In order to run the model, you will have to have present with you the entire Dataset of images. You can download the images from https://www.kaggle.com/kmader/skin-cancer-mnist-ham10000. The model should take roughly 20-30 minutes to train, depending on the speed of your computer.

If you choose not to run the model, go ahead and run the cell to load in the pretrained model. But the visuals will not show and there may be some errors. A pdf is included in the file of the outputs of the notebook.

Loading in all of our packages

In [31]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from warnings import simplefilter

from sklearn.preprocessing import LabelEncoder
from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import RandomOverSampler
from sklearn.model_selection import train_test_split
from keras.preprocessing.image import ImageDataGenerator
from sklearn.metrics import confusion_matrix

import os
from glob import glob
import PIL
from PIL import Image

import tensorflow as tf
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D, BatchNormalization
import json


simplefilter(action='ignore', category=FutureWarning)
```

DATA PREPROCESSING

Reading in Data

In [3]:

```
df_meta = pd.read_csv('Data/HAM10000_metadata.csv', names = ['lesion_id', 'image_id', '
df_meta.drop(index=0, inplace=True)
df_meta = df_meta.reset_index()
df = df_meta.drop(columns='index')
df.head()
```

Out[3]:

	lesion_id	image_id	dx	dx_type	age	sex	localization
0	HAM_0000118	ISIC_0027419	bkl	histo	80.0	male	scalp
1	HAM_0000118	ISIC_0025030	bkl	histo	80.0	male	scalp
2	HAM_0002730	ISIC_0026769	bkl	histo	80.0	male	scalp

	lesion_id	image_id	dx	dx_type	age	sex	localization
3	HAM_0002730	ISIC_0025661	bkl	histo	80.0	male	scalp
4	HAM_0001466	ISIC_0031633	bkl	histo	75.0	male	ear

Checking for null values, using df.info(), we can see that age contains some null values. There are only 57 rows so we won't worry too much about them affecting the model.

In [4]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10015 entries, 0 to 10014
Data columns (total 7 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   lesion_id     10015 non-null  object
 1   image_id      10015 non-null  object
 2   dx            10015 non-null  object
 3   dx_type       10015 non-null  object
 4   age           9958 non-null   object
 5   sex           10015 non-null  object
 6   localization  10015 non-null  object
dtypes: object(7)
memory usage: 547.8+ KB
```

In [5]:

```
df = df.dropna()
df = df[~(df['sex'] == 'unknown')]
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 9948 entries, 0 to 10014
Data columns (total 7 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   lesion_id     9948 non-null   object
 1   image_id      9948 non-null   object
 2   dx            9948 non-null   object
 3   dx_type       9948 non-null   object
 4   age           9948 non-null   object
 5   sex           9948 non-null   object
 6   localization  9948 non-null   object
dtypes: object(7)
memory usage: 621.8+ KB
```

We will One-Hot-Encode the 'dx' column, this label encoder will be used later to easily decode our predictions

In [6]:

```
label_encoder = LabelEncoder()

label_encoder.fit(df['dx'])

#Create a new column named dx_encodings to hold our encoded diagnoses
df['dx_encodings'] = label_encoder.transform(df['dx'])

df.head(5)
```

Out[6]:

	lesion_id	image_id	dx	dx_type	age	sex	localization	dx_encodings

	lesion_id	image_id	dx	dx_type	age	sex	localization	dx_encodings
0	HAM_0000118	ISIC_0027419	bkl	histo	80.0	male	scalp	2
1	HAM_0000118	ISIC_0025030	bkl	histo	80.0	male	scalp	2
2	HAM_0002730	ISIC_0026769	bkl	histo	80.0	male	scalp	2
3	HAM_0002730	ISIC_0025661	bkl	histo	80.0	male	scalp	2
4	HAM_0001466	ISIC_0031633	bkl	histo	75.0	male	ear	2

In [7]:

```
#Checking which numbers correspond to the diagnosis
label_encoder.inverse_transform([0,1,2,3,4,5,6])
```

Out[7]:

```
array(['akiec', 'bcc', 'bkl', 'df', 'mel', 'nv', 'vasc'], dtype=object)
```

Exploratory Data Analysis for Insights into our Data

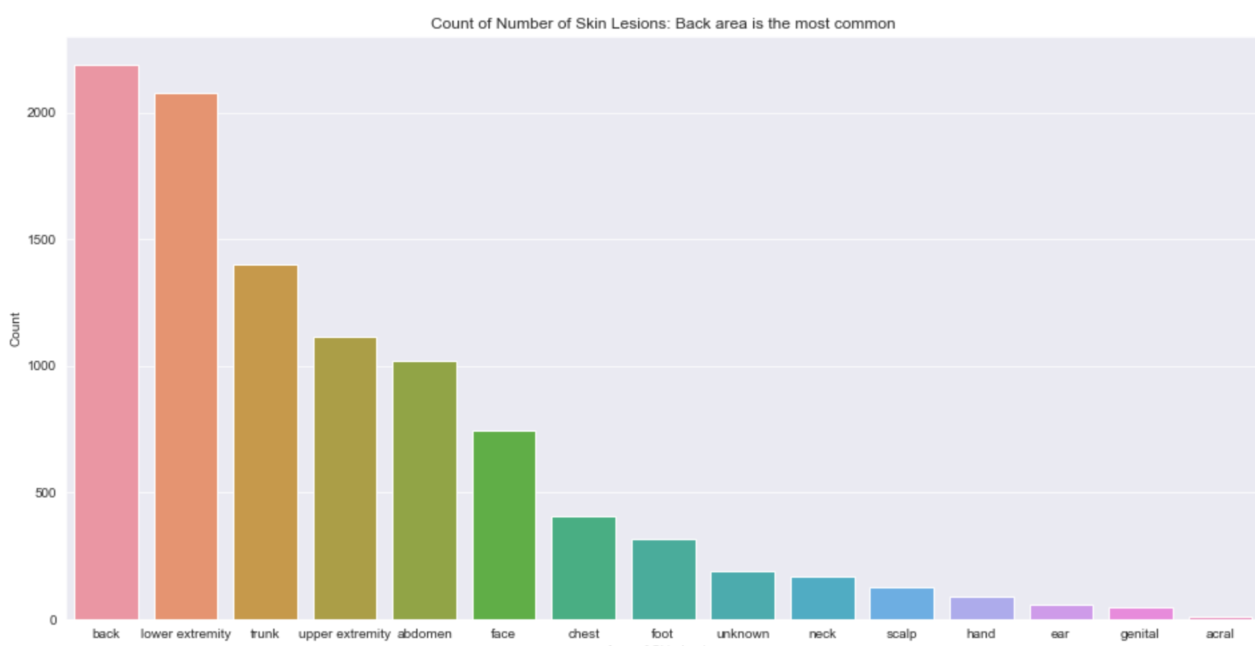
Looking at the most common areas where the skin lesions occur

In [8]:

```
plt.figure(figsize=(16,8))
sns.set_style("darkgrid")

ax = sns.barplot(x = df['localization'].value_counts().index, y = df['localization'].va
ax.set_xlabel('Area of Skin Lesion')
ax.set_ylabel('Count')
ax.set_title("Count of Number of Skin Lesions: Back area is the most common")
```

Out[8]:

```
Text(0.5, 1.0, 'Count of Number of Skin Lesions: Back area is the most common')
```

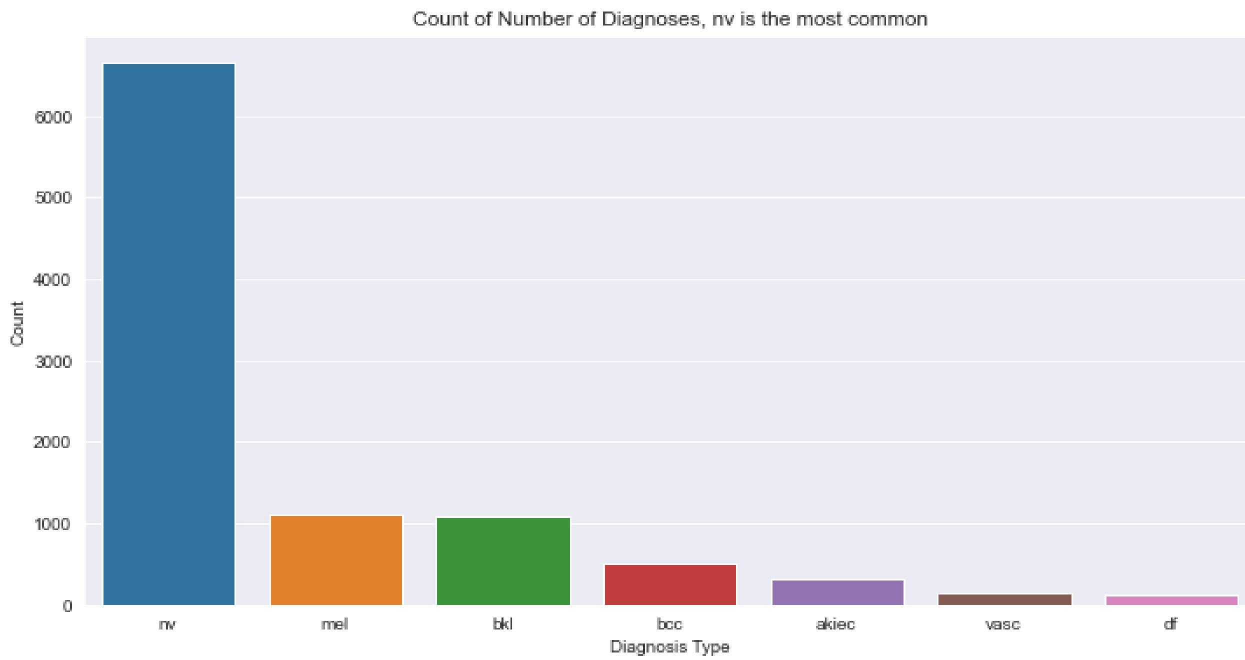


Looking at the amount of different kinds of diagnoses

In [9]:

```
plt.figure(figsize=(12,6))
sns.set_style("darkgrid")

ax1 = sns.barplot(x = df['dx'].value_counts().index, y = df['dx'].value_counts())
ax1.set_xlabel('Diagnosis Type')
ax1.set_ylabel('Count')
ax1.set_title("Count of Number of Diagnoses, nv is the most common")
```

Out[9]:
```
Text(0.5, 1.0, 'Count of Number of Diagnoses, nv is the most common')
```



In [10]:
```
nv_percentage = 100 * len(df[df['dx'] == 'nv']) / len(df)
nv_percentage = '{0:.4g}'.format(nv_percentage) + "%"

print('dx type nv makes up', nv_percentage ,'of the database')
```

```
dx type nv makes up 66.85% of the database
```

From the Description of our diagnoses types, we know that these labels mean this

- melanocytic nevi (nv)
- melanoma (mel)
- benign keratosis-like lesions (bkl)
- basal cell carcinoma (bcc)
- Actinic keratoses and intraepithelial carcinoma / Bowen's disease (akiec)
- vascular lesions (vasc)
- dermatofibroma (df)

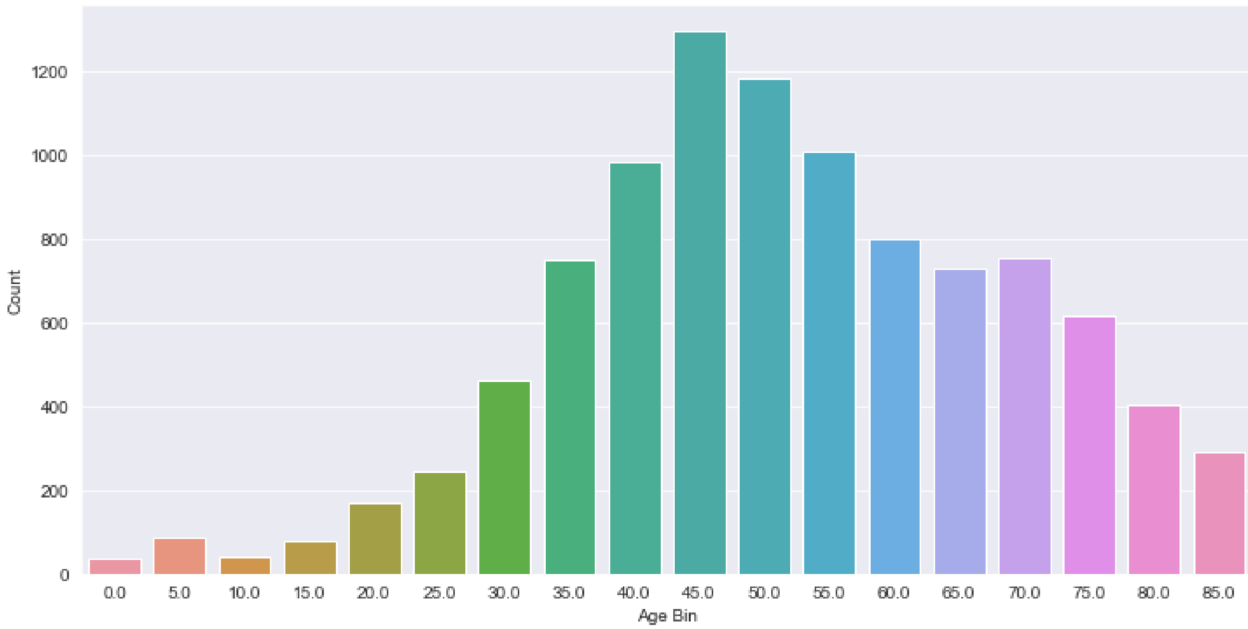
And melanocytic nevi makes up about 2/3rds of all our dx counts, so our data is largely imbalanced, but luckily we have some tools to account for problems such as imbalanced data

Taking a look at the Age distribution, 40-50 are the most commmon

In [11]:
```
plt.figure(figsize=(12,6))
sns.set_style("darkgrid")
```

```
df_age = df['age'].value_counts()
df_age.index = df_age.index.astype(float)
df_age = df_age.sort_index(ascending=True)

ax2 = sns.barplot(x = df_age.index, y = df_age)
ax2.set_xlabel("Age Bin")
ax2.set_ylabel("Count")
```

Out[11]: Text(0, 0.5, 'Count')



Looking at genders to see if there's an imbalance

In [12]:

```
df['sex'].value_counts()
```

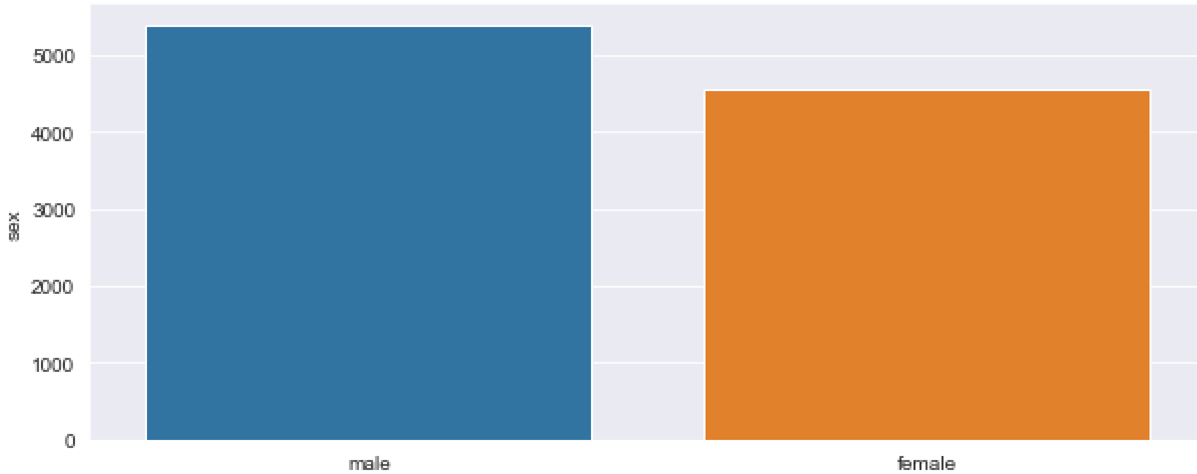
Out[12]:

```
male      5400
female    4548
Name: sex, dtype: int64
```

In [13]:

```
plt.figure(figsize=(10,4))
sns.set_style("darkgrid")

sns.barplot(x = df['sex'].value_counts().index, y= df['sex'].value_counts())
```

Out[13]: <AxesSubplot:ylabel='sex'>



LEARNINGS FROM EDA

A huge problem that stands out from this data set is that there is a major imbalance in the dx columns. Melanocytic nevi accounts for the majority of values in our dx column. To account for this, we will have to use certain techniques like image generation, oversampling, and weighting the classes.

CONVERTING JPG IMAGES TO RGB PIXEL DATA

We will be resizing our images to 32 x 32 images so we can fit all the picutres into our input layers of our model

In [14]:

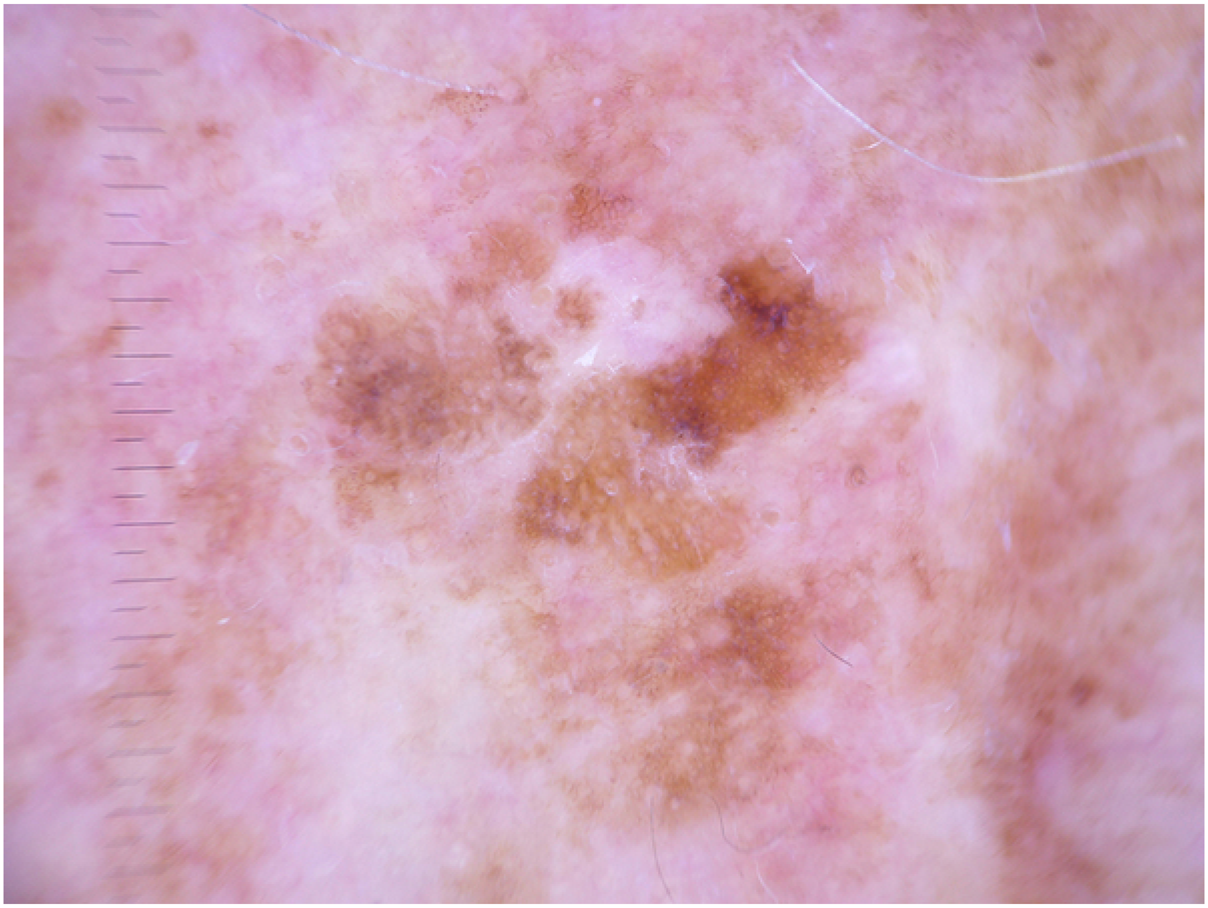
```
IMAGE_SIZE = 32

# This cell finds all of the jpg images in directory and creates a full path of where t
image_path = {os.path.splitext(os.path.basename(x))[0]: x
               for x in glob(os.path.join('Data/', '*', '*.jpg'))}

# Creating a new column in our dataframe and our dictionary to set the respective file 
df['image_path'] = df['image_id'].map(image_path.get)

# We create a lambda function to open each image file in our 'image_path' column and co
df['image_data'] = df['image_path'].map(lambda x: np.asarray(Image.open(x).resize((IMAG
```

Comparing Original Image to Pixel Image

In [15]:

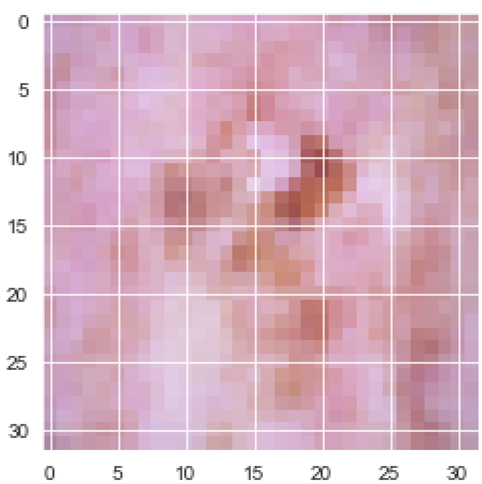
```
PIL.Image.open(df['image_path'].iloc[0])
```

Out[15]:



In [16]:

```
# Image Pixel Data mapped out into a 32 x 32 rgb image

plt.imshow(df['image_data'].iloc[0].reshape(32,32,3))
plt.show()
```



NORMALIZING PIXEL DATA AND SPLITTING INTO TRAINING AND TESTING

In [143…

```
X = np.asarray(df['image_data'].tolist())

X = X / 255.
```

```python
# One-Hot-Encoding our Labels
Y = df['dx_encodings']
Y = tf.keras.utils.to_categorical(Y, num_classes=7)

X_train, X_test, y_train, y_test = train_test_split(X,Y,test_size= .3, random_state = 3

print("X_train:", len(X_train),", y_train: ", len(y_train),", X_test: ", len(X_test),",
```

```
X_train: 6963 , y_train:  6963 , X_test:  2985 , y_test: 2985
```

In [144…

```python
#Successfully Encoded our labels
pd.DataFrame(Y).head(5)
```

Out[144…

	0	1	2	3	4	5	6
0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
1	0.0	0.0	1.0	0.0	0.0	0.0	0.0
2	0.0	0.0	1.0	0.0	0.0	0.0	0.0
3	0.0	0.0	1.0	0.0	0.0	0.0	0.0
4	0.0	0.0	1.0	0.0	0.0	0.0	0.0

Splitting our Data into train and testing datasets

BALANCING OUR DATA

Using Random Under Sampler, it will undersample from dx types such as 'nv' and oversample other minority 'dx' classes.

In [145…

```python
pd.DataFrame(y_train).value_counts()
```

Out[145…

```
0    1    2    3    4    5    6
0.0  0.0  0.0  0.0  0.0  1.0  0.0    4635
          1.0  0.0  0.0  0.0  0.0     796
          0.0  0.0  1.0  0.0  0.0     789
     1.0  0.0  0.0  0.0  0.0  0.0     350
1.0  0.0  0.0  0.0  0.0  0.0  0.0     218
0.0  0.0  0.0  0.0  0.0  0.0  1.0      97
               1.0  0.0  0.0  0.0      78
dtype: int64
```

First use undersampling, to cut down the values of our Majority class.

We can actually exclude this part, but this would result in having a very large data set, which would cause our model to take about an hour and a half to train

In [148…

```python
#Our data set is heavily imbalanced, so we will first undersample from the majority cla

sampling_strategy = {5: 3000}

#Number chosen after trial and error, experimenting with different undersampling thresh

num_samples, dim_x, dim_y, dim_z = X_train.shape
```

```
X_train = X_train.reshape((num_samples,dim_x*dim_y*dim_z))

random_undersampler = RandomUnderSampler(sampling_strategy=sampling_strategy)

X_train, y_train = random_undersampler.fit_resample(X_train, y_train)

#new_length = int(X_train.size / (32 * 32 * 3))

X_train = X_train.reshape((len(X_train), dim_x,dim_y,dim_z))
```

In [149…

```
num_samples, dim_x, dim_y, dim_z = X_train.shape

X_train = X_train.reshape((num_samples,dim_x*dim_y*dim_z))

random_oversampler = RandomOverSampler()

X_train, y_train = random_oversampler.fit_resample(X_train, y_train)

X_train = X_train.reshape((len(X_train),dim_x,dim_y,dim_z))
```

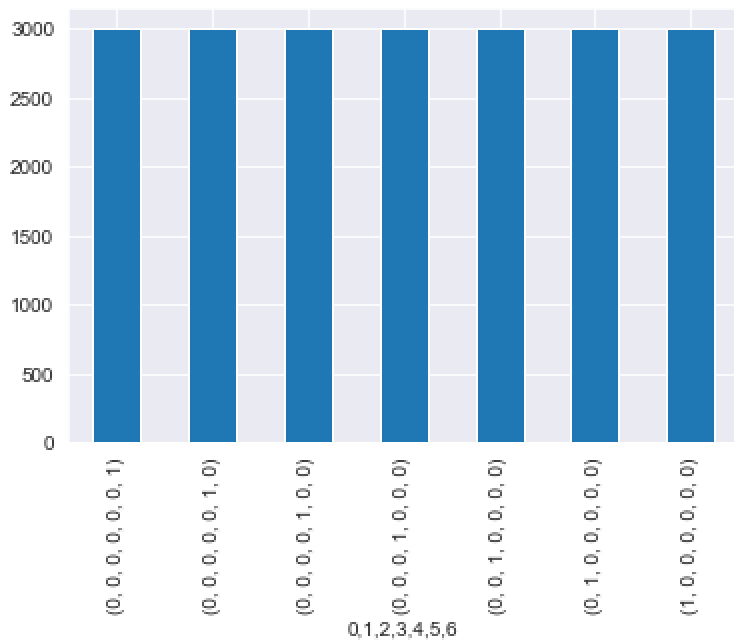
The classes are now balanced at 3000 samples each

In [150…

```
pd.DataFrame(y).value_counts().plot(kind='bar')
```

Out[150…

```
<AxesSubplot:xlabel='0,1,2,3,4,5,6'>
```



Final length of our training data

In [152…

```
print("X_train:", len(X_train),", y_train: ", len(y_train))
```

```
X_train: 21000 , y_train:  21000
```

Convolutional Neural Networks are not Scale or Rotation Invariant, to account for this, we use Data Augmentation to prevent overfitting

TRAINING AND TESTING THE MODEL

Generate Images Through Image Generator

In [153…

```
Image_Data_Generator = ImageDataGenerator(height_shift_range = .15,
                                          width_shift_range = .15,
                                          horizontal_flip = True,
                                          vertical_flip = True,
                                          rotation_range = 30,
                                          zoom_range = .1)

Image_Data_Generator.fit(X_train)
```

In [51]:

```
print(len(X_train), len(X_test), len(y_train), len(y_test))
```

```
6963 2985 6963 2985
```

In [156…

```
input_shape = (IMAGE_SIZE, IMAGE_SIZE , 3)

model = Sequential([
    #Input Layer
    Conv2D(64, kernel_size = (3, 3), padding ='same',activation="relu", input_shape=inp
    Conv2D(64, kernel_size = (3, 3), padding ='same',activation="relu"),
    MaxPool2D(pool_size=(2, 2)),
    BatchNormalization(),

    Conv2D(128, kernel_size = (3, 3), padding ='same',activation="relu"),
    MaxPool2D(pool_size=(2, 2)),
    BatchNormalization(),

    Conv2D(256, kernel_size = (3, 3),padding ='same',activation='relu'),
    MaxPool2D(pool_size=(2, 2)),
    BatchNormalization(),

    Conv2D(64, kernel_size = (3, 3),padding ='same',activation='relu'),
    Conv2D(64, kernel_size = (3, 3),padding ='same',activation='relu'),
    MaxPool2D(pool_size=(2, 2)),
    Dropout(.25),
    BatchNormalization(),

    Flatten(),
    Dense(128, activation = 'relu'),
    Dense(64, activation ='relu'),
    Dense(7,activation = 'softmax')
])

model.summary()

model.compile(loss='categorical_crossentropy', optimizer='Adam', metrics=['acc'])
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                Output Shape              Param #
=================================================================
conv2d_6 (Conv2D)           (None, 32, 32, 64)        1792
_________________________________________________________________
conv2d_7 (Conv2D)           (None, 32, 32, 64)        36928
_________________________________________________________________
```

```
max_pooling2d_4 (MaxPooling2 (None, 16, 16, 64)         0
_________________________________________________________________
batch_normalization_4 (Batch (None, 16, 16, 64)         256
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 16, 16, 128)        73856
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 8, 8, 128)          0
_________________________________________________________________
batch_normalization_5 (Batch (None, 8, 8, 128)          512
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 8, 8, 256)          295168
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 4, 4, 256)          0
_________________________________________________________________
batch_normalization_6 (Batch (None, 4, 4, 256)          1024
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 4, 4, 64)           147520
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 4, 4, 64)           36928
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 2, 2, 64)           0
_________________________________________________________________
dropout_1 (Dropout)          (None, 2, 2, 64)           0
_________________________________________________________________
batch_normalization_7 (Batch (None, 2, 2, 64)           256
_________________________________________________________________
flatten_1 (Flatten)          (None, 256)                0
_________________________________________________________________
dense_3 (Dense)              (None, 128)                32896
_________________________________________________________________
dense_4 (Dense)              (None, 64)                 8256
_________________________________________________________________
dense_5 (Dense)              (None, 7)                  455
=================================================================
Total params: 635,847
Trainable params: 634,823
Non-trainable params: 1,024
_________________________________________________________________
```

In [139…

```
batch_size = 64
epochs = 50

history = model.fit(
    X_train, y_train,
    epochs=epochs,
    batch_size = batch_size,
    validation_data=(X_test, y_test),
    verbose=2)
```

```
Epoch 1/50
230/230 - 43s - loss: 1.6046 - acc: 0.3716 - val_loss: 3.0057 - val_acc: 0.1429
Epoch 2/50
230/230 - 42s - loss: 1.1211 - acc: 0.5751 - val_loss: 1.5550 - val_acc: 0.4310
Epoch 3/50
230/230 - 40s - loss: 0.8883 - acc: 0.6651 - val_loss: 1.5358 - val_acc: 0.4879
Epoch 4/50
230/230 - 40s - loss: 0.7100 - acc: 0.7305 - val_loss: 1.9904 - val_acc: 0.4622
Epoch 5/50
230/230 - 40s - loss: 0.6072 - acc: 0.7730 - val_loss: 2.4635 - val_acc: 0.4121
Epoch 6/50
230/230 - 43s - loss: 0.5116 - acc: 0.8047 - val_loss: 0.4820 - val_acc: 0.8125
Epoch 7/50
230/230 - 40s - loss: 0.4421 - acc: 0.8300 - val_loss: 1.5101 - val_acc: 0.5814
```

```
Epoch 8/50
230/230 - 40s - loss: 0.4069 - acc: 0.8449 - val_loss: 1.8128 - val_acc: 0.5095
Epoch 9/50
230/230 - 40s - loss: 0.3642 - acc: 0.8627 - val_loss: 0.4595 - val_acc: 0.8317
Epoch 10/50
230/230 - 40s - loss: 0.3321 - acc: 0.8707 - val_loss: 0.4310 - val_acc: 0.8395
Epoch 11/50
230/230 - 41s - loss: 0.3052 - acc: 0.8851 - val_loss: 0.5837 - val_acc: 0.7841
Epoch 12/50
230/230 - 44s - loss: 0.2757 - acc: 0.8969 - val_loss: 0.7307 - val_acc: 0.7629
Epoch 13/50
230/230 - 44s - loss: 0.2721 - acc: 0.8980 - val_loss: 0.3113 - val_acc: 0.8852
Epoch 14/50
230/230 - 42s - loss: 0.2212 - acc: 0.9171 - val_loss: 0.2527 - val_acc: 0.9083
Epoch 15/50
230/230 - 44s - loss: 0.2134 - acc: 0.9199 - val_loss: 0.2637 - val_acc: 0.9140
Epoch 16/50
230/230 - 41s - loss: 0.2029 - acc: 0.9239 - val_loss: 0.6037 - val_acc: 0.7971
Epoch 17/50
230/230 - 40s - loss: 0.1923 - acc: 0.9273 - val_loss: 0.5075 - val_acc: 0.8351
Epoch 18/50
230/230 - 40s - loss: 0.1965 - acc: 0.9250 - val_loss: 1.2036 - val_acc: 0.7019
Epoch 19/50
230/230 - 40s - loss: 0.1656 - acc: 0.9376 - val_loss: 0.2878 - val_acc: 0.9000
Epoch 20/50
230/230 - 41s - loss: 0.1640 - acc: 0.9401 - val_loss: 0.9488 - val_acc: 0.7451
Epoch 21/50
230/230 - 43s - loss: 0.1479 - acc: 0.9438 - val_loss: 2.0427 - val_acc: 0.6146
Epoch 22/50
230/230 - 43s - loss: 0.1451 - acc: 0.9465 - val_loss: 0.6001 - val_acc: 0.8217
Epoch 23/50
230/230 - 43s - loss: 0.1366 - acc: 0.9514 - val_loss: 0.2195 - val_acc: 0.9295
Epoch 24/50
230/230 - 43s - loss: 0.1335 - acc: 0.9505 - val_loss: 0.2214 - val_acc: 0.9360
Epoch 25/50
230/230 - 43s - loss: 0.1218 - acc: 0.9535 - val_loss: 0.1974 - val_acc: 0.9410
Epoch 26/50
230/230 - 43s - loss: 0.1205 - acc: 0.9549 - val_loss: 0.4295 - val_acc: 0.8732
Epoch 27/50
230/230 - 44s - loss: 0.1221 - acc: 0.9562 - val_loss: 0.4078 - val_acc: 0.8637
Epoch 28/50
230/230 - 44s - loss: 0.1047 - acc: 0.9603 - val_loss: 0.3128 - val_acc: 0.9044
Epoch 29/50
230/230 - 43s - loss: 0.1004 - acc: 0.9625 - val_loss: 0.2867 - val_acc: 0.9168
Epoch 30/50
230/230 - 43s - loss: 0.1073 - acc: 0.9612 - val_loss: 0.3968 - val_acc: 0.8825
Epoch 31/50
230/230 - 42s - loss: 0.0960 - acc: 0.9641 - val_loss: 0.2453 - val_acc: 0.9313
Epoch 32/50
230/230 - 43s - loss: 0.0915 - acc: 0.9657 - val_loss: 0.2538 - val_acc: 0.9376
Epoch 33/50
230/230 - 42s - loss: 0.0902 - acc: 0.9664 - val_loss: 0.2043 - val_acc: 0.9494
Epoch 34/50
230/230 - 42s - loss: 0.0841 - acc: 0.9686 - val_loss: 0.2122 - val_acc: 0.9468
Epoch 35/50
230/230 - 43s - loss: 0.0928 - acc: 0.9678 - val_loss: 0.3604 - val_acc: 0.9010
Epoch 36/50
230/230 - 43s - loss: 0.0903 - acc: 0.9656 - val_loss: 0.3255 - val_acc: 0.9025
Epoch 37/50
230/230 - 43s - loss: 0.0696 - acc: 0.9761 - val_loss: 0.2767 - val_acc: 0.9316
Epoch 38/50
230/230 - 42s - loss: 0.0745 - acc: 0.9729 - val_loss: 0.2227 - val_acc: 0.9457
Epoch 39/50
230/230 - 43s - loss: 0.0675 - acc: 0.9768 - val_loss: 0.2129 - val_acc: 0.9386
Epoch 40/50
```

```
230/230 - 40s - loss: 0.0618 - acc: 0.9785 - val_loss: 0.2247 - val_acc: 0.9441
Epoch 41/50
230/230 - 42s - loss: 0.0673 - acc: 0.9766 - val_loss: 0.3966 - val_acc: 0.9125
Epoch 42/50
230/230 - 43s - loss: 0.0693 - acc: 0.9767 - val_loss: 0.2506 - val_acc: 0.9395
Epoch 43/50
230/230 - 41s - loss: 0.0738 - acc: 0.9723 - val_loss: 0.2644 - val_acc: 0.9316
Epoch 44/50
230/230 - 42s - loss: 0.0597 - acc: 0.9797 - val_loss: 0.3979 - val_acc: 0.8933
Epoch 45/50
230/230 - 41s - loss: 0.0558 - acc: 0.9806 - val_loss: 0.2345 - val_acc: 0.9476
Epoch 46/50
230/230 - 41s - loss: 0.0584 - acc: 0.9801 - val_loss: 0.5700 - val_acc: 0.8641
Epoch 47/50
230/230 - 42s - loss: 0.0590 - acc: 0.9788 - val_loss: 0.3603 - val_acc: 0.9148
Epoch 48/50
230/230 - 40s - loss: 0.0620 - acc: 0.9777 - val_loss: 0.2142 - val_acc: 0.9506
Epoch 49/50
230/230 - 40s - loss: 0.0549 - acc: 0.9815 - val_loss: 0.2818 - val_acc: 0.9265
Epoch 50/50
230/230 - 40s - loss: 0.0514 - acc: 0.9819 - val_loss: 0.2563 - val_acc: 0.9430
```

Run this cell to load in the model

model = keras.models.load_model('CNN_skin_lesion_model')

In [150…

```
score = model.evaluate(X_test, y_test)
print('Test accuracy:', score[1])
```

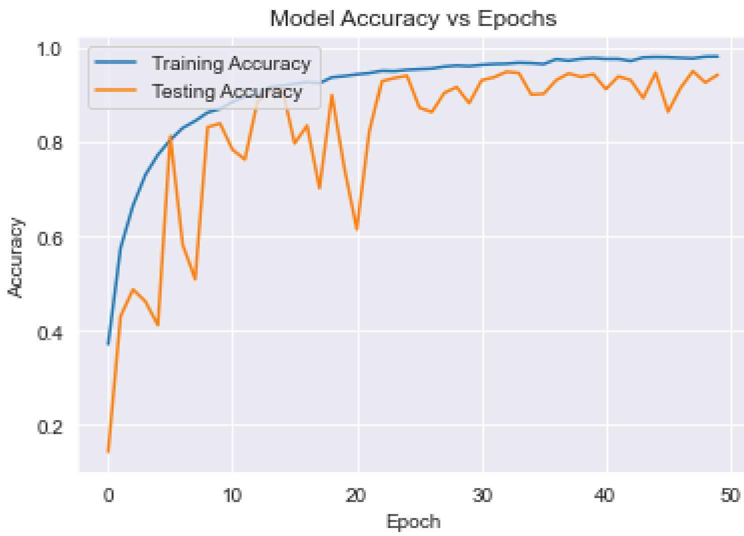
```
197/197 [==============================] - 9s 44ms/step - loss: 0.2563 - acc: 0.9430: 0s
- loss: 0.2581 - acc
Test accuracy: 0.9430158734321594
```

In [151…

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

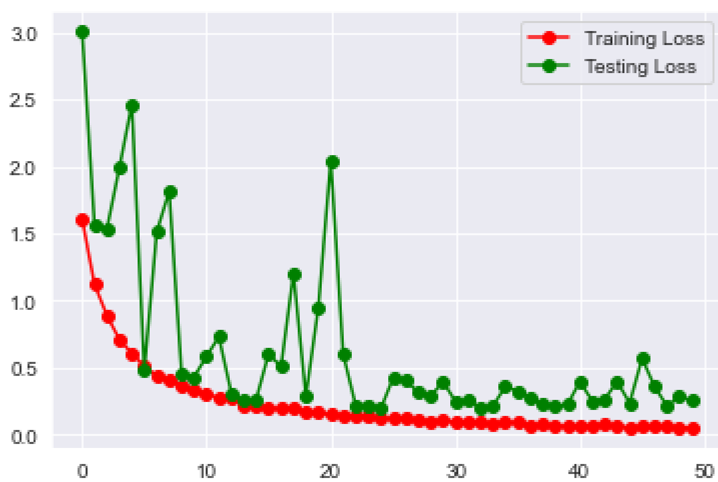
In [153…

```
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('Model Accuracy vs Epochs')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Training Accuracy', 'Testing Accuracy'], loc='upper left')
plt.show()
```



In [143…

```
plt.plot(history.history["loss"] , 'ro-' , label = "Training Loss")
plt.plot(history.history["val_loss"] , 'go-' , label = "Testing Loss")
plt.legend()
plt.show()
```



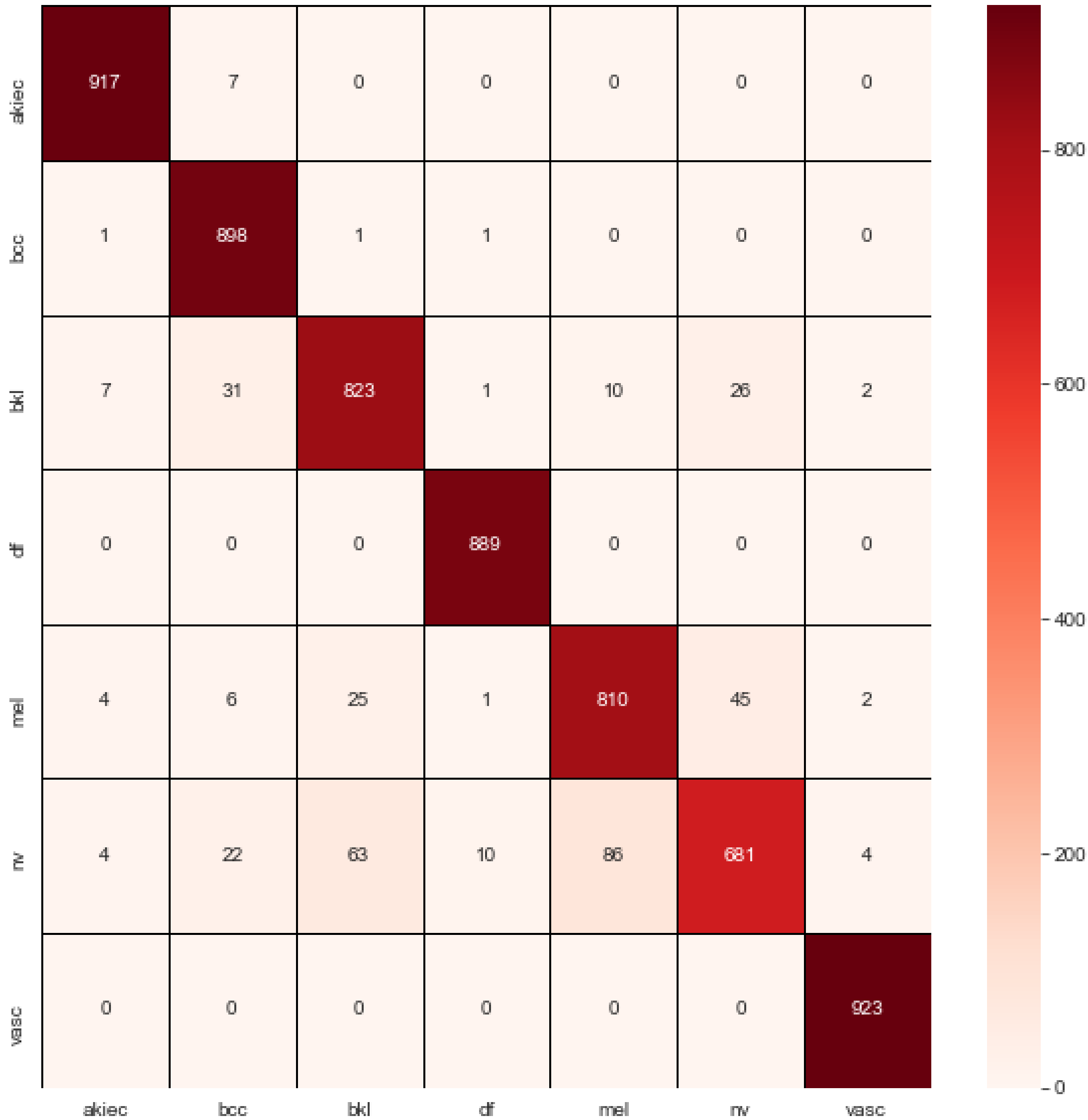
model.save('CNN_skin_lesion_model_Image_Gen

In [162…

```
cm = confusion_matrix(y_test.argmax(axis = 1) , y_pred.argmax(axis = 1))
cm = pd.DataFrame(cm , index = ['akiec','bcc', 'bkl', 'df','mel' ,'nv', 'vasc'] , colum
plt.figure(figsize = (10,10))
sns.heatmap(cm,cmap= "Reds", linecolor = 'black' , linewidth = 1 , annot = True, fmt=''
```

Out[162…

```
<AxesSubplot:>
```


akiec
bcc
bkl
df
mel
nv
vasc
917
7
0
0
0
0
0
1
898
1
1
0
0
0
7
31
823
1
10
26
2
0
0
0
889
0
0
0
4
6
25
1
810
45
2
4
22
63
10
86
681
4
0
0
0
0
0
0
923
akiec
bcc
bkl
df
mel
nv
vasc
800
600
400
200
0