CS 412 Project Report

To run code, please first [READ ME](https://docs.google.com/document/d/1MBKA2ERi1SLBhWiGCeB-u2QcmEVVCv83jBJg9BamhKw/edit?usp=sharing) & [Codelink](https://colab.research.google.com/drive/1GocCfVuSsnKpKJLDRrYoTereBz64whXo?usp=sharing)

To access the [Drive Folder](https://drive.google.com/drive/folders/1au0Wyw3vBsi35-WU0H3NB9twzlCaPpS8?usp=sharing) & [Final Notebooks](https://drive.google.com/drive/folders/1CXAV_pJvLH4ywxssUgAAFuySiKubkjLX?usp=sharing)

# 1) Problem Definition

Our goal is to build a machine learning model that classifies 5 different types of skin cancers with best accuracy possible based on the skin images which were provided to us. These cancer types are: Melanoma (MEL), Melanocytic nevus (NV), Basal cell carcinoma (BCC), Actinic keratosis (AK), Benign keratosis (BKL).

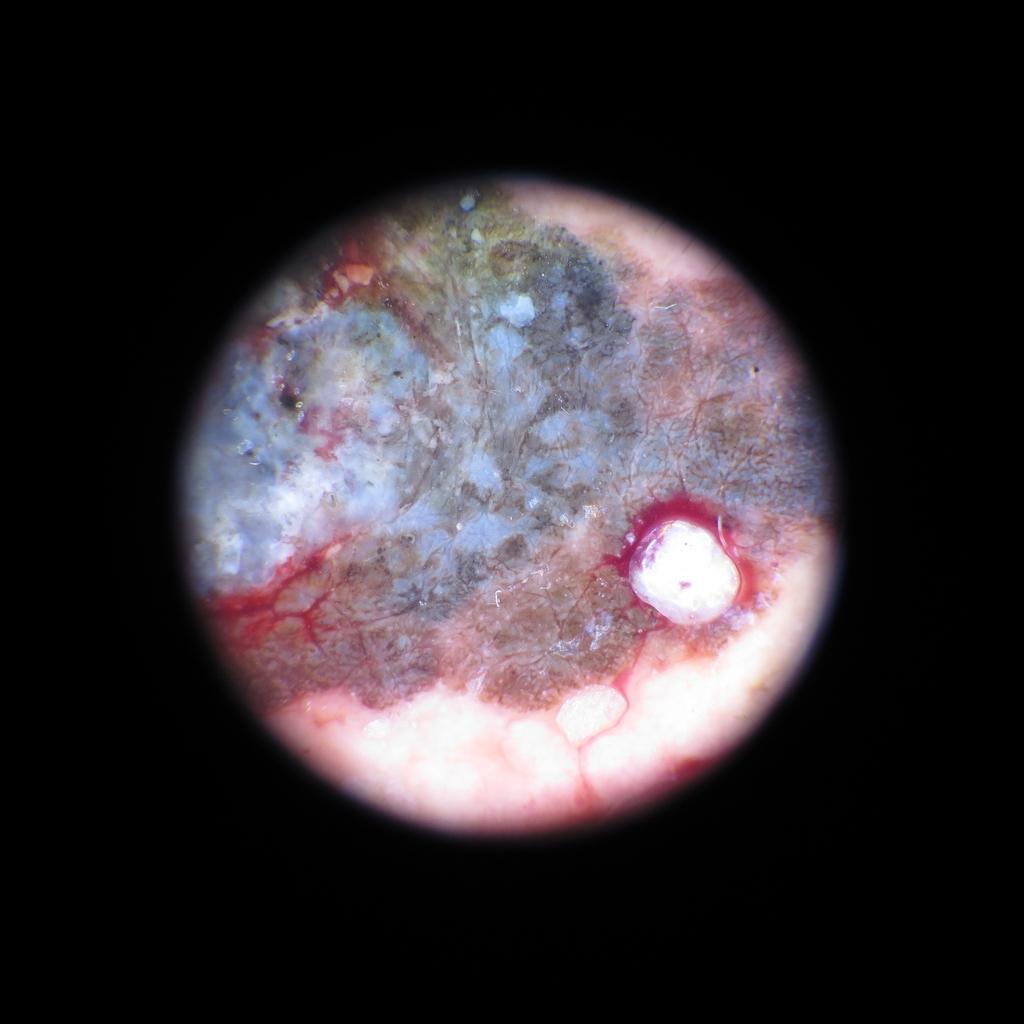
# 2) Discussion About Data

## 2.1) Data Provided

Data provided to us to train the model consists of 10.000 images with their corresponding labels. We have another 5.000 images without their corresponding labels for us, and our goal is to predict their labels based on our model and test the accuracies from Kaggle website by submitting our predictions as a csv file. All these 15.000 images are in the RGB format. Some samples from the data:

## 

## 2.2) Circular Black Border Problem

While we were inspecting the images given to us, we observed that some images have some black circles around them. You can see an example of an image below:

## 2.3) Ratio of Classes

As you can observe from the graph below, our training data was unbalanced meaning that the number of images that belong to specific class, were not the same proportionally.

# 

# 3) Preprocessing

## 3.1) Trimming

As we stated in the above [paragraph](#_nai1hgvot1vd), there were black circles around some of the images and we thought that this may cause a problem while training our model. Therefore we decided to apply trimming to the black circled images. Such that, we start checking all images and when we see it is a black circled one we crop the black parts from that image and write it back to drive for further usage. You can see an example of the trimmed version of the image below.

|  |  |
| --- | --- |
|  |  |
| **Trimmed Not Cropped Example** | **Trimmed Cropped Example**  **(used as final data)** |

## 3.2) Balancing and Augmentation

Since our training data was unbalanced and it’s size was relatively small, we decided to use the oversampling method in order to balance the data. For this reason, we augmented our data such that each class had nearly the same number of samples. For instance, if we augment an image with label 2 twice, we augment an image with label 5 ten times to get the same proportions.

## 3.3) Pickling

In order not to load the image data whenever we need to train some model, we store the processed images in a binary file (pickle file) in Google Drive. Thus, we only load the data once and are able to speed up the overall training process.

# 4) Baseline model

As a baseline model, we used basic CNN architecture which consists of

Conv2D(kernel size = 64)

MaxPooling2D(pool\_size=(2,2), padding='same'))

Conv2D(kernel size = 128)

Conv2D(kernel size = 128)

MaxPooling2D(pool\_size=(2,2), padding='same'))

Conv2D(kernel size = 256)

MaxPooling2D(pool\_size=(2,2), padding='same'))

We used “relu” as an activation function except for the last layer, and we used “softmax” with 5 classes output for the last layer. We got nearly 63% validation accuracy and 61% real test accuracy at Kaggle.

# 5) Methods Used

## 5.1) Augmentation

As we mentioned before, we used augmentation techniques to increase the size of our training data. We performed the augmentation process during the preprocessing part and saved those new images in Google Drive. Afterwards, we used this newly created data to train our models. We will talk about the transfer learning method in detail at the below part. While using transfer learning, we did not use our augmented images, instead we augmented the data during the training time at each time by using data generator’s flow method which is supported by Keras. The reason for this approach is because, we used colab to use its GPU’s so we are restricted with the Colab’s ram size which is 12 GB. When we tried our own data with image size more than 75x75 pixels, Colab’s ram was not enough and it restarted the kernel automatically. Therefore, we used the data generator’s flow function, which augments the data only for that batch while training and then just continues keeping the original image.

To tell more about augmentation, let’s mention the approaches that we used during augmenting the data. We used rotations, so that the rotated versions of the data can be understood more easily by our model. We used width and height shifting, zooming to make the image closer by making it bigger and shearing to the image. You can see the augmented version of the two images below.



## 5.2) Transfer Learning

In order to get better accuracy results, we used pre-trained models and changed their last layer accordingly. Afterwards, we performed hyperparameter tuning to get the optimal scores. Since there are many pre-trained models, we first performed literature review to get a better understanding of the cutting edge technologies that are used in image classification and chose the models that perform the best. After spending enough time on that, we decided to try DenseNet201,InceptionResNet,InceptionV3, VGG16, Xception and Resnet152V2 models.

While training the pre-processed training data with the models, we discovered that the sizes of the images mattered and changed the accuracies. We tried 75x75, 100x100 and 128x128 image sizes and learned that the best results are the ones with the higher sizes. We wanted to try bigger sizes but the Colab’s ram size was not sufficient for our approach. You can access the detailed results and performance comparisons of the algorithms and their models based on size [here](https://docs.google.com/spreadsheets/d/1DBFb4IfHjudCvixfgQfeUYrGQJfC_W6a2qPvKzYWqHo/edit?usp=sharing).

The other problem we had was the type of the data we need to use. Therefore, we created another log table to test our models with different data types. You can access the detailed results [here](https://docs.google.com/spreadsheets/d/1DBFb4IfHjudCvixfgQfeUYrGQJfC_W6a2qPvKzYWqHo/edit#gid=1268898128).

In the end, we have decided that using trimmed and cropped images with size 128x128 is the best available solution for this task. Now, the goal is to train the pre-trained models with our pre-processed training data. You can see the train and validation accuracies below for the models we used to get our final score.

|  |  |  |
| --- | --- | --- |
| **Model** | **Train Accuracy** | **Validation Accuracy** |
| Xception | 96.44 | 78.50 |
| Xception FineTuned | 95.06 | 79.00 |
| DenseNet201 | 94.40 | 77.40 |
| DenseNet201 FineTuned | 93.86 | 75.90 |
| InceptionV3 | 64.35 | 65.10 |
| InceptionV3 FineTuned | 84.64 | 72.20 |
| InceptionResNet | 85.78 | 71.49 |

After we trained our models, we saved the model and its weights for the parameters in order to use for later trials and combine them with the ensemble method. One can also use these weights to obtain nearly the same result for themselves.

# 6) Best Performance and Why

For the final testing we used a trick to combine all the pre-trained models with our weights initialized and prepared all the models for this. We used ensemble and defined a function which combines the given models and averages the outputs of them. By doing this we get the best result because we believe that if one model fails to identify a label the other models could fix it’s mistake. You can see the function here.

def ensemble(models, model\_input):

outputs = [model.outputs[0] for model in models]

y = layers.Average()(outputs)

model = Model(model\_input, y, name='ensemble')

return model

We used many different combinations of models yet the best test score (according to Kaggle) obtained from the combination:

DenseNet201 & Xception & InceptionResNet & Inception-FineTune

Please check the [Colab File](https://colab.research.google.com/drive/1GocCfVuSsnKpKJLDRrYoTereBz64whXo?usp=sharing) for the ensemble models to see all the codes.

# 7) Overall Evaluate

Thanks to the transfer learning, we were able to obtain the best accuracy results by using the ensemble method which contains DenseNet201, Xception , InceptionResNet, Inception-FineTune combination and got 81.485% test accuracy in Kaggle by using our pre-processed data. When we use trimmed and cropped data, we get a higher accuracy score compared with the actual provided data. Hence, we were satisfied by our preprocessing method and overall accuracy score from the transfer learning methods.

# 8) Future Work

Since colab was used while training data, our RAM was limited. These codes can be converted to run in local and can be runned with bigger sized images to get a better accuracy.

We trained our final models with 9.000 instances of the given training data in order to provide some validation accuracies. As a future work, it can be trained with all given instances at the training data.