Machine Learning Engineer Nanodegree

Skin Cancer Prediction using Deep Learning

Gaurav Rajput

February 6, 2020

I. Definition

Skin cancer is the most deadly type of cancer which directly affects the skin of the patient. This deadly type of cancer is on the rise beacause of the ozone layer depletion. There is 50 percent rise in the cases of skin cancer all over the world.

With the rise in the cases of skin cancer it is necessary to find the computer-aided solution instead of the manual method of analysing the patterns in the skin moles by the experts, which is very tedious and time-consuming process. If machine learning and image processing are used in the solution, we can make this process quick and easy.

Project Overview

To build a system for skin cancer prediction that will predict which type of skincancer the patient have with the probability also shown by each type of cancer.

To automate the process of skin cancer detection, this project aims to develop a deep convolutional neural network . As the transfer learning will not require heavy computational power to achieve the better performance.

The deeper CNN modelswill give much more better performance compared to shallower models. Using Residual Network models like ResNet-50 for trasfer learning can save us alot of computational power for the training of the model.

Problem Statement

The goal is to create a simple application that will help the specialists find the exact skin cancer types

- To successfully find a way to balance out the unbalanced data.
- To train a model to classify the skin cancer types.

Metrics

For the evaluation of the model for multi-class classification, accuracy of the model is most useful metrics.

Apart from that the F-score is also the most useful metrics for the unbalanced dataset.

	Predicted Positive	**Predicted Negative**
Reference Positive	TP	FN
Reference Negative	FP	TN
	$\text{Precision } (\mathbf{P}) = \frac{TP}{TP+FP}$	
	$\text{Recall } (\mathbf{R}) = \frac{TP}{TP+FN}$	
	$\text{Overall Accuracy} = \frac{TP+TN}{TP+FP+TN+FN}$	
	$\mathbf{F} - \text{score} = 2 \times \frac{R \times P}{R+P}$	

II. Analysis

Data Exploration

The data for this problem statement can be easily acquired from ISIC archives, in case if extra data is needed.

Apart from that the data can also accessed from Kaggle Dataset.

The above dataset has 10015 images of 7 types of skin cancer which are

- Benign Keratosis
- Melanocytic Nevi
- Dermatofibroma
- Melanoma
- Vascular Lesions
- Basal Cell Carcinoma
- Actinic Keratoses

Apart from the images, the dataset also includes the metadata in form of CSV file, which gives information about the images in the dataset.

The CSV file has 7 columns which are

- Lesion Id:

> Lesion Id is the unique Id of evry skin lesion. Some images in the dataset are augmented images of the same lesions.

- Image Id:

> Image id is a unique id of every image.

- dx:

> It signifies the type of the skin cancer in the image.

- dx type:

> it is the type of the skin cancer cell.

- Age:

> It is the age of patient who have the skin cancer.

- Sex:

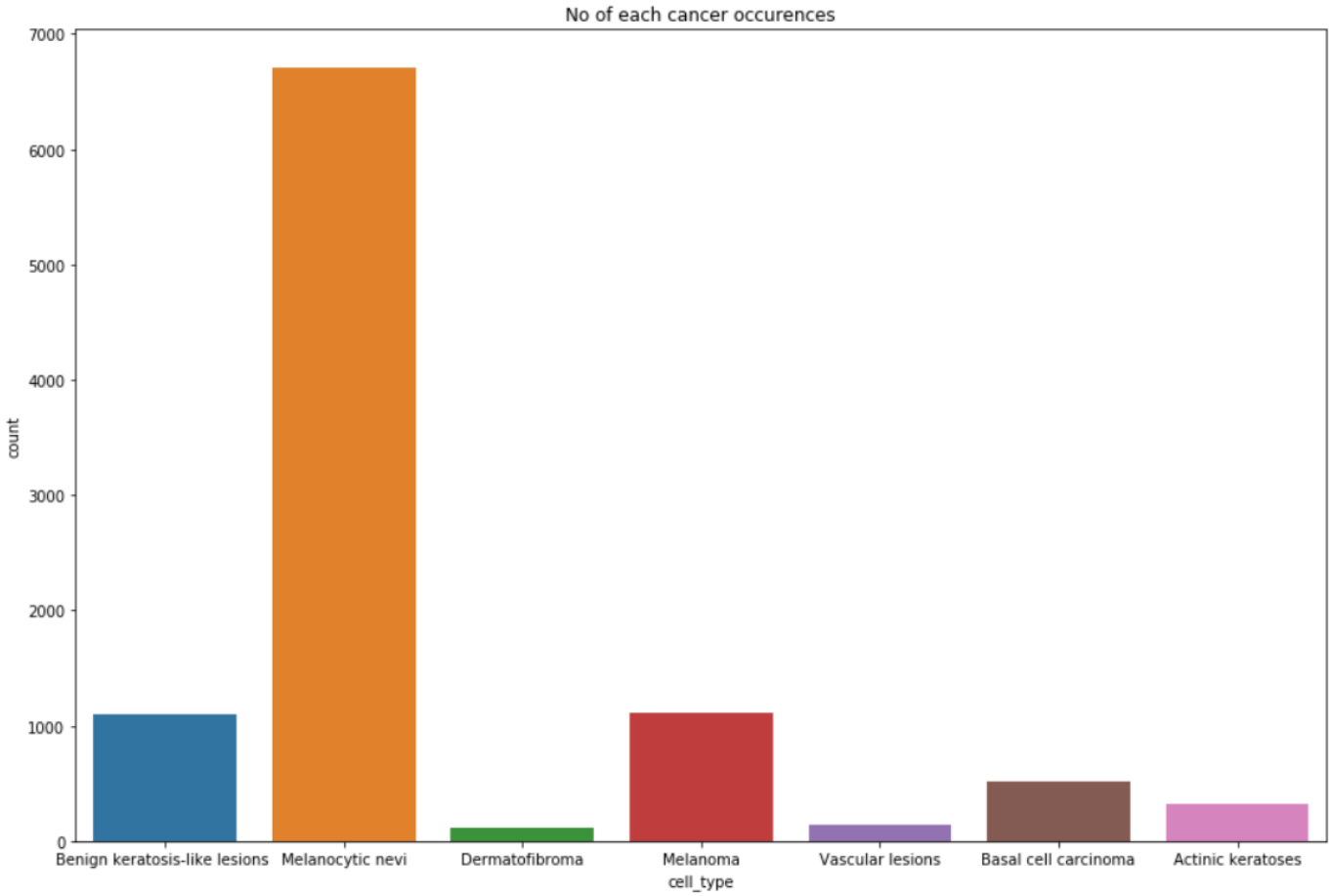
> It is the gender of the patient.

- Localization

> It states the locality on the body where the lesion is there.

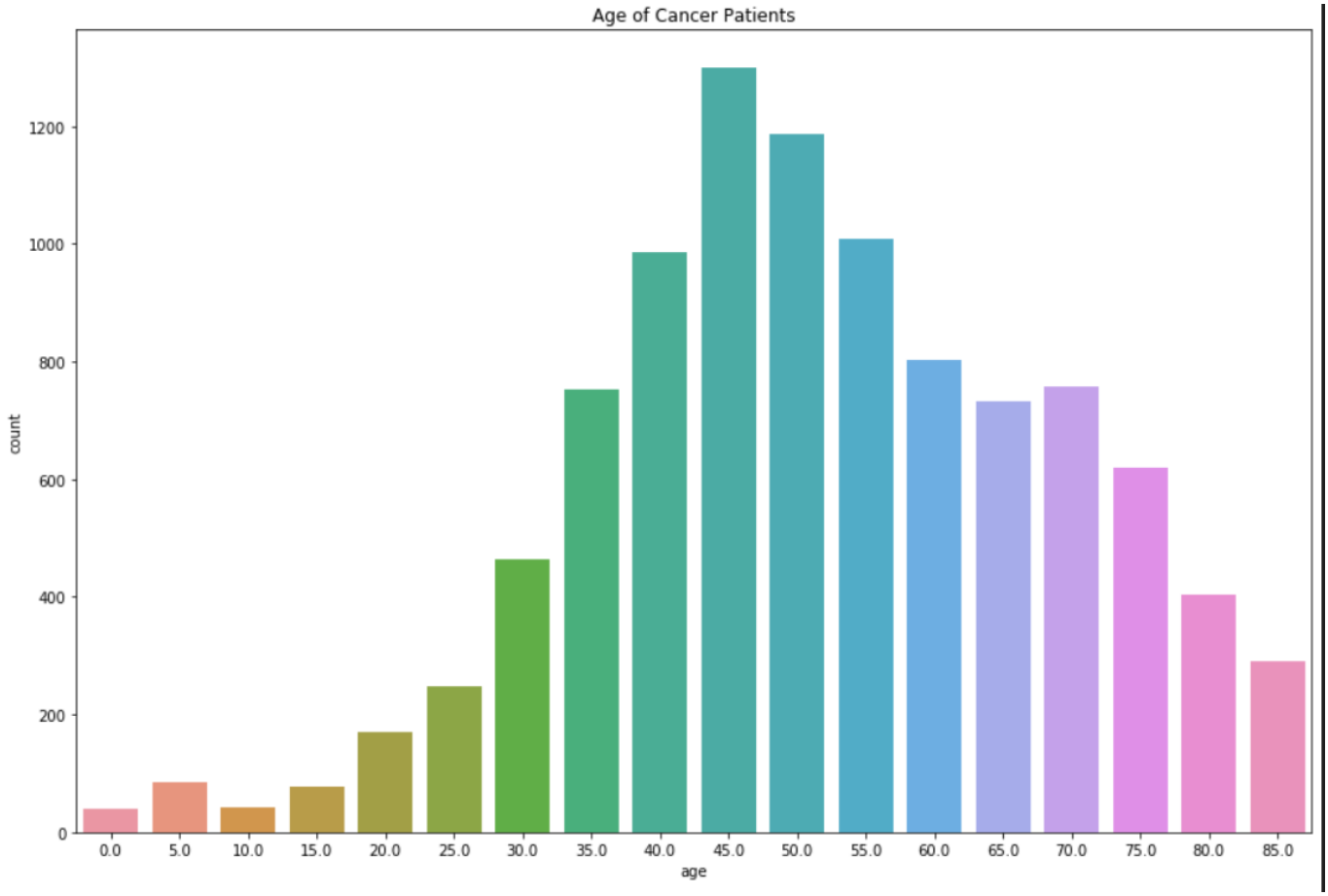
Exploratory Visualization

In the above dataset the images of Melanocytic Nevi have more than 60% of the images. Dermatofibroma and Vascular Lesions have just a 100s of images.



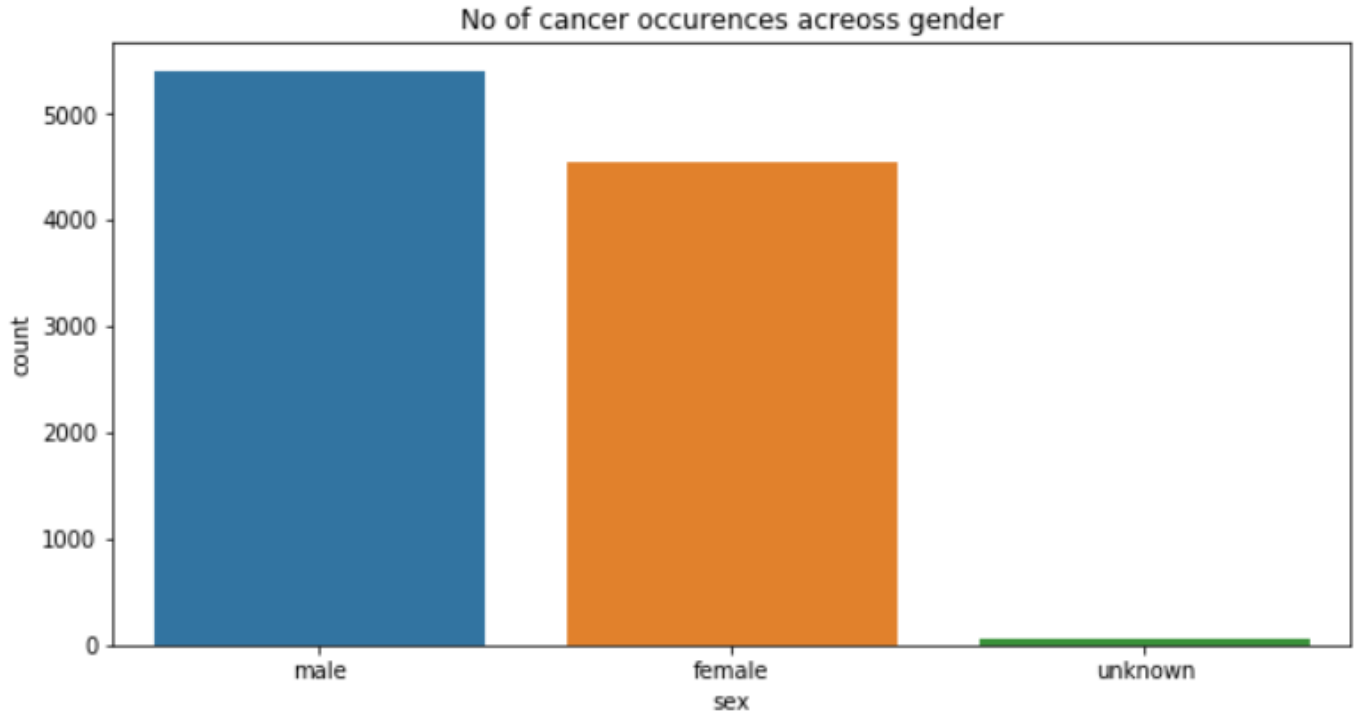
Above scenario tells that the dataset is highly skewd towards one specific type of cancer. This Issue of imbalanced data have to be solved to get better results or there is possibility that model will only learn to classify one specific type of cancer.

The age distribution of the skin cancer patients is like a normal distribution. The patient with age closer to 40 have most number of skin cancer cases.



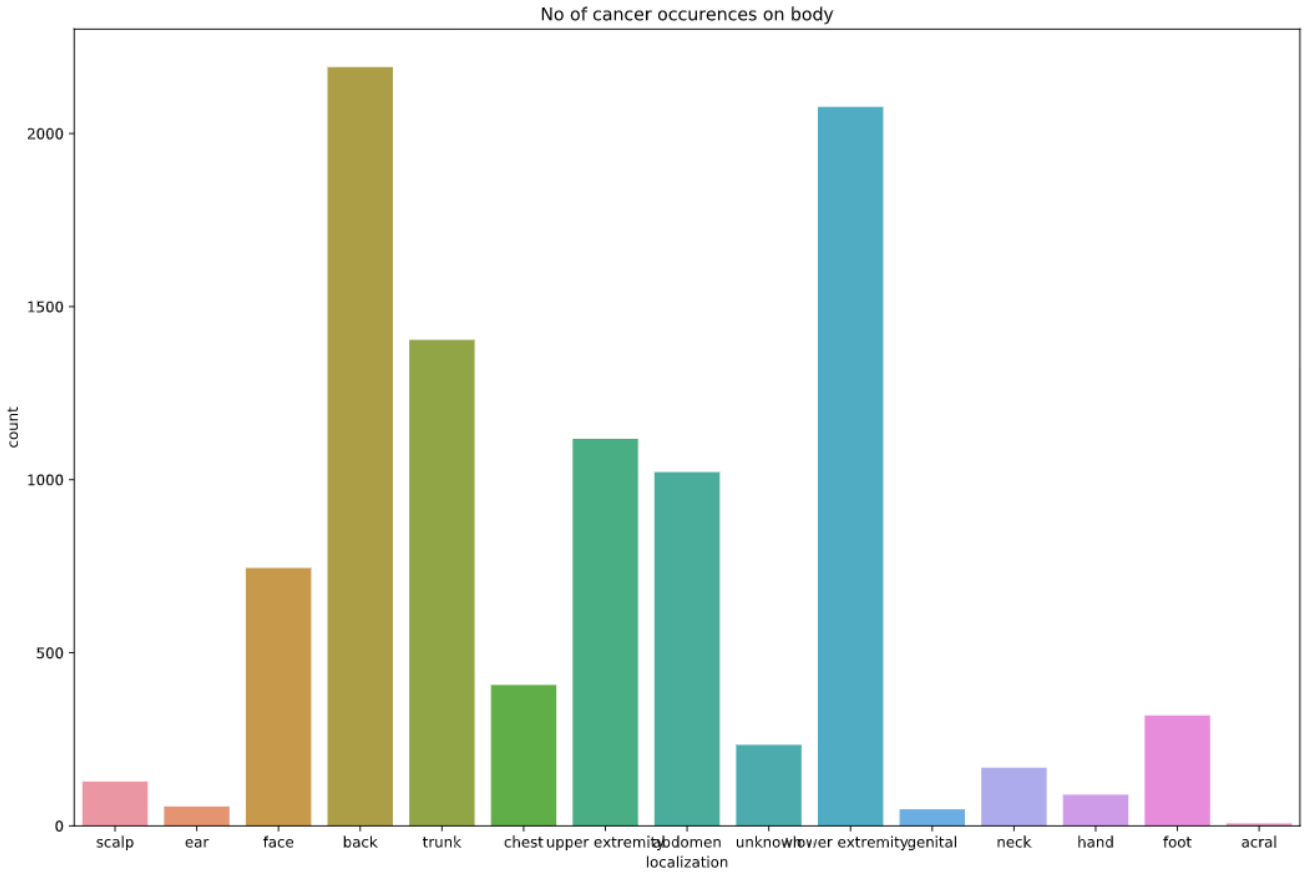
There were some cases where there was missing data of age which were replaced with the mean age.

The cases of skin cancer along the gender can be shown in below image



As is shown in above image, the no of cases are slightly higher in males compared to females. There are some missing cases.

The Location of skin Lesions is concentrated in some parts of the body like back and lower extermity.



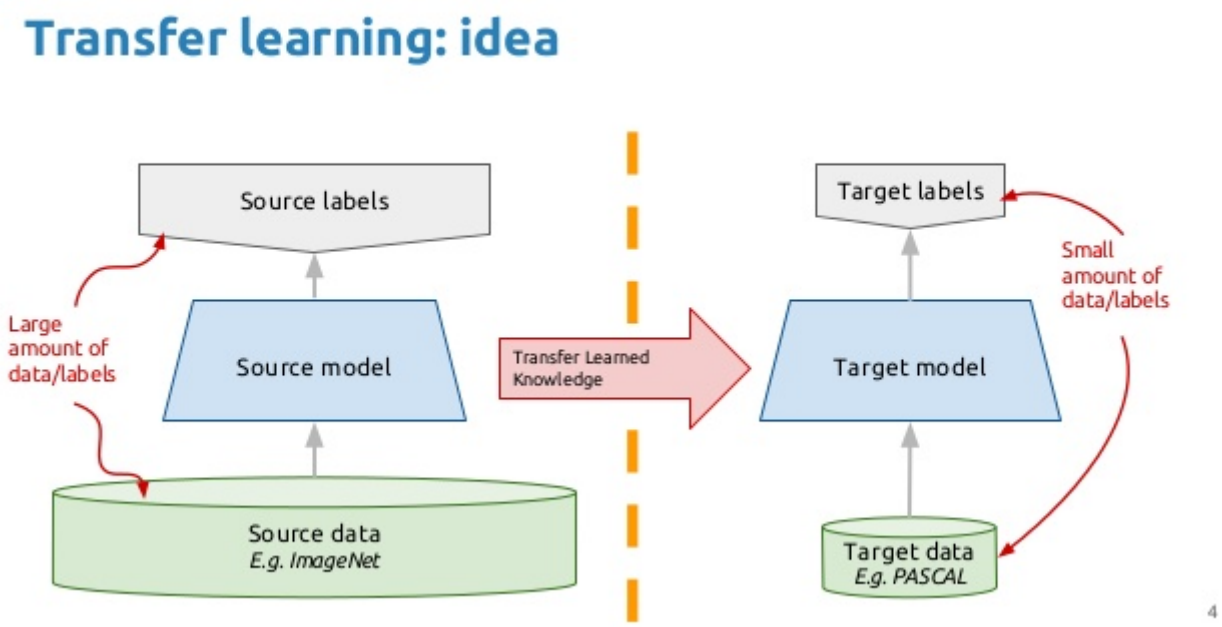
Algorithms and Techniques

- Image Preprocessing

> - Image Preprocessing must be done on the images in the dataset before the actual training. Various preprocessing like Image Resizing, conversion to tensor and normalization.
> - Various other strategies like mole segmentation, skin color normalization, brightness normalization can also be experimented after successful implementation of above techniques.
> - The dataset is imbalanced to a specic class that problem can be solved by using data upsampling in which the minority data is scaled up to the data of maximum levels, using the data upsampling with combination with the data augmentation can help in the better performence of the model.

- Classification Model

> - The above problem can be most efficintly solved by using Convolutional Neural Network, rather than other machine learning approach like SVM, Naive Bayes algorithm or Random Forest Trees. These models lack the complexity to handle the diverse data like images.
> - When we use our own custom trained CNN models, they also have their own set of problems as the deep CNN models need lots of computational power for the training of these models.
> - To circumvent these problems, the best approach is using transfer learning approach. By using the models trained on diverse dataset like ImageNet, we can get very good performence which we can again enhance by training the pretrained model on the task specific dataset.



Benchmark

The performance of above model will be compared with the 2016 paper "Melanoma Detection by Analysis of Clinical Images Using Convolutional Neural Network" by Nasr Esfahani. In the above paper the author have used a 2 layer deep CNN model to achieve the task. The model got accuracy of 0.81 percent.

III. Methodology

Data Preprocessing

Before the actual task of training a classifier model, the data needs to be preprocessed. The data is needed to be braught in specific format before feeding the data to the models. First of all we created new csv file in which added additional columns like `cell_type_idx`, `path`, and `duplicates`

- cell_type_idx

> This column has unique number associated with every type of skin cancer.

- path

> This column has the path of the image. As every image has its unique image id, we used that image id for getting the pathnof image.

- duplicates

> This column describes if there are multiple image of single lesion id. If there are multiple images of single lesion then that image is `duplicated` else `original`. We have prepared this column to make sure that our validation set do not have images with common lesion id.

After the above process, we split our data into 70:30 format for training and validation set which will be used for the training of the model. We have created two different csv files for the training and validation set. We are using this approach as it will be easy to load and transform our data.

To solve the problem of imbalanced data, we have used upsampling for the minority classes, by using upsampling our dataset becomes of size 28K images. The above process is done in `Data-Exploration.ipynb` notebook

In the `Model Creation.ipynb` notebook, the actual data augmentation and transformation is done.

We have created custom dataset to load the above dataset. Below is the code.

```
class HamDataset(Dataset):
    def __init__(self, csvpath, transform = None):
        self.df = pd.read_csv(csvpath)
        self.transform = transform

    def __len__(self):
            return len(self.df)

    def __getitem__(self, index):
        X = Image.open(self.df['path'][index])
        y = torch.tensor(int(self.df['cell_type_idx'][index]))

        if self.transform:
            X = self.transform(X)

        return X, y
```

To load the specific dataloader to be loaded for the pytorch model we are using the specific function.

```
def get_data_loader(csvpath, transform = test_transform, batch_size = 32):
    if csvpath == 'train.csv':
        transform = train_transform
    csvpath = os.path.join(data_dir, csvpath)
    dataset = HamDataset(csvpath, transform)
    dataloader = DataLoader(dataset, batch_size=batch_size, shuffle=True,
num_workers=0)

    return dataloader
```

The transformations used for the data for normalizing, regularizing and augmenting the the data we are using data transformations.

In below transformations for training dataset, images are being resized into required size before the actual training purpose.

The images are being ranomly flipped into horizontal and vertical way, are randomly rotated, there brightness, contrast and hue are changed by factor of 0.1.

Then they are being converted into tensor and then normalized into predefined standard format according to ImageNet dataset.

```
norm_mean = [0.485, 0.456, 0.406]
norm_std = [0.229, 0.224, 0.225]

train_transform = transforms.Compose([transforms.Resize(input_size),
                                      transforms.RandomHorizontalFlip(),
                                      transforms.RandomVerticalFlip(),
                                      transforms.RandomRotation(20),
                                      transforms.ColorJitter(brightness=0.1,
contrast=0.1, hue=0.1),
                                      transforms.ToTensor(),
                                      transforms.Normalize(norm_mean, norm_std)])

test_transform = transforms.Compose([transforms.Resize(input_size),
                                     transforms.ToTensor(),
                                     transforms.Normalize(norm_mean, norm_std)])
```

Implementation

For the training of the actual model we have used mainly two pretrained model which are ResNet50 and DenseNet121 model.

We have used `cuda` device for the training of the algorithm to parallelize the training algorithms.

For DenseNet121 Model, we are using the ImageNet weights for the inference. Before training the model we have freezed its base layers and only training the final layer for inference.

We are using CrossEntropyLoss for loss calculation and we have used Adam optimizer with default parameters.

We have used ReduceLROnPlateau learning rate scheduler as keeping the LR constant does not have any effect on the validation loss after some epochs.

The hyperparameters of the learning rate scheduler are set as this:

```
mode : 'min',
patience : 3
verbose = True
```

These hyperparameters mean that if there is no change in loss of the model for last 3 epochs then learning rate will be decreased by factor of 0.1. The initial Learning rate is 0.001.

The batch size of the dataset is set at 32 which means the change in the model weights will happen after the iteration of 32 datapoints or images.

The training script can be seen below:

```
def train_model(model, criterion, optimizer, scheduler, num_epochs=25, best_acc =
0.0):
    since = time.time()

    val_acc, val_loss = [],[]
    train_acc, train_loss = [],[]
    lr_rate = []

    best_model_wts = copy.deepcopy(model.state_dict())
#     best_acc = 0.0

    for epoch in range(num_epochs):
        print('Epoch {}/{}'.format(epoch, num_epochs - 1))
        print('-' * 10)

        # Each epoch has a training and validation phase
        for phase in ['train', 'val']:
            if phase == 'train':
#                 print('phase train')
                for param_group in optimizer.param_groups:
                    print('Learning Rate {}'.format(param_group['lr']))
                    lr_rate.append(param_group['lr'])
                model.train()  # Set model to training mode
            else:
#                 print('phase val')
                model.eval()   # Set model to evaluate mode
```

```
            running_loss = 0.0
            running_corrects = 0

            # Iterate over data.
            for inputs, labels in tqdm(dataloader[phase]):
                inputs = inputs.to(device)
                labels = labels.to(device)

                # zero the parameter gradients
                optimizer.zero_grad()

                # forward
                # track history if only in train
                with torch.set_grad_enabled(phase == 'train'):
                    outputs = model(inputs)
                    _, preds = torch.max(outputs, 1)
                    loss = criterion(outputs, labels)

                    # backward + optimize only if in training phase
                    if phase == 'train':
                        loss.backward()
                        optimizer.step()

                # statistics
                running_loss += loss.item() * inputs.size(0)
                running_corrects += torch.sum(preds == labels.data)

            del inputs
            del labels
            torch.cuda.empty_cache()

            if phase == 'train':
                scheduler.step(loss)

            epoch_loss = running_loss / size[phase]
            epoch_acc = running_corrects.double() / size[phase]

            if phase == 'train':
                val_loss.append(epoch_loss)
                val_acc.append(epoch_acc)
            else:
                train_loss.append(epoch_loss)
                train_acc.append(epoch_acc)

            print('{} Loss: {:.4f} Acc: {:.4f}'.format(
                phase, epoch_loss, epoch_acc))

            # deep copy the model
            if phase == 'val' and epoch_acc > best_acc:
                best_acc = epoch_acc
                best_model_wts = copy.deepcopy(model.state_dict())

    time_elapsed = time.time() - since
    print('Training complete in {:.0f}m {:.0f}s'.format(
```

```
            time_elapsed // 60, time_elapsed % 60))
        print('Best val Acc: {:4f}'.format(best_acc))

        # load best model weights
        model.load_state_dict(best_model_wts)
        return model, val_loss, val_acc, train_loss, train_acc, best_acc
```

The Best model weights are preserved to get just the best of the performence of the model. The list of various metrics are returned to get visualization and analysis of training process.

The model is trained for 25 epochs.

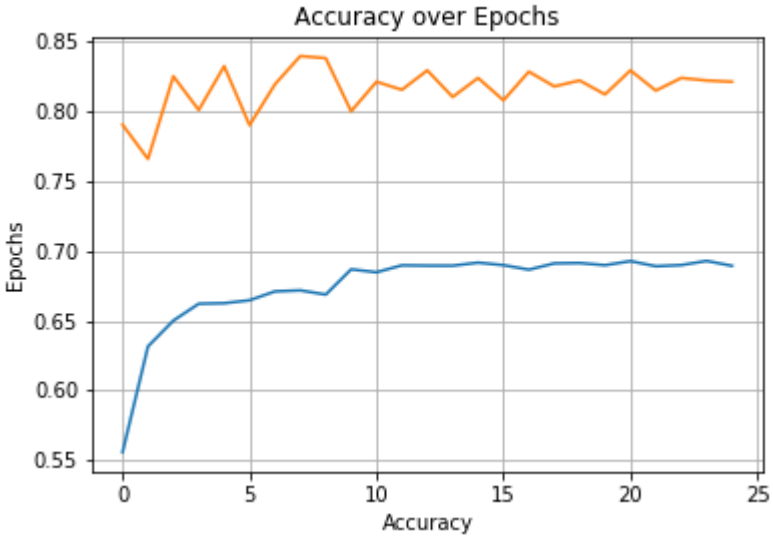
As seen in the script, the useless variable are deleted, so that the performence and the memory of the cuda device is optimized.

IV. Results

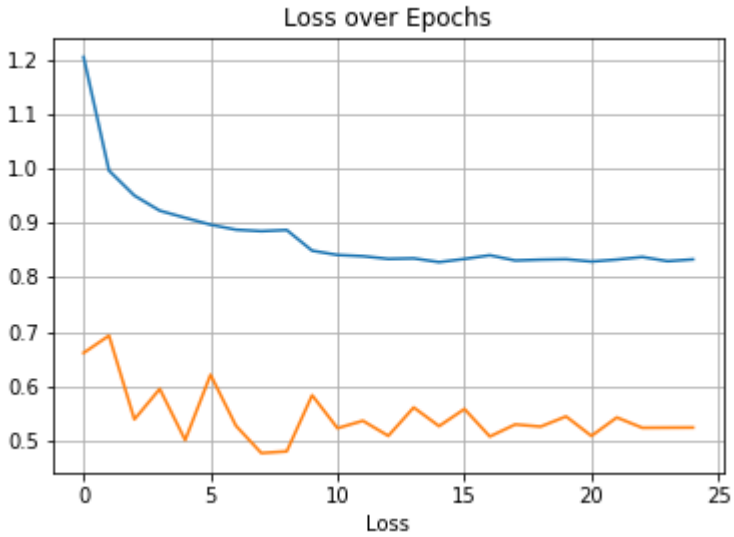
Model Evaluation and Validation

The model after training for 25 epochs, gave validation accuracy of 84% and validation loss of 0.5, which can be further be improved by training and experimenting on more hyperparameters.

The below graph of accuracy tells us about accuracy over epochs



Below is the graph of loss over epochs



As seen in the above twor graphs, the graph is flattening out after some number of epochs and then again decreasing slightly, that is because of the Learning Rate scheduler. Using and experimenting with the LRscheduler can give us better results.

It took about 8 hours to train the model for 25 epochs. Without any finetuning or any hyperparameter tuning the model is giving 84% of accuracy. Hyperparameter tuning and finetuning can give us better results, experimenting with other pretrained models can give us another perspective to the solution.

Justification

Compared with the previous benchmark, the results of this model much better considering the fact that there were very little efforts in tuning the hyperparameters and without any finetuning.

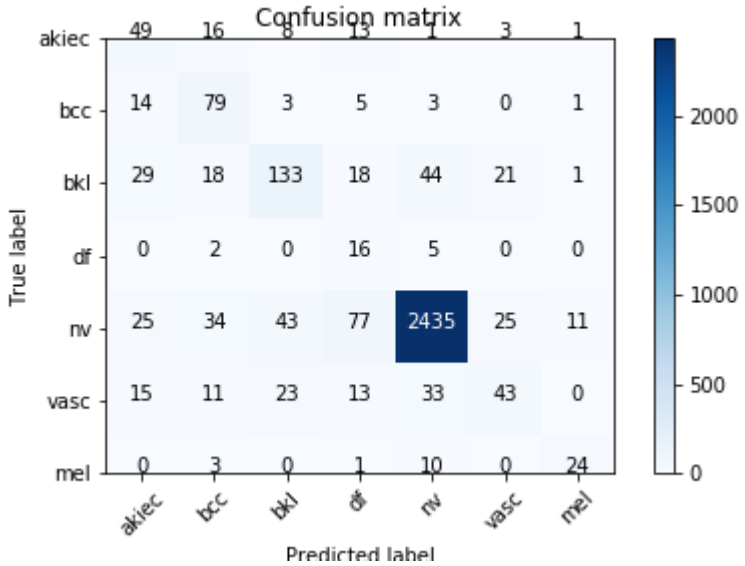
We solved the problem of data imbalance by upsampling the data and augmented the data. By finetuning our model we can make our model much more better, but that is not possible beacause of hardware constraints.

There are some better benchmarks than this, but they dont have any data or hardware constraints as well as this solutions are using ensemble type of solutions, which is not very practical way to solve this problem, as again there are hardware constraints in the real worls as well.

V. Conclusion

Free-Form Visualization

The model is able to complete its objective of classification of the skin cancer lesions with good accuracy of 84% on test set. But there is very much room for improvement as the above loss graph shows. But considering the hardware and data constraints, this performence is also very good.



Because there is a data imbalance in the test set, above confusion matrix is not very good matrix for the evaluation of the model.

	precision	recall	f1-score	support
akiec	0.37	0.54	0.44	91
bcc	0.48	0.75	0.59	105
bkl	0.63	0.50	0.56	264
df	0.11	0.70	0.19	23
nv	0.96	0.92	0.94	2650
vasc	0.47	0.31	0.37	138
mel	0.63	0.63	0.63	38
accuracy			0.84	3309
macro avg	0.52	0.62	0.53	3309
weighted avg	0.87	0.84	0.85	3309

Above report shows a more balanced picture of the model. As above report shows, there is still room for improvement.

Reflection

In this project, the main problem statement was to classify the skin cancer lesions into 7 categories. To do that, there were some challenges like data imbalance, lack of data, lack of computational power. We tried to solve these issues by various means.

We tried to solve the problem of data imbalance and lack of data by using data upsampling in which we just increase data of minority classes, and on top of that we used data augmentation to solve lack of data problem.

To solve the problem of lack of computational power, we tried to use transfer learning to develop a classification model, and on top of that we tried to parallelize the training and evaluation process. Using this both methods, we tried to get most of the hardware and prexisting solutions.

Using this methods, our model was gave good performence for just 25 epochs without any need of the hyperparameter tuning.

Improvement

Th dataset is so much diverse in nature, this fact have its own advantages and disadvantages. Advantages were our model was able to learn practical and real life data. Cons of this fact is there were skin lesions of different skin colours, which made it difficult to get the right patterns of the data. Using various skin colour normalization process would improve the performence of the models.

Many solution on health related problems use some kind of ensemble models, as it is better to depend on multiple models rather than one. But this ensemble models can also improve the performence of the system.

Hyperparameter tuning and fine tuning can also give a good boost to the model's performence. Apart from that, using multiple datasets can also solve the problem of data imbalance and data constraints.
