Deliverable 2 (Gargi Singh and Jehan Dastoor)

Problem Statement

We are attempting to create a CNN model that can classify different types of cancerous skin lesions. We also want to be able to give the user the best possible method for diagnosis of that type of skin lesion.

Data Preprocessing (https://challenge2018.isic-archive.com/task3/)

We are currently working with the ISIC 2018 Task 3: Lesion Diagnosis dataset with no changes having been made. The dataset has 10,015 training images and a metadata.csv file that maps the images to their respective class. For preprocessing, we are doing the following:

1. Normalizing our images (i.e. dividing by 255) so that the CNN receives normalized data that it can work with better.
2. Resizing our images so that they are 64x64px instead of 600x450px in order to save some processing time and reduce the number of variables needed for each layer in the network.
3. Shuffling the dataset so that there is no possibility of correlations existing between images that were taken at certain times etc.
4. Converting the images to tensors in a tf.data.Dataset object to be used later.

Further preprocessing to be done later that will increase the number of images used to train and make the predictions more rigorous:

1. Applying rotations to each image.
2. Small zooming to each image.
3. Horizontal flipping.
4. Shears.

This will make our model more likely to predict the right class when given slightly different images.

Machine Learning Model

a. The model was written in tensorflow.keras. As of now, the model is fairly simple (with the goals of adding an already pre-trained classification model from tensorflow's library at the beginning later) with the following layers
    i. Layer 1
        1. Conv2D with 32, 3x3, filters chosen as this is the standard used.
        2. MaxPooling with a 2x2 window to reduce our feature map by a factor of 2 for faster processing.
    ii. Layer 2
        1. Conv2D with 64, 3x3, filters as we have less data now so we can use more filters.
        2. MaxPooling with a 2x2 window to reduce the feature map by a factor of 2.

iii. Layer 3
   1. Conv2D with 64, 3x3, filters to provide a final mapping of features.

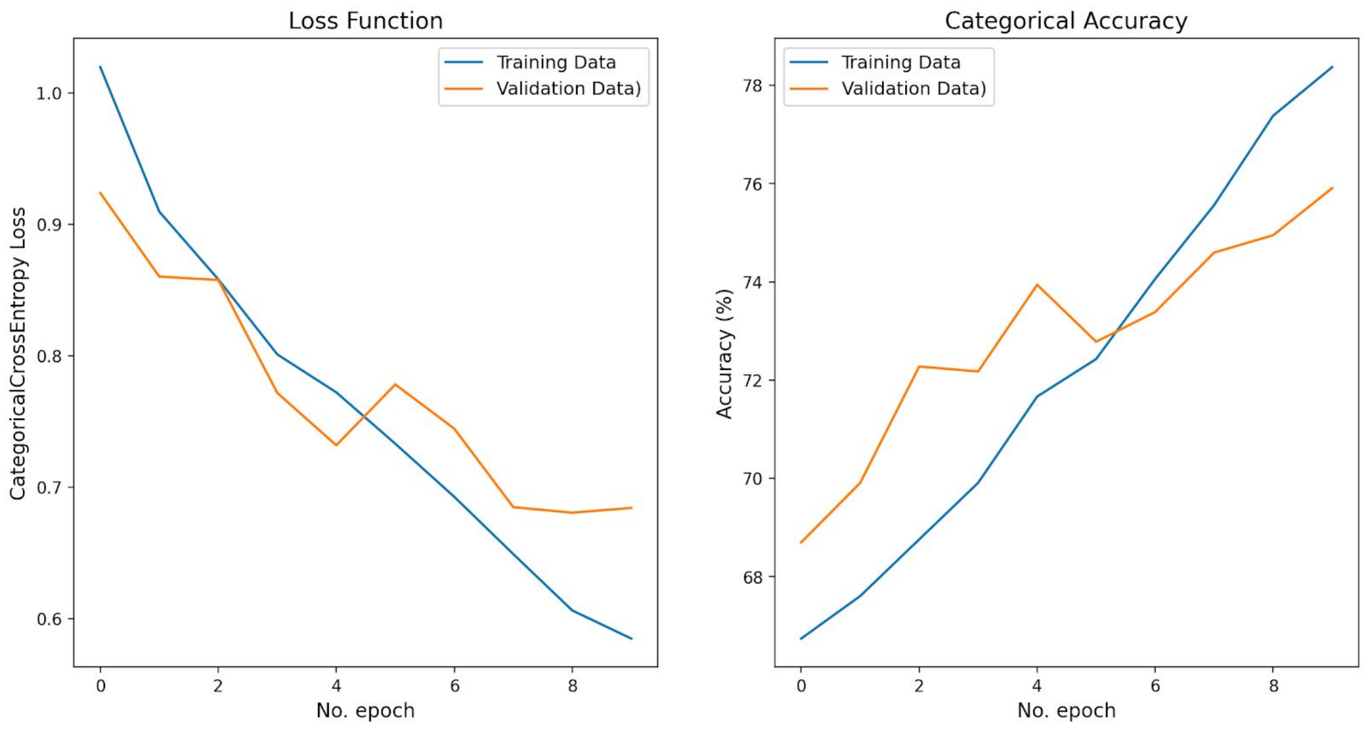
iv. Layer 4
   1. Flatten to create a 1D list of features.
   2. Dense layer to feed all the outputs from the previous layer to 64 neurons.
   3. Another Dense layer to feed the previous dense layers outputs to 7 neurons that make up our 7 different classes we want to identify.

Not much experimentation has been done on the number of filters being used as the goal is first to add a pretrained model to improve accuracy and then parameter tune. The training data was split into 20% validation data so that there would be plenty of data to be used for testing. At the same time, the percentage was not made too high as even though we have 10k+ images, with 7 classes that gives about 1.4k+ images per class which is not that much. For regularization, ImageDataGenerator was used in keras_preprocessing.image. Lastly, the training was done for 10 epochs so that it would not take too long at this preliminary stage, yet we would still be able to get a decent accuracy.

b. The model was tested against the validation data using a "categorical_accuracy" metric and a CategoricalCrossEntropy loss function. As of now, the model is achieving only ~80% accuracy, so it is underfitting a little (the creators of the task said that 90-95% is acceptable). However, I cannot check with their test data as they have not provided a .csv file mapping the images to classes for the test data. I have emailed to ask for this file and am waiting for a response.

c. Importing the images into my notebook was causing problems: I was getting a "Too many files open" error. To solve this I ended up using flow_from_dataframe to import the images using the metadata.csv file to classify the images at the same time. This took quite a bit of time to figure out, but after some StackOverflow digging it worked. I also did not realise that I could not use a SpareCategoricalCrossEntropy loss function as that requires normal integers where we have hot encoding, as well as the fact that sometimes two classes can have the probability 0.5 as the classification is unsure. Although, we might look into using SpareCategoricalCrossEntropy as it is faster and valid for most of the dataset.

Preliminary Results



If we look at the figure above, we see that the loss function is decreasing and the accuracy function is increasing (somewhat) linearly with the number of epochs. We have not started to plateau yet, which implies that a greater number of epochs should be used to have our model reach convergence. The model is performing at around 78% categorical_accuracy after 10 epochs which is significantly higher than random at 50%. We cannot be sure whether we are over or under fitting as we do not have the metadata file for the test images (which hopefully we get soon). Looking at these results, we should be easily able to get about 85% categorical_accuracy if we increase the number of epochs. Furthermore, if we tune our hyperparameters/add more layers/add a pre-existing model to the beginning of our current model, we should be on track for much better results.

Next Steps

To decide on one and add a pre trained tensorflow classification model to the beginning of our model so that we have a better base to build up from that is built by data scientists. We can fix the parameters of these layers and then work on the other layers. After this is done, we can begin playing around more with the number of filters being used for our Conv2D layers and the relevant padding and stride parameters etc. Furthermore, I would like to see how much cropping the image to 64x64px impacts our final accuracy. Once the mode is performing well, it can be left to train for many more epochs (e.g. 100+). However, this might become difficult to monitor for overfitting at that point.