Importing Libraries

```
import seaborn as sns
import matplotlib.pyplot as plt
from imblearn.over_sampling import RandomOverSampler
import numpy as np
from sklearn.model_selection import train_test_split
import os, cv2
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, Flatten, Dense, MaxPool2D

from sklearn.metrics import  precision_score, recall_score, accuracy_score,classification_report
```

Import Data

```
import pandas as pd
data = pd.read_csv("/content/hmnist_28_28_RGB.csv")
data.head()
```

	pixel0000	pixel0001	pixel0002	pixel0003	pixel0004	pixel0005	pixel0006	pixel(
0	192	153	193	195	155	192	197	
1	25	14	30	68	48	75	123	
2	192	138	153	200	145	163	201	
3	38	19	30	95	59	72	143	
4	158	113	139	194	144	174	215	

5 rows × 2353 columns

```
data=data.dropna()
```

```
y = data['label']
x = data.drop(columns = ['label'])
```

Exploratory Data Analysis (EDA)

```
tabular_data = pd.read_csv("/content/ISIC2018_Task3_Test_GroundTruth.csv")
tabular_data.head()
```

	lesion_id	image_id	dx	dx_type	age	sex	localization	data
0	HAMTEST_0000000	ISIC_0034524	nv	follow_up	40.0	female	back	vidir_moler
1	HAMTEST_0000001	ISIC_0034525	nv	histo	70.0	male	abdomen	rosenc
2	HAMTEST_0000002	ISIC_0034526	bkl	histo	70.0	male	back	rosenc
3	HAMTEST_0000003	ISIC_0034527	nv	histo	35.0	male	trunk	vienna_c
4	HAMTEST_0000004	ISIC_0034528	nv	follow_up	75.0	female	trunk	vidir_moler

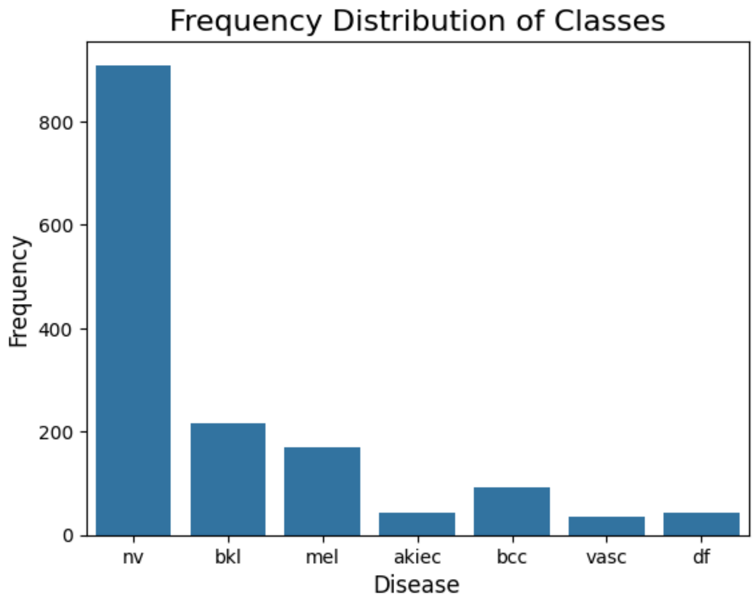
```
tabular_data.columns
```

```
Index(['lesion_id', 'image_id', 'dx', 'dx_type', 'age', 'sex', 'localization',
       'dataset'],
      dtype='object')
```

```
classes = {4: ('nv', ' melanocytic nevi'), 6: ('mel', 'melanoma'), 2 :('bkl', 'benign keratosis-like lesions'), 1:('bcc' , ' basal cell
```

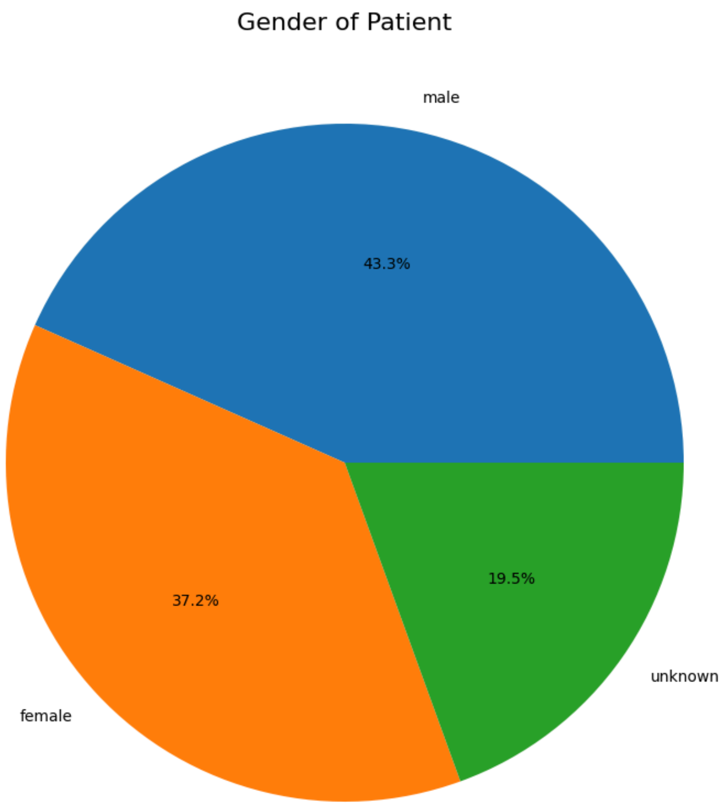
Frequency Distribution of Classes

```
sns.countplot(x = 'dx', data = tabular_data)
plt.xlabel('Disease', size=12)
plt.ylabel('Frequency', size=12)
plt.title('Frequency Distribution of Classes', size=16)
plt.show()
```



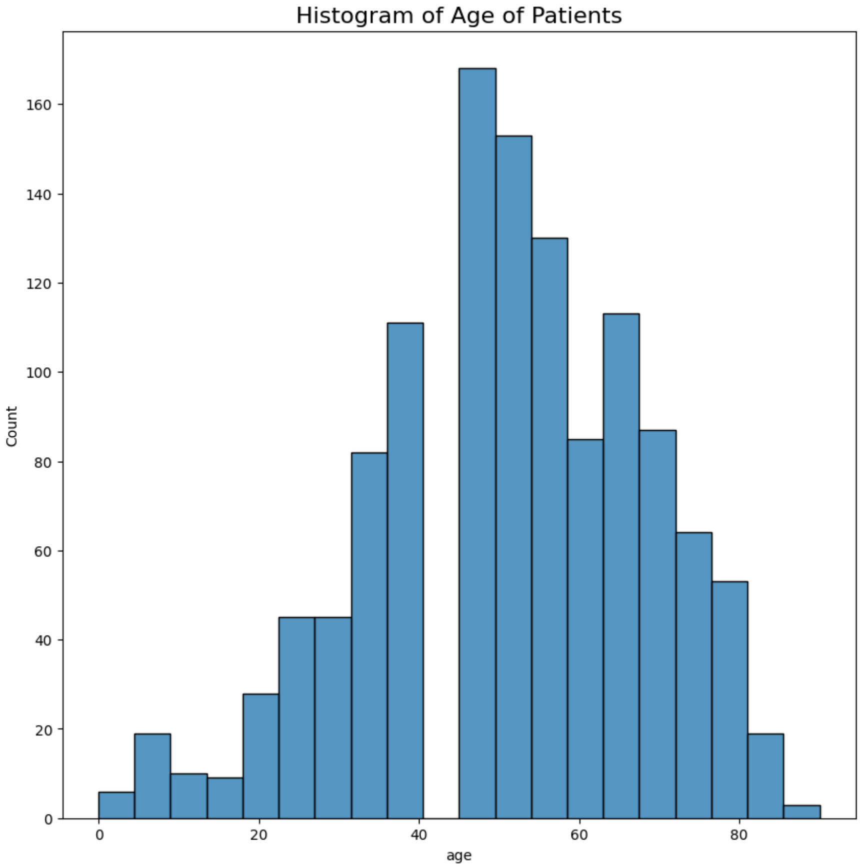
Distribution of Disease over Gender

```
bar, ax = plt.subplots(figsize = (10,10))
plt.pie(tabular_data['sex'].value_counts(), labels = tabular_data['sex'].value_counts().index, autopct="%.1f%%")
plt.title('Gender of Patient', size=16)
plt.show()
```



Histogram of Age of Patients

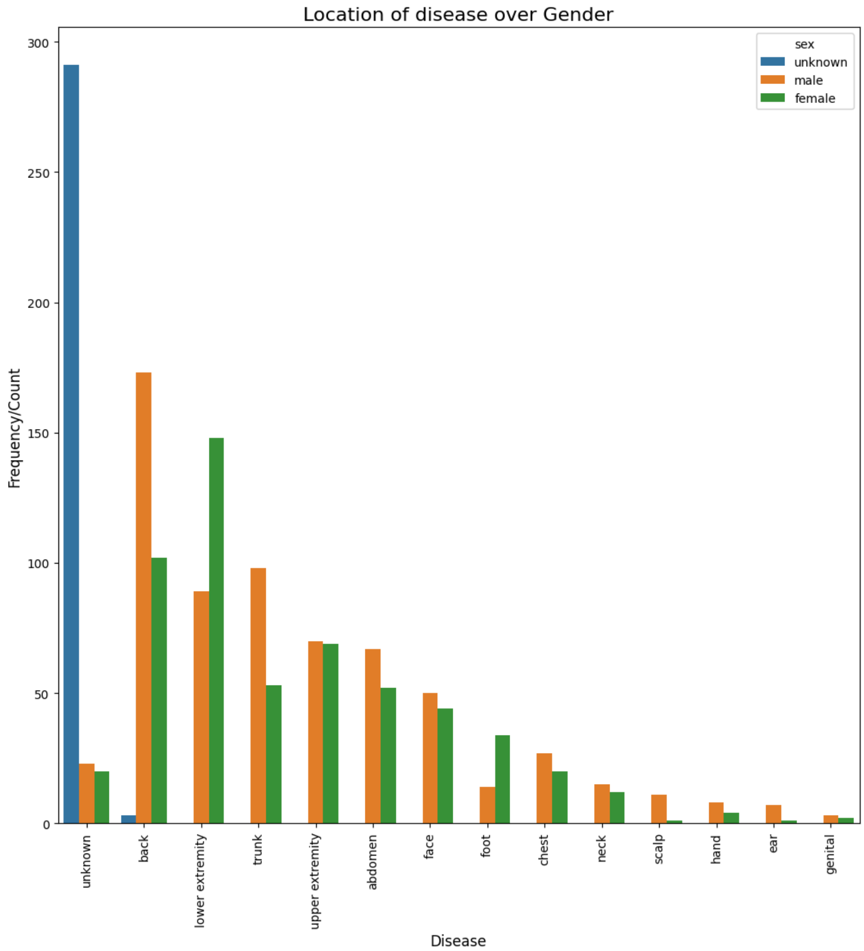
```
bar, ax = plt.subplots(figsize=(10,10))
sns.histplot(tabular_data['age'])
plt.title('Histogram of Age of Patients', size=16)
plt.show()
```



Location of disease over Gender

```
value = tabular_data[['localization', 'sex']].value_counts().to_frame()
value.reset_index(level=[1,0 ], inplace=True)
temp = value.rename(columns = {'localization':'location', 0: 'count'})

bar, ax = plt.subplots(figsize = (12, 12))
sns.barplot(x = 'location',  y='count', hue = 'sex', data = temp)
plt.title('Location of disease over Gender', size = 16)
plt.xlabel('Disease', size=12)
plt.ylabel('Frequency/Count', size=12)
plt.xticks(rotation = 90)
plt.show()
```



Oversampling

To overcome class imbalace

```
oversample = RandomOverSampler()
x,y  = oversample.fit_resample(x,y)
```

```
x = np.array(x).reshape(-1,28,28,3)
print('Shape of X :',x.shape)
```

```
Shape of X : (46935, 28, 28, 3)
```

```
print('Shape of Y :',y.shape)
```

```
Shape of Y : (46935,)
```

Standardization and Splitting Data

```
x = (x-np.mean(x))/np.std(x)
X_train, X_test, Y_train, Y_test = train_test_split(x,y, test_size=0.2, random_state=1)
```

Custom Model

```
#Soft Attention

from keras import backend as K
from keras.layers import Layer,InputSpec,Input,BatchNormalization,MaxPooling2D,concatenate,Activation,Dropout
import keras.layers as kl
import tensorflow as tf
from tensorflow.keras import Model



class SoftAttention(Layer):
    def __init__(self,ch,m,concat_with_x=False,aggregate=False,**kwargs):
        self.channels=int(ch)
        self.multiheads = m
        self.aggregate_channels = aggregate
        self.concat_input_with_scaled = concat_with_x


        super(SoftAttention,self).__init__(**kwargs)

    def build(self,input_shape):

        self.i_shape = input_shape

        kernel_shape_conv3d = (self.channels, 3, 3) + (1, self.multiheads) # DHWC

        self.out_attention_maps_shape = input_shape[0:1]+(self.multiheads,)+input_shape[1:-1]

        if self.aggregate_channels==False:

            self.out_features_shape = input_shape[:-1]+(input_shape[-1]+(input_shape[-1]*self.multiheads),)
        else:
            if self.concat_input_with_scaled:
                self.out_features_shape = input_shape[:-1]+(input_shape[-1]*2,)
            else:
                self.out_features_shape = input_shape


        self.kernel_conv3d = self.add_weight(shape=kernel_shape_conv3d,
                                        initializer='he_uniform',
                                        name='kernel_conv3d')
        self.bias_conv3d = self.add_weight(shape=(self.multiheads,),
                                      initializer='zeros',
                                      name='bias_conv3d')

        super(SoftAttention, self).build(input_shape)

    def call(self, x):

        exp_x = K.expand_dims(x,axis=-1)

        c3d = K.conv3d(exp_x,
                     kernel=self.kernel_conv3d,
                     strides=(1,1,self.i_shape[-1]), padding='same', data_format='channels_last')
        conv3d = K.bias_add(c3d,
                        self.bias_conv3d)
        conv3d = kl.Activation('relu')(conv3d)

        conv3d = K.permute_dimensions(conv3d,pattern=(0,4,1,2,3))


        conv3d = K.squeeze(conv3d, axis=-1)
        conv3d = K.reshape(conv3d,shape=(-1, self.multiheads ,self.i_shape[1]*self.i_shape[2]))

        softmax_alpha = K.softmax(conv3d, axis=-1)
        softmax_alpha = kl.Reshape(target_shape=(self.multiheads, self.i_shape[1],self.i_shape[2]))(softmax_alpha)


        if self.aggregate_channels==False:
            exp_softmax_alpha = K.expand_dims(softmax_alpha, axis=-1)
            exp_softmax_alpha = K.permute_dimensions(exp_softmax_alpha,pattern=(0,2,3,1,4))

            x_exp = K.expand_dims(x,axis=-2)

            u = kl.Multiply()([exp_softmax_alpha, x_exp])

            u = kl.Reshape(target_shape=(self.i_shape[1],self.i_shape[2],u.shape[-1]*u.shape[-2]))(u)

        else:
            exp_softmax_alpha = K.permute_dimensions(softmax_alpha,pattern=(0,2,3,1))

            exp_softmax_alpha = K.sum(exp_softmax_alpha,axis=-1)
```

```
            exp_softmax_alpha = K.expand_dims(exp_softmax_alpha, axis=-1)

            u = kl.Multiply()([exp_softmax_alpha, x])

        if self.concat_input_with_scaled:
            o = kl.Concatenate(axis=-1)([u,x])
        else:
            o = u

        return [o, softmax_alpha]

    def compute_output_shape(self, input_shape):
        return [self.out_features_shape, self.out_attention_maps_shape]


    def get_config(self):
        return super(SoftAttention,self).get_config()
```

Start coding or generate with AI.

```
# MainInput=Input(shape=(28, 28, 3))

# conv=(Conv2D(filters=64,kernel_size=(3,3), activation="relu",padding="same",kernel_initializer='he_normal')(MainInput))
# conv=(BatchNormalization()(conv))
# conv=(Conv2D(filters=64,kernel_size=(1,1), activation="relu",padding="same",kernel_initializer='he_normal')(conv))
# conv=(BatchNormalization()(conv))

# conv=(MaxPooling2D(strides=(2, 2),padding="same")(conv))

# conv=(Conv2D(filters=128,kernel_size=(3,3), activation="relu",padding="same",kernel_initializer='he_normal')(conv))
# conv=(BatchNormalization()(conv))
# conv=(Conv2D(filters=128,kernel_size=(1,1), activation="relu",padding="same",kernel_initializer='he_normal')(conv))
# conv=(BatchNormalization()(conv))

# conv=(MaxPooling2D()(conv))


# conv=(Conv2D(filters=256,kernel_size=(3,3), activation="relu",padding="same",kernel_initializer='he_normal')(conv))
# conv=(BatchNormalization()(conv))
# conv=(Conv2D(filters=256,kernel_size=(3,3), activation="relu",padding="same",kernel_initializer='he_normal')(conv))
# conv=(BatchNormalization()(conv))
# conv=(Conv2D(filters=256,kernel_size=(1,1), activation="relu",padding="same",kernel_initializer='he_normal')(conv))
# conv=(BatchNormalization()(conv))

# conv=(MaxPooling2D()(conv))

# conv=(Conv2D(filters=512,kernel_size=(3,3), activation="relu",padding="same",kernel_initializer='he_normal')(conv))
# conv=(BatchNormalization()(conv))
# conv=(Conv2D(filters=512,kernel_size=(3,3), activation="relu",padding="same",kernel_initializer='he_normal')(conv))
# conv=(BatchNormalization()(conv))
# conv=(Conv2D(filters=512,kernel_size=(1,1), activation="relu",padding="same",kernel_initializer='he_normal')(conv))
# conv=(BatchNormalization()(conv))

# attention_layer,map2 = SoftAttention(aggregate=True,m=16,concat_with_x=False,ch=int(conv.shape[-1]),name='soft_attention')(conv)
# attention_layer=(MaxPooling2D(pool_size=(2, 2),padding="same")(attention_layer))
# conv=(MaxPooling2D(pool_size=(2, 2),padding="same")(conv))

# conv = concatenate([conv,attention_layer])
# conv=Activation("relu")(conv)
# conv= Dropout(0.5)(conv)

# conv=(Conv2D(filters=512,kernel_size=(3,3), activation="relu",padding="same",kernel_initializer='he_normal')(conv))
# conv=(BatchNormalization()(conv))
# conv=(Conv2D(filters=512,kernel_size=(3,3), activation="relu",padding="same",kernel_initializer='he_normal')(conv))
# conv=(BatchNormalization()(conv))
# conv=(Conv2D(filters=512,kernel_size=(1,1), activation="relu",padding="same",kernel_initializer='he_normal')(conv))
# conv=(BatchNormalization()(conv))

# conv=(MaxPooling2D(pool_size=(4, 4),padding="same")(conv))

# conv=(Flatten()(conv))
# conv=(Dense(4096,activation="relu")(conv))
# conv=(Dense(4096,activation="relu")(conv))
# conv=(Dense(7, activation="softmax")(conv))
```

```
# from tensorflow.keras.models import Model
# model = Model(inputs=MainInput, outputs=conv)


# model.summary()
```

Start coding or generate with AI.

```
from tensorflow.keras.applications import EfficientNetB0,ResNet50,MobileNetV2,ResNet101
from tensorflow.keras import layers

inputs = layers.Input(shape=(28, 28, 3))
model = ResNet101(include_top=False, input_tensor=inputs, weights="imagenet")
model.trainable = False

# Rebuild top
conv = MaxPooling2D(pool_size=(2, 2),padding="same")(model.output)
conv = (BatchNormalization()(conv))


attention_layer,map2 = SoftAttention(aggregate=True,m=16,concat_with_x=False,ch=int(conv.shape[-1]),name='soft_attention')(conv)
attention_layer=(MaxPooling2D(pool_size=(2, 2),padding="same")(attention_layer))
conv=(MaxPooling2D(pool_size=(2, 2),padding="same")(conv))

conv = concatenate([conv,attention_layer])
conv=Activation("relu")(conv)
conv= Dropout(0.5)(conv)

conv=(Conv2D(filters=512,kernel_size=(3,3), activation="relu",padding="same",kernel_initializer='he_normal')(conv))
conv=(BatchNormalization()(conv))
conv=(Conv2D(filters=512,kernel_size=(3,3), activation="relu",padding="same",kernel_initializer='he_normal')(conv))
conv=(BatchNormalization()(conv))
conv=(Conv2D(filters=512,kernel_size=(1,1), activation="relu",padding="same",kernel_initializer='he_normal')(conv))
conv=(BatchNormalization()(conv))

conv=(MaxPooling2D(pool_size=(4, 4),padding="same")(conv))

conv=(Flatten()(conv))
conv=(Dense(4096,activation="relu")(conv))
conv=(Dense(4096,activation="relu")(conv))
conv=(Dense(7, activation="softmax")(conv))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet101_weights_tf_dim_ordering_tf_kerne
171446536/171446536 [==============================] - 5s 0us/step
```

```
from tensorflow.keras.models import Model
model = Model(inputs=inputs, outputs=conv,name="ResNet101")
model.summary()
```

```
 conv2d_1 (Conv2D)           (None, 1, 1, 512)            2359808   ['batch_normalization_1[0][0]
                                                                     ']

 batch_normalization_2 (Bat  (None, 1, 1, 512)            2048      ['conv2d_1[0][0]']
 chNormalization)

 conv2d_2 (Conv2D)           (None, 1, 1, 512)            262656    ['batch_normalization_2[0][0]'
                                                                     ]

 batch_normalization_3 (Bat  (None, 1, 1, 512)            2048      ['conv2d_2[0][0]']
 chNormalization)

 max_pooling2d_3 (MaxPoolin  (None, 1, 1, 512)            0         ['batch_normalization_3[0][0]'
 g2D)                                                                ]

 flatten (Flatten)           (None, 512)                  0         ['max_pooling2d_3[0][0]']

 dense (Dense)               (None, 4096)                 2101248   ['flatten[0][0]']

 dense_1 (Dense)             (None, 4096)                 1678131   ['dense[0][0]']
                                                          2

 dense_2 (Dense)             (None, 7)                    28679     ['dense_1[0][0]']

==================================================================================================
Total params: 83376023 (318.05 MB)
Trainable params: 40710679 (155.30 MB)
Non-trainable params: 42665344 (162.76 MB)
__________________________________________________________________________________________________
```

Start coding or generate with AI.

Model Training

```
# model = Sequential()
# model.add(Conv2D(16, kernel_size = (3,3), input_shape = (28, 28, 3), activation = 'relu', padding = 'same'))
# model.add(Conv2D(32, kernel_size = (3,3), activation = 'relu'))
# model.add(MaxPool2D(pool_size = (2,2)))
# model.add(Conv2D(32, kernel_size = (3,3), activation = 'relu', padding = 'same'))
# model.add(Conv2D(64, kernel_size = (3,3), activation = 'relu'))
# model.add(MaxPool2D(pool_size = (2,2), padding = 'same'))
# model.add(Flatten())
# model.add(Dense(64, activation='relu'))
# model.add(Dense(32, activation='relu'))
# model.add(Dense(7, activation='softmax'))
# model.summary()
```

```
callback = tf.keras.callbacks.ModelCheckpoint(filepath='best_model.h5',
                                                  monitor='val_acc', mode='max',
                                                 verbose=1)
```

```
Y_train.shape
```

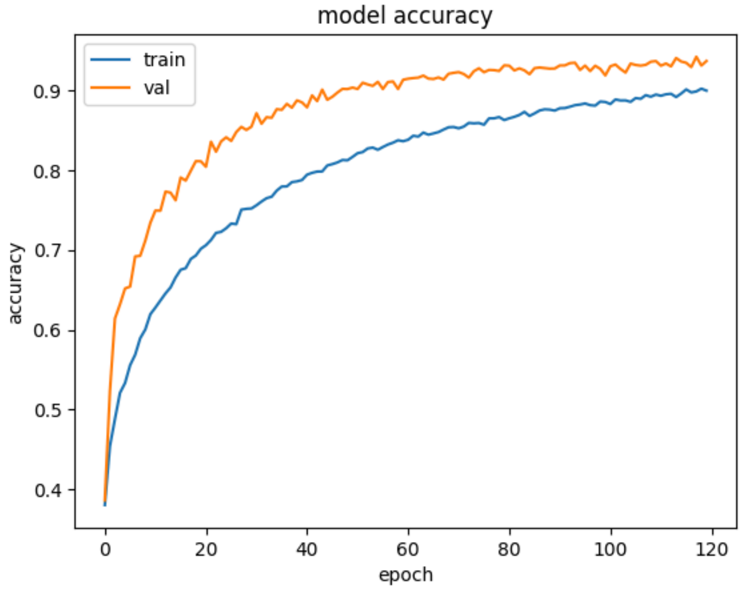
```
(37548,)
```

```
# model.compile(loss = 'sparse_categorical_crossentropy',
#               optimizer = 'adam',
#                metrics = ['accuracy'])
# history = model.fit(X_train,
#                     Y_train,
#                     validation_split=0.2,
#                     batch_size = 128,
#                     epochs = 20,
#                     callbacks=[callback])
model.compile(loss = 'sparse_categorical_crossentropy',
             optimizer = 'adam',
              metrics = ['accuracy'])
history = model.fit(X_train,
                    Y_train,
                    validation_split=0.3,
                    batch_size = 128,
                    epochs = 120,
                    callbacks=[callback])
```

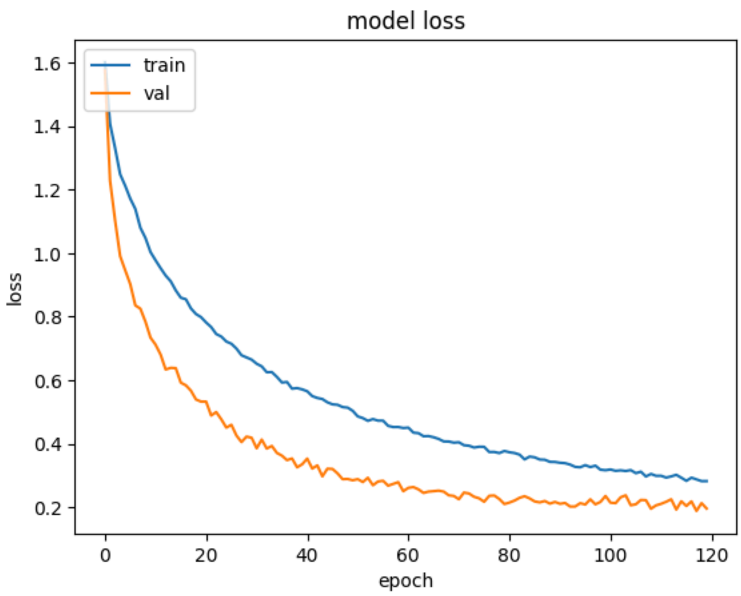
```
Epoch 108: saving model to best_model.h5
206/206 [==============================] - 33s 162ms/step - loss: 0.2966 - accuracy: 0.8940 - val_loss: 0.2219 - val_accuracy: 0.
Epoch 109/120
206/206 [==============================] - ETA: 0s - loss: 0.3046 - accuracy: 0.8920
Epoch 109: saving model to best_model.h5
206/206 [==============================] - 26s 125ms/step - loss: 0.3046 - accuracy: 0.8920 - val_loss: 0.1950 - val_accuracy: 0.
Epoch 110/120
206/206 [==============================] - ETA: 0s - loss: 0.2986 - accuracy: 0.8948
Epoch 110: saving model to best_model.h5
206/206 [==============================] - 32s 157ms/step - loss: 0.2986 - accuracy: 0.8948 - val_loss: 0.2056 - val_accuracy: 0.
Epoch 111/120
206/206 [==============================] - ETA: 0s - loss: 0.2987 - accuracy: 0.8933
Epoch 111: saving model to best_model.h5
206/206 [==============================] - 27s 131ms/step - loss: 0.2987 - accuracy: 0.8933 - val_loss: 0.2105 - val_accuracy: 0.
Epoch 112/120
206/206 [==============================] - ETA: 0s - loss: 0.2931 - accuracy: 0.8951
Epoch 112: saving model to best_model.h5
206/206 [==============================] - 27s 132ms/step - loss: 0.2931 - accuracy: 0.8951 - val_loss: 0.2169 - val_accuracy: 0.
Epoch 113/120
206/206 [==============================] - ETA: 0s - loss: 0.2968 - accuracy: 0.8957
Epoch 113: saving model to best_model.h5
206/206 [==============================] - 32s 157ms/step - loss: 0.2968 - accuracy: 0.8957 - val_loss: 0.2248 - val_accuracy: 0.
Epoch 114/120
206/206 [==============================] - ETA: 0s - loss: 0.3018 - accuracy: 0.8916
Epoch 114: saving model to best_model.h5
206/206 [==============================] - 28s 137ms/step - loss: 0.3018 - accuracy: 0.8916 - val_loss: 0.1923 - val_accuracy: 0.
Epoch 115/120
206/206 [==============================] - ETA: 0s - loss: 0.2926 - accuracy: 0.8960
Epoch 115: saving model to best_model.h5
206/206 [==============================] - 29s 141ms/step - loss: 0.2926 - accuracy: 0.8960 - val_loss: 0.2185 - val_accuracy: 0.
Epoch 116/120
206/206 [==============================] - ETA: 0s - loss: 0.2829 - accuracy: 0.9012
Epoch 116: saving model to best_model.h5
206/206 [==============================] - 26s 127ms/step - loss: 0.2829 - accuracy: 0.9012 - val_loss: 0.2037 - val_accuracy: 0.
Epoch 117/120
206/206 [==============================] - ETA: 0s - loss: 0.2928 - accuracy: 0.8974
Epoch 117: saving model to best_model.h5
206/206 [==============================] - 32s 157ms/step - loss: 0.2928 - accuracy: 0.8974 - val_loss: 0.2181 - val_accuracy: 0.
Epoch 118/120
206/206 [==============================] - ETA: 0s - loss: 0.2874 - accuracy: 0.8987
Epoch 118: saving model to best_model.h5
206/206 [==============================] - 31s 149ms/step - loss: 0.2874 - accuracy: 0.8987 - val_loss: 0.1881 - val_accuracy: 0.
Epoch 119/120
206/206 [==============================] - ETA: 0s - loss: 0.2820 - accuracy: 0.9021
Epoch 119: saving model to best_model.h5
206/206 [==============================] - 33s 163ms/step - loss: 0.2820 - accuracy: 0.9021 - val_loss: 0.2127 - val_accuracy: 0.
Epoch 120/120
206/206 [==============================] - ETA: 0s - loss: 0.2820 - accuracy: 0.8997
Epoch 120: saving model to best_model.h5
206/206 [==============================] - 33s 160ms/step - loss: 0.2820 - accuracy: 0.8997 - val_loss: 0.1957 - val_accuracy: 0.
```

Plot Accuracy and Loss

```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```




```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```




```
model.save('SkinSense.h5')
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-a26de8fecd8f> in <cell line: 1>()
----> 1 model.save('SkinSense.h5')

NameError: name 'model' is not defined
```

```
model.load_weights('best_model.h5')
```

```
classes_labels=[]
for key in classes.keys():
    classes_labels.append(key)
print(classes_labels)
```

```
[4, 6, 2, 1, 5, 0, 3]
```

```
from sklearn.metrics import confusion_matrix , classification_report
```

```
y_true = np.array(Y_test)
y_pred = model.predict(X_test)
y_pred = np.array(list(map(lambda x: np.argmax(x), y_pred)))
print(y_true)
print(y_pred)
```

```
294/294 [==============================] - 11s 26ms/step
[5 1 4 ... 2 6 0]
[5 1 4 ... 2 6 0]
```

```
pred=model.predict(X_test)
pred
```

```
294/294 [==============================] - 7s 24ms/step
array([[1.03346088e-06, 2.72406269e-05, 1.43908997e-04, ...,
        5.90842159e-04, 9.99204218e-01, 2.95835798e-05],
       [9.03610180e-06, 9.99931216e-01, 2.74267113e-05, ...,
        3.04716996e-05, 1.07574033e-06, 2.44911149e-07],
       [1.98594498e-05, 3.73432995e-04, 2.63358746e-02, ...,
        9.42691684e-01, 1.55584916e-04, 3.03406678e-02],
       ...,
       [2.22518042e-06, 7.23289195e-05, 9.91607904e-01, ...,
        8.31333920e-03, 9.92855803e-07, 3.23264180e-06],
       [2.66652165e-07, 7.76843592e-07, 6.62892999e-05, ...,
        1.15143340e-02, 2.12508300e-09, 9.88418281e-01],
       [9.99999404e-01, 6.79679744e-08, 1.00183627e-07, ...,
        2.33203163e-08, 1.15260275e-11, 3.58218813e-07]], dtype=float32)
```

```
classes_labels
```

```
[4, 6, 2, 1, 5, 0, 3]
```

```
report = classification_report(y_true, y_pred)

print("\nClassification Report:")
print(report)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.95      1.00      0.97      1359
           1       0.93      0.99      0.95      1318
           2       0.89      0.92      0.91      1262
           3       0.98      1.00      0.99      1351
           4       0.95      0.69      0.80      1374
           5       0.98      1.00      0.99      1358
           6       0.88      0.96      0.92      1365

    accuracy                           0.94      9387
   macro avg       0.94      0.94      0.93      9387
weighted avg       0.94      0.94      0.93      9387
```