mnist_train

August 29, 2024

[ ]:
```
%matplotlib inline
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
np.random.seed(123)
from sklearn.preprocessing import label_binarize
from sklearn.metrics import confusion_matrix
import itertools

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
from keras import backend as K
import itertools
from keras.layers import BatchNormalization
from keras.utils import to_categorical # convert to one-hot-encoding

from keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ReduceLROnPlateau
from sklearn.model_selection import train_test_split
```

[ ]:
```
base_skin_dir = 'archive/'

# Merging images from both folders HAM10000_images_part1.zip and␣
↪HAM10000_images_part2.zip into one dictionary

imageid_path_dict = {os.path.splitext(os.path.basename(x))[0]: x
                     for x in glob(os.path.join(base_skin_dir, '*', '*.jpg'))}

# This dictionary is useful for displaying more human-friendly labels later on
```

```python
lesion_type_dict = {
    'nv': 'Melanocytic nevi',
    'mel': 'Melanoma',
    'bkl': 'Benign keratosis-like lesions ',
    'bcc': 'Basal cell carcinoma',
    'akiec': 'Actinic keratoses',
    'vasc': 'Vascular lesions',
    'df': 'Dermatofibroma'
}
```

[ ]:
```python
skin_df = pd.read_csv(os.path.join(base_skin_dir, 'HAM10000_metadata.csv'))

# Creating New Columns for better readability

skin_df['path'] = skin_df['image_id'].map(imageid_path_dict.get)
skin_df['cell_type'] = skin_df['dx'].map(lesion_type_dict.get)
skin_df['cell_type_idx'] = pd.Categorical(skin_df['cell_type']).codes
```

[ ]:
```python
# Now lets see the sample of tile_df to look on newly made columns
skin_df.head()
```

[ ]:
```
     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

                                               path  \
0  archive/HAM10000_images_part_1/ISIC_0027419.jpg
1  archive/HAM10000_images_part_1/ISIC_0025030.jpg
2  archive/HAM10000_images_part_1/ISIC_0026769.jpg
3  archive/HAM10000_images_part_1/ISIC_0025661.jpg
4  archive/HAM10000_images_part_2/ISIC_0031633.jpg

                        cell_type  cell_type_idx
0  Benign keratosis-like lesions              2
1  Benign keratosis-like lesions              2
2  Benign keratosis-like lesions              2
3  Benign keratosis-like lesions              2
4  Benign keratosis-like lesions              2
```

[ ]:
```python
# skrink dataset
print(f"Original Size: {skin_df.shape}")
#skin_df = skin_df.sample(frac=0.1) # shuffle the dataset
print(f"Shrunk Size: {skin_df.shape}")
```

```
Original Size: (10015, 10)
Shrunk Size: (10015, 10)
```

[ ]:
```
skin_df.isnull().sum()
```

[ ]:
```
lesion_id         0
image_id          0
dx                0
dx_type           0
age              57
sex               0
localization      0
path              0
cell_type         0
cell_type_idx     0
dtype: int64
```

[ ]:
```
skin_df['age'].fillna((skin_df['age'].mean()), inplace=True)
```

[ ]:
```
skin_df.isnull().sum()
```

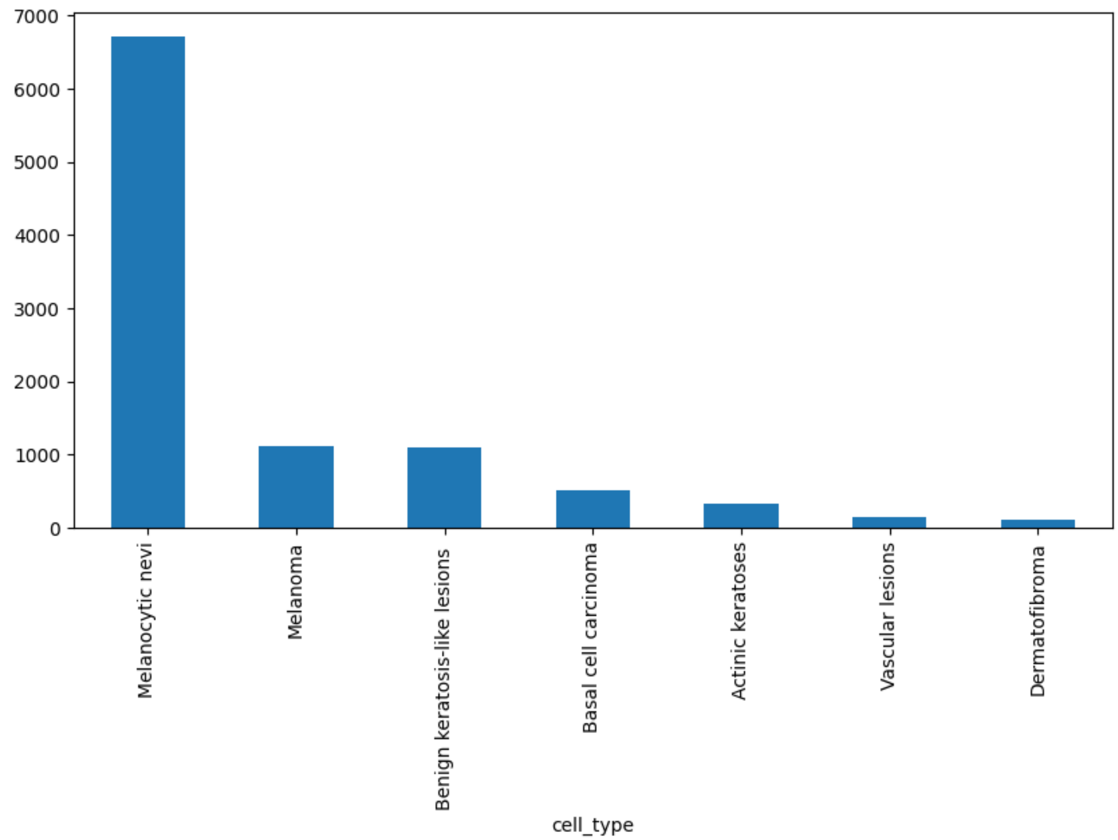
[ ]:
```
lesion_id        0
image_id         0
dx               0
dx_type          0
age              0
sex              0
localization     0
path             0
cell_type        0
cell_type_idx    0
dtype: int64
```

[ ]:
```
print(skin_df.dtypes)
```

```
lesion_id         object
image_id          object
dx                object
dx_type           object
age              float64
sex               object
localization      object
path              object
cell_type         object
cell_type_idx       int8
dtype: object
```

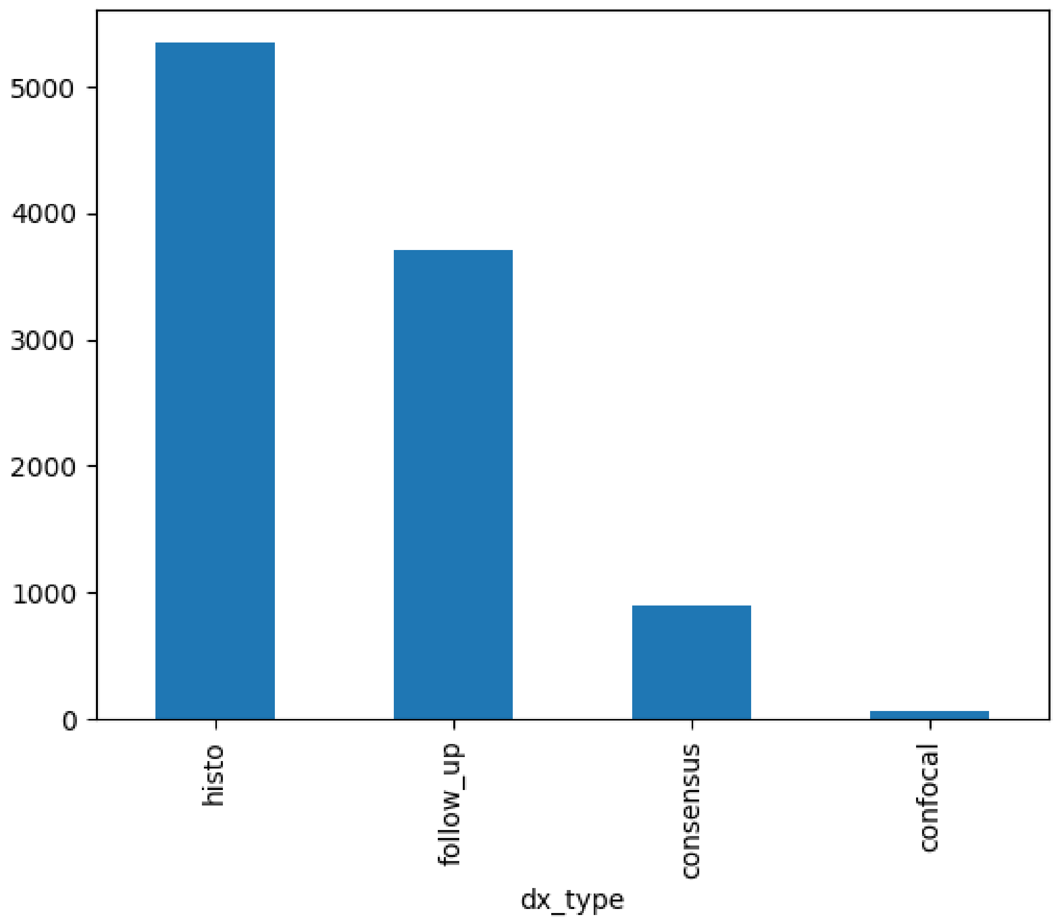
[ ]:
```
fig, ax1 = plt.subplots(1, 1, figsize= (10, 5))
skin_df['cell_type'].value_counts().plot(kind='bar', ax=ax1)
```

[ ]: <Axes: xlabel='cell_type'>



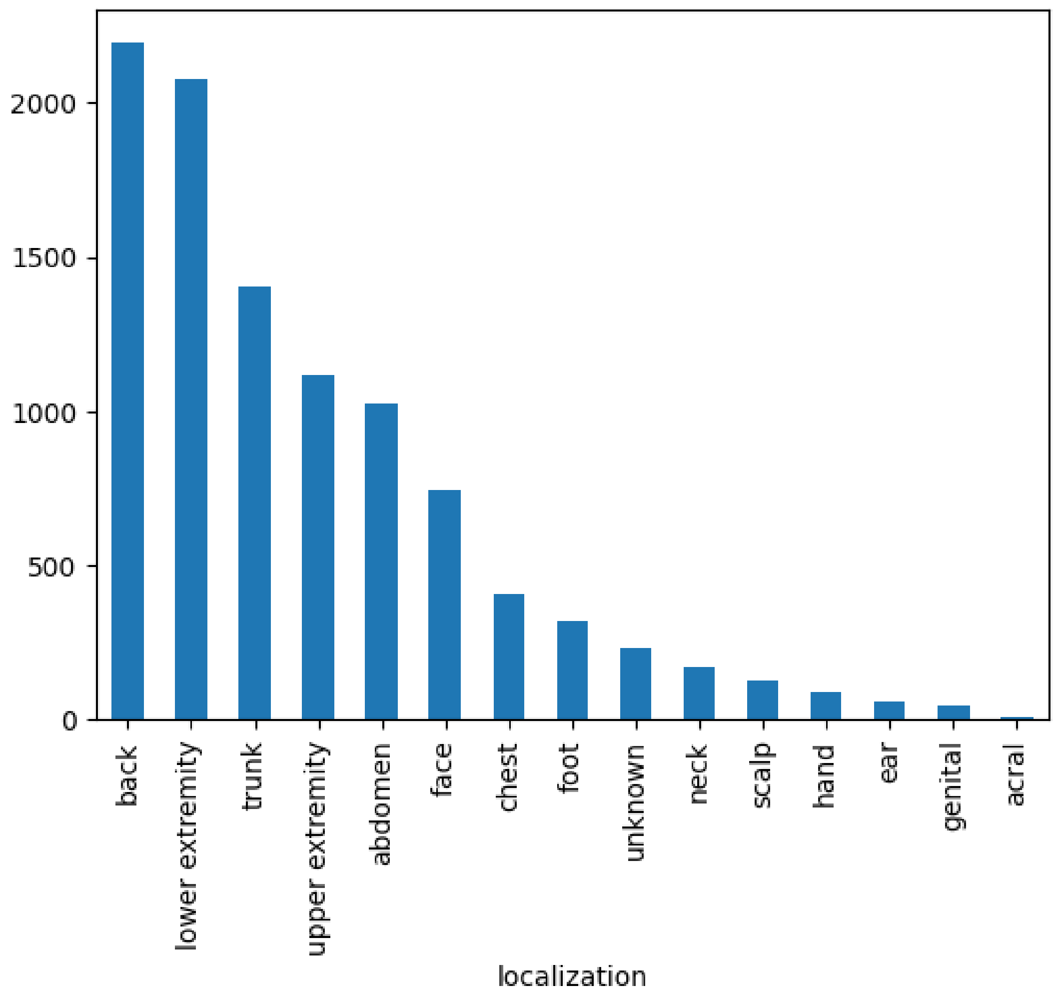
[ ]:
```
skin_df['dx_type'].value_counts().plot(kind='bar')
```

[ ]: <Axes: xlabel='dx_type'>



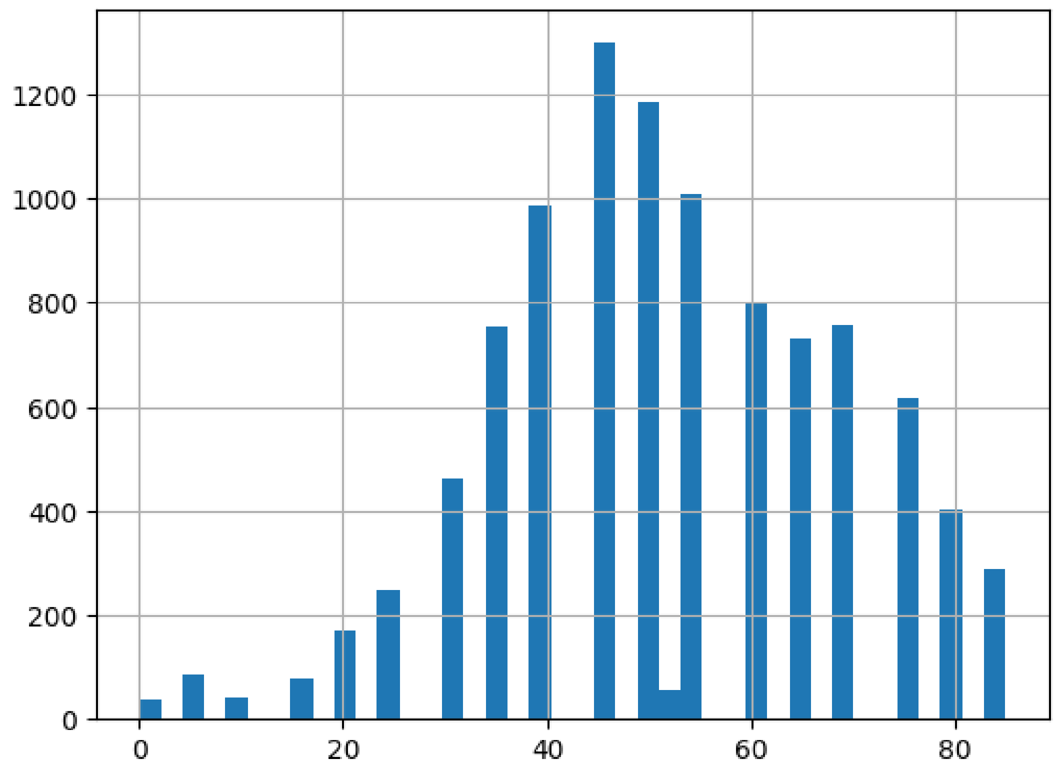
[ ]:
```
skin_df['localization'].value_counts().plot(kind='bar')
```

[ ]: <Axes: xlabel='localization'>



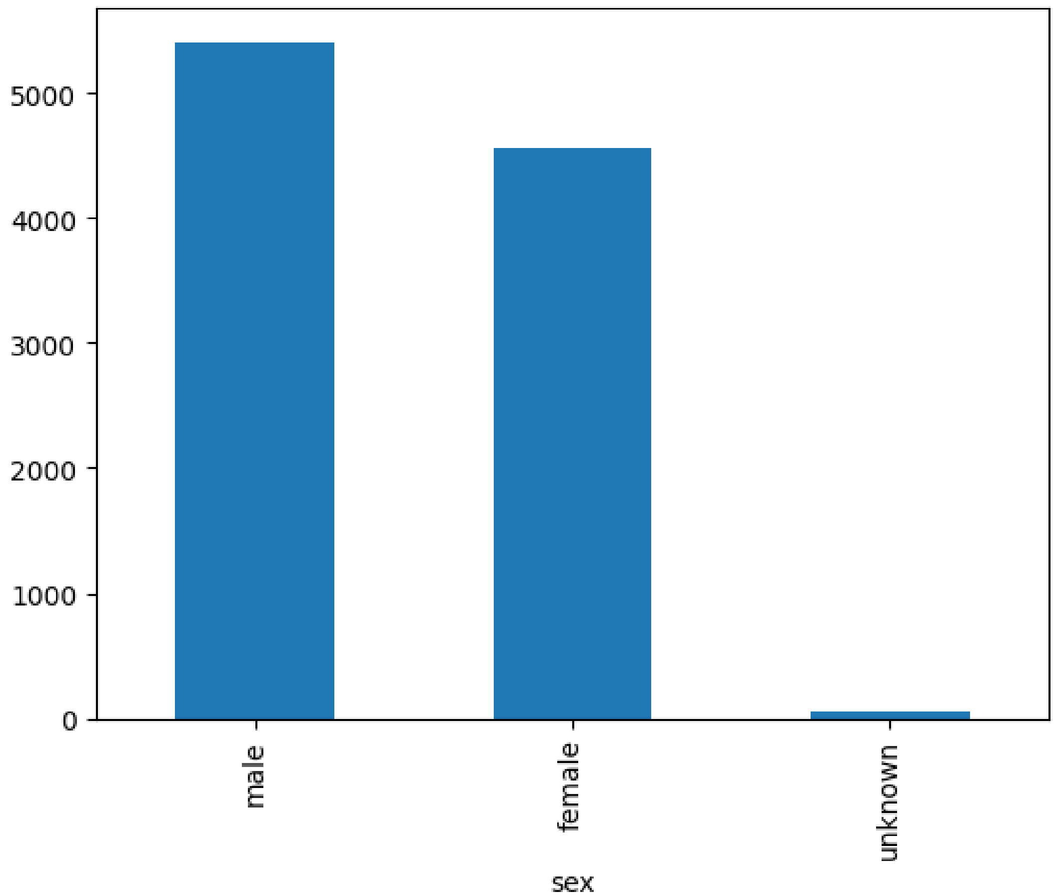
[ ]:
```
skin_df['age'].hist(bins=40)
```

[ ]: <Axes: >



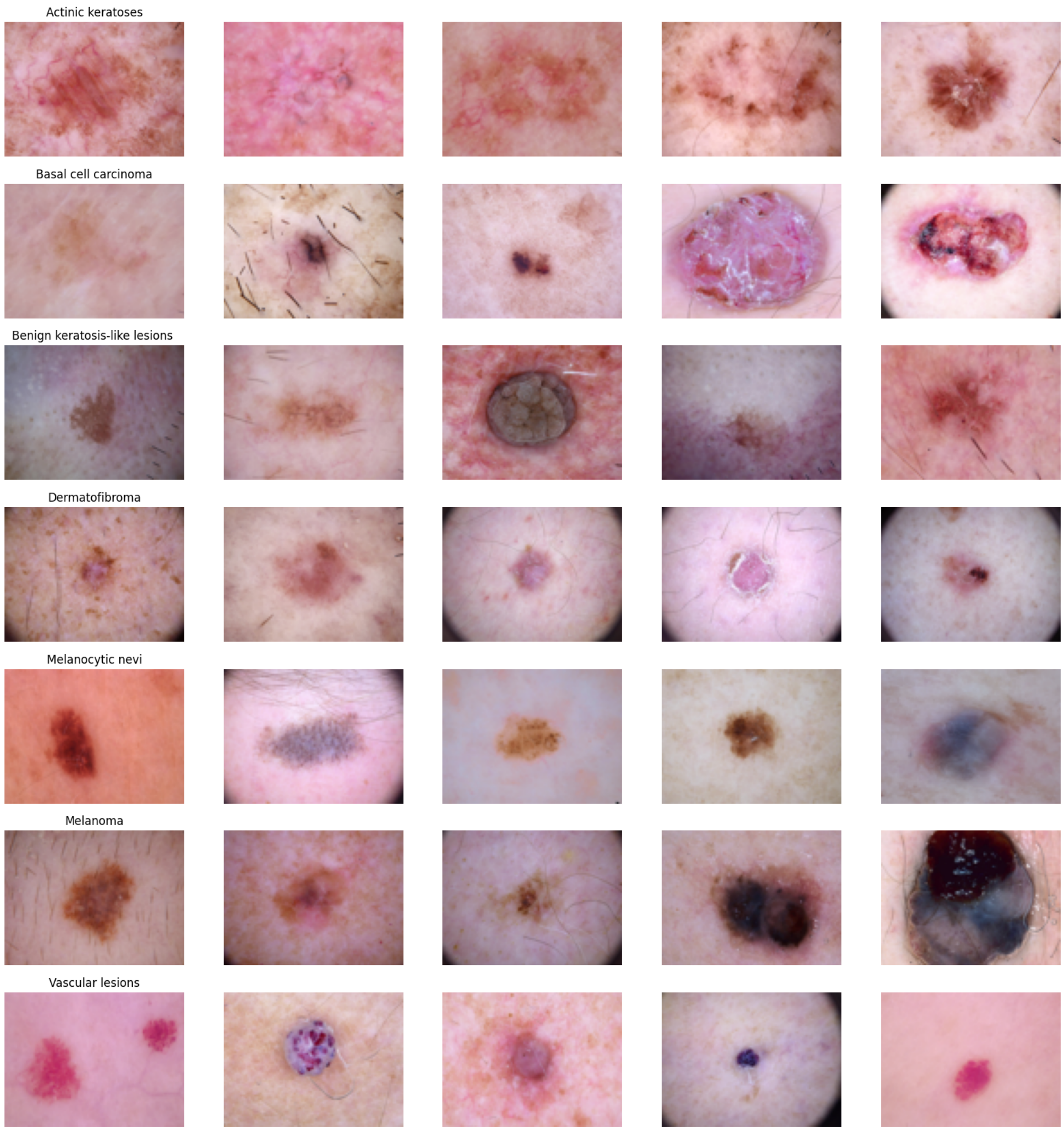
[ ]:
```
skin_df['sex'].value_counts().plot(kind='bar')
```

[ ]: <Axes: xlabel='sex'>




```
[ ]: skin_df['image'] = skin_df['path'].map(lambda x: np.asarray(Image.open(x).
      ↪resize((100,75))))
```

```
[ ]: 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.sort_values(['cell_type']).
      ↪groupby('cell_type')):
         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')

     #fig.savefig('category_samples.png', dpi=300)
```




```
[ ]: # Checking the image size distribution
     skin_df['image'].map(lambda x: x.shape).value_counts()
```

```
[ ]: image
     (75, 100, 3)    10015
     Name: count, dtype: int64
```

```
[ ]: features=skin_df.drop(columns=['cell_type_idx'],axis=1)
     target=skin_df['cell_type_idx']
```

```
[ ]: x_train_o, x_test_o, y_train_o, y_test_o = train_test_split(features, target,␣
      ↪test_size=0.20,random_state=1234)
```

```
[ ]: x_train = np.asarray(x_train_o['image'].tolist())
     x_test = np.asarray(x_test_o['image'].tolist())

     x_train_mean = np.mean(x_train)
     x_train_std = np.std(x_train)

     x_test_mean = np.mean(x_test)
     x_test_std = np.std(x_test)

     x_train = (x_train - x_train_mean)/x_train_std
     x_test = (x_test - x_test_mean)/x_test_std
```

```
[ ]: # Perform one-hot encoding on the labels
     y_train = to_categorical(y_train_o, num_classes = 7)
     y_test = to_categorical(y_test_o, num_classes = 7)
```

```
[ ]: x_train, x_validate, y_train, y_validate = train_test_split(x_train, y_train,␣
      ↪test_size = 0.1, random_state = 2)
```

```
[ ]: # Reshape image in 3 dimensions (height = 75px, width = 100px , canal = 3)
     x_train = x_train.reshape(x_train.shape[0], *(75, 100, 3))
     x_test = x_test.reshape(x_test.shape[0], *(75, 100, 3))
     x_validate = x_validate.reshape(x_validate.shape[0], *(75, 100, 3))
```

```
[ ]: # Set the CNN model
     # my CNN architechture is In -> [[Conv2D->relu]*2 -> MaxPool2D -> Dropout]*2 ->␣
      ↪Flatten -> Dense -> Dropout -> Out
     input_shape = (75, 100, 3)
     num_classes = 7

     model = Sequential()
     model.add(Conv2D(32, kernel_size=(3, 3),activation='relu',padding =␣
      ↪'Same',input_shape=input_shape))
     model.add(Conv2D(32,kernel_size=(3, 3), activation='relu',padding = 'Same',))
     model.add(MaxPool2D(pool_size = (2, 2)))
     model.add(Dropout(0.25))


     model.add(Conv2D(64, (3, 3), activation='relu',padding = 'Same'))
     model.add(Conv2D(64, (3, 3), activation='relu',padding = 'Same'))
     model.add(MaxPool2D(pool_size=(2, 2)))
     model.add(Dropout(0.40))

     model.add(Flatten())
```

```python
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
model.summary()
```

```
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-
packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not
pass an `input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in the model
instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential_18"

Layer (type)	Output Shape	Param #
conv2d_74 (Conv2D)	(None, 75, 100, 32)	896
conv2d_75 (Conv2D)	(None, 75, 100, 32)	9,248
max_pooling2d_37 (MaxPooling2D)	(None, 37, 50, 32)	0
dropout_56 (Dropout)	(None, 37, 50, 32)	0
conv2d_76 (Conv2D)	(None, 37, 50, 64)	18,496
conv2d_77 (Conv2D)	(None, 37, 50, 64)	36,928
max_pooling2d_38 (MaxPooling2D)	(None, 18, 25, 64)	0
dropout_57 (Dropout)	(None, 18, 25, 64)	0
flatten_18 (Flatten)	(None, 28800)	0
dense_37 (Dense)	(None, 128)	3,686,528
dropout_58 (Dropout)	(None, 128)	0
dense_38 (Dense)	(None, 7)	903

Total params: 3,752,999 (14.32 MB)

Trainable params: 3,752,999 (14.32 MB)

Non-trainable params: 0 (0.00 B)

[ ]:
```
# Define the optimizer
optimizer = Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999, amsgrad=False)
```

[ ]:
```
# Compile the model
model.compile(optimizer = optimizer , loss = "categorical_crossentropy",␣
↪metrics=["accuracy"])
```

[ ]:
```
# Set a learning rate annealer
learning_rate_reduction = ReduceLROnPlateau(monitor='val_accuracy',
                                            patience=3,
                                            verbose=1,
                                            factor=0.5,
                                            min_lr=0.00001)
```

[ ]:
```
# With data augmentation to prevent overfitting

datagen = ImageDataGenerator(
        featurewise_center=False,  # set input mean to 0 over the dataset
        samplewise_center=False,  # set each sample mean to 0
        featurewise_std_normalization=False,  # divide inputs by std of the␣
↪dataset
        samplewise_std_normalization=False,  # divide each input by its std
        zca_whitening=False,  # apply ZCA whitening
        rotation_range=10,  # randomly rotate images in the range (degrees, 0␣
↪to 180)
        zoom_range = 0.1, # Randomly zoom image
        width_shift_range=0.1,  # randomly shift images horizontally (fraction␣
↪of total width)
        height_shift_range=0.1,  # randomly shift images vertically (fraction␣
↪of total height)
        horizontal_flip=False,  # randomly flip images
        vertical_flip=False)  # randomly flip images

datagen.fit(x_train)
```

[ ]:
```
# Fit the model
EPOCHS = 25
batch_size = 32

history = model.fit(datagen.flow(x_train,y_train, batch_size=batch_size),
                              epochs = EPOCHS, validation_data =␣
↪(x_validate,y_validate),
                              verbose = 1, steps_per_epoch=x_train.shape[0] //␣
↪batch_size
```

```
                              , callbacks=[learning_rate_reduction])
```

```
Epoch 1/25

/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-
packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:122:
UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in
its constructor. `**kwargs` can include `workers`, `use_multiprocessing`,
`max_queue_size`. Do not pass these arguments to `fit()`, as they will be
ignored.
  self._warn_if_super_not_called()

225/225             34s 148ms/step -
accuracy: 0.6495 - loss: 1.1965 - val_accuracy: 0.6808 - val_loss: 0.8905 -
learning_rate: 0.0010
Epoch 2/25
  1/225             45s 201ms/step - accuracy:
0.7188 - loss: 0.6985

/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/contextlib.py:
155: UserWarning: Your input ran out of data; interrupting training. Make sure
that your dataset or generator can generate at least `steps_per_epoch * epochs`
batches. You may need to use the `.repeat()` function when building your
dataset.
  self.gen.throw(typ, value, traceback)

225/225             1s 4ms/step -
accuracy: 0.7188 - loss: 0.6985 - val_accuracy: 0.6808 - val_loss: 0.8875 -
learning_rate: 0.0010
Epoch 3/25
Epoch 3/25
225/225             34s 149ms/step -
accuracy: 0.6730 - loss: 0.9239 - val_accuracy: 0.6858 - val_loss: 0.8886 -
learning_rate: 0.0010
Epoch 4/25
225/225             1s 4ms/step -
accuracy: 0.5938 - loss: 1.1290 - val_accuracy: 0.6845 - val_loss: 0.8513 -
learning_rate: 0.0010
Epoch 5/25
225/225             33s 148ms/step -
accuracy: 0.6792 - loss: 0.8862 - val_accuracy: 0.6945 - val_loss: 0.7833 -
learning_rate: 0.0010
Epoch 6/25
225/225             1s 4ms/step -
accuracy: 0.6562 - loss: 0.9194 - val_accuracy: 0.6945 - val_loss: 0.7813 -
learning_rate: 0.0010
Epoch 7/25
225/225             33s 148ms/step -
accuracy: 0.6827 - loss: 0.8550 - val_accuracy: 0.7020 - val_loss: 0.7971 -
learning_rate: 0.0010
```

```
Epoch 8/25
225/225              1s 4ms/step -
accuracy: 0.7500 - loss: 0.6406 - val_accuracy: 0.7045 - val_loss: 0.8081 -
learning_rate: 0.0010
Epoch 9/25
225/225              34s 149ms/step -
accuracy: 0.6896 - loss: 0.8287 - val_accuracy: 0.7269 - val_loss: 0.7328 -
learning_rate: 0.0010
Epoch 10/25
225/225              1s 4ms/step -
accuracy: 0.7500 - loss: 0.7513 - val_accuracy: 0.7257 - val_loss: 0.7376 -
learning_rate: 0.0010
Epoch 11/25
225/225              34s 149ms/step -
accuracy: 0.7062 - loss: 0.7945 - val_accuracy: 0.7419 - val_loss: 0.7099 -
learning_rate: 0.0010
Epoch 12/25
225/225              1s 4ms/step -
accuracy: 0.6562 - loss: 0.7024 - val_accuracy: 0.7382 - val_loss: 0.7104 -
learning_rate: 0.0010
Epoch 13/25
225/225              34s 149ms/step -
accuracy: 0.7070 - loss: 0.7999 - val_accuracy: 0.7332 - val_loss: 0.7191 -
learning_rate: 0.0010
Epoch 14/25
  1/225              31s 142ms/step - accuracy:
0.6250 - loss: 0.7672
Epoch 14: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.
225/225              1s 4ms/step -
accuracy: 0.6250 - loss: 0.7672 - val_accuracy: 0.7319 - val_loss: 0.7229 -
learning_rate: 0.0010
Epoch 15/25
225/225              33s 146ms/step -
accuracy: 0.7327 - loss: 0.7345 - val_accuracy: 0.7494 - val_loss: 0.6926 -
learning_rate: 5.0000e-04
Epoch 16/25
225/225              1s 4ms/step -
accuracy: 0.7500 - loss: 0.6122 - val_accuracy: 0.7494 - val_loss: 0.6907 -
learning_rate: 5.0000e-04
Epoch 17/25
225/225              34s 148ms/step -
accuracy: 0.7412 - loss: 0.7250 - val_accuracy: 0.7718 - val_loss: 0.6604 -
learning_rate: 5.0000e-04
Epoch 18/25
225/225              1s 4ms/step -
accuracy: 0.6562 - loss: 0.7531 - val_accuracy: 0.7706 - val_loss: 0.6602 -
learning_rate: 5.0000e-04
Epoch 19/25
```

```
225/225 34s 149ms/step -
accuracy: 0.7381 - loss: 0.7221 - val_accuracy: 0.7544 - val_loss: 0.6809 -
learning_rate: 5.0000e-04
Epoch 20/25
  1/225 31s 140ms/step - accuracy:
0.6875 - loss: 0.5997
Epoch 20: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.
225/225 1s 4ms/step -
accuracy: 0.6875 - loss: 0.5997 - val_accuracy: 0.7494 - val_loss: 0.6840 -
learning_rate: 5.0000e-04
Epoch 21/25
225/225 34s 150ms/step -
accuracy: 0.7348 - loss: 0.7056 - val_accuracy: 0.7681 - val_loss: 0.6436 -
learning_rate: 2.5000e-04
Epoch 22/25
225/225 1s 4ms/step -
accuracy: 0.8125 - loss: 0.5895 - val_accuracy: 0.7668 - val_loss: 0.6437 -
learning_rate: 2.5000e-04
Epoch 23/25
225/225 0s 147ms/step -
accuracy: 0.7529 - loss: 0.6692
Epoch 23: ReduceLROnPlateau reducing learning rate to 0.0001250000059371814.
225/225 34s 151ms/step -
accuracy: 0.7529 - loss: 0.6692 - val_accuracy: 0.7643 - val_loss: 0.6600 -
learning_rate: 2.5000e-04
Epoch 24/25
225/225 1s 5ms/step -
accuracy: 0.7188 - loss: 0.7034 - val_accuracy: 0.7631 - val_loss: 0.6592 -
learning_rate: 1.2500e-04
Epoch 25/25
225/225 34s 151ms/step -
accuracy: 0.7596 - loss: 0.6532 - val_accuracy: 0.7656 - val_loss: 0.6295 -
learning_rate: 1.2500e-04
```

[ ]:
```
loss, accuracy = model.evaluate(x_test, y_test, verbose=1)
loss_v, accuracy_v = model.evaluate(x_validate, y_validate, verbose=1)

print("Validation: accuracy = %f  ;  loss_v = %f" % (accuracy_v, loss_v))
print("Test: accuracy = %f  ;  loss = %f" % (accuracy, loss))
```

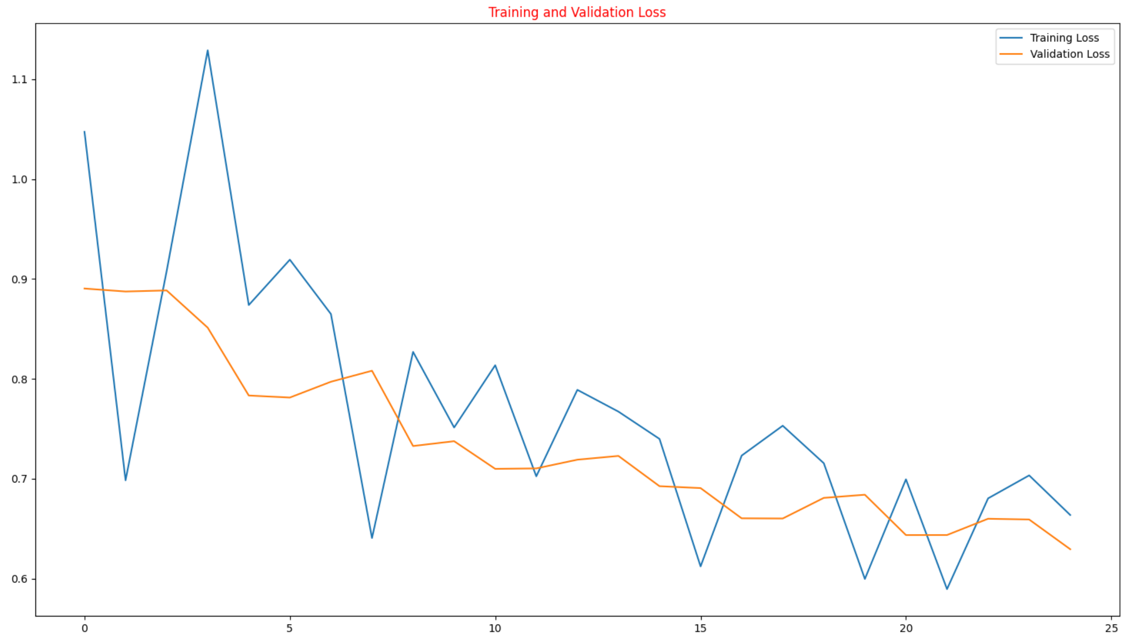
```
63/63 2s 36ms/step -
accuracy: 0.7663 - loss: 0.6267
26/26 1s 37ms/step -
accuracy: 0.7821 - loss: 0.6014
Validation: accuracy = 0.765586  ;  loss_v = 0.629457
Test: accuracy = 0.754368  ;  loss = 0.639162
```

[ ]:
```
model.save(f"model_test_acc_{accuracy:.2f}.keras")
```

[ ]:
```python
# Graph of Binary Accuracy
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss=history.history['loss']
val_loss=history.history['val_loss']

epochs_range = range(EPOCHS)

plt.figure(figsize=(40, 10))
plt.subplot(1, 2, 1)
#plt.grid()
plt.plot(epochs_range, acc, label='Training Binary Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Binary Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Binary Accuracy', color='Green')
plt.savefig('TrainingValidationAccuracy.png')
```

[ ]:
```python
# Graph of Loss
plt.figure(figsize=(40, 10))
plt.subplot(1, 2, 2)
#plt.grid()
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss', color='red')
```

```python
plt.show()
plt.savefig('TrainingValidationLoss.png')
```

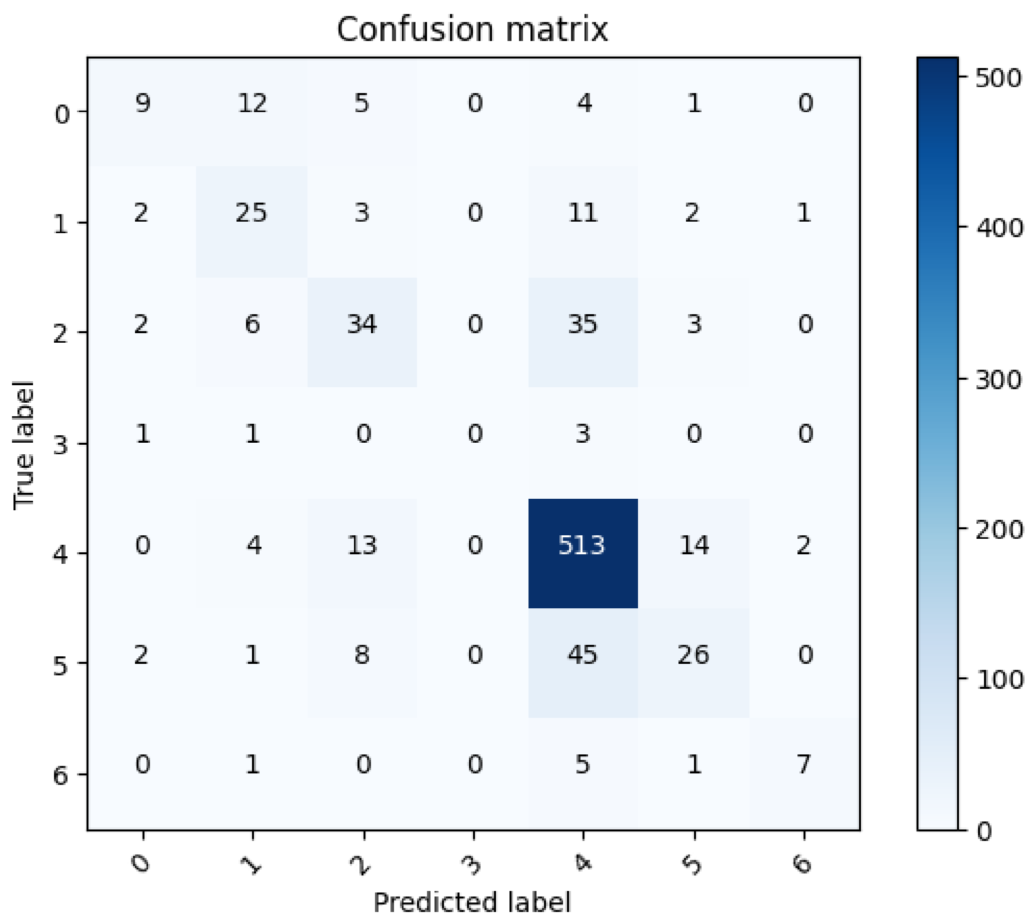


<Figure size 640x480 with 0 Axes>

[ ]:
```python
# Function to plot confusion matrix
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
```

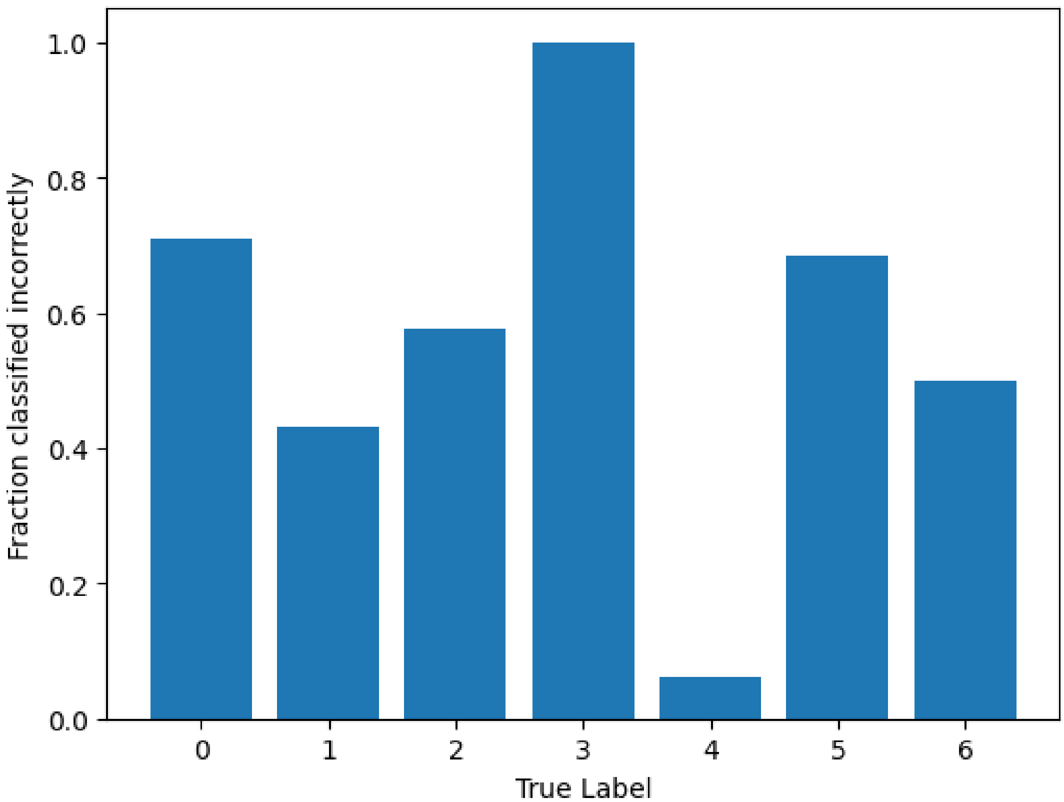
```
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

# Predict the values from the validation dataset
Y_pred = model.predict(x_validate)
# Convert predictions classes to one hot vectors
Y_pred_classes = np.argmax(Y_pred,axis = 1)
# Convert validation observations to one hot vectors
Y_true = np.argmax(y_validate,axis = 1)
# compute the confusion matrix
confusion_mtx = confusion_matrix(Y_true, Y_pred_classes)



# plot the confusion matrix
plot_confusion_matrix(confusion_mtx, classes = range(7))
```

```
26/26                 1s 37ms/step
```



[ ]:
```
label_frac_error = 1 - np.diag(confusion_mtx) / np.sum(confusion_mtx, axis=1)
plt.bar(np.arange(7),label_frac_error)
plt.xlabel('True Label')
plt.ylabel('Fraction classified incorrectly')
```

[ ]:
```
Text(0, 0.5, 'Fraction classified incorrectly')
```


1.0
0.8
0.6
0.4
0.2
0.0
Fraction classified incorrectly
0
1
2
3
4
5
6
True Label