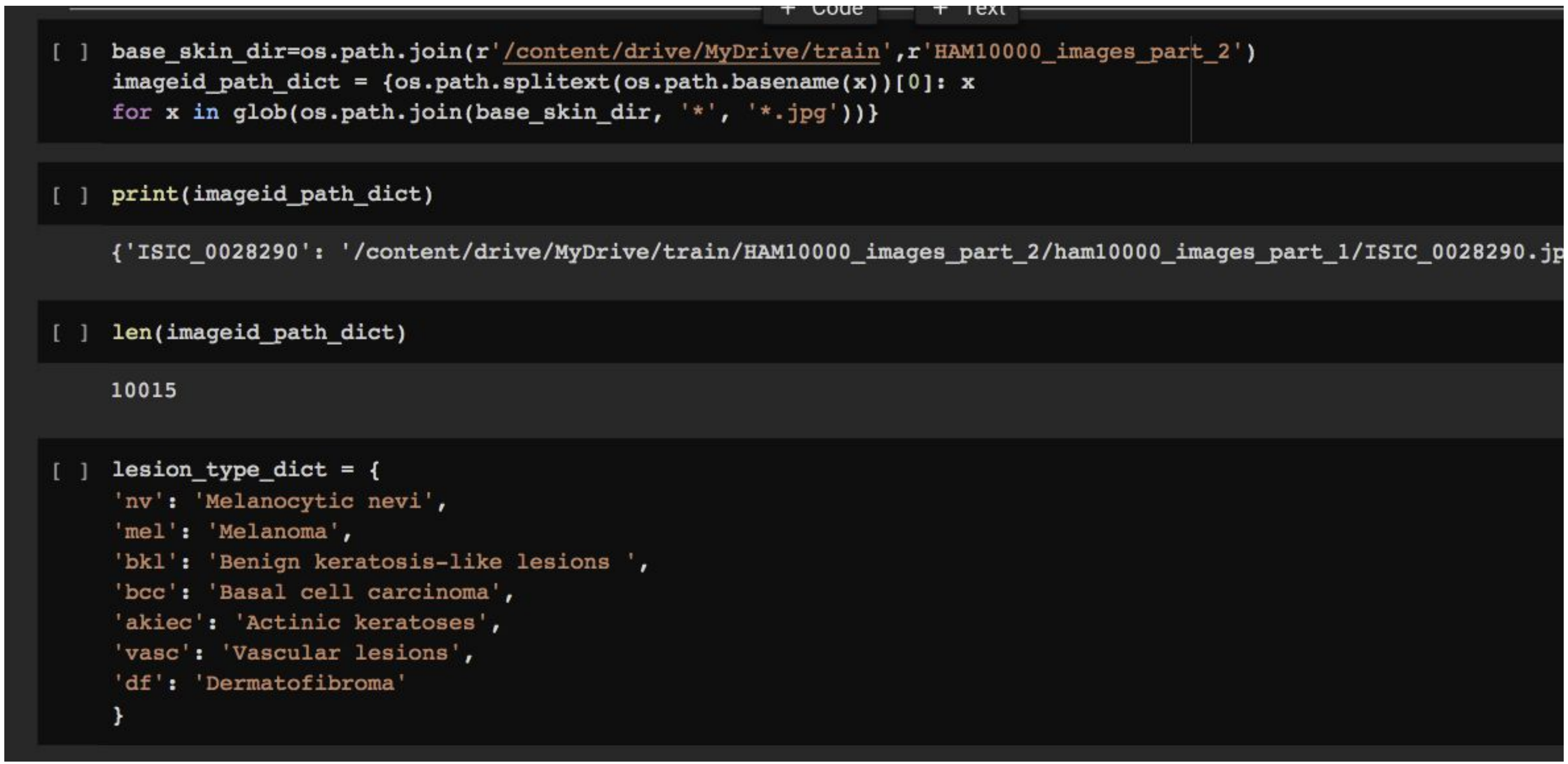
SKIN CANCER DETECTION

Group-13

13.Mohammed Anees
15.Mohammed Ashnab
19.Mohammed Rasif
21.Muhamed Mohsin

Making Dictionary of Images and also Labels

There are two folders in our dataset. Merging images from both folders into one dictionary as required. Labelling seven types of Skin Cancer. This dictionary is used for displaying labels later on. It is very human friendly.



```
[ ] base_skin_dir=os.path.join(r'/content/drive/MyDrive/train',r'HAM10000_images_part_2')
    imageid_path_dict = {os.path.splitext(os.path.basename(x))[0]: x
    for x in glob(os.path.join(base_skin_dir, '*', '*.jpg'))}
```

```
[ ] print(imageid_path_dict)
```

```
{'ISIC_0028290': '/content/drive/MyDrive/train/HAM10000_images_part_2/ham10000_images_part_1/ISIC_0028290.jp
```

```
[ ] len(imageid_path_dict)
```

```
10015
```

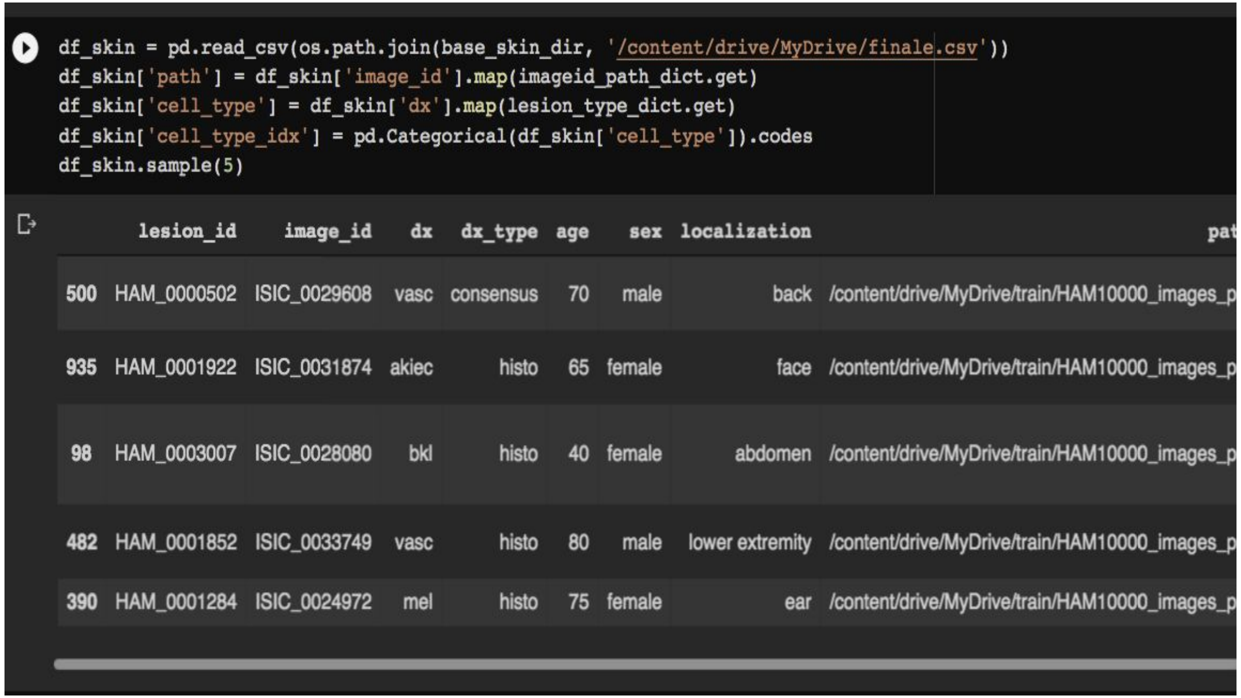
```
[ ] lesion_type_dict = {
    'nv': 'Melanocytic nevi',
    'mel': 'Melanoma',
    'bkl': 'Benign keratosis-like lesions ',
    'bcc': 'Basal cell carcinoma',
    'akiec': 'Actinic keratoses',
    'vasc': 'Vascular lesions',
    'df': 'Dermatofibroma'
    }
```

Reading the Data and Processing

To join the image folder and the csv folder a path is created.

Also creating new columns for organization of data and better readability.

Sample of newly made columns using sample extension is shown here.



```
df_skin = pd.read_csv(os.path.join(base_skin_dir, '/content/drive/MyDrive/finale.csv'))
df_skin['path'] = df_skin['image_id'].map(imageid_path_dict.get)
df_skin['cell_type'] = df_skin['dx'].map(lesion_type_dict.get)
df_skin['cell_type_idx'] = pd.Categorical(df_skin['cell_type']).codes
df_skin.sample(5)
```

```
     lesion_id     image_id      dx  dx_type    age  sex     localization     pat
500  HAM_0000502   ISIC_0029608  vasc  consensus  70  male    back             /content/drive/MyDrive/train/HAM10000_images_p
935  HAM_0001922   ISIC_0031874  akiec  histo     65  female  face             /content/drive/MyDrive/train/HAM10000_images_p
98   HAM_0003007   ISIC_0028080  bkl   histo      40  female  abdomen          /content/drive/MyDrive/train/HAM10000_images_p
482  HAM_0001852   ISIC_0033749  vasc  histo      80  male    lower extremity  /content/drive/MyDrive/train/HAM10000_images_p
390  HAM_0001284   ISIC_0024972  mel   histo      75  female  ear              /content/drive/MyDrive/train/HAM10000_images_p
```

Loading and adjusting pretrained models in pytorch

Here we are going to feature extract torchvision model. Due to pytorch, a well established models are obtained. We are using model resnet50. Here we updating all models parameters from our new task. Loading the pretrained resnet50 and adjusting the last layer is for feature extraction. We update only last layer weights from which we derive predictions. It is called feature extraction because we use the pretrained CNN as a fixed feature-extractor, and only change the output layer. We can see that the last layer is a linear layer, having 2048 input neurons and having 1000 output neurons.This is useful if you have 1000 different classes. However, we only have to deal with 7 different classes - the 7 different skin cancer types - therefore we have to change the last layer. Load model on the GPU, because that is the place where we want to train the model in the end.

Split the data

The data is being split into training and validation sets. The Dataset consists of 10,015 images. This is the step where the dataset will be split into training data, testing data and validation data. Training dataset is a sample of data used to fit the model which is the actual dataset that we use to train the model.

Train the model

In the proposed work we set the number of iterations i.e., epochs as 40. We calculate the training error and testing error after training the model per epoch. Initially it will be zero and so on. So basically, after 100 images are passed, the error will be appended to training error. It will be stored in temporary. Mean is computed, average of 100 images are stored in training error. Then we train the model where the model will learn from the image i.e., pixels and value of the image. We will store it in a CPU. We get the predicted as well actual output. Criterion is used to calculate the difference between the predicted and the actual output. We apply backward propagation and optimizer to make the model better over time. Same steps are performed for validation error. Plot the graph based on the error generated.

Test the actual classification ability of the model

The proposed model is tested on the testing data set for accuracy.

We will evaluate the model again. Test generator will be created

and we have two have two models, one is predicted and other one

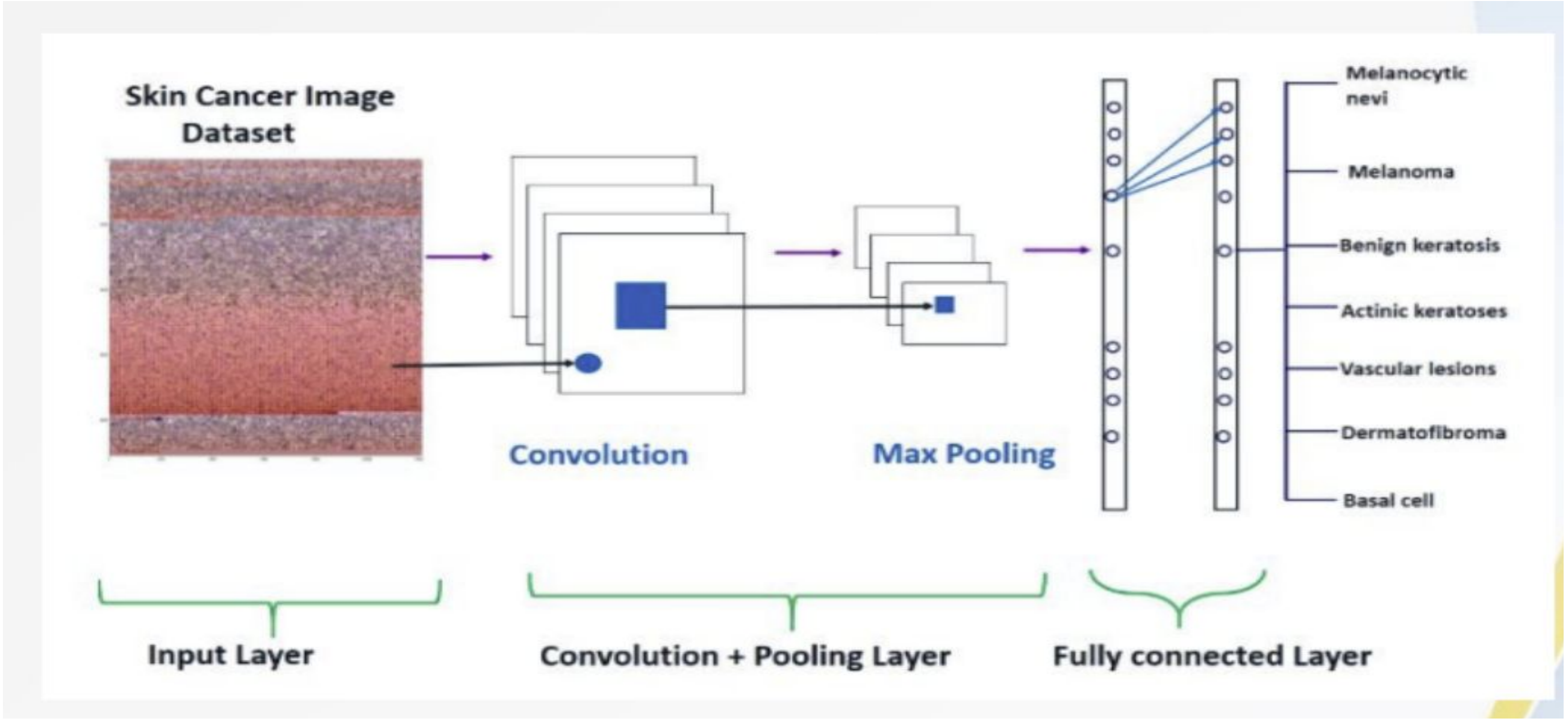
is actual output. The probability of 7 values associated with each class, any one

will have high probability. This will be stored in the result.

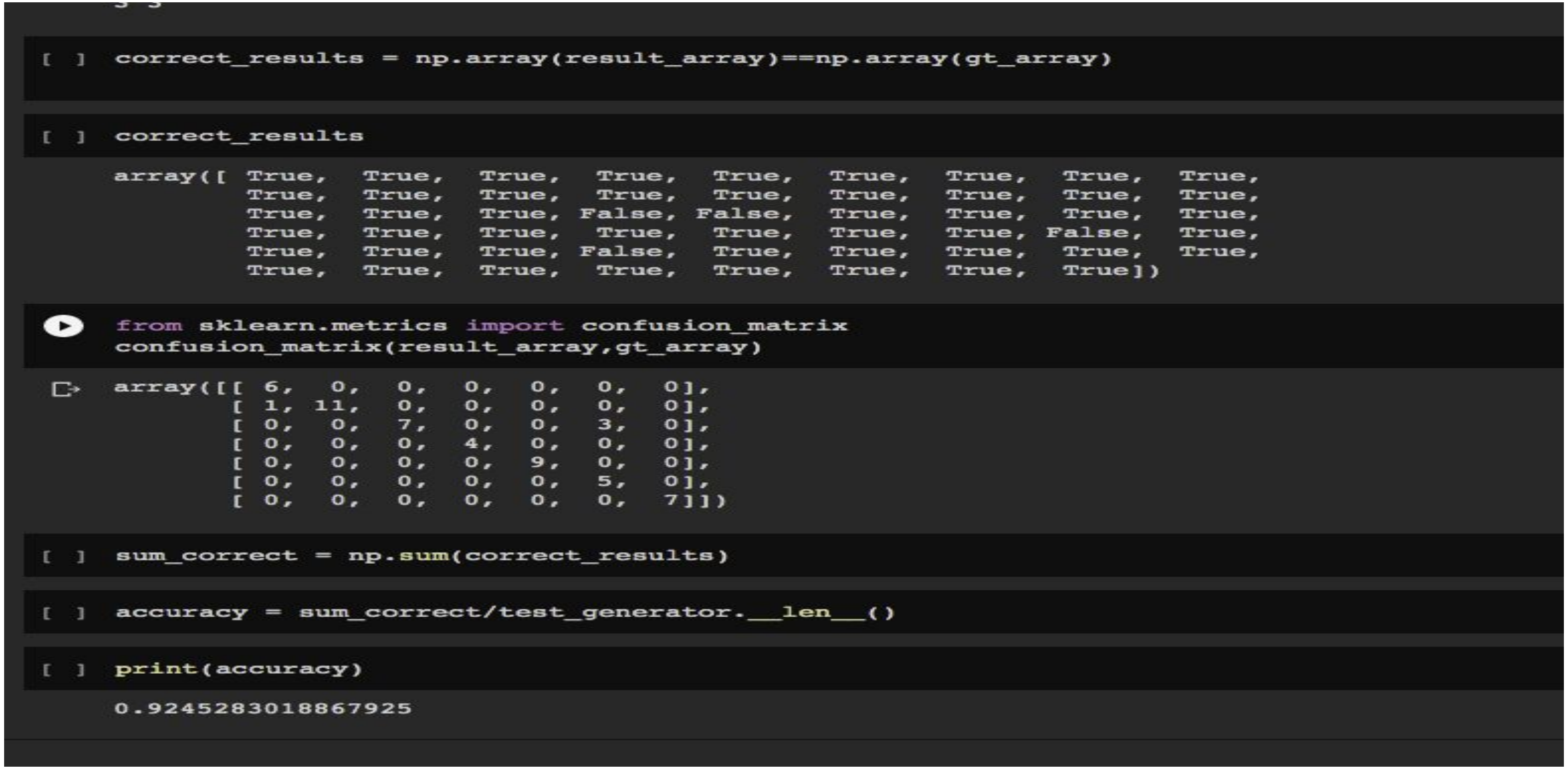
Check both the predicted as well the actual value and assign if

true if matching else false. Creating a confusion matrix for the

above. The proposed model obtained 92% accuracy rate.



CNN applied on a Skin image



```
[ ]  correct_results = np.array(result_array)==np.array(gt_array)
```

```
[ ]  correct_results
```

```
     array([ True,  True,  True,  True,  True,  True,  True,  True,  True,
             True,  True,  True,  True,  True,  True,  True,  True,  True,
             True,  True,  True, False, False,  True,  True,  True,  True,
             True,  True,  True,  True,  True,  True,  True, False,  True,
             True,  True,  True, False,  True,  True,  True,  True,  True,
             True,  True,  True,  True,  True,  True,  True,  True])
```

```
     from sklearn.metrics import confusion_matrix
     confusion_matrix(result_array,gt_array)
```

```
     array([[ 6,  0,  0,  0,  0,  0,  0],
            [ 1, 11,  0,  0,  0,  0,  0],
            [ 0,  0,  7,  0,  0,  3,  0],
            [ 0,  0,  0,  4,  0,  0,  0],
            [ 0,  0,  0,  0,  9,  0,  0],
            [ 0,  0,  0,  0,  0,  5,  0],
            [ 0,  0,  0,  0,  0,  0,  7]])
```

```
[ ]  sum_correct = np.sum(correct_results)
```

```
[ ]  accuracy = sum_correct/test_generator.__len__()
```

```
[ ]  print(accuracy)
```

```
     0.9245283018867925
```

Accuracy obtained from the model