```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

```
!pwd
```

```
/content
```

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import os
from glob import glob
import seaborn as sns
from PIL import Image

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, MaxPool2D, BatchNormalization
from sklearn.model_selection import train_test_split
from scipy import stats
from sklearn.preprocessing import LabelEncoder
```

```
np.random.seed(42)
skin_df = pd.read_csv('/content/gdrive/MyDrive/IIITA Documents/M.Tech._2nd_SEM_Project_(mit2020068)/Datasets/HAM1(

SIZE=32
```

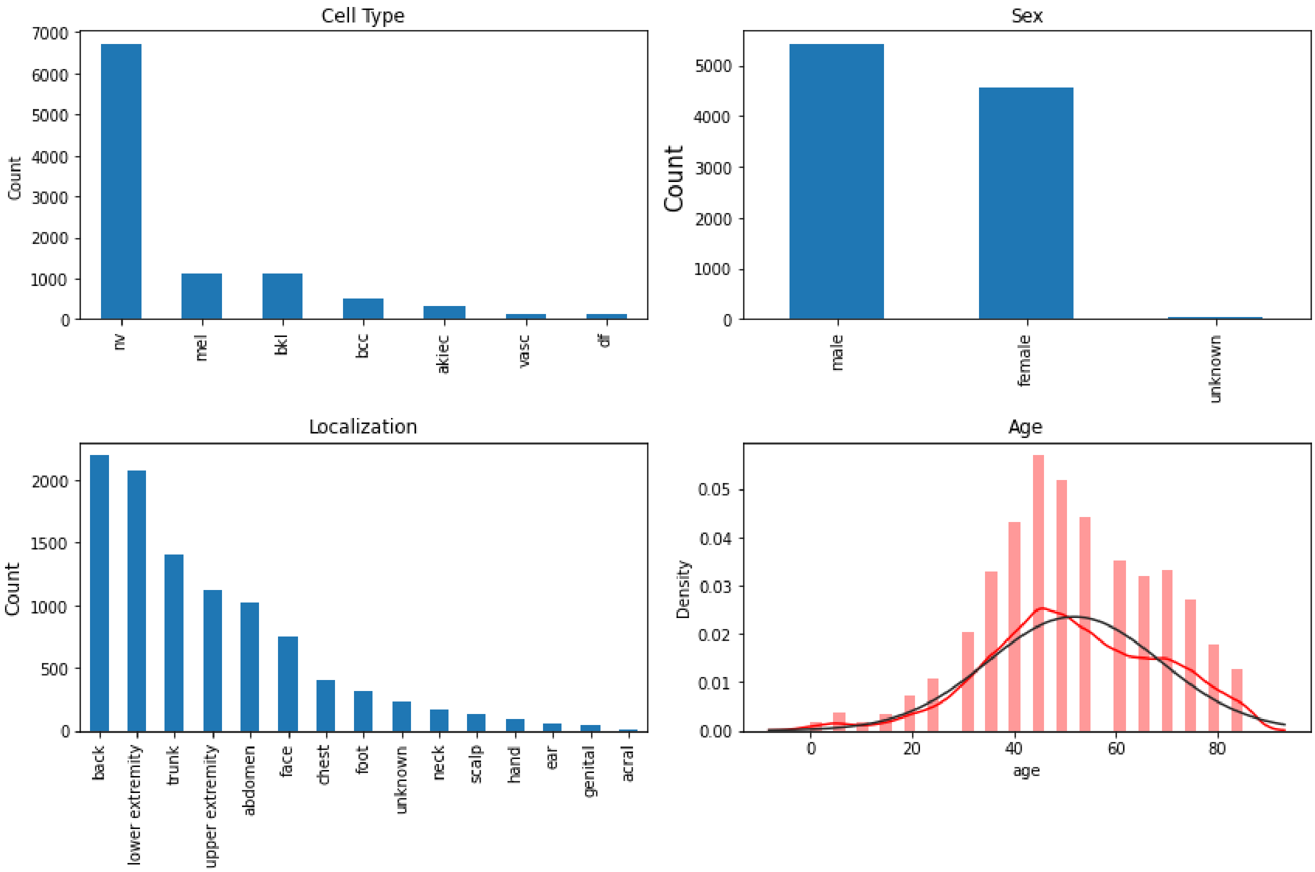
```
# label encoding to numeric values from text
le = LabelEncoder()
le.fit(skin_df['dx'])
LabelEncoder()
print(list(le.classes_))

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

```
['akiec', 'bcc', 'bkl', 'df', 'mel', 'nv', 'vasc']
        lesion_id      image_id   dx  ...     sex     localization label
1617  HAM_0007180  ISIC_0033272  mel  ...    male             face     4
8128  HAM_0007195  ISIC_0031923   nv  ...  female  lower extremity     5
2168  HAM_0001835  ISIC_0026652  mel  ...    male             back     4
1090  HAM_0000465  ISIC_0030583  bkl  ...  female            trunk     2
7754  HAM_0001720  ISIC_0034010   nv  ...    male          abdomen     5
8071  HAM_0006333  ISIC_0024424   nv  ...    male            trunk     5
7423  HAM_0004548  ISIC_0032832   nv  ...  female  upper extremity     5
8984  HAM_0006526  ISIC_0026671   nv  ...    male  lower extremity     5
2310  HAM_0003102  ISIC_0032389  mel  ...    male             face     4
7256  HAM_0004260  ISIC_0025525   nv  ...    male             back     5

[10 rows x 8 columns]
```

```
# Data distribution visualization
fig = plt.figure(figsize=(12,8))

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

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

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

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

plt.tight_layout()
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2557: FutureWarning: `distplot` is a depreca
  warnings.warn(msg, FutureWarning)
```




```
# Distribution of data into various classes
from sklearn.utils import resample
print("label Frequency")
print(skin_df['label'].value_counts())
print()
print("Class Frequency")
print(skin_df['dx'].value_counts())
```

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

```
Class Frequency
nv       6705
mel      1113
bkl      1099
bcc       514
akiec     327
vasc      142
df        115
Name: dx, dtype: int64
```

```
#Balance data.

df_0 = skin_df[skin_df['label'] == 0]
df_1 = skin_df[skin_df['label'] == 1]
df_2 = skin_df[skin_df['label'] == 2]
df_3 = skin_df[skin_df['label'] == 3]
df_4 = skin_df[skin_df['label'] == 4]
df_5 = skin_df[skin_df['label'] == 5]
df_6 = skin_df[skin_df['label'] == 6]

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)
```

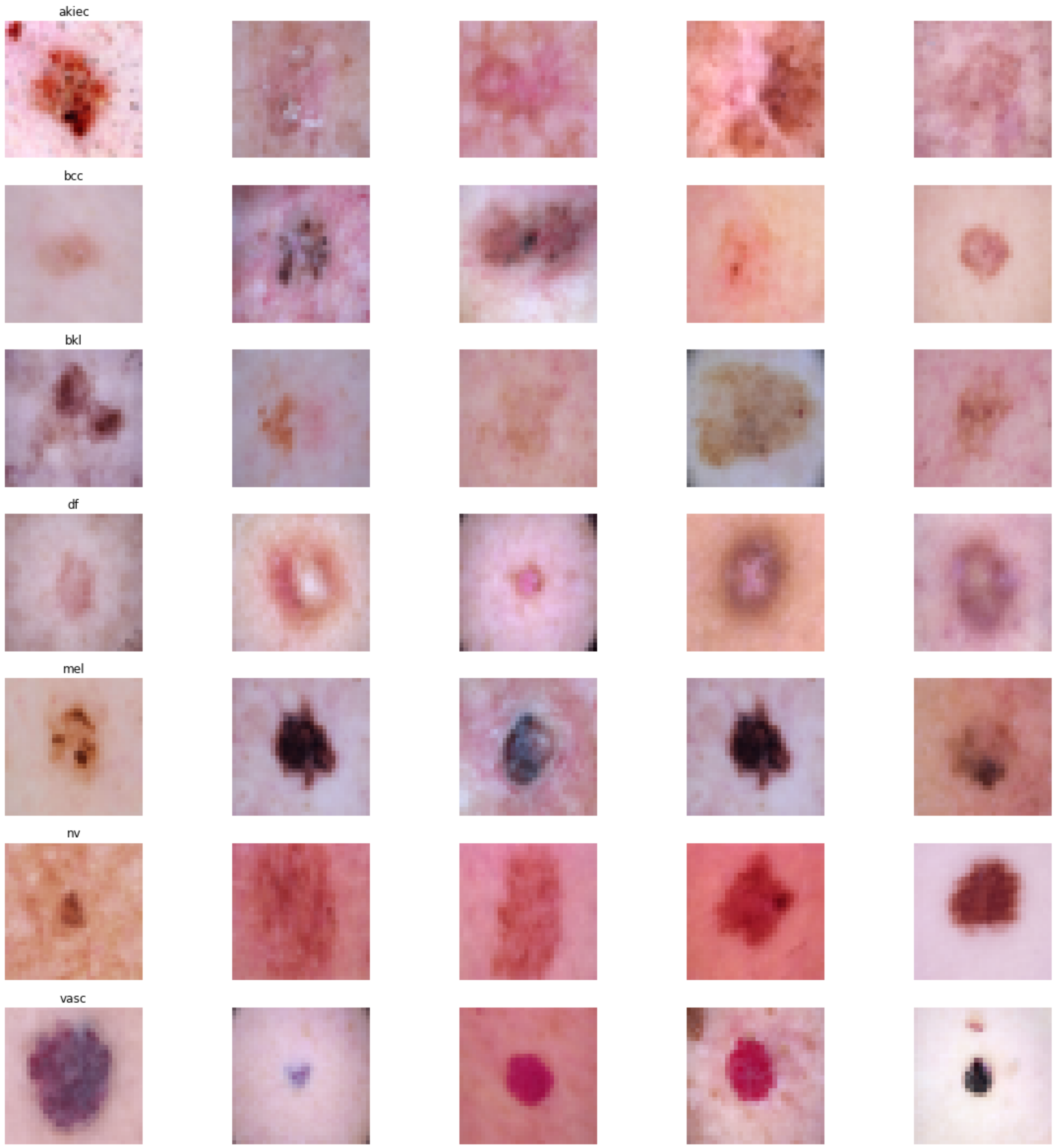
```
#Combined back to a single dataframe
skin_df_balanced = pd.concat([df_0_balanced, df_1_balanced,
                              df_2_balanced, df_3_balanced,
                              df_4_balanced, df_5_balanced, df_6_balanced])
```

```
#Check the distribution. All classes should be balanced now.
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
```

```
time to read images based on image ID from the CSV file
 is the safest way to read images as it ensures the right image is read for the right ID
_path = {os.path.splitext(os.path.basename(x))[0]: x
                for x in glob(os.path.join('/content/gdrive/MyDrive/IIITA Documents/M.Tech._2nd_SEM_Project_(mit20
```

```
#Define the path and add as a new column
skin_df_balanced['path'] = skin_df['image_id'].map(image_path.get)
#Use the path to read images.
skin_df_balanced['image'] = skin_df_balanced['path'].map(lambda x: np.asarray(Image.open(x).resize((SIZE,SIZE))))
n_samples = 5  # number of samples for plotting
# Plotting
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')
```




```
#Convert dataframe column of images into numpy array
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=0.25, random_state=42)
```

```
#Define the model.

num_classes = 7

model = Sequential()
model.add(Conv2D(256, (3, 3), activation="relu", input_shape=(SIZE, SIZE, 3)))
#model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(0.3))

model.add(Conv2D(128, (3, 3),activation='relu'))
#model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(0.3))

model.add(Conv2D(64, (3, 3),activation='relu'))
#model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(0.3))
model.add(Flatten())

model.add(Dense(32))
model.add(Dense(7, activation='softmax'))
model.summary()

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

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 30, 30, 256)       7168
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 15, 15, 256)       0
_________________________________________________________________
dropout_3 (Dropout)          (None, 15, 15, 256)       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 13, 13, 128)       295040
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 6, 6, 128)         0
_________________________________________________________________
dropout_4 (Dropout)          (None, 6, 6, 128)         0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 4, 4, 64)          73792
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 2, 2, 64)          0
_________________________________________________________________
dropout_5 (Dropout)          (None, 2, 2, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 32)                8224
_________________________________________________________________
dense_3 (Dense)              (None, 7)                 231
=================================================================
Total params: 384,455
Trainable params: 384,455
Non-trainable params: 0
_________________________________________________________________
```

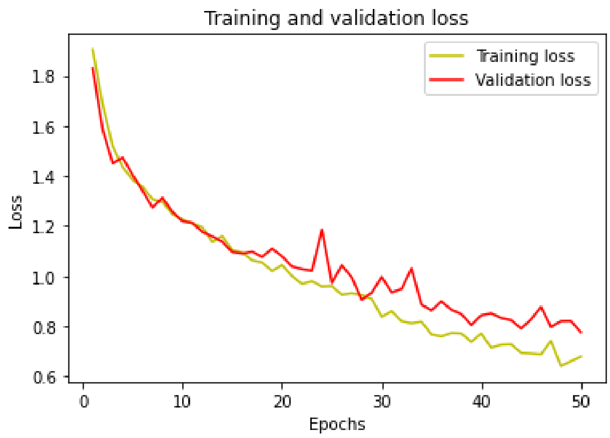
```
# Train
#You can also use generator to use augmentation during training.

batch_size = 16
epochs = 50

history = model.fit(
```

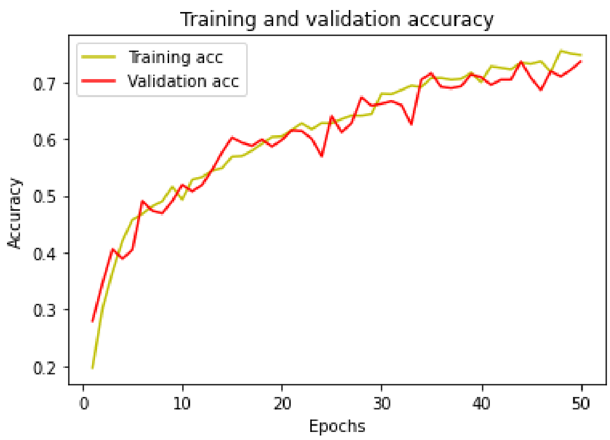
```
    x_train, y_train,
    epochs=epochs,
    batch_size = batch_size,
    validation_data=(x_test, y_test),
    verbose=2)

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

```
165/165 - 22s - loss: 0.9683 - acc: 0.6274 - val_loss: 1.0277 - val_acc: 0.6137
Epoch 23/50
165/165 - 23s - loss: 0.9795 - acc: 0.6168 - val_loss: 1.0212 - val_acc: 0.6000
Epoch 24/50
165/165 - 22s - loss: 0.9582 - acc: 0.6278 - val_loss: 1.1848 - val_acc: 0.5691
Epoch 25/50
165/165 - 22s - loss: 0.9599 - acc: 0.6274 - val_loss: 0.9722 - val_acc: 0.6400
Epoch 26/50
165/165 - 22s - loss: 0.9256 - acc: 0.6347 - val_loss: 1.0433 - val_acc: 0.6114
Epoch 27/50
165/165 - 22s - loss: 0.9315 - acc: 0.6411 - val_loss: 0.9946 - val_acc: 0.6274
Epoch 28/50
165/165 - 22s - loss: 0.9243 - acc: 0.6408 - val_loss: 0.9055 - val_acc: 0.6731
Epoch 29/50
165/165 - 23s - loss: 0.9086 - acc: 0.6438 - val_loss: 0.9335 - val_acc: 0.6583
Epoch 30/50
165/165 - 22s - loss: 0.8373 - acc: 0.6792 - val_loss: 0.9966 - val_acc: 0.6617
Epoch 31/50
165/165 - 23s - loss: 0.8593 - acc: 0.6785 - val_loss: 0.9349 - val_acc: 0.6663
Epoch 32/50
165/165 - 23s - loss: 0.8195 - acc: 0.6857 - val_loss: 0.9484 - val_acc: 0.6594
Epoch 33/50
165/165 - 22s - loss: 0.8113 - acc: 0.6937 - val_loss: 1.0306 - val_acc: 0.6251
Epoch 34/50
165/165 - 22s - loss: 0.8177 - acc: 0.6910 - val_loss: 0.8861 - val_acc: 0.7040
Epoch 35/50
165/165 - 22s - loss: 0.7668 - acc: 0.7070 - val_loss: 0.8626 - val_acc: 0.7154
Epoch 36/50
165/165 - 22s - loss: 0.7589 - acc: 0.7070 - val_loss: 0.8988 - val_acc: 0.6914
Epoch 37/50
165/165 - 22s - loss: 0.7724 - acc: 0.7040 - val_loss: 0.8652 - val_acc: 0.6891
Epoch 38/50
165/165 - 22s - loss: 0.7702 - acc: 0.7055 - val_loss: 0.8482 - val_acc: 0.6926
Epoch 39/50
165/165 - 22s - loss: 0.7373 - acc: 0.7162 - val_loss: 0.8046 - val_acc: 0.7131
Epoch 40/50
165/165 - 22s - loss: 0.7700 - acc: 0.6998 - val_loss: 0.8431 - val_acc: 0.7086
Epoch 41/50
165/165 - 22s - loss: 0.7137 - acc: 0.7276 - val_loss: 0.8511 - val_acc: 0.6949
Epoch 42/50
165/165 - 22s - loss: 0.7256 - acc: 0.7250 - val_loss: 0.8323 - val_acc: 0.7040
Epoch 43/50
165/165 - 23s - loss: 0.7274 - acc: 0.7223 - val_loss: 0.8241 - val_acc: 0.7040
Epoch 44/50
165/165 - 22s - loss: 0.6931 - acc: 0.7341 - val_loss: 0.7903 - val_acc: 0.7360
Epoch 45/50
165/165 - 22s - loss: 0.6908 - acc: 0.7318 - val_loss: 0.8270 - val_acc: 0.7074
Epoch 46/50
165/165 - 22s - loss: 0.6871 - acc: 0.7364 - val_loss: 0.8765 - val_acc: 0.6857
Epoch 47/50
165/165 - 21s - loss: 0.7406 - acc: 0.7181 - val_loss: 0.7961 - val_acc: 0.7189
Epoch 48/50
165/165 - 22s - loss: 0.6403 - acc: 0.7550 - val_loss: 0.8198 - val_acc: 0.7097
Epoch 49/50
165/165 - 21s - loss: 0.6595 - acc: 0.7501 - val_loss: 0.8207 - val_acc: 0.7211
Epoch 50/50
165/165 - 21s - loss: 0.6781 - acc: 0.7474 - val_loss: 0.7753 - val_acc: 0.7360
28/28 [==============================] - 2s 61ms/step - loss: 0.7753 - acc: 0.7360
Test accuracy: 0.7360000014305115
```

```
#plot the training and validation accuracy and loss at each epoch
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()


acc = history.history['acc']
val_acc = history.history['val_acc']
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()
```

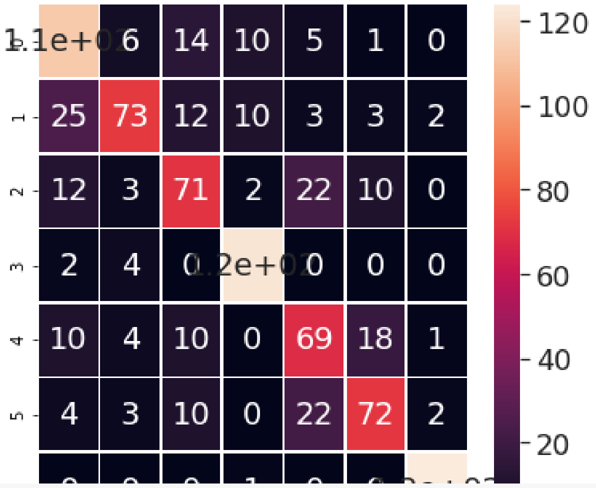





```
# Prediction on test data
y_pred = model.predict(x_test)
# Convert predictions classes to one hot vectors
y_pred_classes = np.argmax(y_pred, axis = 1)
# Convert test data to one hot vectors
y_true = np.argmax(y_test, axis = 1)
```

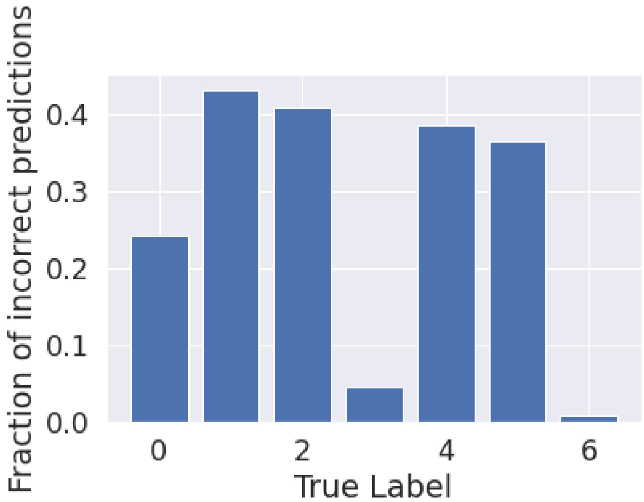
```
#Print confusion matrix
cm = confusion_matrix(y_true, y_pred_classes)

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

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd1b26968d0>
```




```
#PLot fractional incorrect misclassifications
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')
```

```
Text(0, 0.5, 'Fraction of incorrect predictions')
```



✓ 0s completed at 1:09 PM