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
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
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])
```

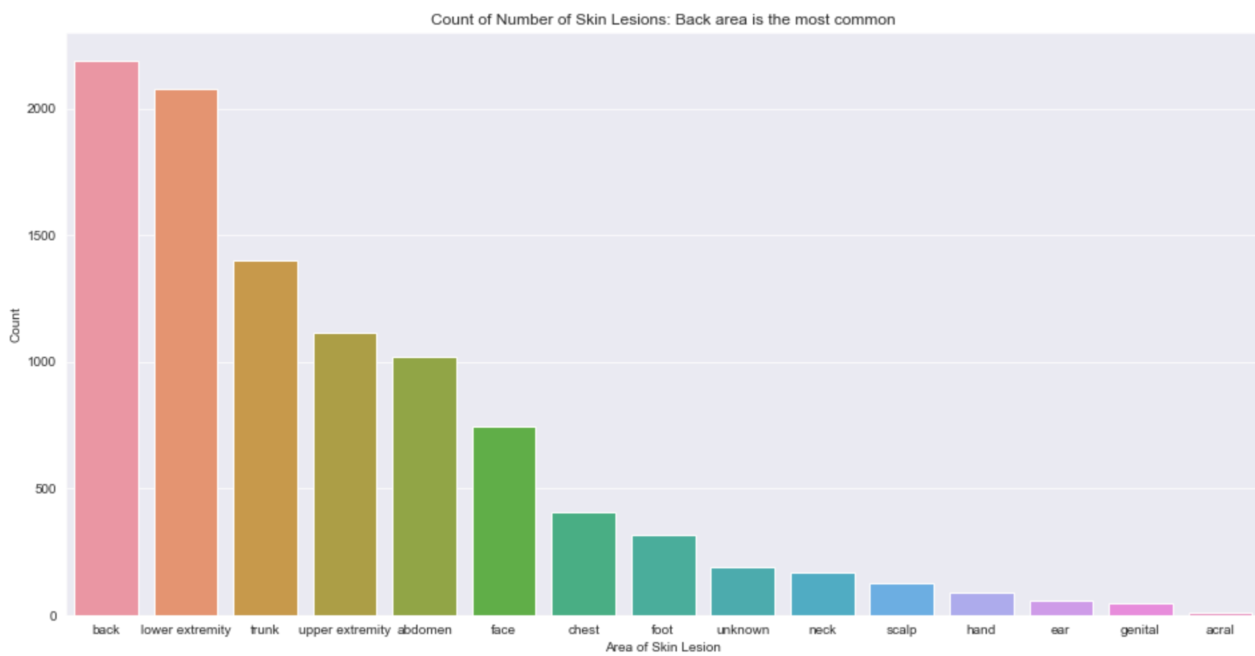
```
Out[7]: array(['akiec', 'bcc', 'bkl', 'df', 'mel', 'nv', 'vasc'], dtype=object)
```

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")
```

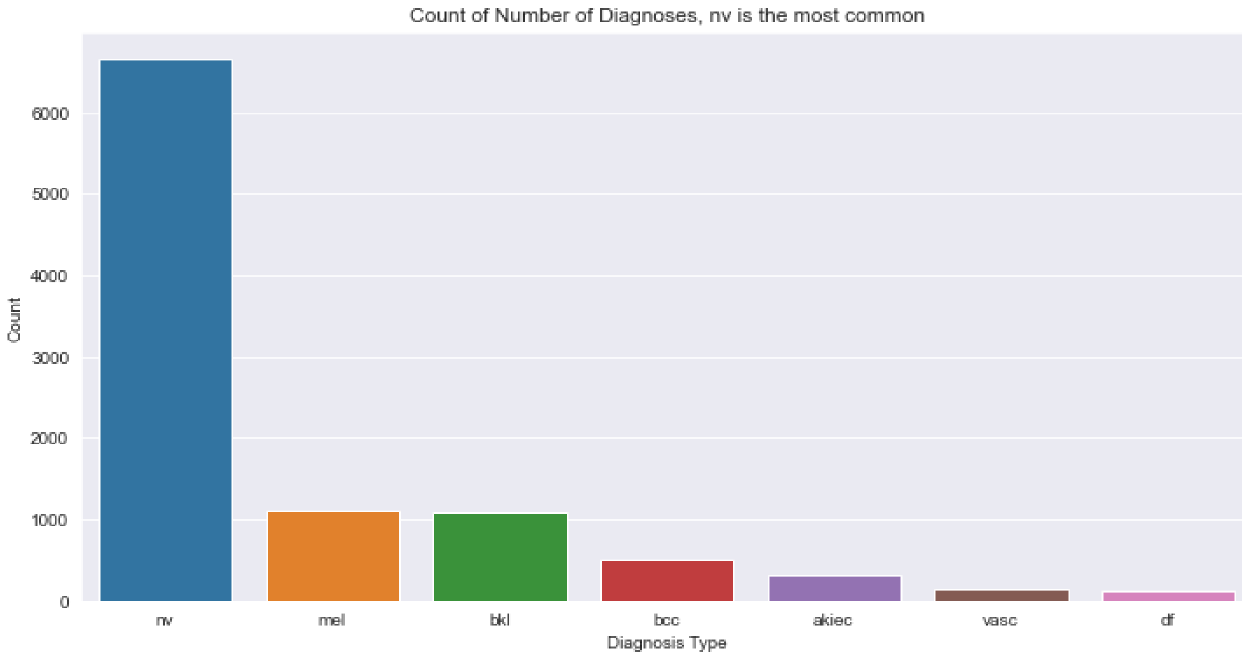
```
Out[8]: Text(0.5, 1.0, 'Count of Number of Skin Lesions: Back area is the most common')
```



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

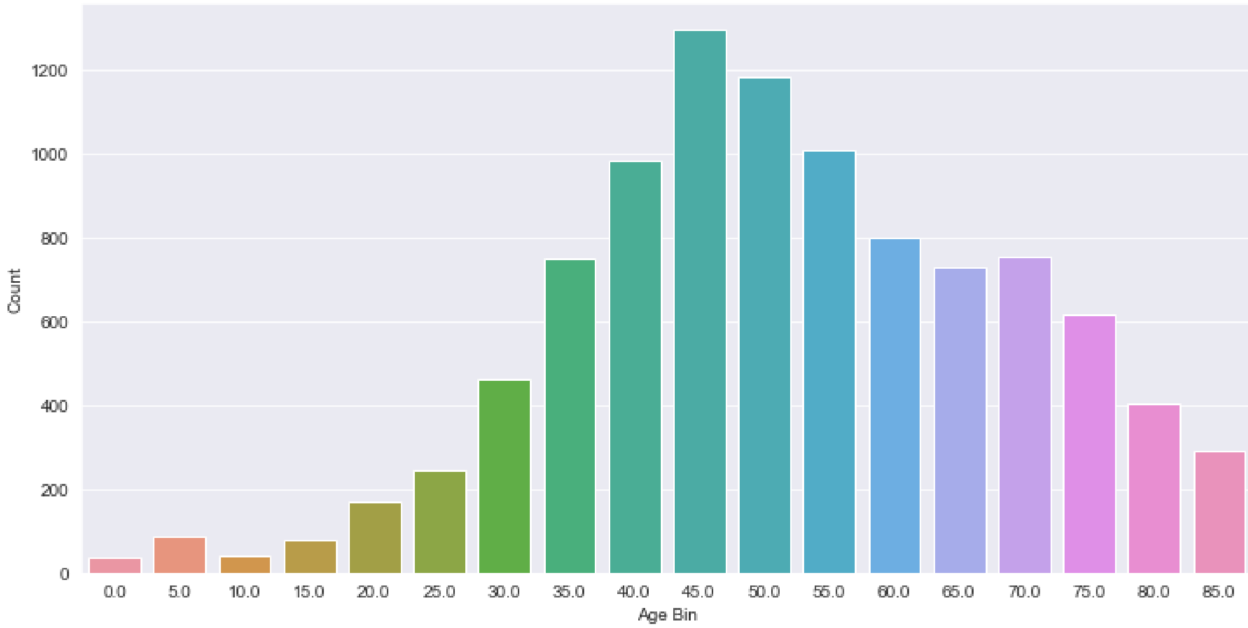
- melanocytic nevi (nv)
- melanoma (mel)
- benign keratosis-like lesions (bkl)
- basal cell carcinoma (bcc)
- Actinic keratoses and intraepithelial carcinoma / Bowen's disease (akiec)
- vascular lesions (vasc)
- dermatofibroma (df)

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()
```

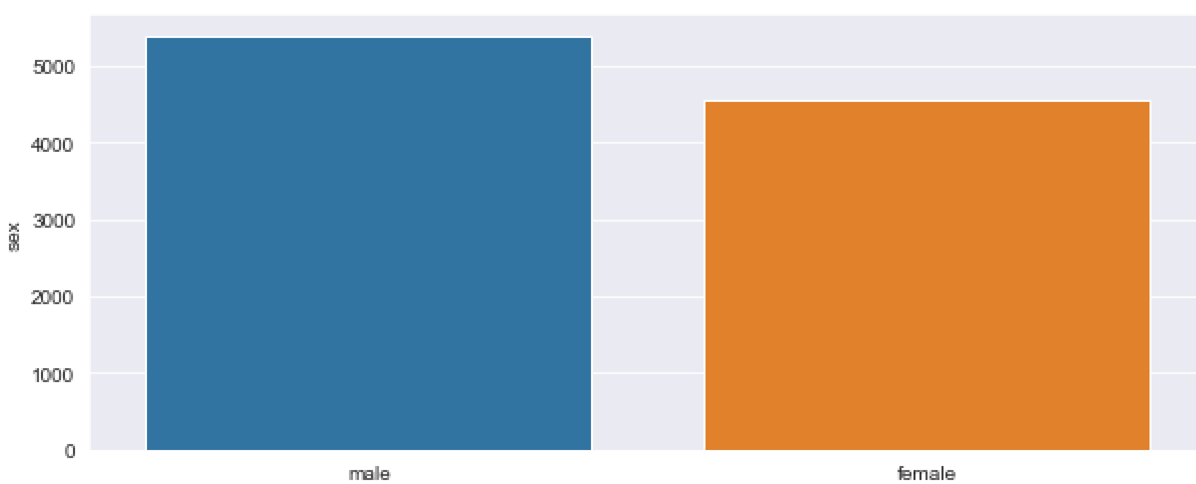
Out[12]:

```
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]:

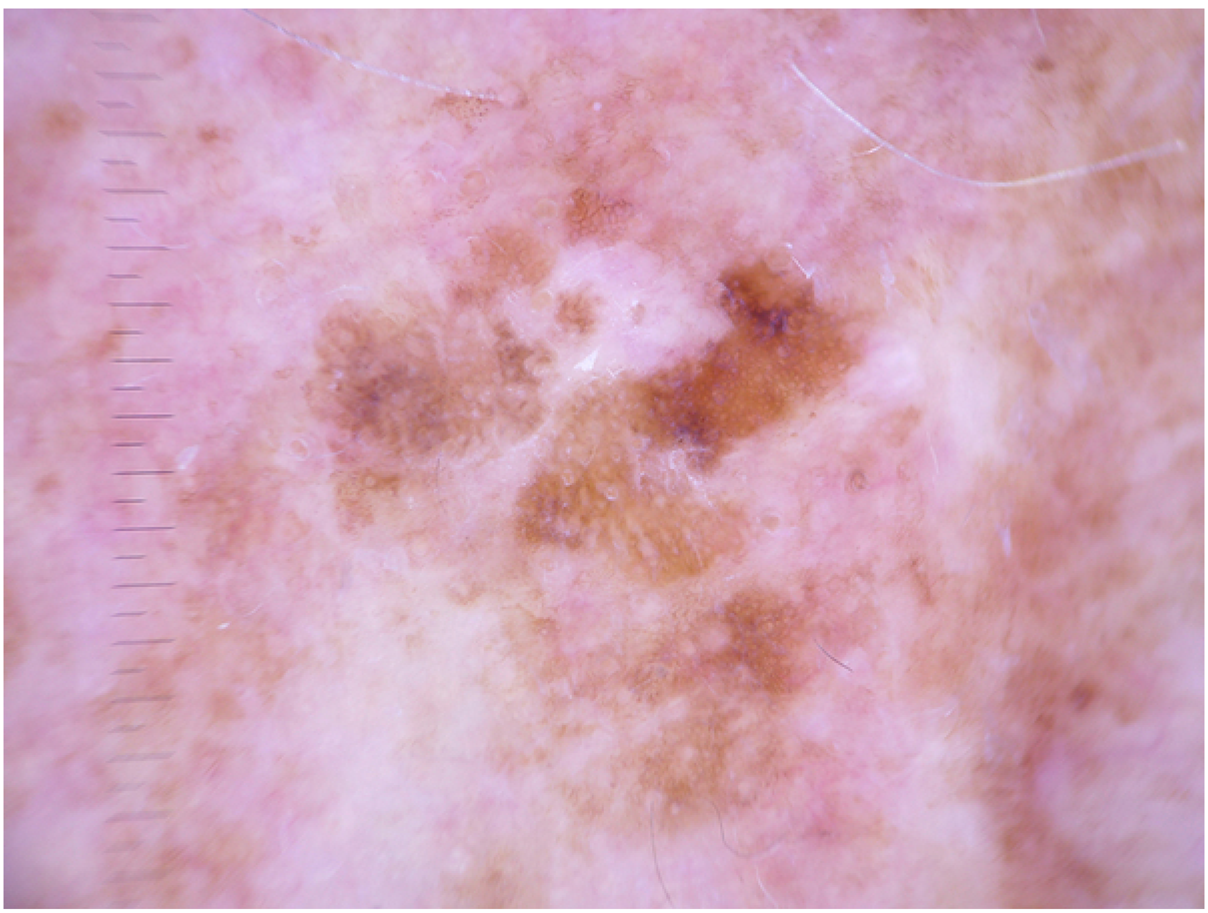
```
IMAGE_SIZE = 32

# 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])
```

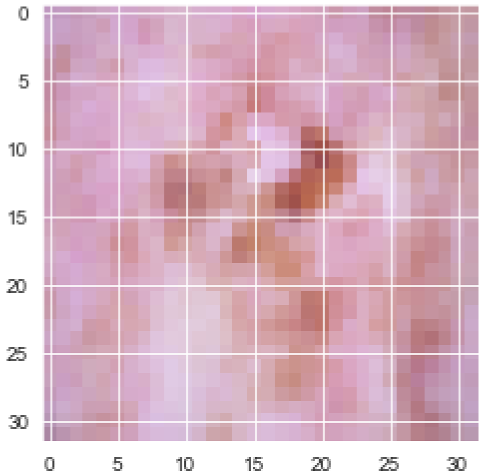
Out[15]:



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 [158...

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

X = X / 255.

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

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

In [144...

```
#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...

```
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

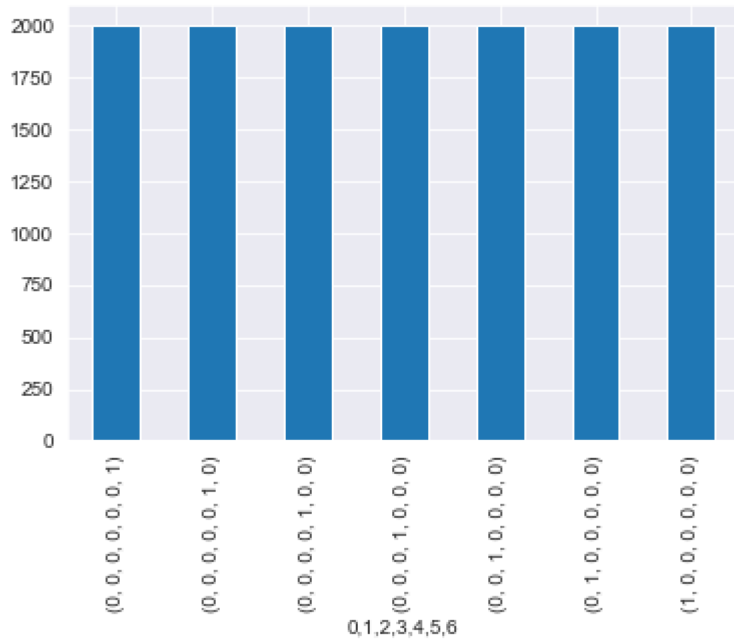
In [159...
```
#Our data set is heavily imbalanced, so we will first undersample from the majority cla

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 [160...
```
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 2000 samples each

In [162...
```
pd.DataFrame(y_train).value_counts().plot(kind='bar')
```

Out[162... <AxesSubplot:xlabel='0,1,2,3,4,5,6'>



Final length of our training data

In [163…

```
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: 14000 , y_train:  14000
```

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 [164…

```
#This Image Generator will apply various methods of flipping, rotating, and shifting du
#prevent our model from overfitting and become better accustomed to images outside of o

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 [165…

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

```
14000 2985 14000 2985
```

In [166…

```
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_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_12 (Conv2D)           (None, 32, 32, 64)        1792
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 32, 32, 64)        36928
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 16, 16, 64)        0
_________________________________________________________________
batch_normalization_8 (Batch (None, 16, 16, 64)        256
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 16, 16, 128)       73856
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 8, 8, 128)         0
_________________________________________________________________
batch_normalization_9 (Batch (None, 8, 8, 128)         512
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 8, 8, 256)         295168
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 4, 4, 256)         0
_________________________________________________________________
batch_normalization_10 (Batc (None, 4, 4, 256)         1024
_________________________________________________________________
conv2d_16 (Conv2D)           (None, 4, 4, 64)          147520
_________________________________________________________________
conv2d_17 (Conv2D)           (None, 4, 4, 64)          36928
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 2, 2, 64)          0
_________________________________________________________________
dropout_2 (Dropout)          (None, 2, 2, 64)          0
_________________________________________________________________
batch_normalization_11 (Batc (None, 2, 2, 64)          256
```

```
_________________________________________________________________
flatten_2 (Flatten)          (None, 256)               0
_________________________________________________________________
dense_6 (Dense)              (None, 128)               32896
_________________________________________________________________
dense_7 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_8 (Dense)              (None, 7)                 455
=================================================================
Total params: 635,847
Trainable params: 634,823
Non-trainable params: 1,024
_________________________________________________________________
```

In [167…

```
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
219/219 - 79s - loss: 1.2156 - acc: 0.5350 - val_loss: 2.3711 - val_acc: 0.0905
Epoch 2/50
219/219 - 73s - loss: 0.7040 - acc: 0.7346 - val_loss: 1.0826 - val_acc: 0.5765
Epoch 3/50
219/219 - 73s - loss: 0.4703 - acc: 0.8223 - val_loss: 1.3565 - val_acc: 0.5209
Epoch 4/50
219/219 - 74s - loss: 0.3713 - acc: 0.8611 - val_loss: 1.3994 - val_acc: 0.5116
Epoch 5/50
219/219 - 75s - loss: 0.2856 - acc: 0.8956 - val_loss: 1.1861 - val_acc: 0.5970
Epoch 6/50
219/219 - 77s - loss: 0.2430 - acc: 0.9095 - val_loss: 1.7048 - val_acc: 0.5434
Epoch 7/50
219/219 - 76s - loss: 0.2143 - acc: 0.9194 - val_loss: 1.1199 - val_acc: 0.6429
Epoch 8/50
219/219 - 76s - loss: 0.1754 - acc: 0.9332 - val_loss: 2.1792 - val_acc: 0.4077
Epoch 9/50
219/219 - 73s - loss: 0.1599 - acc: 0.9392 - val_loss: 1.2340 - val_acc: 0.6506
Epoch 10/50
219/219 - 72s - loss: 0.1284 - acc: 0.9526 - val_loss: 1.1618 - val_acc: 0.6884
Epoch 11/50
219/219 - 76s - loss: 0.1221 - acc: 0.9556 - val_loss: 1.2174 - val_acc: 0.6794
Epoch 12/50
219/219 - 73s - loss: 0.0938 - acc: 0.9656 - val_loss: 1.2860 - val_acc: 0.6878
Epoch 13/50
219/219 - 73s - loss: 0.0977 - acc: 0.9639 - val_loss: 1.4597 - val_acc: 0.6620
Epoch 14/50
219/219 - 74s - loss: 0.0893 - acc: 0.9671 - val_loss: 1.7324 - val_acc: 0.6422
Epoch 15/50
219/219 - 73s - loss: 0.0726 - acc: 0.9744 - val_loss: 1.3666 - val_acc: 0.6948
Epoch 16/50
219/219 - 72s - loss: 0.0693 - acc: 0.9745 - val_loss: 1.9975 - val_acc: 0.5903
Epoch 17/50
219/219 - 72s - loss: 0.0596 - acc: 0.9786 - val_loss: 1.3491 - val_acc: 0.6814
Epoch 18/50
219/219 - 73s - loss: 0.0584 - acc: 0.9801 - val_loss: 1.4387 - val_acc: 0.6948
Epoch 19/50
219/219 - 73s - loss: 0.0516 - acc: 0.9800 - val_loss: 1.4334 - val_acc: 0.6824
Epoch 20/50
```

```
219/219 - 73s - loss: 0.0430 - acc: 0.9841 - val_loss: 1.3367 - val_acc: 0.7183
Epoch 21/50
219/219 - 72s - loss: 0.0522 - acc: 0.9816 - val_loss: 1.6287 - val_acc: 0.6660
Epoch 22/50
219/219 - 72s - loss: 0.0493 - acc: 0.9833 - val_loss: 1.7704 - val_acc: 0.6476
Epoch 23/50
219/219 - 71s - loss: 0.0320 - acc: 0.9889 - val_loss: 1.6871 - val_acc: 0.6881
Epoch 24/50
219/219 - 72s - loss: 0.0363 - acc: 0.9876 - val_loss: 1.9015 - val_acc: 0.6680
Epoch 25/50
219/219 - 72s - loss: 0.0349 - acc: 0.9876 - val_loss: 1.8306 - val_acc: 0.6566
Epoch 26/50
219/219 - 72s - loss: 0.0410 - acc: 0.9853 - val_loss: 1.4605 - val_acc: 0.7082
Epoch 27/50
219/219 - 72s - loss: 0.0175 - acc: 0.9934 - val_loss: 1.5737 - val_acc: 0.7109
Epoch 28/50
219/219 - 74s - loss: 0.0280 - acc: 0.9899 - val_loss: 1.4470 - val_acc: 0.7179
Epoch 29/50
219/219 - 75s - loss: 0.0408 - acc: 0.9868 - val_loss: 1.4145 - val_acc: 0.7243
Epoch 30/50
219/219 - 77s - loss: 0.0570 - acc: 0.9815 - val_loss: 1.6611 - val_acc: 0.6905
Epoch 31/50
219/219 - 75s - loss: 0.0212 - acc: 0.9924 - val_loss: 2.0227 - val_acc: 0.6087
Epoch 32/50
219/219 - 73s - loss: 0.0205 - acc: 0.9923 - val_loss: 1.7116 - val_acc: 0.7142
Epoch 33/50
219/219 - 73s - loss: 0.0106 - acc: 0.9962 - val_loss: 1.8671 - val_acc: 0.6965
Epoch 34/50
219/219 - 74s - loss: 0.0290 - acc: 0.9903 - val_loss: 1.3786 - val_acc: 0.7300
Epoch 35/50
219/219 - 73s - loss: 0.0330 - acc: 0.9881 - val_loss: 1.3964 - val_acc: 0.7350
Epoch 36/50
219/219 - 71s - loss: 0.0176 - acc: 0.9938 - val_loss: 1.6416 - val_acc: 0.7136
Epoch 37/50
219/219 - 74s - loss: 0.0143 - acc: 0.9952 - val_loss: 1.8698 - val_acc: 0.7162
Epoch 38/50
219/219 - 74s - loss: 0.0308 - acc: 0.9901 - val_loss: 1.5048 - val_acc: 0.7159
Epoch 39/50
219/219 - 73s - loss: 0.0192 - acc: 0.9943 - val_loss: 1.7173 - val_acc: 0.7032
Epoch 40/50
219/219 - 73s - loss: 0.0118 - acc: 0.9957 - val_loss: 1.6839 - val_acc: 0.7240
Epoch 41/50
219/219 - 74s - loss: 0.0253 - acc: 0.9906 - val_loss: 2.0915 - val_acc: 0.6576
Epoch 42/50
219/219 - 73s - loss: 0.0285 - acc: 0.9909 - val_loss: 2.0375 - val_acc: 0.6389
Epoch 43/50
219/219 - 74s - loss: 0.0158 - acc: 0.9948 - val_loss: 2.4504 - val_acc: 0.6379
Epoch 44/50
219/219 - 73s - loss: 0.0165 - acc: 0.9945 - val_loss: 1.7640 - val_acc: 0.7250
Epoch 45/50
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-167-a5d7c0671131> in <module>
      2 epochs = 50
      3 
----> 4 history = model.fit(
      5     X_train, y_train,
      6     epochs=epochs,

~\anaconda3\envs\ECS 171\lib\site-packages\keras\engine\training.py in fit(self, x, y, b
atch_size, epochs, verbose, callbacks, validation_split, validation_data, shuffle, class
_weight, sample_weight, initial_epoch, steps_per_epoch, validation_steps, validation_bat
ch_size, validation_freq, max_queue_size, workers, use_multiprocessing)
   1187                 model=self,
   1188                 steps_per_execution=self._steps_per_execution)
```

```
-> 1189            val_logs = self.evaluate(
   1190                x=val_x,
   1191                y=val_y,

~\anaconda3\envs\ECS 171\lib\site-packages\keras\engine\training.py in evaluate(self, x, y, batch_size, verbose, sample_weight, steps, callbacks, max_queue_size, workers, use_multiprocessing, return_dict, **kwargs)
   1462              with tf.profiler.experimental.Trace('test', step_num=step, _r=1):
   1463                callbacks.on_test_batch_begin(step)
-> 1464                tmp_logs = self.test_function(iterator)
   1465                if data_handler.should_sync:
   1466                  context.async_wait()

~\anaconda3\envs\ECS 171\lib\site-packages\tensorflow\python\eager\def_function.py in __call__(self, *args, **kwds)
    887
    888        with OptionalXlaContext(self._jit_compile):
--> 889          result = self._call(*args, **kwds)
    890
    891        new_tracing_count = self.experimental_get_tracing_count()

~\anaconda3\envs\ECS 171\lib\site-packages\tensorflow\python\eager\def_function.py in _call(self, *args, **kwds)
    922        # In this case we have not created variables on the first call. So we can
    923        # run the first trace but we should fail if variables are created.
--> 924        results = self._stateful_fn(*args, **kwds)
    925        if self._created_variables:
    926          raise ValueError("Creating variables on a non-first call to a function"

~\anaconda3\envs\ECS 171\lib\site-packages\tensorflow\python\eager\function.py in __call__(self, *args, **kwargs)
   3021        (graph_function,
   3022         filtered_flat_args) = self._maybe_define_function(args, kwargs)
-> 3023      return graph_function._call_flat(
   3024          filtered_flat_args, captured_inputs=graph_function.captured_inputs)  # pylint: disable=protected-access
   3025

~\anaconda3\envs\ECS 171\lib\site-packages\tensorflow\python\eager\function.py in _call_flat(self, args, captured_inputs, cancellation_manager)
   1958          and executing_eagerly):
   1959        # No tape is watching; skip to running the function.
-> 1960        return self._build_call_outputs(self._inference_function.call(
   1961            ctx, args, cancellation_manager=cancellation_manager))
   1962      forward_backward = self._select_forward_and_backward_functions(

~\anaconda3\envs\ECS 171\lib\site-packages\tensorflow\python\eager\function.py in call(self, ctx, args, cancellation_manager)
    589        with _InterpolateFunctionError(self):
    590          if cancellation_manager is None:
--> 591            outputs = execute.execute(
    592                str(self.signature.name),
    593                num_outputs=self._num_outputs,

~\anaconda3\envs\ECS 171\lib\site-packages\tensorflow\python\eager\execute.py in quick_execute(op_name, num_outputs, inputs, attrs, ctx, name)
     57    try:
     58      ctx.ensure_initialized()
---> 59      tensors = pywrap_tfe.TFE_Py_Execute(ctx._handle, device_name, op_name,
     60                                          inputs, attrs, num_outputs)
     61    except core._NotOkStatusException as e:

KeyboardInterrupt:
```

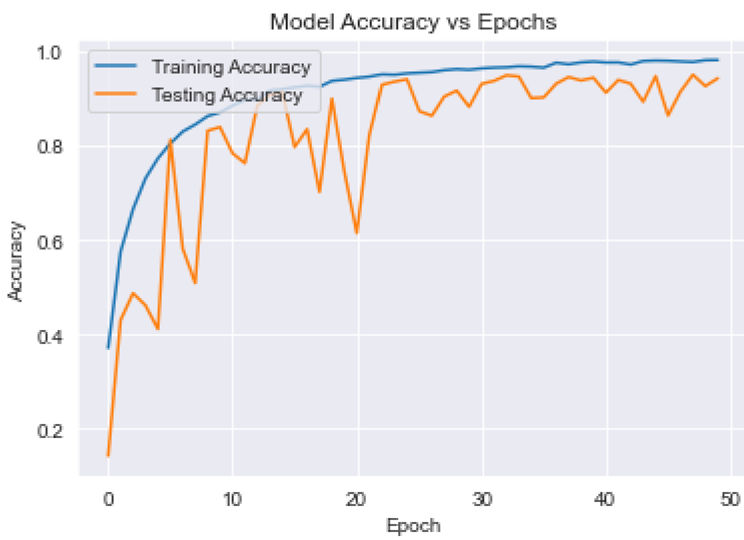
In [150…

```
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
```

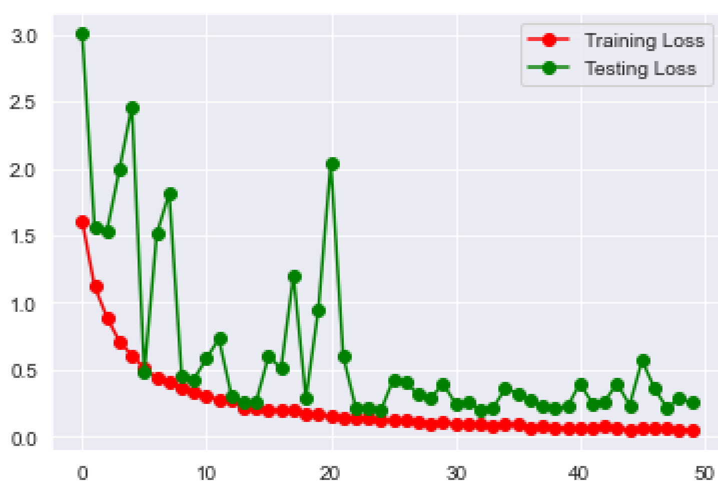
In [153…

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

plt.show()
```



In [151…

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

In [162…

```
confusion_matrix = confusion_matrix(y_test.argmax(axis = 1) , y_pred.argmax(axis = 1))

confusion_matrix = pd.DataFrame(confusion_matrix , index = ['akiec','bcc', 'bkl', 'df',
                    columns = ['akiec','bcc', 'bkl', 'df','mel' ,'nv', 'vasc'])

plt.figure(figsize = (12,12))
sns.heatmap(cm,cmap= "Reds", linecolor = 'black' , linewidth = 1 , annot = True, fmt=''
```

Out[162…

```
<AxesSubplot:>
```