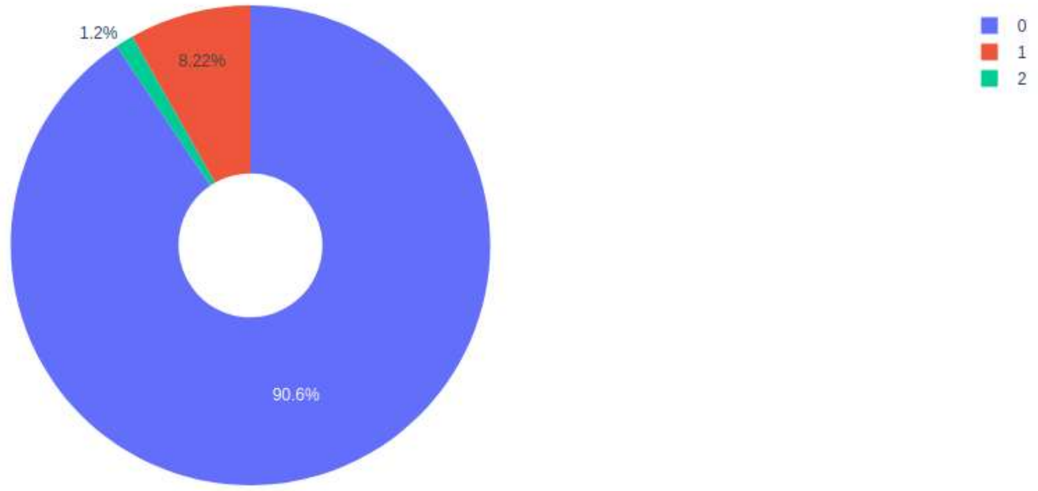

Figure 8 - Distribution for glasses

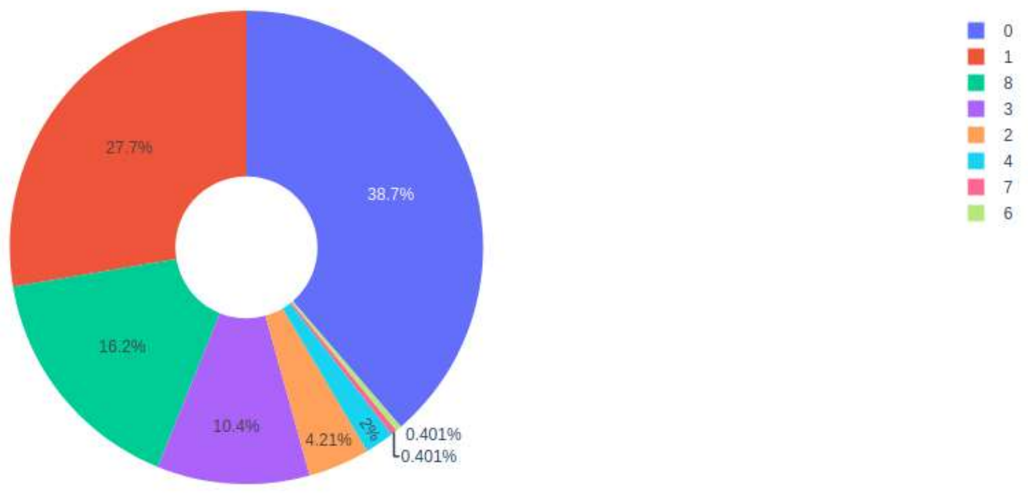

Figure 9 - Distribution for hair colour

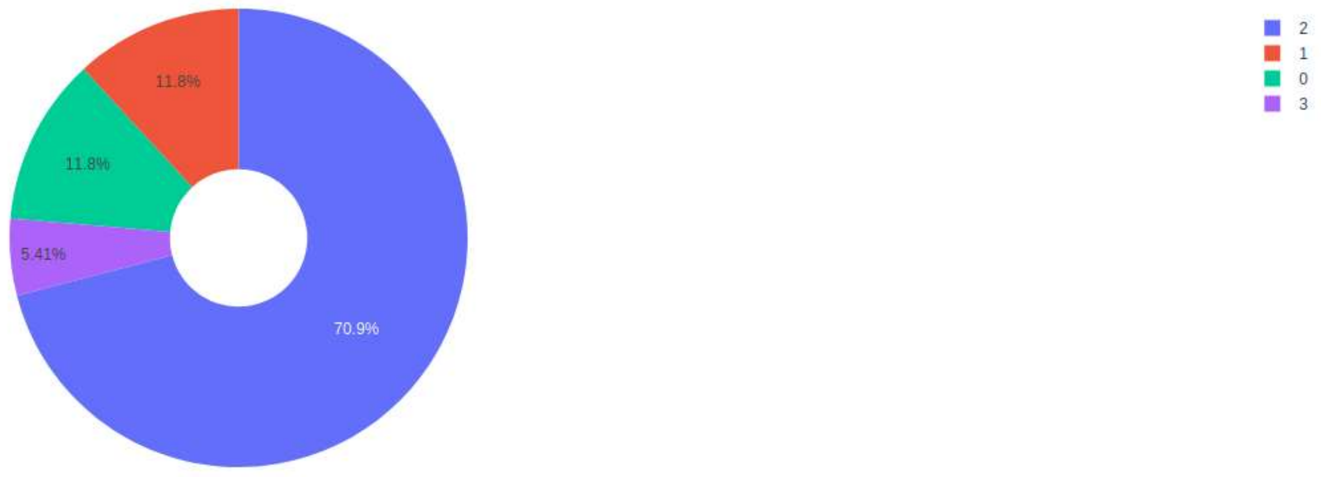

Figure 10 - Distribution for hair top

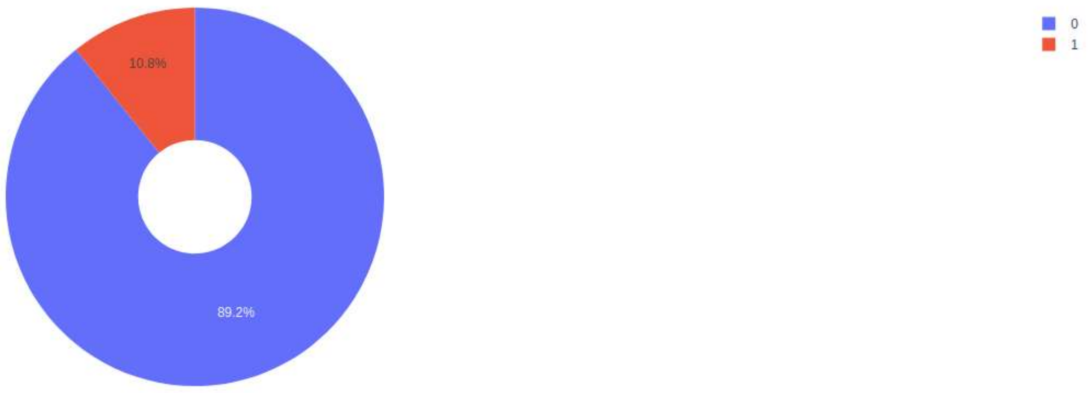

Figure 11 - Distribution for not human

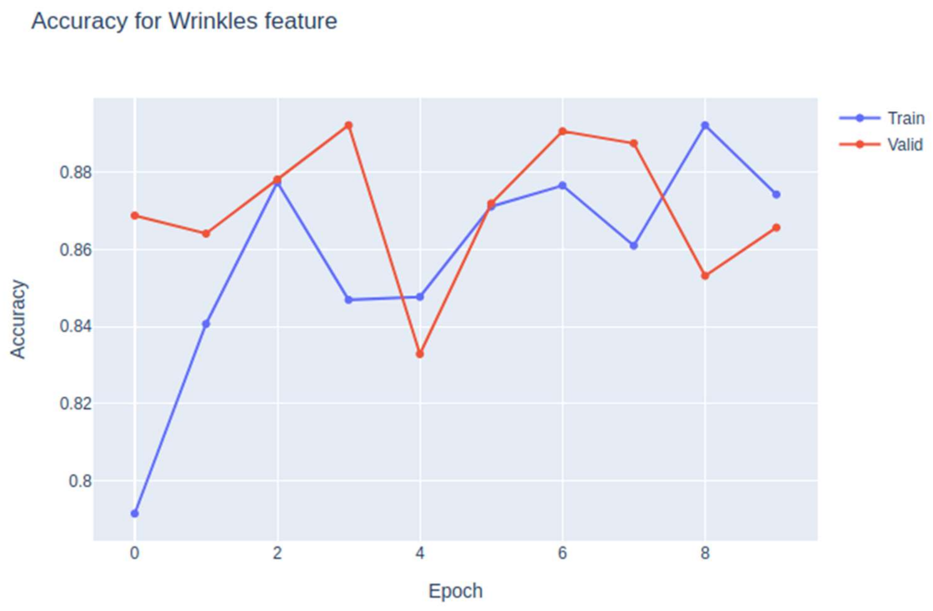

Figure 12 - Accuracy for wrinkles

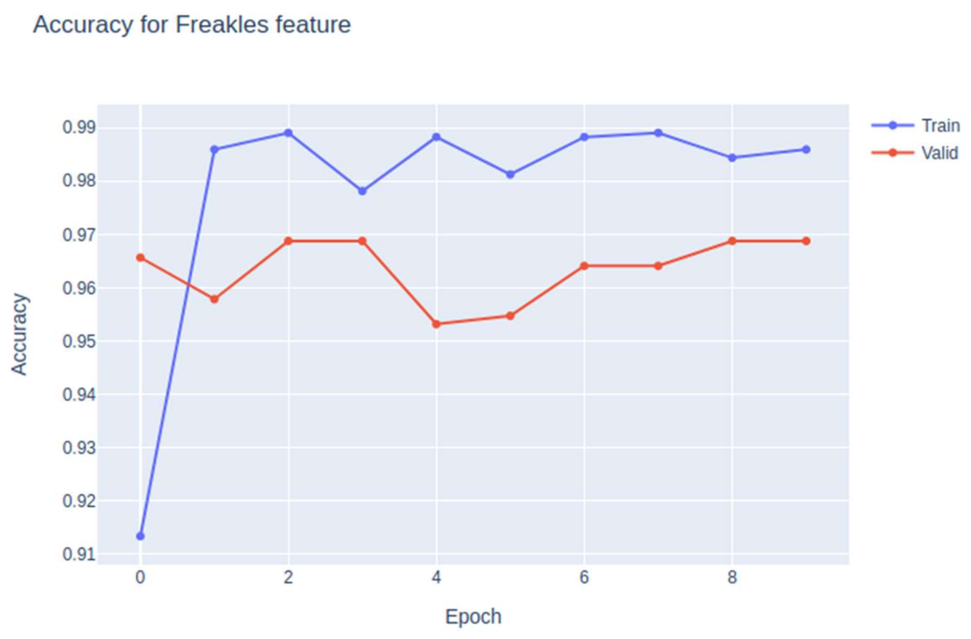

Figure 13 - Accuracy for freckles

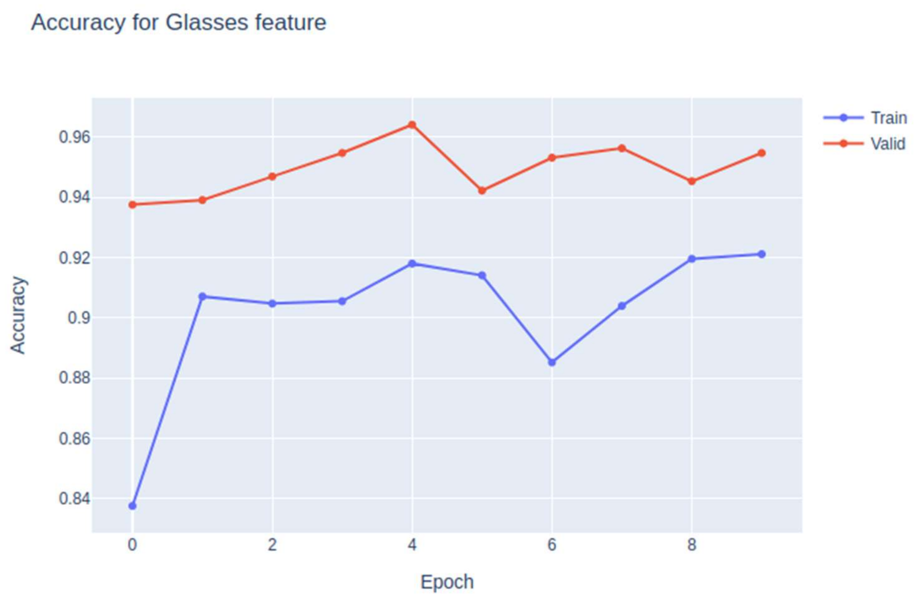

Figure 14 - Accuracy for glasses

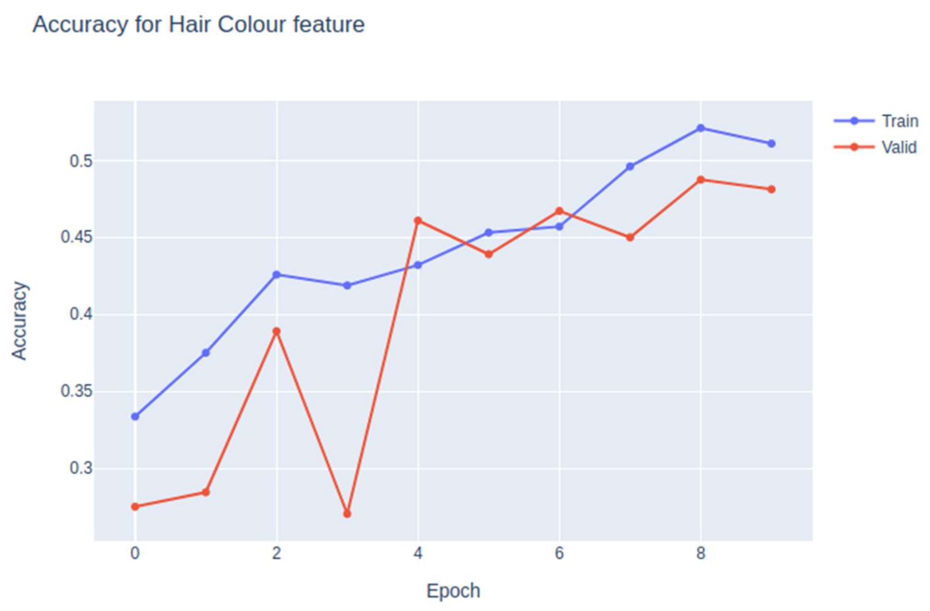

Figure 15 - Accuracy for hair colour

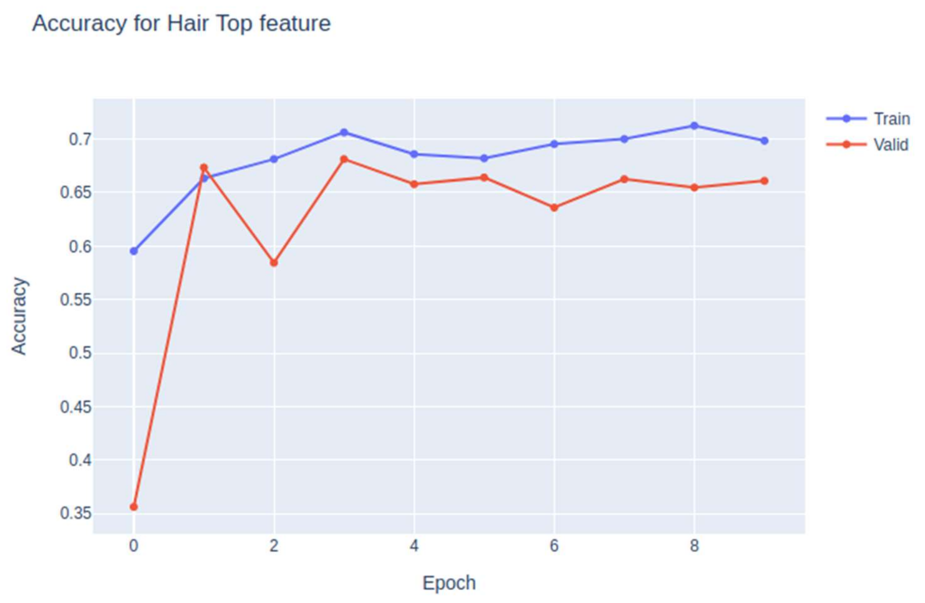

Figure 16 - Accuracy for hair top

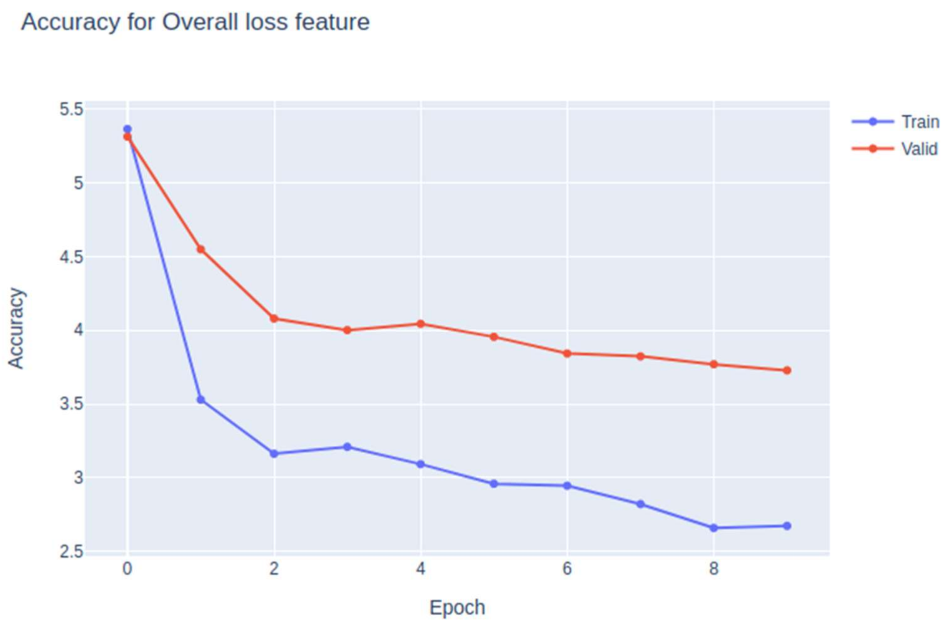

Figure 17 - Accuracy for overall loss