Skin Cancer Classification

Introduction

The task is to build a machine learning model to predict the type of the skin cancer out of 5 different types. The given .csv file that consists of two columns as “Id” which is the name of the image file and “Category” (label) column is representing the category where,

Category = 1 represents Melanoma (MEL)
Category = 2 represents Melanocytic nevus (NV)
Category = 3 represents Basal cell carcinoma (BCC)
Category = 4 represents Actinic keratosis (AK)
Category = 5 represents Benign keratosis (BKL)

Baseline

For building the model, we need to evaluate each image in the dataset and extract features from them and represent the results according to extracted features. Since the CNN algorithm is one of the most popular algorithms for image processing, we chose CNN to build our machine learning model.

Using *ImageDataGenerator* and *flow_from_dataframe* from *keras_preprocessing.image* library, we merged the .csv file and image file. We splitted the whole training data into training and validation data by validation_split = 0.25.

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Conv2D(filters=64, kernel_size=3, strides=2, activation='relu', input_shape=(32,32,3)),### Conv2D layer
  tf.keras.layers.MaxPooling2D(pool_size=(2,2),strides=1,padding='same'),### MaxPooling2D layer
  tf.keras.layers.Conv2D(filters=128,kernel_size=3, strides=1, activation='relu'),### Conv2D layer
  tf.keras.layers.MaxPooling2D(pool_size=(2, 2), strides=1, padding='same'),### MaxPooling2D layer
  tf.keras.layers.Flatten(),### Flatten
  tf.keras.layers.Dense(64, activation='relu'), ###Dense
  tf.keras.layers.Dense(5, activation='softmax'),### Dense softmax
])

```

In CNN algorithm, we built a Sequential model using ReLU activation function except the last layer. In the output layer we used softmax function since we are doing a multiclass classification.

Since we have 10000 images and they all have really big sizes, epochs took a really long time. Overall the accuracy was between 0.59 and 0.61 range.

In the baseline model, we did not implement any preprocessing methods or data augmentation.

Improving the model

- Preprocessing
  The image data has files in various sizes and some of them have black borders. In order to focus on the image more properly and to not waste our time while processing the irrelevant parts of images, we need to get rid of the black borders. We used *PythonImageLibrary (PIL)* for cropping the images.

  We resized the images to (256, 256) size to evaluate our model faster. Preprocessing affected performance positively, making the training process work faster.

- Data augmentation
  We need data augmentation to increase the amount of data by using methods like flipping, zooming, rotating etc. This technique makes the model more robust to slight variations hence prevents the model from overfitting.

  We applied this technique by using ImageDataGenerator class from Keras because it does not store the augmented data in memory which is an efficient and practical way. ImageDataGenerator generates batches of image data with real-time augmentation.

```
train_datagen = ImageDataGenerator(rescale=1./255, validation_split=0.25,
                                   rotation_range=20, width_shift_range=0.1, height_shift_range=0.1, shear_range=0.1, zoom_range=0.2,
                                   horizontal_flip=True, vertical_flip=True)

```

- Assigning class weights for controlling imbalanced data
  We observed that the data has more samples for class 2. Since our model will have more chance to observe characteristics of class 2, it will be prone to overfitting. Hence we need to assign weights to our class labels so that the cost function penalizes loss on the majority class more heavily. This may make the model adapt better to characteristics of minority class. Theoretically, this needs to increase the accuracy but it was not the case for our model. (Implementation of it can be seen in the notebook)

- Adding new layers to model, increasing the number of epochs
  We tried various models for implementing CNN. We added and removed layers from the model. These kinds of operations did not affect our accuracy. Increasing the number of epochs increased the training accuracy since it meant more time to learn the weights but it did not have a significant effect on the test dataset. High numbers of epochs can cause overfitting so we should keep it in a small range.

- Transfer learning (epoch range = (10,20))
  We tried ResNet50 as the pre-trained model. By using this model, we used pre-trained weights that were learned with “imagenets” dataset. We adapted the model to our own task by adding extra layers at the end and training them with our dataset.

Transfer learning improved our training accuracy to 76%. However, validation accuracy was 54% which is lower than our baseline's accuracy. This difference between accuracy and validation accuracy is a significant indicator of overfitting. **To overcome the overfitting,** we added *dropout rate as 0.5* to our model. Furthermore, we added *early stopping* which monitors validation loss and stops the model if it tends to overfit. We also *decreased the number of nodes of fully connected layers* to 256, 128 from 1024,1024. We used the *preprocessing function* which is defined in the ResNet50 library and removed rescaling in ImageDataGenerator. We added *kernel regularizers* to the extra layers that we implemented in order to keep the weights in a reasonable range. For the regularized model, we noticed that it starts overfitting in the same epoch as in the baseline model. But the loss increases much more slowly afterwards.These techniques lowered the difference between training and validation accuracy by decreasing training accuracy but the validation/testing accuracy did not increase.

In conclusion, after adding all these techniques including transfer learning, even though our model has a slightly better training accuracy than our baseline model we could not increase the validation accuracy.

We could not increase the accuracy by using the ResNet50 model, so we switched our model to VGG16. Implemented the model with the same properties but it also did not change the result.

References

- https://towardsdatascience.com/transfer-learning-for-image-classification-using-keras-c47ccf09c8c8
- https://keras.io/guides/transfer_learning/
- https://medium.com/@vijayabhaskar96/tutorial-on-keras-flow-from-dataframe-1fd4493d237c
- https://towardsdatascience.com/vggnet-vs-resnet-924e9573ca5c
- https://towardsdatascience.com/exploring-image-data-augmentation-with-keras-and-tensorflow-a8162d89b844
- https://machinelearningmastery.com/weight-regularization-to-reduce-overfitting-of-deep-learning-models/
- https://elitedatascience.com/overfitting-in-machine-learning#how-to-detect
- https://www.kaggle.com/suniliitb96/tutorial-keras-transfer-learning-with-resnet50
- https://keras.io/api/layers/regularizers/
- https://towardsdatascience.com/handling-overfitting-in-deep-learning-models-c760ee047c6e
- https://www.kaggle.com/sid321axn/step-wise-approach-cnn-model-77-0344-accuracy