Skin Cancer Classification and Segmentation using Machine Learning

Contents

1. Introduction

Skin lesions are characterized by distinct patches, abnormal bumps, pus filled sacs, scars, blisters or discoloration that is distinct from skin on normal areas of the body. This could be as mild as from a mosquito bite or as severe as it may occur as a symptom of an underlying disease like diabetes and various different carcinoma. This project explores the potential of artificial neural networks to design an automated diagnosis method for various skin lesions by making use of the HAM 10000 datasets which encompasses a huge dataset of dermatoscopic images that serve as a training dataset for academic machine learning purposes.
The final dataset consists of 10015 images with a 53.3% pathological verification. Among these 10015 images 327 datasets were detected for akiec, 514 for bcc, 1099 for bkl, 115 for df, 1113 for mel, 6705 for nv and 142 for vasc.

The following are the different types of skin lesions dealt in this project:
akiec:

Actinic keratoses(Solar keratoses) and Intraepithelial carcinoma(Bowen's Disease). Actinic keratoses are rough scaly patches on skin that develop from chronic exposure to sun. This is dermatoscopically characterized by a flat to slightly raised rough dry patch which is usually less than 2.5cm. This is completely curable if detected early and properly treated, however if left untreated can progress into squamous cell carcinoma. Thus early detection is required.

Bowen's disease is a very early form of skin cancer that is characterized by a red, scaly patch on the skin. This is caused by long term exposure to the sun and people with immunodeficiency are more prone to it. This can eventually develop to various forms of skin cancers like squamous cell skin cancer if left undiagnosed. Proper detection is important to seek required medical care to avoid unwanted complications.

bcc:

Basal cell carcinoma is a type of skin cancer which starts with the basal cells. This occurs mostly due to the long term exposure to UV light and in accordance with this it is found mainly in areas exposed to sunlight. Mutations are seen in the DNA of basal cells causing rapid proliferation and accumulation of abnormal cells. Apart from chronic sun exposure radiation therapy and use of immunosuppressive drugs are some of the risk factors for the same. The lesions are characterized by pearly white, skin colored or pink bumps with dark brown or blue spots, often with a raised edge or waxy appearance. This can lead to an increased risk of other types of skin cancer such as squamous cell carcinoma and may also spread to the nearest lymph node if left untreated hence timely detection and proper treatment is required.

bkl:

Benign keratosis includes seborrheic keratoses, solar lentigo and lichen-planus like keratoses. Seborrheic keratoses usually have brown black waxy looking appearance. It is more likely associated with some genetic factors although sun exposure is also thought to have an impact. At times it could be misunderstood with an actual melanoma which shows the importance of proper detection. Solar lentigo is a harmless patch of darkened skin by melanin accumulation due to UV exposure. It also raises a possibility of malignant lesions which should be detected and treated at the earliest to avoid further damages. Lesions of lichen-planus like keratoses appear abruptly as a solitary macule, papule, or plaque caused by an inflammatory reaction by the immune system attacking our skin. Exposure to UV radiation, skin irritation or some medications is considered as a trigger for this. Oral lichen planus can lead to oral cancer and those that affect ears can cause hearing impairment thus early detection and proper treatment is important.

df:

Dermatofibroma also called as cutaneous fibrous histiocytoma is a common benign fibrous nodule usually found on the skin of the lower legs and is more frequent in women. It is caused by an overgrowth of a mixture of different cell types in the dermis layer of the skin. The growths often develop after some type of small trauma to the skin, including a puncture from a splinter or bug bite. This is dermatoscopically distinguished by a central white area surrounded by a faint pigmented network. Most of the lesions are 7-10mm in diameter usually seen as a solitary firm papulae or nodule in the limb and varies from pink, grey, and red or to brown in colour. A poor prognosis of this is associated with metastasis and may turn painful.

mel:

Melanoma is a malignant neoplasm that develops from melanocytes. A combination of environmental factors like exposure to UV and genetic factors like mutations leading to activation of oncogenes and repression of tumour suppressing genes is attributed as a cause for the same. The lesions are characterized by ragged, notched or blurred outlines. Shades of black, brown and tan are often seen. This is highly invasive and rapidly spread to other organs thus early detection is important for treatment to be successful.

nv:

Melanocytic nevi are benign neoplasms caused by the proliferation of neural crest derived melanocytes present at the time of birth or shortly after birth forming a small collection of cells called nests. This causes benign freckles, moles and malignant melanoma. Exposure to UV rays as well as genetic factors is considered as a cause for this neoplasm.

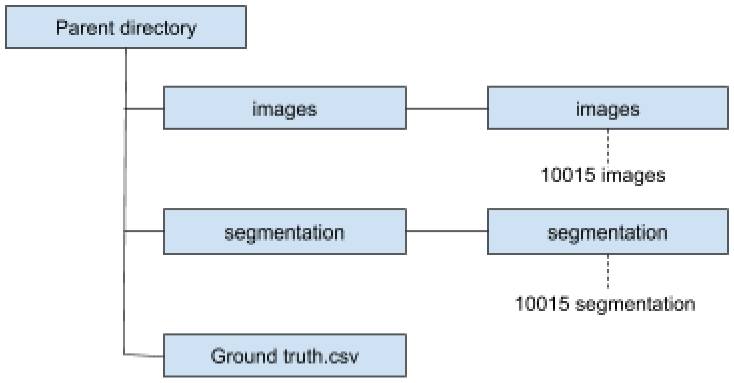
vasc:

Cherry angiomas, angiokeratomas and pyogenic granulomas are the three vascular lesions covered in this dataset:

Cherry angiomas are harmless benign tumours characterized by red papules on the skin. The cause can be genetic factors, exposure to certain chemicals, medical conditions or climate. Angiokeratomas are small dark spots that happen when capillaries dilate or widen near the surface of skin. Hypertension in veins near the skin, inguinal hernia or haemorrhoids could be a possible cause. This could be an indication of Fabry disease and if left untreated can lead to kidney failure, heart failure and stroke thus proper diagnosis is important. Pyogenic granulomas are benign vascular tumors characterized by small round bloody red skin growth caused by injury, bug bites or hormonal changes.

2. Data Preprocessing

The structure of the dataset is as follows:



With the original images and the segmentation, there is a .csv file that contains the ground truth labels of the classes of the images. The 10015 images consist of images from 7 classes which have different sample sizes leading to class imbalance of the dataset.

2.1 Importing the ground truth labels into Pandas Dataframe

The original .csv file contains ground truth labels of each corresponding image. Read the `HAM10000_metadata.csv` file into a dataframe and deep copy the pandas Dataframe so that any changes made to the copied file does not affect the original file.

```
metadata = pd.read_csv('HAM10000_metadata.csv')
data = metadata.copy(deep=True)
data.head()
```

2.2 Converting the string labels to numerical labels

The original .csv file contains labels **akiec, bcc, bkl, df, mel, nv, vasc**. We convert these string labels to numerical labels using the LabelEncoder from scikit-learn. This function enables us to encoded string labels to numerical labels as follows:

String label	akiec	bcc	bkl	df	mel	nv	vasc
Integer label	0	1	2	3	4	5	6

```
label_encoder = LabelEncoder()
label_encoder.fit(data['dx'])
data['label'] = label_encoder.transform(data['dx'])
data = data.sort_values('image_id')
data.head()
```

2.3 Stratified train validation split

In order to have the same distribution of class in the train and validation set, we split the dataset with stratification. The original 10015 image dataset is split into a validation set with 992 images (nearly 1% of the total dataset) and a training set with the rest of the image.

```
#Here 5514 is the number of samples with only one lesion_id
#i.e. no augmented copies of the image is present in the dataset
val_labels_stratify = np.zeros(5514, dtype=int)

#Array containing the number of samples that each class should contain
#in order to have same distribution in the original dataset
temp = np.array(data['label'].value_counts(sort=False)
        /data['label'].value_counts().sum()*5514).astype(np.int)

x = np.zeros_like(temp)
for i in range(len(temp)):
    for j in range(i+1):
        x[i] +=temp[j]

lower_limit = 0
for i in range(len(x)):
    val_labels_stratify[lower_limit:x[i]] = int(i)
    lower_limit = x[i]
```

```
from collections import Counter
Counter(val_labels_stratify)
```

```
Counter({0: 183, 1: 282, 2: 605, 3: 63, 4: 612, 5:3691, 6: 78})
```

To add to the complexity, some of the images in the original dataset have been added more than one time with different augmentations. This can be found by looking at the lesion_id of each image. In order to avoid those images being present in the validation set, we chose only images that had one lesion_id in the dataset. So all 992 images in the validation set were chosen such that there is no other augmented copy of the same image in the training set. Here is the code for splitting the dataset into training and validation sets.

```
#getting lesion_id with only one copy for the validation set
val_df = data.groupby('lesion_id').filter(lambda x:len(x)==1).drop_duplicates(
                                                                        subset='lesion_id')
val_image_id = val_df['image_id']
val_labels_id = val_df['label']

#getting lesion_id with more than one copies for the training set
train_df = data.groupby('lesion_id').filter(lambda x: len(x)>1)
train_image_id = train_df['image_id']
train_labels_id = train_df['label']

#splitting of data which has only one copy of lesion_id
X_train, X_val, y_train, y_val = train_test_split(val_image_id, val_labels_id,
                                                  test_size = 992,random_state=1,
                                                  stratify=val_labels_stratify)

#concating lesion_id with one copy in the X_train a lesion_id with more than one copy
X_train = pd.concat([X_train, train_image_id])
y_train = pd.concat([y_train, train_labels_id])

#defining image path, segmentation path and labels of the training set
train_input_img_paths = 'images/images/' + X_train.values + '.jpg'
train_target_img_paths = 'segmentation/segmentation/'+ X_train.values +'_segmentation.png'
train_labels = y_train.values

#defining image path, segmentation path and labels of the validation set
val_input_img_paths = 'images/images/' + X_val.values + '.jpg'
val_target_img_paths = 'segmentation/segmentation/' + X_val.values + '_segmentation.png'
val_labels = y_val.values
```

Printing image path, segmentation path and corresponding labels of the training set.

```
#printing training image path, segmentation path and corresponding labels
print('Number of samples in the training set:{}'.format(len(train_labels)))
for input_path, target_path, label in zip(train_input_img_paths[:10],
train_target_img_paths[:10], train_labels[:10]):
    print(input_path, "|", target_path, "|", label)
```

```
Number of samples in the training set:9023
images/images/ISIC_0027981.jpg | segmentation/segmentation/ISIC_0027981_segmentation.png | 5
images/images/ISIC_0028236.jpg | segmentation/segmentation/ISIC_0028236_segmentation.png | 2
images/images/ISIC_0028854.jpg | segmentation/segmentation/ISIC_0028854_segmentation.png | 0
images/images/ISIC_0029574.jpg | segmentation/segmentation/ISIC_0029574_segmentation.png | 4
images/images/ISIC_0032093.jpg | segmentation/segmentation/ISIC_0032093_segmentation.png | 5
images/images/ISIC_0031306.jpg | segmentation/segmentation/ISIC_0031306_segmentation.png | 5
images/images/ISIC_0027296.jpg | segmentation/segmentation/ISIC_0027296_segmentation.png | 5
images/images/ISIC_0030100.jpg | segmentation/segmentation/ISIC_0030100_segmentation.png | 5
images/images/ISIC_0026682.jpg | segmentation/segmentation/ISIC_0026682_segmentation.png | 5
images/images/ISIC_0027218.jpg | segmentation/segmentation/ISIC_0027218_segmentation.png | 2
```

Printing image path, segmentation path and corresponding labels of the validation set.

```
#printing validation image path, segmentation path and corresponding labels
print('Number of samples in the validation set:{}'.format(len(val_labels)))
for input_path, target_path, label in zip(val_input_img_paths[:10],
val_target_img_paths[:10], val_labels[:10]):
    print(input_path, "|", target_path, "|", label)
```

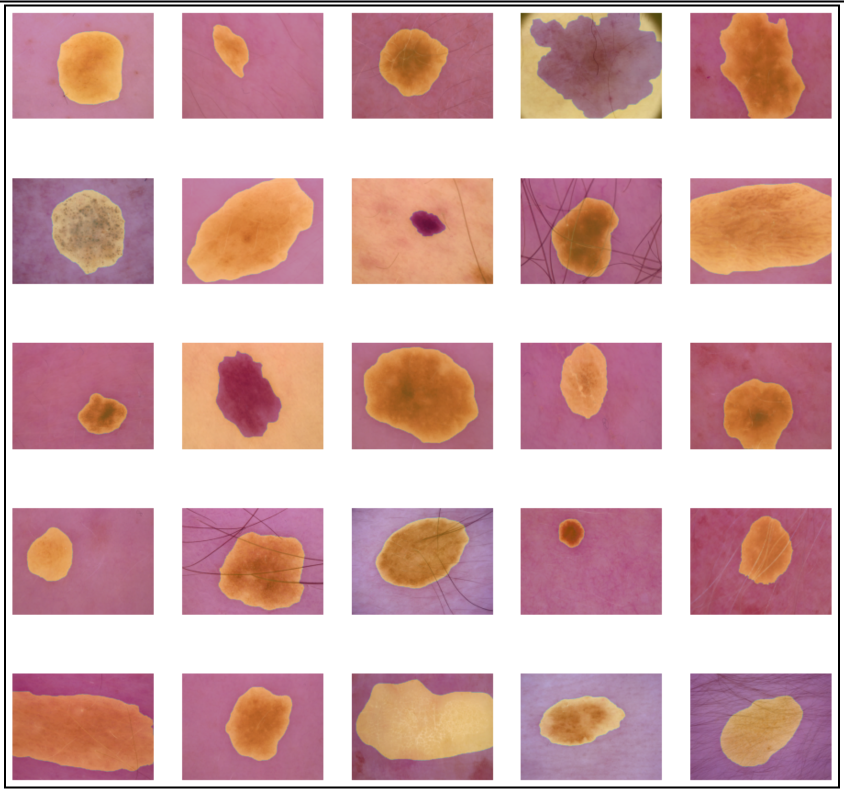
```
Number of samples in the validation set:992
images/images/ISIC_0028090.jpg | segmentation/segmentation/ISIC_0028090_segmentation.png | 5
images/images/ISIC_0026130.jpg | segmentation/segmentation/ISIC_0026130_segmentation.png | 5
images/images/ISIC_0032054.jpg | segmentation/segmentation/ISIC_0032054_segmentation.png | 5
images/images/ISIC_0024371.jpg | segmentation/segmentation/ISIC_0024371_segmentation.png | 2
images/images/ISIC_0028189.jpg | segmentation/segmentation/ISIC_0028189_segmentation.png | 5
```

```
images/images/ISIC_0025883.jpg | segmentation/segmentation/ISIC_0025883_segmentation.png | 2
images/images/ISIC_0030612.jpg | segmentation/segmentation/ISIC_0030612_segmentation.png | 5
images/images/ISIC_0024389.jpg | segmentation/segmentation/ISIC_0024389_segmentation.png | 5
images/images/ISIC_0031341.jpg | segmentation/segmentation/ISIC_0031341_segmentation.png | 5
images/images/ISIC_0026867.jpg | segmentation/segmentation/ISIC_0026867_segmentation.png | 2
```

2.4 Inverting segmentation for uniform labeling of pixel

Some of the segmentation images in the dataset are labeled in the wrong way. Such segmentation images have the lesion coded as 0 and the background coded as 1. Printing some images in the validation set with segmentation overlay

```python
fig = plt.figure()
fig.set_size_inches(16,16)
for i in range(25):
    plt.subplot(5, 5, 1 + i)
    plt.axis('Off')
    plt.imshow(np.array(Image.open(val_input_img_paths[i])))
    plt.imshow(np.array(Image.open(val_target_img_paths[i])), alpha=0.3)
```

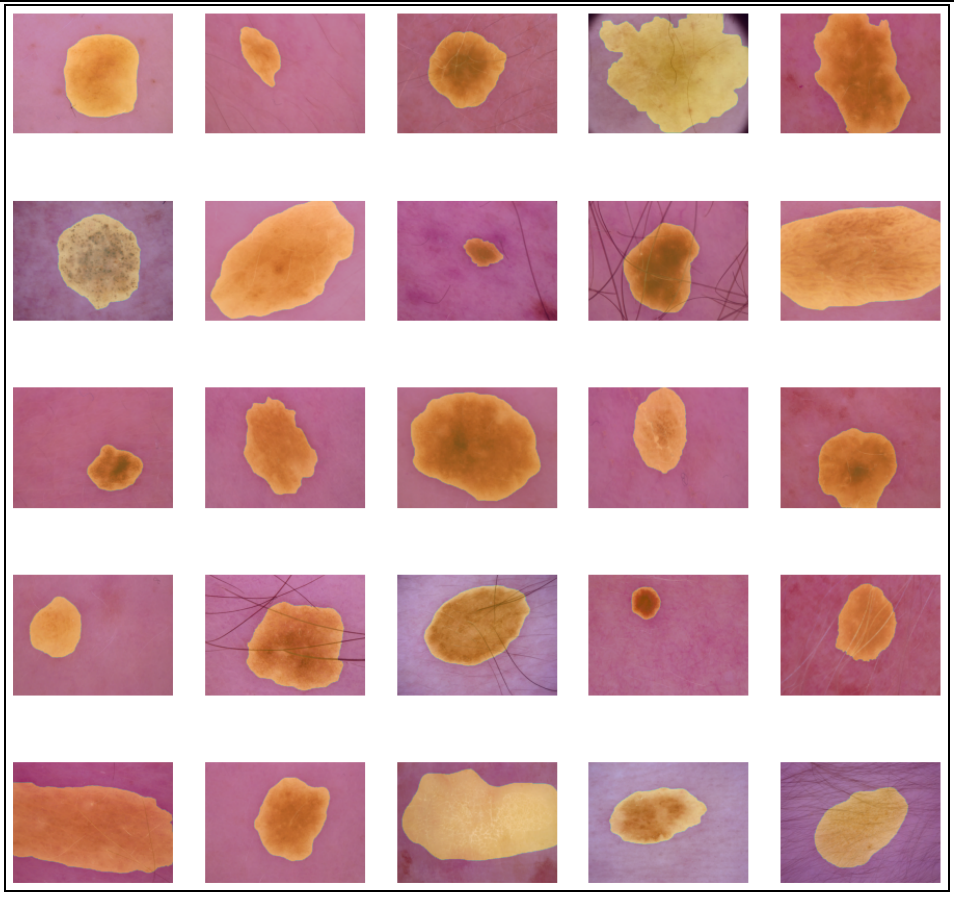


To invert the code so that all segmentation images are uniformly coded we have defined a helper function to invert the segmentation.

```python
def invert_segmentation(image_data):
    """Inverts the segmentation data of the image if the skin lesion
    is 0 and the background is 1."""
    if image_data[0][0]>0:
        inverted_image_data = np.logical_not(image_data)
        inverted_image_data = inverted_image_data.astype(int)
        return inverted_image_data
    else :
        return image_data
```

Now printing the same images to check whether the segmentation are uniformly coded.

```python
fig = plt.figure()
fig.set_size_inches(16,16)
for i in range(25):
    plt.subplot(5, 5, 1 + i)
    plt.axis('Off')
    plt.imshow(np.array(Image.open(val_input_img_paths[i])))
    plt.imshow(invert_segmentation(np.array(Image.open(val_target_img_paths[i]))),alpha=0.3)
```



3. Model Architecture

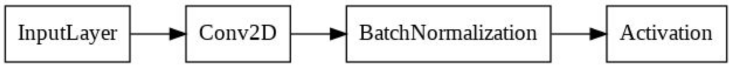
We are implementing a Unet model with MobileNetv2 encoder. The Unet model predicts the skin lesion pixels as 1 and background pixels as 0 without any classification of labels. To classify the skin lesions into the 7 classes, we are implementing a classification model with input as mobilenet backbone's output. We take the pretrained MobileNetv2 trained on the imagenet dataset and freeze the layers. Next we take the outputs of the following feature layers and concatenate them to the encoder as skip connections.

```
feature_layers = ('block_13_expand_relu', 'block_6_expand_relu', 'block_3_expand_relu',
                  'block_1_expand_relu')
```

The following code takes the input tensor and performs convolution, batch normalization and activation.

```
def Conv2dBN(filters, kernel_size, activation=None, padding='valid',
    kernel_initializer='glorot_uniform', use_batchnorm=False, name = None):
    """Extension of Conv2D layer with batchnorm"""
    convolution_name, activation_name, batchnorm_name = None, None, None
    block_name = name
    if block_name is not None:
        convolution_name = block_name + '_conv'
        if activation is not None:
            activation_str=activation.__name__ if callable(activation) else str(activation)
            activation_name = block_name + '_' + activation_str
            if use_batchnorm:
                batchnorm_name = block_name + '_bn'

    def encoder(input_tensor):
        x = Conv2D(filters = filters,
                kernel_size=kernel_size,
                padding=padding,
                use_bias=not (use_batchnorm),
                kernel_initializer = kernel_initializer,
                name = convolution_name)(input_tensor)

        if use_batchnorm:
            x = BatchNormalization(name=batchnorm_name)(x)
        if activation:
            x = Activation(activation, name=activation_name)(x)
        return x
    return encoder

def Conv3x3BNReLU(filters, use_batchnorm, name=None):
    def wrapper(input_tensor):
        return Conv2dBN(filters, kernel_size=3,
                        activation='relu',
                        kernel_initializer='he_uniform',
                        padding='same',
                        use_batchnorm=use_batchnorm,
                        name=name)(input_tensor)

    return wrapper
```

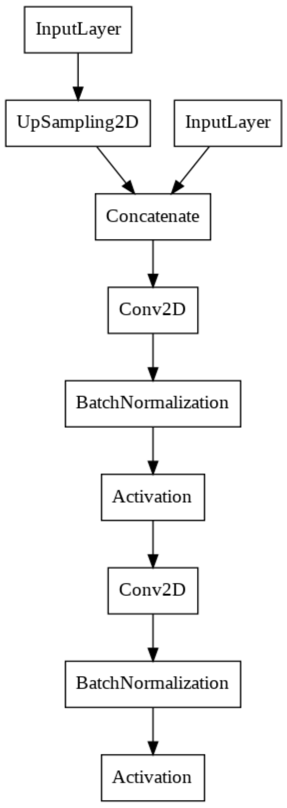
A schematic diagram of the above code is given below:



The following function defines the decoder block of the Unet model. It performs 2D UpSampling of the input and concatenates with the skip connections of feature layers and performs 2D convolution, batch normalization and ReLU activation.

```
def DecoderUpsamplingX2Block(filters, stage, use_batchnorm=False):
    up_name = 'decoder_stage{}_upsampling'.format(stage)
    conv1_name = 'decoder_stage{}a'.format(stage)
    conv2_name = 'decoder_stage{}b'.format(stage)
    concat_name = 'decoder_stage{}_concat'.format(stage)

    def wrapper(input_tensor, skip=None):
        x = UpSampling2D(size=2, name=up_name)(input_tensor)

        if skip is not None:
            x = Concatenate(axis=-1, name=concat_name)([x, skip])

        x = Conv3x3BNReLU(filters, use_batchnorm, name=conv1_name)(x)
        x = Conv3x3BNReLU(filters, use_batchnorm, name=conv2_name)(x)
        return x
    return wrapper
```

The schematic diagram of the decoder block is given below.



The Unet model for segmentation is defined using the following code

```
def Unet(skip_connection_layers, input_shape=(224, 224, 3), encoder_freeze=False,
         decoder_filters=(256, 128,64, 32, 16), n_upsample_blocks=5, classes=1,
         activation='sigmoid', use_batchnorm=True):

    backbone = MobileNetV2(include_top=False,input_shape=input_shape,
                           weights='imagenet')
    input_ = backbone.input
    x = backbone.output

    skips = ([backbone.get_layer(name=i).output for i in
             skip_connection_layers])

    #building decoder blocks
    for i in range(len(decoder_filters)):
        if i < len(skips):
            skip = skips[i]
        else :
            skip = None
        x = DecoderUpsamplingX2Block(decoder_filters[i], stage=i,
                                     use_batchnorm=use_batchnorm)(x, skip)

    x = Conv2D(filters=classes, kernel_size=(3,3),
               padding='same', name='final_conv')(x)
    x = Activation(activation, name=activation)(x)

    if encoder_freeze:
        backbone.trainable = False

    model = Model(input_, x)

    return model
```

The above Unet model only predicts the segmentation of the skin lesion and does not predict the class of the skin lesion. This is because almost all skin lesions are very similar in nature with only minute differences. Also each image has only one class of skin lesion. So it will be very difficult for the model to predict the class with the segmentation. Trail Unet models for multi-class semantic segmentation of HAM 10000 dataset have shown poor performance with multiple class prediction for a single input image.
For classification of the input image into 7 different classes, a Ynet model is proposed. This is a modification of the Unet model where the last layer of the enoder branch is given as the input for multi-class classification.
NOTE : I personally came up with the Ynet model as I could not find any scientific literature about it.

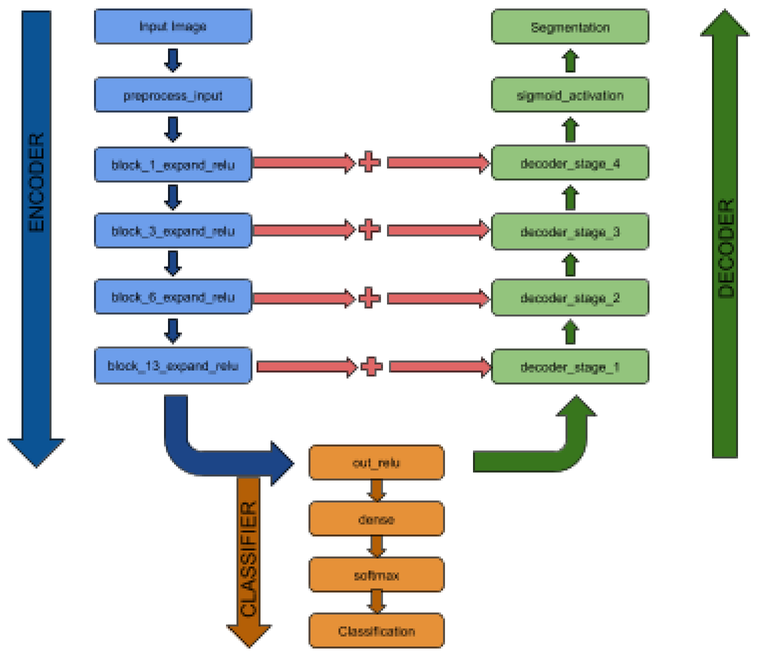
NOTE: The proposed Ynet model is a modification of the already existing Unet model. This model can be only used for datasets where the input image has only one class for segmentation (here skin lesion) and multi-class classification(here 7 class). This Ynet model performs better if the segmentation of the 7 classes are very similar and each image has only one class.

NOTE: Here the backbone for the Ynet is pretrained MobileNetv2 trained on the imagenet dataset. The backbone layers are freezed for training. For better accuracy in both segmentation and classification, initial training should be done for the encoder and classification branch only and after a few epochs the layers should be freezed. This is recommended so that the feature maps of the backbone have the feature of the HAM dataset and not the imagenet dataset. This will increase the performance of the model in both segmentation and classification heads. Due to resource constraints, here the model backbone is freezed for the imagenet dataset feature maps.

For classification of the input images into 7 classes a separate classification model with input as the output of the mobilenet backbone of Unet segmentation model.

```
def Unet_classification(Unet_model, num_class, encoder_freeze):
    unet_model = Unet_model(skip_connection_layers=feature_layers,
                            encoder_freeze=encoder_freeze)
    x = unet_model.input
    out = unet_model.get_layer(name='out_relu').output
    out = GlobalAveragePooling2D()(out)
    out = Dense(128, activation = 'relu')(out)
    out = Dropout(0.4)(out)
    out = Dense(128, activation = 'relu')(out)
    out = Dense(num_class, activation='softmax',
                name='classification')(out)
    model = Model(x, [unet_model.output, out])

    return model

model = Unet_classification(Unet, 7, encoder_freeze=False)
#model.summary()
#plot_model(model)
```

A brief schematic representation of the final model is given below:



4. DataGenerator

A custom datagenerator is created inorder to accommodate the specific details of the dataset.No data augmentation has been included in the datagenerator due to time and resource constraints. A large dataset like this (10015 images) is expected to train the model with decent accuracy without any data augmentation.

Here is the code to create the datagenerator for the model.

```
class HamDatasetClassification(keras.utils.Sequence):
    """Helper to iterate over the data (as Numpy arrays)."""

    def __init__(self, batch_size, img_size, classes, input_img_paths, target_img_paths,
                 labels):
        self.batch_size = batch_size
        self.img_size = img_size
        self.classes = classes
        self.input_img_paths = input_img_paths
        self.target_img_paths = target_img_paths
        self.labels = labels

    def __len__(self):
        return len(self.input_img_paths) // self.batch_size

    def __getitem__(self, idx):
        """Returns tuple (input, target) correspond to batch #idx."""
        i = idx * self.batch_size
        batch_input_img_paths = self.input_img_paths[i : i + self.batch_size]
        batch_target_img_paths = self.target_img_paths[i : i + self.batch_size]
        batch_labels = self.labels[i : i + self.batch_size]
        x = np.zeros((self.batch_size,) + self.img_size + (3,), dtype="float32")
        for j, path in enumerate(batch_input_img_paths):
            img = load_img(path, target_size=self.img_size)
            x[j] = preprocess_input(np.array(img))

        y = np.zeros((self.batch_size,) + self.img_size + (1,))
        for j, path in enumerate(batch_target_img_paths):
            img = load_img(path, target_size=self.img_size, color_mode="grayscale")
            img_data = np.expand_dims(img, axis = 2)
            img_data = invert_segmentation(np.array(img_data))
            y[j] = img_data/255

        label = to_categorical(batch_labels, self.classes)
        return x, [y,label]

train_gen = HamDatasetClassification(batch_size, img_size, classes, train_input_img_paths,
                                     train_target_img_paths, train_labels)
val_gen = HamDatasetClassification(batch_size, img_size, classes, val_input_img_paths,
                                   val_target_img_paths, val_labels)
```

The above code helps in iterating over the data as Numpy arrays without consuming too much RAM. This helps in training the model with loads of data.

The input image data is created using the load_img function as is then passed on to the preprocess_input function where the image is rescaled form [0, 255] to [-1, 1] range. This is then fed into the model as input.

The model has two output heads, segmentation and class. Target segmentation data is converted into Numpy arrays using load_img function. Then it is passed on to the invert_segmentation function to invert the segmentation coding if required, then the target segmentation data is rescaled from [0, 255] range to [0, 1] range. The corresponding labels of the input images are converted to one hot vector using the to_categorical function.

Training datagenerator (`train_gen`) and validation datagenerator (`val_gen`) are then correspondingly created the custom DataGenerator.

5. Custom Loss Function

In order to overcome the class imbalance problem in our dataset, we define a custom weighted categorical loss function for training the model. This loss function treats every class differently according to the weights of that class.
Here is the code:

```
def weighted_categorical_crossentropy(weights):
    weights = tf.constant(weights, dtype = tf.float32)
    def loss(y_true, y_pred):
        l = tf.keras.losses.CategoricalCrossentropy()(y_true, y_pred)
        l = l*weights[tf.math.argmax(y_true[0])]
        return l

    return loss

#calculating class weight
from sklearn.utils.class_weight import compute_class_weight
labels = data['label'].to_numpy(dtype=int)

#print('Number of samples in each class:')
#print(data['label'].value_counts())

weights = compute_class_weight('balanced', np.unique(labels), labels)

#defining the loss function using the class weights
loss = weighted_categorical_crossentropy(weights)
```

In this loss function each class has a weight assigned. The categorical loss between the prediction and ground truth is multiplied by the corresponding weight of the ground truth class. The class weights have been computed by the compute_class_weight function imported from scikit- learn.

6. Compiling and Fitting the Model

As the model has two prediction heads (segmentation and classification), I have used two loss functions. For segmentation, since it is a binary classification of whether the pixel is skin lesion or background, the loss function used is binary cross entropy. For classification, due to the class imbalance seen before, the loss function used is the custom weighted categorical cross entropy. Loss weight of the binary cross entropy has been scaled 5 times with respect to the categorical cross entropy for better accuracy in both prediction heads.

I have used Adam as the optimizer and accuracy as the metric.

The model is then fitted to the training data generator (train_gen) and validation data generator (val_gen) for with batch size as 32. The model is trained for 30 epochs and after each epoch the model is saved if the classification accuracy has improved.

```
model.compile(optimizer="adam", loss=["binary_crossentropy", loss], metrics=['accuracy'],
              loss_weights=[5,1])
mc = ModelCheckpoint('best_model.h5', monitor='val_classification_accuracy', mode='max',
                     verbose=1, save_best_only=True)
model.fit(train_gen, epochs=30, validation_data=val_gen, callbacks=[mc])
```

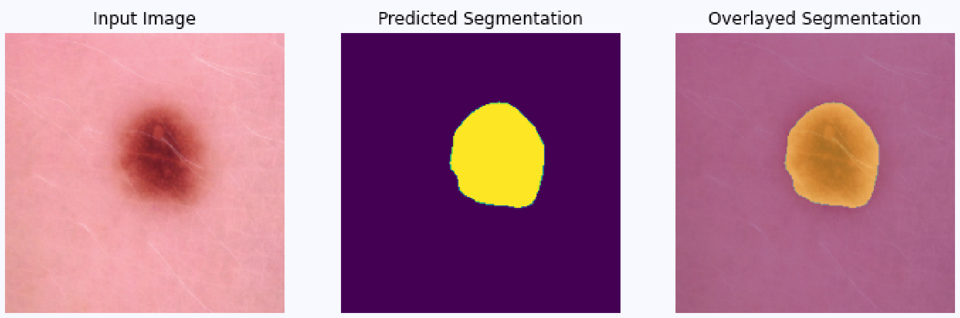
6. Evaluating the model

In order to visualize the prediction of the model, a helper function is created as below:

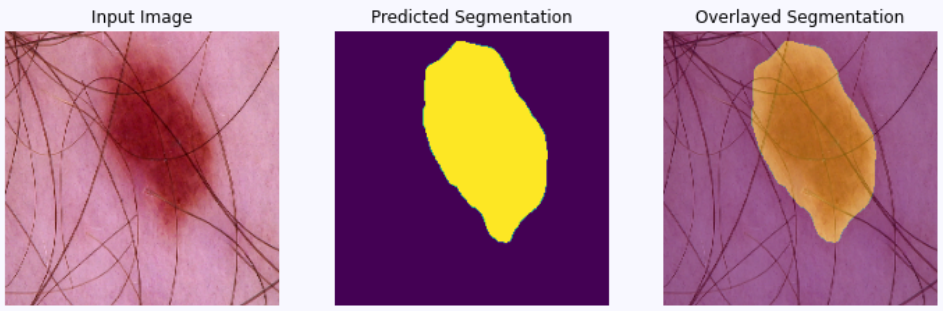
```python
def predict_output(model, input_data, threshold = 0.5):
    """Predicts the segmentation and class of the input image."""
    if isinstance(input_data, str):
        data = load_img(input_data, target_size=(224,224))
        data = np.array(data)
    if isinstance(input_data, np.ndarray):
        data = input_data

    height, width, channel = data.shape
    if height == 224 and width == 224 and channel == 3:
        data = np.expand_dims(data, axis = 0)
        preprocessed_data = preprocess_input(data)
        prediction = model.predict(preprocessed_data)
        segmentation = prediction[0][0,:,:,0]
        predicted_class = np.argmax(prediction[1][0])
        predicted_score = max(prediction[1][0])

        result = np.zeros((height, width))
        for i in range(height):
            for j in range(width):
                if segmentation[i][j] > threshold:
                    result[i][j] = 1
                else :
                    result[i][j] = 0

        print('Predicted Class : {}'.format(predicted_class))
        print('Predicted score : {}'.format(predicted_score))
        fig = plt.figure()
        fig.set_size_inches((12, 5))
        plt.subplot(1,3,1)
        plt.title('Input Image')
        plt.imshow(data[0])
        plt.axis('Off')
        plt.subplot(1,3,2)
        plt.title('Predicted Segmentation')
        plt.imshow(result)
        plt.axis('Off')
        plt.subplot(1,3,3)
        plt.title('Overlayed Segmentation')
        plt.imshow(data[0])
        plt.imshow(result, alpha=0.4)
        plt.axis('Off')
        plt.show()
    else :
        print('Input Image shape not valid. Should be (224, 224, 3) but got
{}'.format(data.shape))
```

Below are some of the predictions from the validation set:

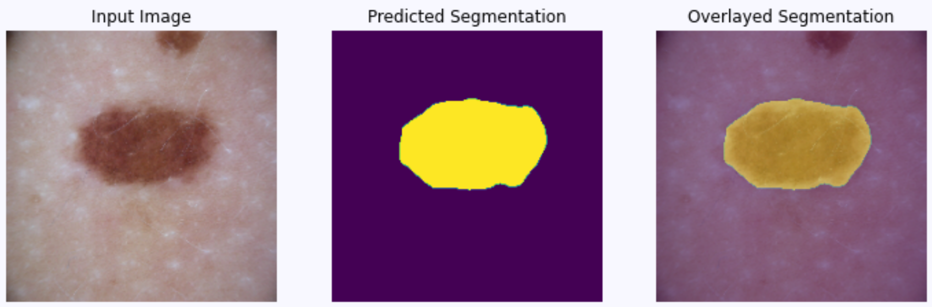
```
Predicted Class : 5
Predicted Label : nv
Predicted score : 0.9999933242797852
```



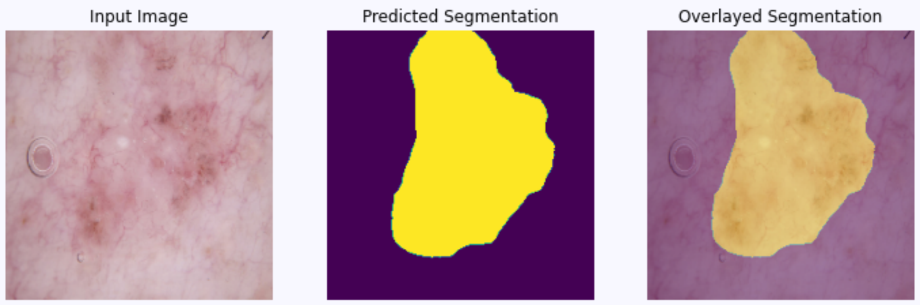
```
Predicted Class : 5
Predicted Label : nv
Predicted score : 0.9999324083328247
```



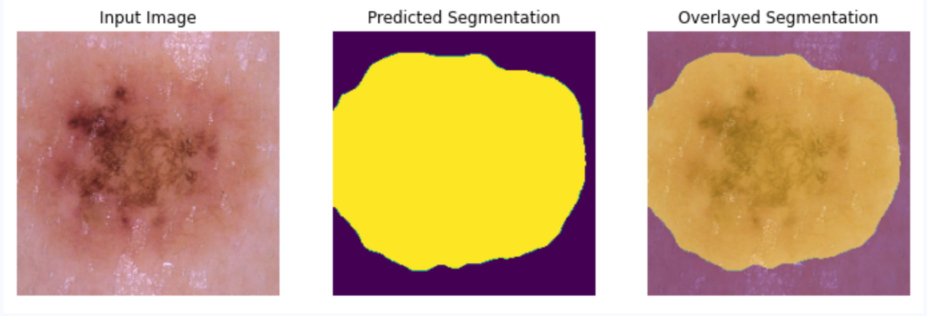
```
Predicted Class : 5
Predicted Label : nv
Predicted score : 0.9705167412757874
```



```
Predicted Class : 1
Predicted Label : bcc
Predicted score : 0.7348884344100952
```

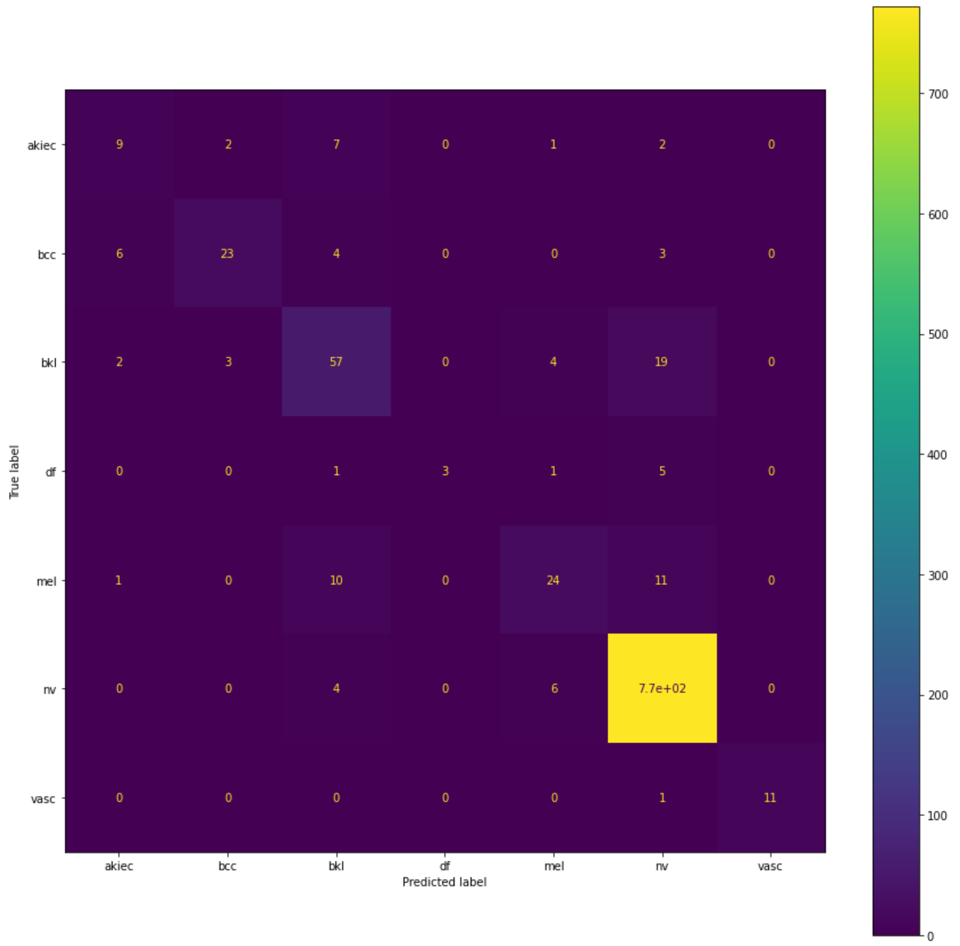


```
Predicted Class : 4
Predicted Label : mel
Predicted score : 0.4512506127357483
```



In order to understand the performance of the classification output, a confusion matrix is created from which we can compute the number of correct and incorrect classifications.

```
_,validation_prediction = model.predict(val_gen)
from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay
cm = confusion_matrix(val_labels, np.argmax(validation_prediction, axis = -1))
fig, ax = plt.subplots(figsize=(15, 15))
x=ConfusionMatrixDisplay(cm,
                   display_labels =['akiec','bcc','bkl','df','mel','nv','vasc']).plot(ax=ax)
```



8. Further Improvements

1. The proposed Ynet (modified Unet with separate segmentation and classification branch) can be further improved by first training the model only with the encoder and classifier branch and after a few epochs, the decoder branch with skip connection is trained with the encoder branch freezed.This is recommended so that the feature maps

of the encoder branch are relevant to our dataset and this will increase the performance of the model both in the segmentation and classification heads.

2. Data Augmentation can be implemented so as to increase the number of training samples which can further improve the performance of the model.

3. Dilated Convolutions have shown promising results in biomedical image segmentation and that can be implemented in our model.

4. Further testing can be done by changing the backbone of the model from MobileNetv2 to Xception or any other model.

5. Dice loss or IoU can be implemented for accurate segmentation

9. References

1. HAM 10000 Dataset
2. Tschandl, P., Rosendahl, C. & Kittler, H. The HAM10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions. Sci. Data 5, 180161 (2018).
3. U-Net: Convolutional Networks for Biomedical Image Segmentation
4. MobileNetV2: Inverted Residuals and Linear Bottlenecks