pro1

June 23, 2021

[37]:
```
import pandas as pd
import os
import shutil
import matplotlib.pyplot as plt
import numpy as np
from glob import glob
import seaborn as sns
from PIL import Image
from multiprocessing import Queue
np.random.seed(42)
from sklearn.metrics import confusion_matrix

import keras
from keras.utils.np_utils import to_categorical # used for converting labels to␣
↪one-hot-encoding
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D,␣
↪BatchNormalization
from keras import regularizers
from keras.applications.resnet50 import ResNet50

from tensorflow.keras.metrics import Recall
from tensorflow.keras.callbacks import EarlyStopping,ReduceLROnPlateau
from sklearn.model_selection import train_test_split
from scipy import stats
from sklearn.preprocessing import LabelEncoder
```

[38]:
```
data_dir = os.getcwd() + "/all images"
```

[39]:
```
dest_dir = os.getcwd() + "/organized"
```

[40]:
```
skin_df2 = pd.read_csv('/home/coder/Desktop/project/skin disease/
↪HAM10000_metadata.csv')
skin_df2['dx'].value_counts()
```

[40]:
```
nv        6705
mel       1113
```

```
bkl       1099
bcc        514
akiec      327
vasc       142
df         115
Name: dx, dtype: int64
```

[41]:
```
label=skin_df2['dx'].unique().tolist()
label_images=[]
```

[42]:
```
# Copy images to new folders
for i in label:
    os.mkdir(dest_dir + str(i) + "/")
    sample = skin_df2[skin_df2['dx'] == i]['image_id']
    label_images.extend(sample)
    for id in label_images:
        shutil.copyfile((data_dir + "/"+ id +".jpg"), (dest_dir + i + "/"+id+".
↪jpg"))
    label_images=[]
```

[44]:
```
from keras.preprocessing.image import ImageDataGenerator
import os
from matplotlib import pyplot as plt
```

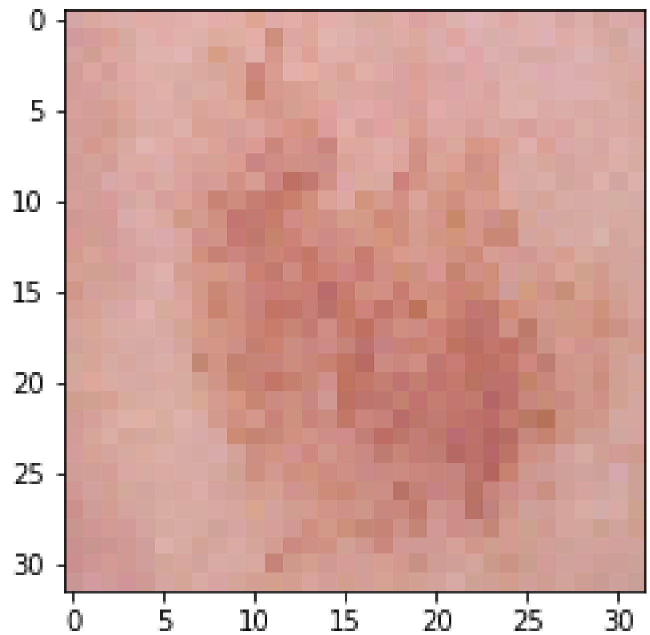
[45]:
```
datagen = ImageDataGenerator()
```

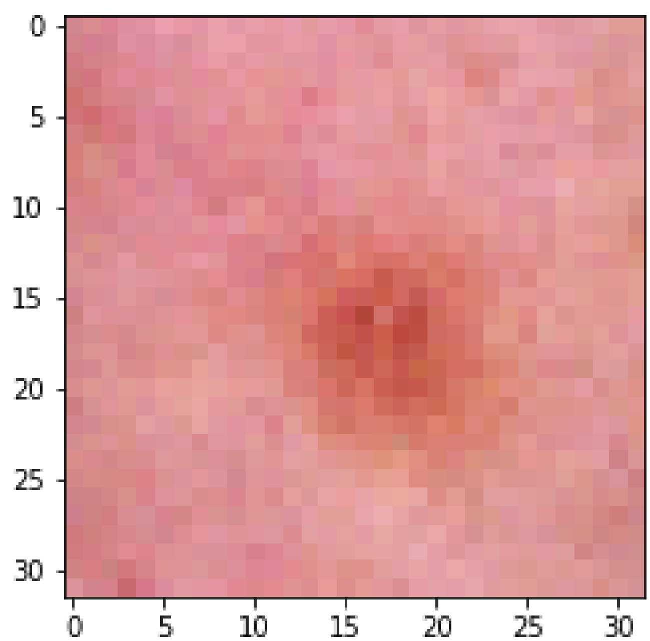
[46]:
```
train_dir = os.getcwd() + "/organized"
#USe flow_from_directory
train_data_keras = datagen.flow_from_directory(directory=train_dir,
                                               class_mode='categorical',
                                               batch_size=16,  #16 images at a time
                                               target_size=(32,32))  #Resize images
```

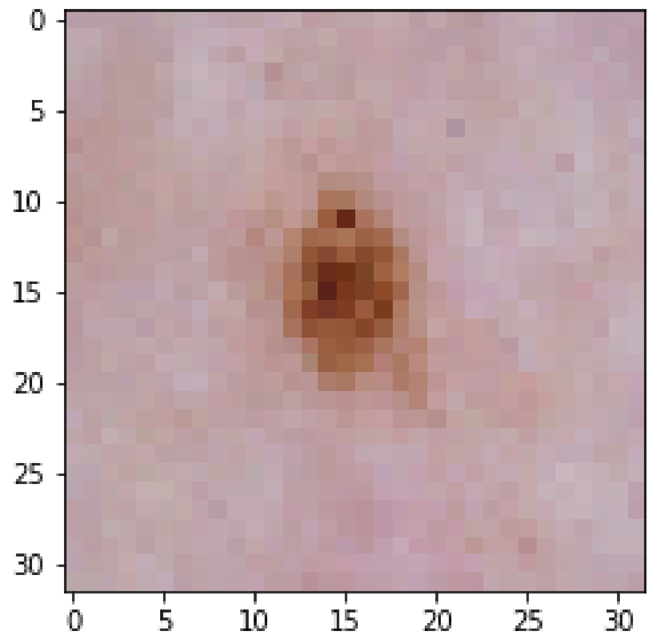
```
Found 10015 images belonging to 7 classes.
```

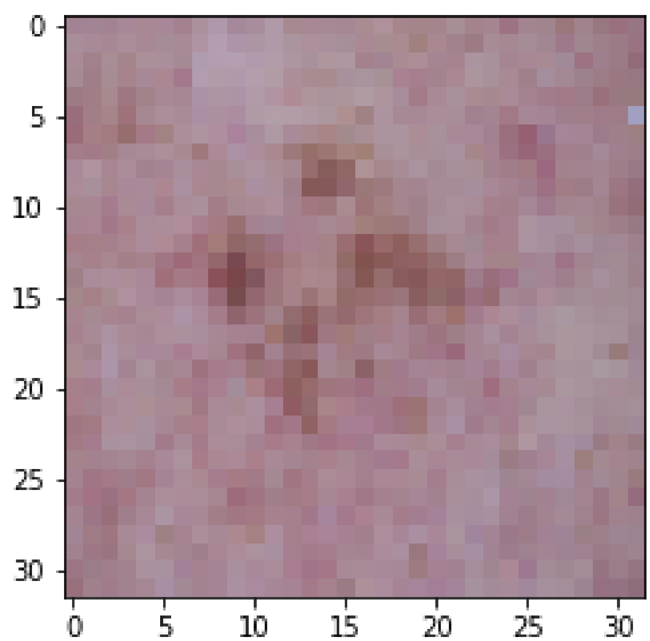
[47]:
```
x, y = next(train_data_keras)
```

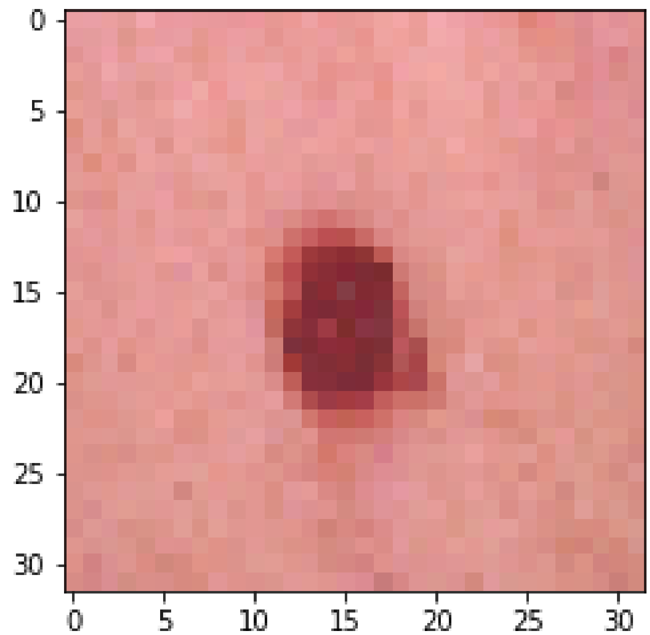
[48]:
```
for i in range (0,15):
    image = x[i].astype(int)
    plt.imshow(image)
    plt.show()
```


0
5
10
15
20
25
30
0 5 10 15 20 25 30

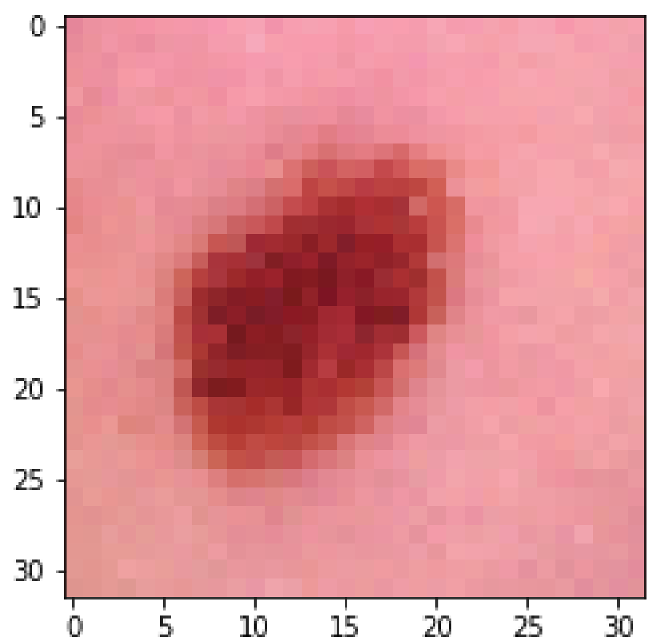

0
5
10
15
20
25
30
0 5 10 15 20 25 30


0
5
10
15
20
25
30
0
5
10
15
20
25
30

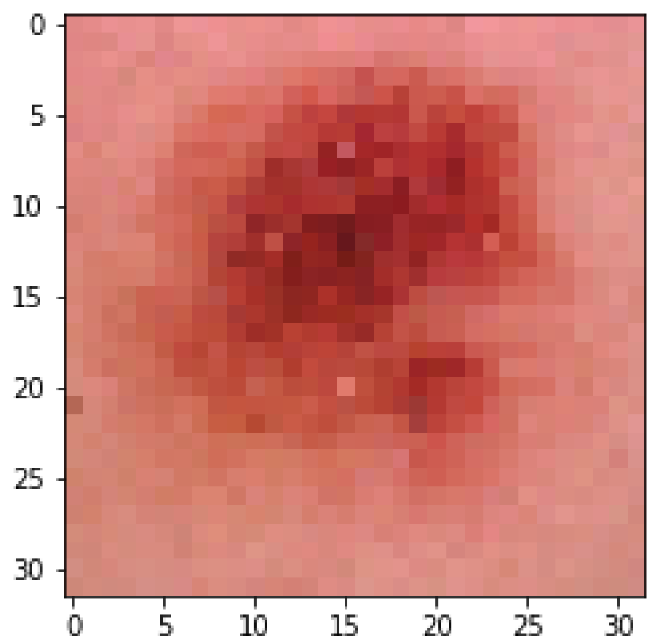

0
5
10
15
20
25
30
0
5
10
15
20
25
30

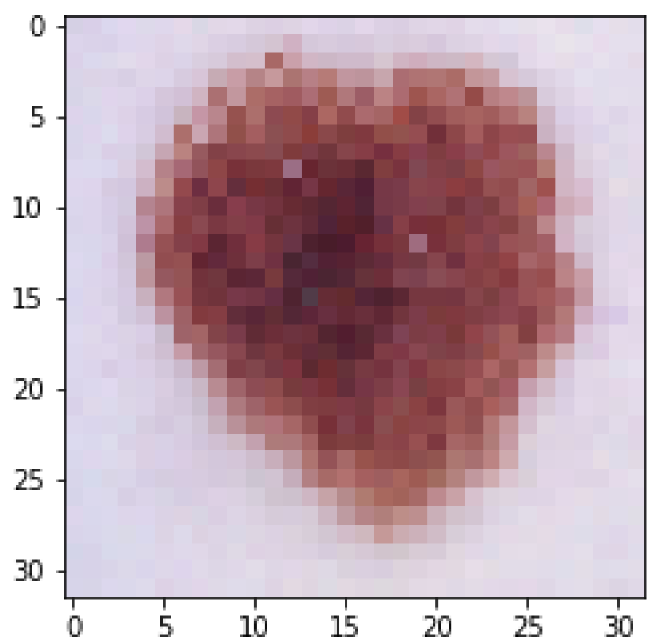


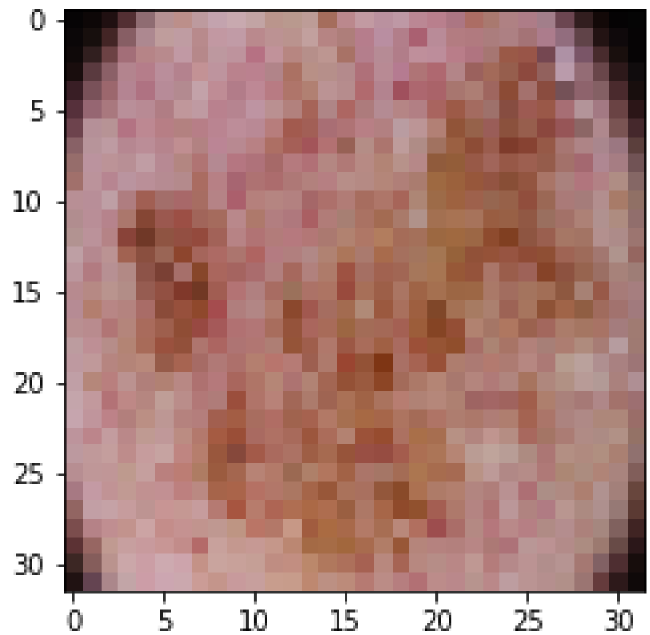
0
5
10
15
20
25
30
0
5
10
15
20
25
30

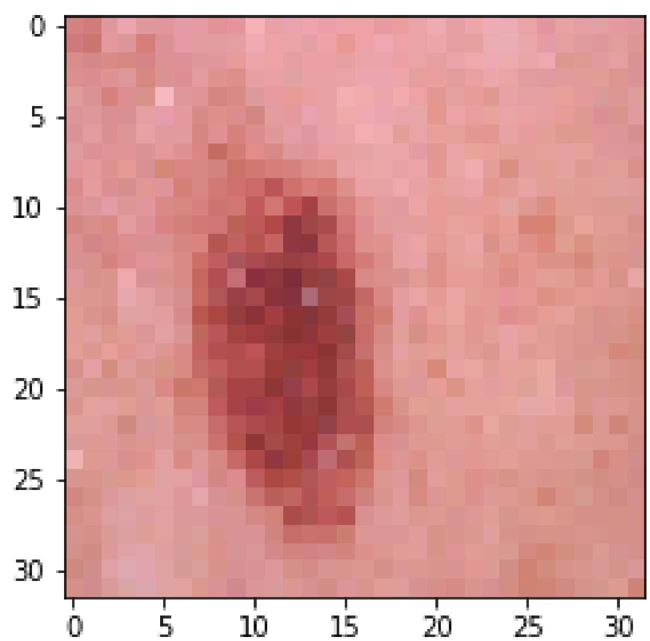


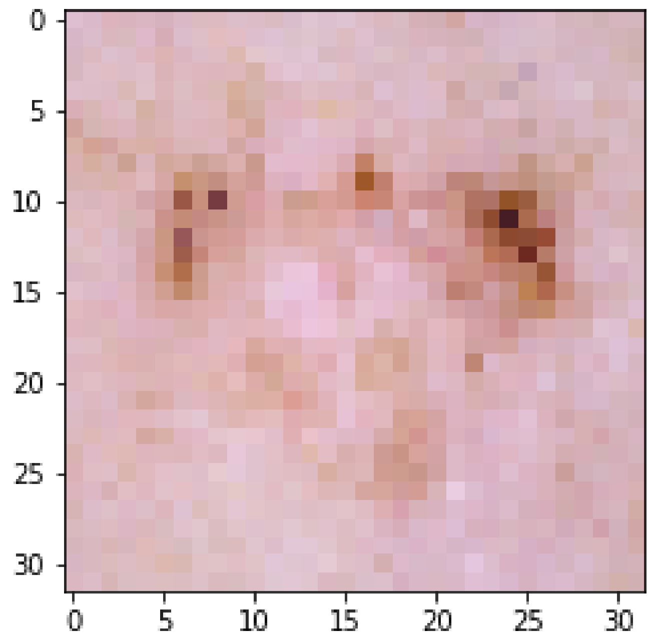
0
5
10
15
20
25
30
0
5
10
15
20
25
30

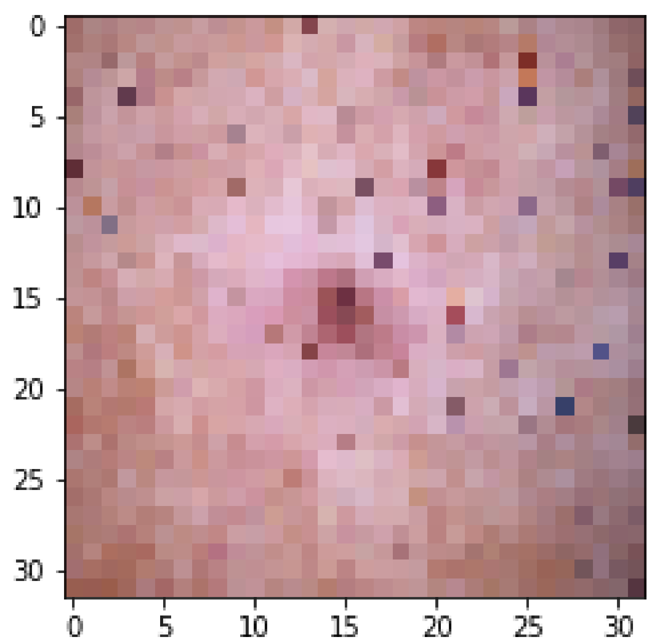

0
5
10
15
20
25
30
0
5
10
15
20
25
30

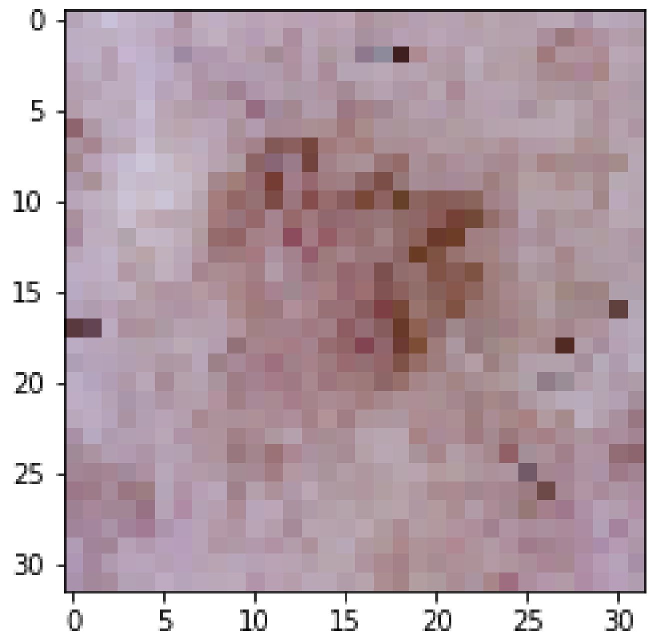

0
5
10
15
20
25
30
0
5
10
15
20
25
30

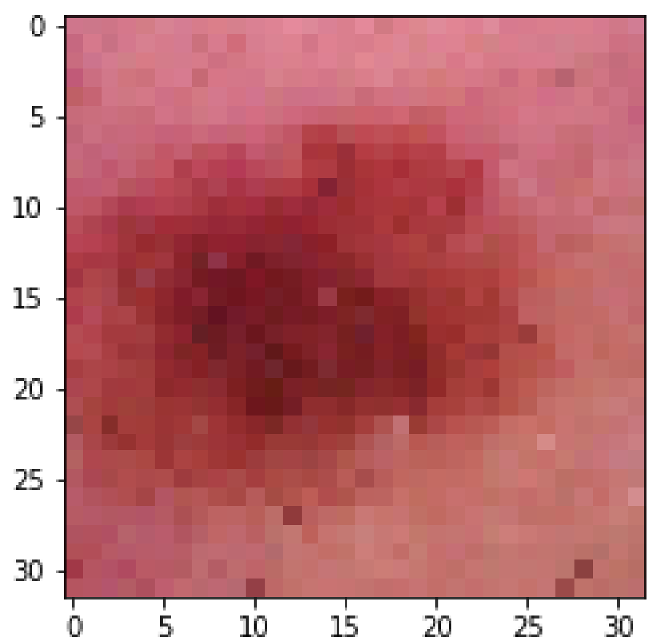

0
5
10
15
20
25
30
0
5
10
15
20
25
30

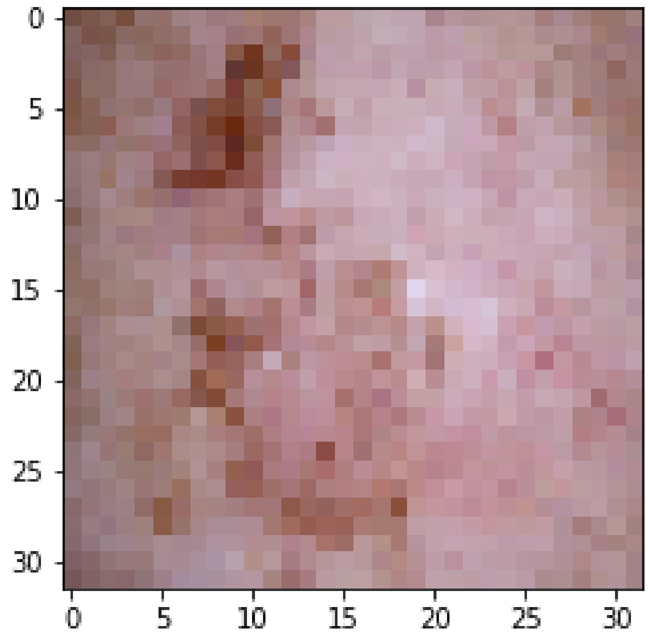

0
5
10
15
20
25
30
0
5
10
15
20
25
30


0
5
10
15
20
25
30
0
5
10
15
20
25
30


0
5
10
15
20
25
30
0
5
10
15
20
25
30


0
5
10
15
20
25
30
0
5
10
15
20
25
30


0
5
10
15
20
25
30
0
5
10
15
20
25
30



[49]:
```
SIZE=32
# label encoding to numeric values from text
le = LabelEncoder()
le.fit(skin_df2['dx'])
LabelEncoder()
print(list(le.classes_))
```

```
['akiec', 'bcc', 'bkl', 'df', 'mel', 'nv', 'vasc']
```

[50]:
```
skin_df2['label'] = le.transform(skin_df2["dx"])
print(skin_df2.sample(10))
```

```
        lesion_id      image_id   dx    dx_type   age      sex  \
971   HAM_0000948  ISIC_0033631  bkl  consensus   NaN  unknown
606   HAM_0005865  ISIC_0031522  bkl      histo  70.0     male
7198  HAM_0005116  ISIC_0032953   nv      histo   5.0   female
695   HAM_0003015  ISIC_0025083  bkl      histo  55.0     male
787   HAM_0002042  ISIC_0028294  bkl   confocal  75.0   female
6936  HAM_0004869  ISIC_0025370   nv      histo  15.0   female
6292  HAM_0004162  ISIC_0031944   nv  follow_up  65.0     male
9323  HAM_0006553  ISIC_0026942   nv  consensus  20.0   female
7480  HAM_0007306  ISIC_0026810   nv      histo  30.0     male
8886  HAM_0004518  ISIC_0024578   nv      histo  35.0   female

         localization  label
971           unknown      2
```

```
606              chest      2
7198   lower extremity      5
695               back      2
787               face      2
6936   lower extremity      5
6292              back      5
9323              face      5
7480              back      5
8886   upper extremity      5
```

[51]:
```
fig = plt.figure(figsize=(12,8))
```

```
<Figure size 864x576 with 0 Axes>
```

[52]:
```
ax1 = fig.add_subplot(221)
skin_df2['dx'].value_counts().plot(kind='bar', ax=ax1)
ax1.set_ylabel('Count')
ax1.set_title('Cell Type')
```

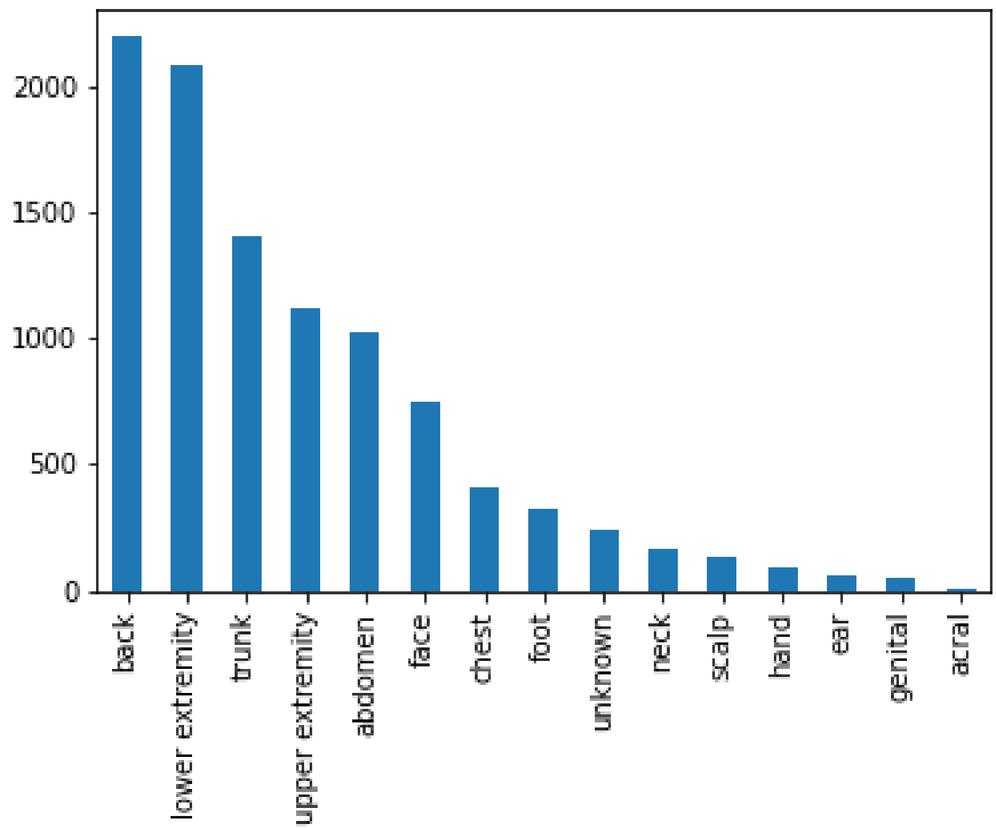
[52]:
```
Text(0.5, 1.0, 'Cell Type')
```

[53]:
```
ax2 = fig.add_subplot(222)
skin_df2['sex'].value_counts().plot(kind='bar', ax=ax2)
ax2.set_ylabel('Count', size=15)
ax2.set_title('Sex')
```

[53]:
```
Text(0.5, 1.0, 'Sex')
```

[54]:
```
ax3 = fig.add_subplot(223)
skin_df2['localization'].value_counts().plot(kind='bar')
ax3.set_ylabel('Count',size=12)
ax3.set_title('Localization')
```

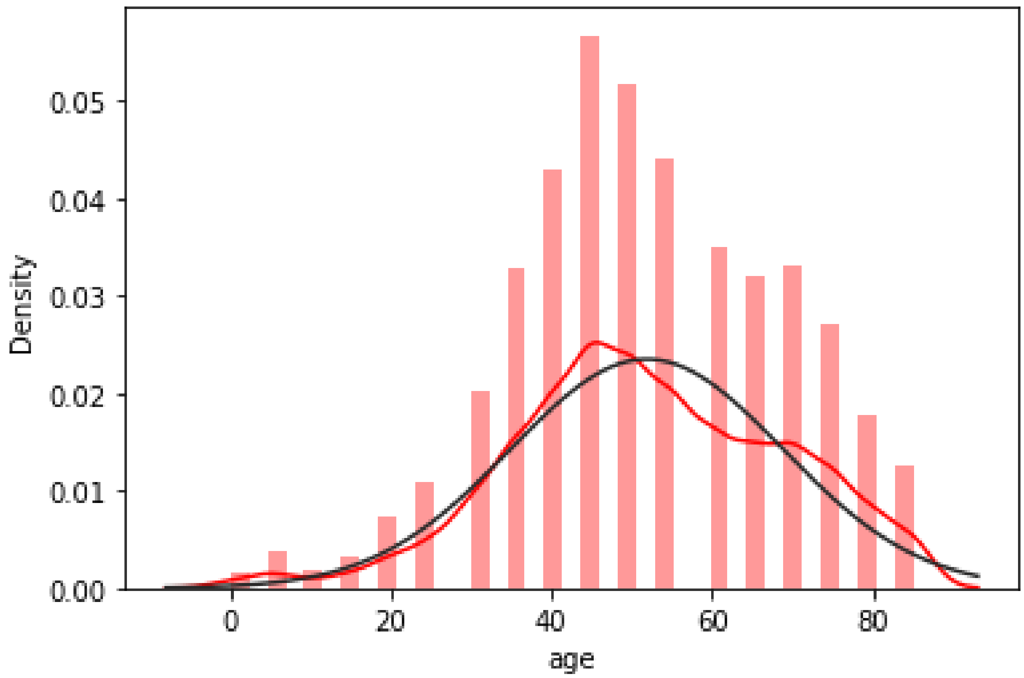
[54]:
```
Text(0.5, 1.0, 'Localization')
```



[55]:
```
ax4 = fig.add_subplot(224)
sample_age = skin_df2[pd.notnull(skin_df2['age'])]
sns.distplot(sample_age['age'], fit=stats.norm, color='red');
ax4.set_title('Age')
```

```
/home/coder/anaconda3/lib/python3.8/site-packages/seaborn/distributions.py:2551:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```

[55]:
```
Text(0.5, 1.0, 'Age')
```



[56]:
```
plt.tight_layout()
plt.show()
```

<Figure size 432x288 with 0 Axes>

[57]:
```
from sklearn.utils import resample
print(skin_df2['label'].value_counts())
```

```
5    6705
4    1113
2    1099
1     514
0     327
6     142
3     115
Name: label, dtype: int64
```

[58]:
```
df_0 = skin_df2[skin_df2['label'] == 0]
df_1 = skin_df2[skin_df2['label'] == 1]
df_2 = skin_df2[skin_df2['label'] == 2]
df_3 = skin_df2[skin_df2['label'] == 3]
df_4 = skin_df2[skin_df2['label'] == 4]
df_5 = skin_df2[skin_df2['label'] == 5]
df_6 = skin_df2[skin_df2['label'] == 6]
```

[59]:
```
n_samples=500
df_0_balanced = resample(df_0, replace=True, n_samples=n_samples,␣
↪random_state=42)
df_1_balanced = resample(df_1, replace=True, n_samples=n_samples,␣
↪random_state=42)
df_2_balanced = resample(df_2, replace=True, n_samples=n_samples,␣
↪random_state=42)
df_3_balanced = resample(df_3, replace=True, n_samples=n_samples,␣
↪random_state=42)
df_4_balanced = resample(df_4, replace=True, n_samples=n_samples,␣
↪random_state=42)
df_5_balanced = resample(df_5, replace=True, n_samples=n_samples,␣
↪random_state=42)
df_6_balanced = resample(df_6, replace=True, n_samples=n_samples,␣
↪random_state=42)
```

[60]:
```
skin_df_balanced = pd.concat([df_0_balanced, df_1_balanced,
                              df_2_balanced, df_3_balanced,
                              df_4_balanced, df_5_balanced, df_6_balanced])
```

[61]:
```
print(skin_df_balanced['label'].value_counts())
```

```
5    500
3    500
1    500
6    500
4    500
2    500
0    500
Name: label, dtype: int64
```

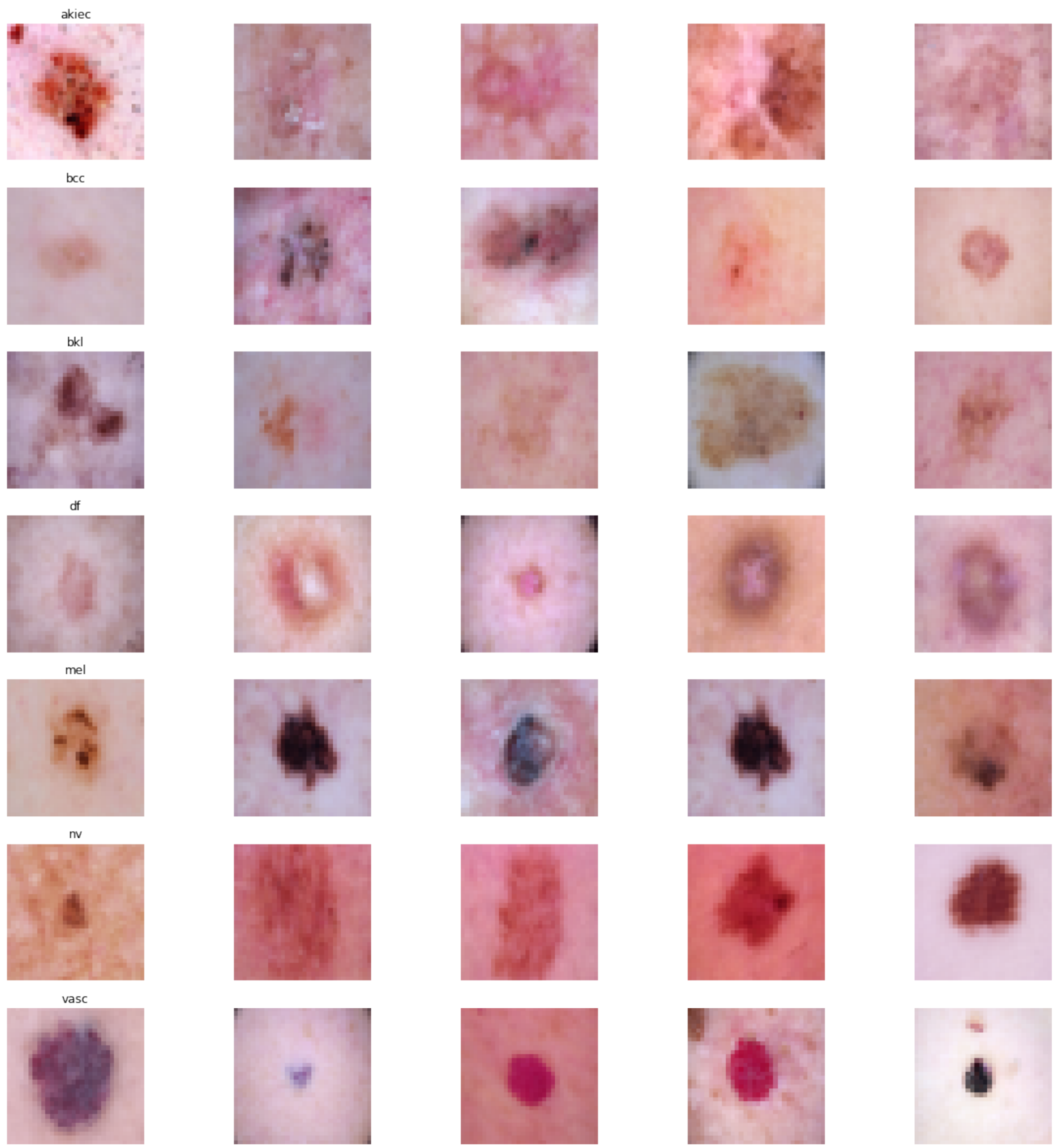
[62]:
```
image_path = {os.path.splitext(os.path.basename(x))[0]: x
                    for x in glob(os.path.join('/home/coder/Desktop/project/
↪skin disease', '*', '*.jpg'))}
```

[63]:
```
skin_df_balanced['path'] = skin_df2['image_id'].map(image_path.get)
```

[64]:
```
skin_df_balanced['image'] = skin_df_balanced['path'].map(lambda x: np.
↪asarray(Image.open(x).resize((SIZE,SIZE))))
# skin_df_balanced['image'] = lambda xs: K.reshape(xs[0], [xs[1][0], xs[1][1]],␣
↪output_shape=(SIZE,SIZE))([x,skin_df_balanced['image']])
```

[65]:
```
n_samples = 5
fig, m_axs = plt.subplots(7, n_samples, figsize = (4*n_samples, 3*7))
for n_axs, (type_name, type_rows) in zip(m_axs,
                                         skin_df_balanced.sort_values(['dx']).
↪groupby('dx')):
```

```
    n_axs[0].set_title(type_name)
    for c_ax, (_, c_row) in zip(n_axs, type_rows.sample(n_samples,␣
 ↪random_state=1234).iterrows()):
        c_ax.imshow(c_row['image'])
        c_ax.axis('off')
```



[66]:
```
X = np.asarray(skin_df_balanced['image'].tolist())
X = X/255.  # Scale values to 0-1. You can also used standardscaler or other␣
 ↪scaling methods.
Y=skin_df_balanced['label']  #Assign label values to Y
```

```
Y_cat = to_categorical(Y, num_classes=7) #Convert to categorical as this is a␣
↪multiclass classification problem
#Split to training and testing
x_train, x_test, y_train, y_test = train_test_split(X, Y_cat, test_size=0.028,␣
↪random_state=42)
```

[71]:
```
input_shape=(32,32,3)

model=Sequential()


model.add(Conv2D(64,(2,2),input_shape=(32,32,3),activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization())


model.add(Conv2D(512,(2,2),input_shape=(32,32,3),activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))



model.add(Dropout(0.3))

model.add(Conv2D(1024,(2,2),input_shape=(32,32,3),activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(BatchNormalization())


model.add(Dropout(0.4))

model.add(Conv2D(1024,(1,1),input_shape=(32,32,3),activation='relu'))
model.add(MaxPooling2D(pool_size=(1, 1)))



model.add(Dropout(0.4))

model.add(Flatten())

model.add(Dense(256,activation='relu',kernel_regularizer=regularizers.l2(0.01)))
model.add(Dropout(0.5))


model.add(Dense(7,activation='softmax'))

model.
↪compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy',Recall()])
```

[72]:
```
model.summary()
```

```
Model: "sequential_5"

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_20 (Conv2D)           (None, 31, 31, 64)        832
_________________________________________________________________
max_pooling2d_20 (MaxPooling (None, 15, 15, 64)        0
_________________________________________________________________
batch_normalization_10 (Batc (None, 15, 15, 64)        256
_________________________________________________________________
conv2d_21 (Conv2D)           (None, 14, 14, 512)       131584
_________________________________________________________________
max_pooling2d_21 (MaxPooling (None, 7, 7, 512)         0
_________________________________________________________________
dropout_20 (Dropout)         (None, 7, 7, 512)         0
_________________________________________________________________
conv2d_22 (Conv2D)           (None, 6, 6, 1024)        2098176
_________________________________________________________________
max_pooling2d_22 (MaxPooling (None, 3, 3, 1024)        0
_________________________________________________________________
batch_normalization_11 (Batc (None, 3, 3, 1024)        4096
_________________________________________________________________
dropout_21 (Dropout)         (None, 3, 3, 1024)        0
_________________________________________________________________
conv2d_23 (Conv2D)           (None, 3, 3, 1024)        1049600
_________________________________________________________________
max_pooling2d_23 (MaxPooling (None, 3, 3, 1024)        0
_________________________________________________________________
dropout_22 (Dropout)         (None, 3, 3, 1024)        0
_________________________________________________________________
flatten_5 (Flatten)          (None, 9216)              0
_________________________________________________________________
dense_10 (Dense)             (None, 256)               2359552
_________________________________________________________________
dropout_23 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_11 (Dense)             (None, 7)                 1799
=================================================================
Total params: 5,645,895
Trainable params: 5,643,719
Non-trainable params: 2,176

_________________________________________________________________
```

[33]:
```
early=EarlyStopping(monitor='accuracy',patience=4,mode='auto')
```

```
reduce_lr = ReduceLROnPlateau(monitor='accuracy', factor=0.5, patience=2,␣
↪verbose=1,cooldown=0, mode='auto',min_delta=0.0001, min_lr=0)
```

[34]:
```
history=model.
↪fit(x_train,y_train,epochs=50,batch_size=90,validation_data=(x_test,␣
↪y_test),callbacks=[early,reduce_lr])
```

```
Epoch 1/50
38/38 [==============================] - 69s 2s/step - loss: 6.8714 - accuracy:
0.2419 - recall: 0.0629 - val_loss: 5.5418 - val_accuracy: 0.1429 - val_recall:
0.0000e+00
Epoch 2/50
38/38 [==============================] - 81s 2s/step - loss: 4.5908 - accuracy:
0.3075 - recall: 0.0841 - val_loss: 4.2043 - val_accuracy: 0.1429 - val_recall:
0.0000e+00
Epoch 3/50
38/38 [==============================] - 93s 2s/step - loss: 3.3887 - accuracy:
0.3471 - recall: 0.1264 - val_loss: 3.3529 - val_accuracy: 0.1633 - val_recall:
0.0000e+00
Epoch 4/50
38/38 [==============================] - 91s 2s/step - loss: 2.6701 - accuracy:
0.3810 - recall: 0.1487 - val_loss: 2.8259 - val_accuracy: 0.1429 - val_recall:
0.0000e+00
Epoch 5/50
38/38 [==============================] - 83s 2s/step - loss: 2.2391 - accuracy:
0.4068 - recall: 0.1796 - val_loss: 2.7171 - val_accuracy: 0.1837 - val_recall:
0.0000e+00
Epoch 6/50
38/38 [==============================] - 65s 2s/step - loss: 1.9102 - accuracy:
0.4606 - recall: 0.2287 - val_loss: 2.9629 - val_accuracy: 0.1429 - val_recall:
0.0000e+00
Epoch 7/50
38/38 [==============================] - 79s 2s/step - loss: 1.7828 - accuracy:
0.4636 - recall: 0.2378 - val_loss: 3.1126 - val_accuracy: 0.1939 - val_recall:
0.1837
Epoch 8/50
38/38 [==============================] - 79s 2s/step - loss: 1.6641 - accuracy:
0.4774 - recall: 0.2716 - val_loss: 2.6893 - val_accuracy: 0.1429 - val_recall:
0.0000e+00
Epoch 9/50
38/38 [==============================] - 82s 2s/step - loss: 1.6139 - accuracy:
0.4979 - recall: 0.2851 - val_loss: 2.7890 - val_accuracy: 0.1224 - val_recall:
0.0204
Epoch 10/50
38/38 [==============================] - 90s 2s/step - loss: 1.5900 - accuracy:
0.5100 - recall: 0.2969 - val_loss: 3.2974 - val_accuracy: 0.2755 - val_recall:
0.0306
```

```
Epoch 11/50
38/38 [==============================] - 85s 2s/step - loss: 1.4723 - accuracy:
0.5300 - recall: 0.3272 - val_loss: 3.3762 - val_accuracy: 0.2551 - val_recall:
0.1735
Epoch 12/50
38/38 [==============================] - 55s 1s/step - loss: 1.4314 - accuracy:
0.5456 - recall: 0.3360 - val_loss: 3.0216 - val_accuracy: 0.2143 - val_recall:
0.0816
Epoch 13/50
38/38 [==============================] - 46s 1s/step - loss: 1.4068 - accuracy:
0.5626 - recall: 0.3595 - val_loss: 3.6369 - val_accuracy: 0.1939 - val_recall:
0.1224
Epoch 14/50
38/38 [==============================] - 49s 1s/step - loss: 1.4363 - accuracy:
0.5755 - recall: 0.3798 - val_loss: 2.7244 - val_accuracy: 0.2551 - val_recall:
0.1735
Epoch 15/50
38/38 [==============================] - 65s 2s/step - loss: 1.3752 - accuracy:
0.5879 - recall: 0.4077 - val_loss: 2.4006 - val_accuracy: 0.3061 - val_recall:
0.2245
Epoch 16/50
38/38 [==============================] - 70s 2s/step - loss: 1.2818 - accuracy:
0.6123 - recall: 0.4342 - val_loss: 1.7348 - val_accuracy: 0.4082 - val_recall:
0.3163
Epoch 17/50
38/38 [==============================] - 70s 2s/step - loss: 1.2684 - accuracy:
0.6314 - recall: 0.4647 - val_loss: 2.2243 - val_accuracy: 0.3673 - val_recall:
0.2959
Epoch 18/50
38/38 [==============================] - 70s 2s/step - loss: 1.2693 - accuracy:
0.6308 - recall: 0.4644 - val_loss: 1.5848 - val_accuracy: 0.4796 - val_recall:
0.3673
Epoch 19/50
38/38 [==============================] - 65s 2s/step - loss: 1.2243 - accuracy:
0.6455 - recall: 0.4871 - val_loss: 1.9190 - val_accuracy: 0.3980 - val_recall:
0.3571
Epoch 20/50
38/38 [==============================] - 55s 1s/step - loss: 1.2026 - accuracy:
0.6599 - recall: 0.5085 - val_loss: 1.5868 - val_accuracy: 0.5714 - val_recall:
0.4592
Epoch 21/50
38/38 [==============================] - 55s 1s/step - loss: 1.1560 - accuracy:
0.6822 - recall: 0.5317 - val_loss: 1.9164 - val_accuracy: 0.4490 - val_recall:
0.3571
Epoch 22/50
38/38 [==============================] - 55s 1s/step - loss: 1.1354 - accuracy:
0.6796 - recall: 0.5603 - val_loss: 1.3746 - val_accuracy: 0.6633 - val_recall:
0.4898
```

```
Epoch 23/50
38/38 [==============================] - 55s 1s/step - loss: 1.0570 - accuracy:
0.7187 - recall: 0.6026 - val_loss: 1.6091 - val_accuracy: 0.4694 - val_recall:
0.3980
Epoch 24/50
38/38 [==============================] - 55s 1s/step - loss: 1.0444 - accuracy:
0.7219 - recall: 0.6035 - val_loss: 2.2228 - val_accuracy: 0.4388 - val_recall:
0.3265
Epoch 25/50
38/38 [==============================] - 55s 1s/step - loss: 1.0276 - accuracy:
0.7375 - recall: 0.6364 - val_loss: 1.1418 - val_accuracy: 0.6633 - val_recall:
0.5918
Epoch 26/50
38/38 [==============================] - 56s 1s/step - loss: 1.0239 - accuracy:
0.7407 - recall: 0.6476 - val_loss: 1.1114 - val_accuracy: 0.6531 - val_recall:
0.5816
Epoch 27/50
38/38 [==============================] - 55s 1s/step - loss: 0.9616 - accuracy:
0.7519 - recall: 0.6678 - val_loss: 1.3127 - val_accuracy: 0.5816 - val_recall:
0.5102
Epoch 28/50
38/38 [==============================] - 55s 1s/step - loss: 0.9576 - accuracy:
0.7604 - recall: 0.6764 - val_loss: 1.4491 - val_accuracy: 0.5714 - val_recall:
0.5000
Epoch 29/50
38/38 [==============================] - 55s 1s/step - loss: 0.8970 - accuracy:
0.7837 - recall: 0.7078 - val_loss: 1.6921 - val_accuracy: 0.5000 - val_recall:
0.4184
Epoch 30/50
38/38 [==============================] - 55s 1s/step - loss: 0.9041 - accuracy:
0.7842 - recall: 0.7099 - val_loss: 1.2262 - val_accuracy: 0.6327 - val_recall:
0.5816
Epoch 31/50
38/38 [==============================] - 55s 1s/step - loss: 0.8704 - accuracy:
0.7978 - recall: 0.7316 - val_loss: 1.7539 - val_accuracy: 0.5204 - val_recall:
0.4796
Epoch 32/50
38/38 [==============================] - 55s 1s/step - loss: 0.8238 - accuracy:
0.8089 - recall: 0.7399 - val_loss: 1.2800 - val_accuracy: 0.6020 - val_recall:
0.5714
Epoch 33/50
38/38 [==============================] - 55s 1s/step - loss: 0.7651 - accuracy:
0.8239 - recall: 0.7704 - val_loss: 1.8625 - val_accuracy: 0.5408 - val_recall:
0.5204
Epoch 34/50
38/38 [==============================] - 55s 1s/step - loss: 0.7912 - accuracy:
0.8078 - recall: 0.7504 - val_loss: 0.8719 - val_accuracy: 0.7857 - val_recall:
0.7245
```

```
Epoch 35/50
38/38 [==============================] - 887s 23s/step - loss: 0.7425 -
accuracy: 0.8325 - recall: 0.7725 - val_loss: 2.5024 - val_accuracy: 0.3878 -
val_recall: 0.3776
Epoch 36/50
38/38 [==============================] - 44s 1s/step - loss: 0.6914 - accuracy:
0.8386 - recall: 0.7907 - val_loss: 0.9617 - val_accuracy: 0.7857 - val_recall:
0.6837
Epoch 37/50
38/38 [==============================] - 45s 1s/step - loss: 0.6909 - accuracy:
0.8404 - recall: 0.7922 - val_loss: 0.9535 - val_accuracy: 0.7653 - val_recall:
0.6837
Epoch 38/50
38/38 [==============================] - 50s 1s/step - loss: 0.6988 - accuracy:
0.8433 - recall: 0.8013 - val_loss: 1.2980 - val_accuracy: 0.6837 - val_recall:
0.6837
Epoch 39/50
38/38 [==============================] - 43s 1s/step - loss: 0.6883 - accuracy:
0.8501 - recall: 0.8113 - val_loss: 0.9811 - val_accuracy: 0.7857 - val_recall:
0.7653
Epoch 40/50
38/38 [==============================] - 44s 1s/step - loss: 0.6566 - accuracy:
0.8648 - recall: 0.8280 - val_loss: 1.8942 - val_accuracy: 0.5408 - val_recall:
0.5000
Epoch 41/50
38/38 [==============================] - 45s 1s/step - loss: 0.6700 - accuracy:
0.8586 - recall: 0.8154 - val_loss: 0.9828 - val_accuracy: 0.7551 - val_recall:
0.6939
Epoch 42/50
38/38 [==============================] - 44s 1s/step - loss: 0.6587 - accuracy:
0.8674 - recall: 0.8266 - val_loss: 1.0834 - val_accuracy: 0.7143 - val_recall:
0.6429
Epoch 43/50
38/38 [==============================] - 46s 1s/step - loss: 0.6342 - accuracy:
0.8751 - recall: 0.8389 - val_loss: 0.9683 - val_accuracy: 0.7245 - val_recall:
0.7143
Epoch 44/50
38/38 [==============================] - 43s 1s/step - loss: 0.6117 - accuracy:
0.8789 - recall: 0.8445 - val_loss: 2.0208 - val_accuracy: 0.4898 - val_recall:
0.4694
Epoch 45/50
38/38 [==============================] - 42s 1s/step - loss: 0.5628 - accuracy:
0.8898 - recall: 0.8624 - val_loss: 1.4999 - val_accuracy: 0.6633 - val_recall:
0.6122
Epoch 46/50
38/38 [==============================] - 43s 1s/step - loss: 0.5433 - accuracy:
0.9015 - recall: 0.8762 - val_loss: 2.4124 - val_accuracy: 0.5102 - val_recall:
0.4796
```

```
Epoch 47/50
38/38 [==============================] - 53s 1s/step - loss: 0.5261 - accuracy:
0.9048 - recall: 0.8812 - val_loss: 1.0716 - val_accuracy: 0.7143 - val_recall:
0.6939
Epoch 48/50
38/38 [==============================] - 71s 2s/step - loss: 0.5286 - accuracy:
0.9048 - recall: 0.8854 - val_loss: 1.1920 - val_accuracy: 0.6735 - val_recall:
0.6224
Epoch 49/50
38/38 [==============================] - ETA: 0s - loss: 0.5372 - accuracy:
0.9015 - recall: 0.8777
Epoch 00049: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.
38/38 [==============================] - 71s 2s/step - loss: 0.5372 - accuracy:
0.9015 - recall: 0.8777 - val_loss: 2.3961 - val_accuracy: 0.4388 - val_recall:
0.4082
Epoch 50/50
38/38 [==============================] - 71s 2s/step - loss: 0.4510 - accuracy:
0.9321 - recall: 0.9165 - val_loss: 0.9191 - val_accuracy: 0.8163 - val_recall:
0.7959
```

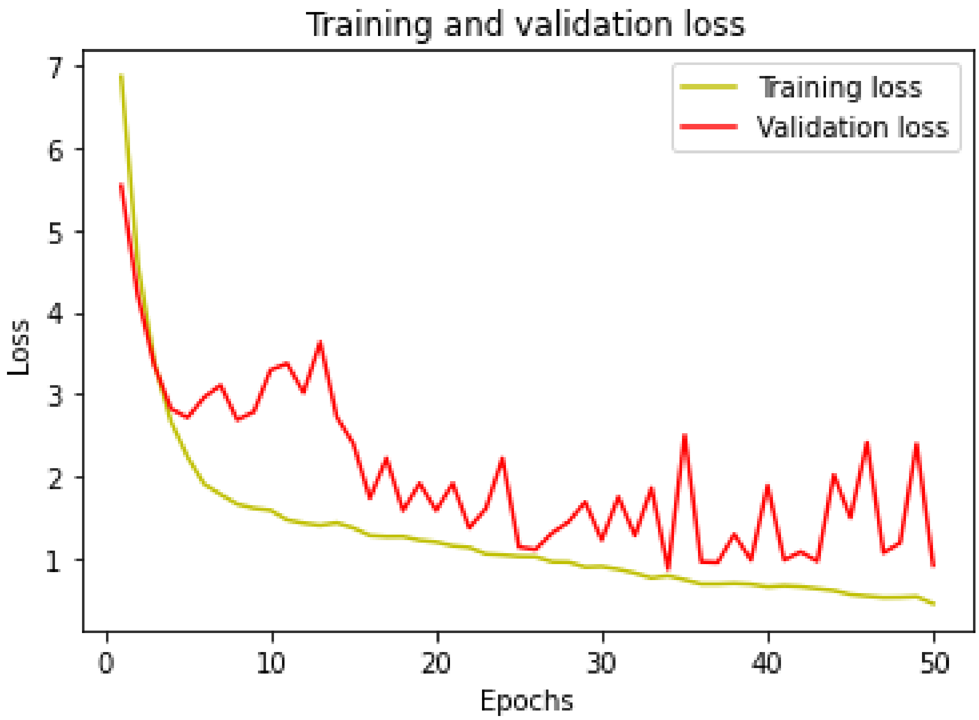
[35]:

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

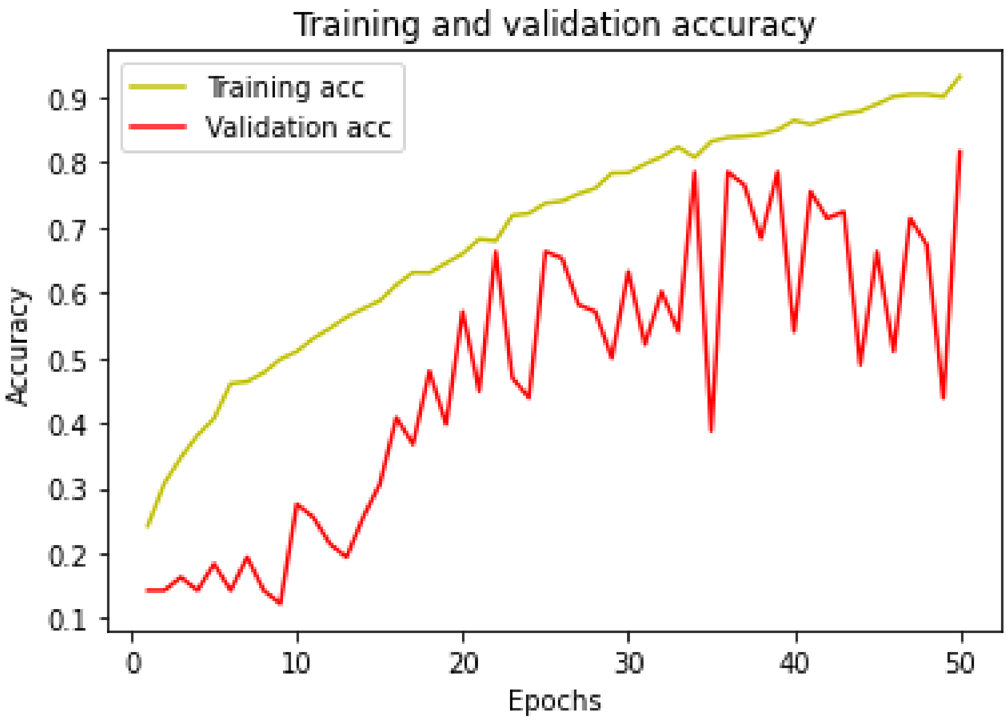
```
4/4 [==============================] - 0s 62ms/step - loss: 0.9191 - accuracy:
0.8163 - recall: 0.7959
Test accuracy: 0.8163265585899353
```

[36]:

```
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(loss) + 1)
plt.plot(epochs, loss, 'y', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



[37]:
```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
plt.plot(epochs, acc, 'y', label='Training acc')
plt.plot(epochs, val_acc, 'r', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



[38]:
```
y_pred = model.predict(x_test)
```

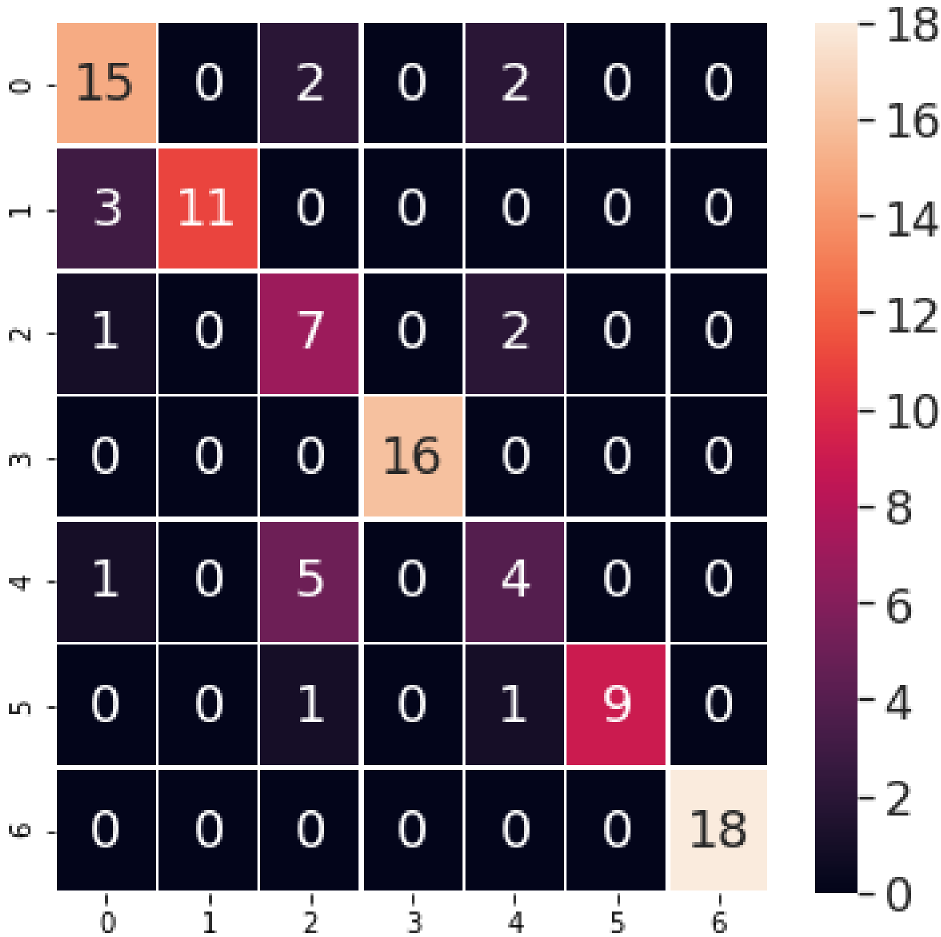
[39]:
```
y_pred_classes = np.argmax(y_pred, axis = 1)
```

[40]:
```
y_true = np.argmax(y_test, axis = 1)
```

[41]:
```
cm = confusion_matrix(y_true, y_pred_classes)
```

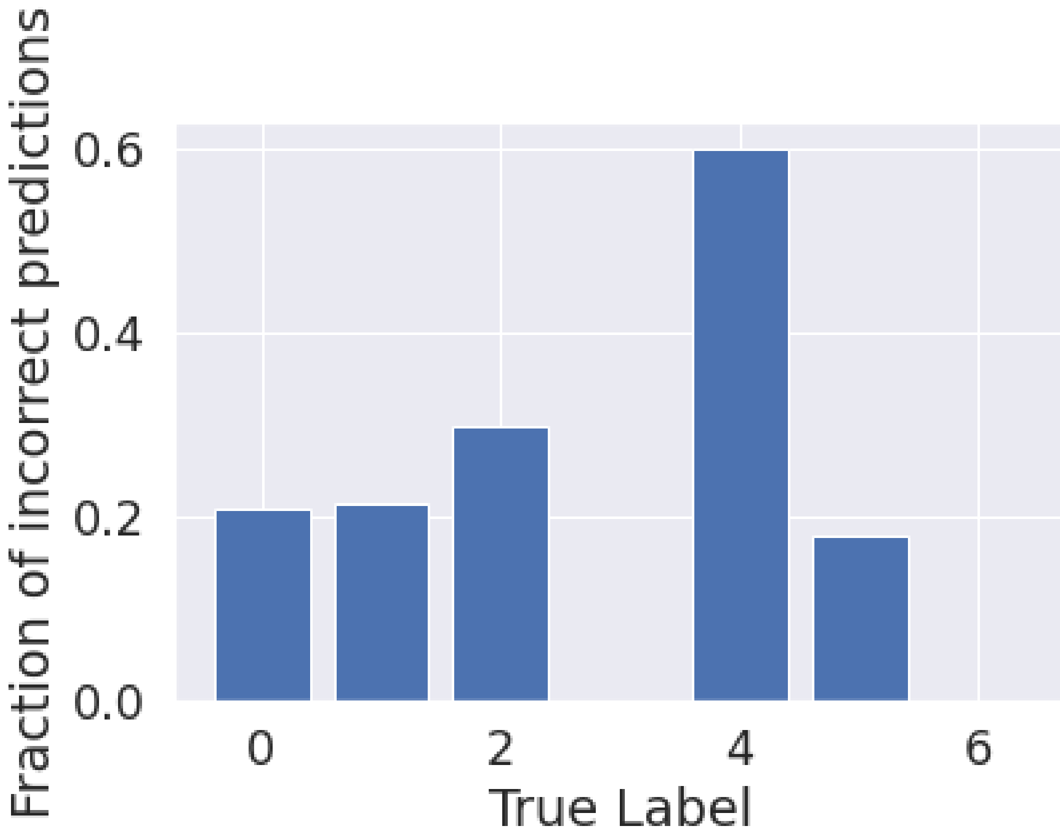
[42]:
```
fig, ax = plt.subplots(figsize=(6,6))
sns.set(font_scale=1.6)
sns.heatmap(cm, annot=True, linewidths=.5, ax=ax)
```

[42]: <AxesSubplot:>



[43]:
```
incorr_fraction = 1 - np.diag(cm) / np.sum(cm, axis=1)
plt.bar(np.arange(7), incorr_fraction)
plt.xlabel('True Label')
plt.ylabel('Fraction of incorrect predictions')
```

[43]:
```
Text(0, 0.5, 'Fraction of incorrect predictions')
```



[6]:
```
model = ResNet50(weights='imagenet')
model.save('/home/coder/Desktop/project/models')
```

```
INFO:tensorflow:Assets written to: /home/coder/Desktop/project/models/assets
```

[7]:
```
model.save('model_resnet50.h5')
```

[ ]:
```

```

```
from __future__ import division, print_function

import sys
import os
import  os.path
import glob
import  re
import numpy as np

from PIL import Image
from keras.applications.imagenet_utils import
preprocess_input,decode_predictions
from keras.models import load_model
from keras.preprocessing import image
# from keras.applications.resnet50 import decode_predictions


from flask import Flask
# from flask.ext.sqlalchemy import SQLAlchemy
from flask import redirect,url_for,request,render_template
from werkzeug.utils import secure_filename
from gevent.pywsgi import WSGIServer

# # Actinic Keratoses         0.akiec
# # Basal cell carcinoma      1.bcc
# # Benign keratosis          2.bkl
# # Dermatofibroma            3.df
# # Melanoma                  4.mel
# # Melanocytic nevi          5.nv
# # Vascular                  6.vasc

# app = Flask(__name__)

MODEL_PATH = "/home/coder/Desktop/project/models/model_resnet50.h5"

mymodel = load_model(MODEL_PATH)
mymodel.make_predict_function()


import os
from flask import request
from flask import Flask
from flask import render_template
import PIL

app = Flask(__name__,static_url_path='/static')
MODEL_PATH = "/home/coder/Desktop/project/models/model.h5"

@app.route('/',methods=["GET","POST"])


def upload_pred():
    if request.method=="POST":
        image_file = request.files["image"]
```

```
        predict="No file uploaded"
        if image_file:
            base_path = os.path.dirname(os.path.abspath(__file__))
            file_path = os.path.join(base_path,'uploads',image_file.filename)
        #   validator
            # chk , not_file = is_valid_file(file_path)
            # if chk :
            image_file.save(file_path)
            predict = model_predict(file_path)
            # else:
            # predict="You Uploaded a " + not_file + " file !!!"
        return render_template("index.html",prediction=predict)
    return render_template("index.html",prediction="Upload Image")


def model_predict(file_path):
    MODEL_PATH = "/home/coder/Desktop/project/models/model.h5"
    mymodel = load_model(MODEL_PATH)
    mymodel.make_predict_function()
    class_name = ["Actinic keratoses","Basal cell carcinoma","Benign keratosis-like lesions","Dernatofibroma","Vascular lesions","Melanoma","Vascular"]
    img = image.load_img(file_path,target_size=(32,32))
    data = np.ndarray(shape=(1, 32, 32, 3), dtype=np.float32)
    x=image.img_to_array(img)
    normalized_image_array = (x.astype(np.float32) / 132.0) - 1
    data[0] = normalized_image_array
    predict1 = (mymodel.predict(data))*100000000000000
    print(predict1)
    index_max = np.argmax(predict1)
    print(index_max)
    return class_name[index_max]

# def is_valid_file(file_path):

#     validity=True
#     not_file=""

#     if file_path==None :
#         return False

#     file_path = str(file_path)

#     file_extension_list = list( map(str , file_path.split(".")) )
#     allowed_extension = ["jpg" , "png" , "webp"]

#     if file_extension_list[len(file_extension_list)-1] in allowed_extension :
#         validity = True
#     else:
#         validity = False
#         not_file += file_extension_list[len(file_extension_list)-1]

#     return ( validity , not_file )

```

```
if __name__ == "__main__":
    app.run(debug=True)
```