In [1]:
```
import matplotlib.pyplot as plt
from PIL import Image
import seaborn as sns
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
import time
```

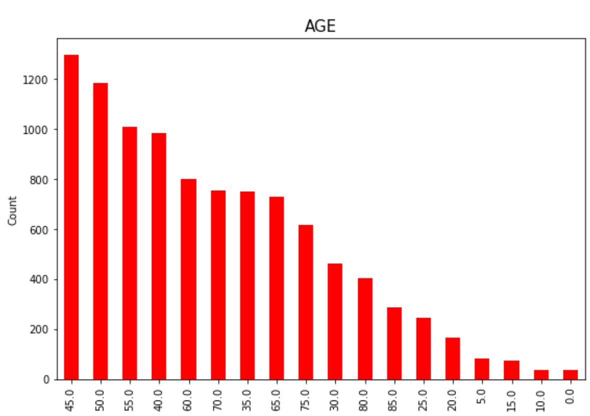
In [3]:
```
import os
os.getcwd()
```

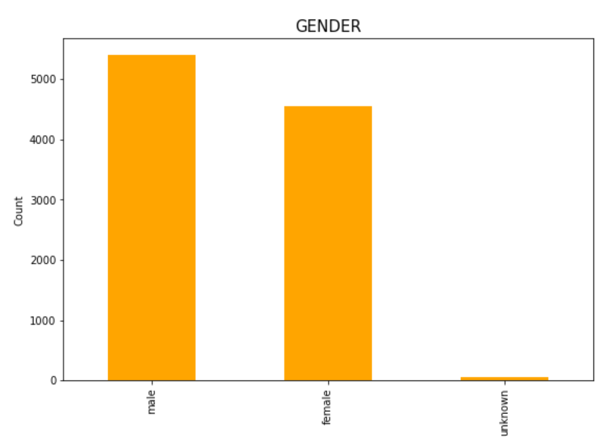
Out[3]:
```
'C:\\Users\\irrro\\OneDrive\\Documents\\Documents\\iro\\uni\\Machine learning
\\ml_skin_cancer_project\\notebooks'
```

In [5]:
```
# import the data of images
dataset_images = pd.read_csv("C:\\Users\\irrro\\OneDrive\\Documents\\Documents\\i


# Reading the data from HAM_metadata.csv
df_metadata = pd.read_csv("C:\\Users\\irrro\\OneDrive\\Documents\\Documents\\iro\
```

In [6]:
```
# Plot side by side plot of count vs age and gender
plt.figure(figsize=(20,10))
plt.subplots_adjust(left=0.125, bottom=1, right=0.9, top=2, hspace=0.2)
plt.subplot(2,2,1)
plt.colormaps()
plt.title("AGE",fontsize=15)
plt.ylabel("Count")
df_metadata['age'].value_counts().plot.bar(color = "red")
plt.subplot(2,2,2)
plt.title("GENDER",fontsize=15)
plt.ylabel("Count")
df_metadata['sex'].value_counts().plot.bar(color = "orange")
```

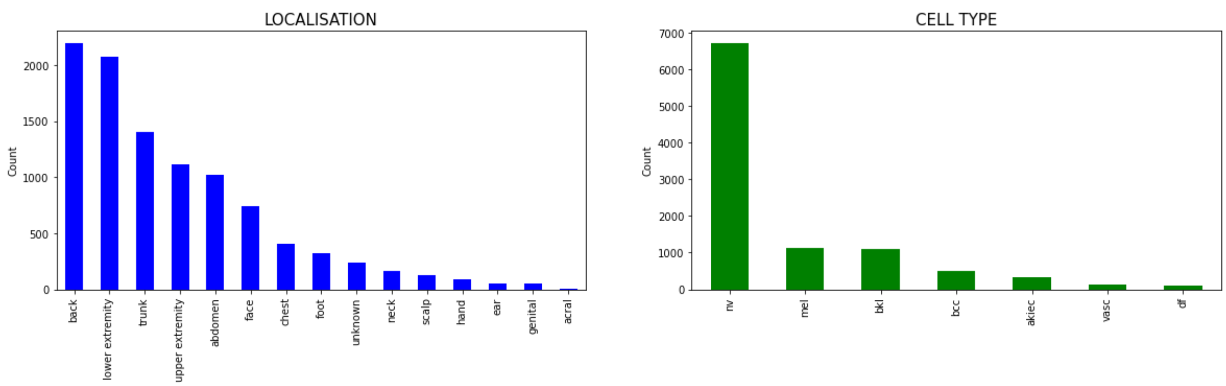
Out[6]:
```
<AxesSubplot:title={'center':'GENDER'}, ylabel='Count'>
```



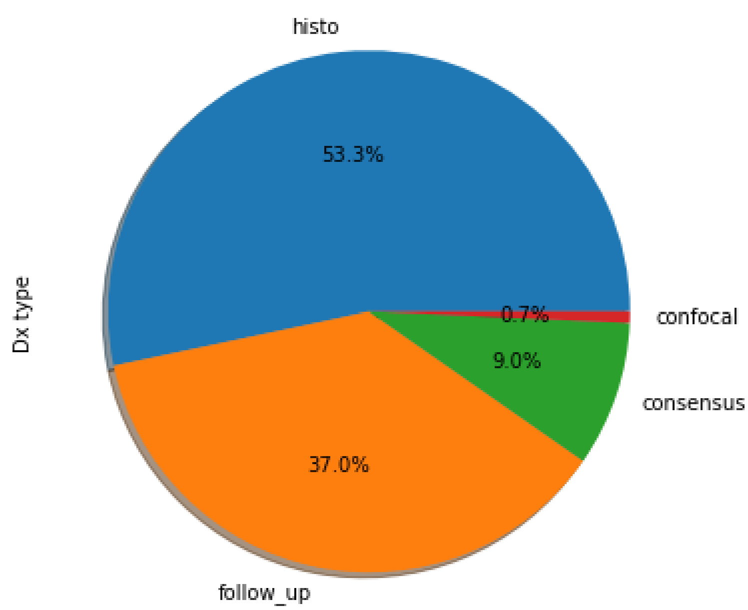


In [7]:
```
# Plot side by side plot of count vs localisation and cell type
plt.figure(figsize=(20,10))
plt.subplot(2,2,1)
plt.title("LOCALISATION",fontsize=15)
plt.ylabel("Count")
plt.xticks(rotation=45)
df_metadata['localization'].value_counts().plot.bar(color = "blue")

plt.subplot(2,2,2)
plt.title("CELL TYPE",fontsize=15)
plt.ylabel("Count")
df_metadata['dx'].value_counts().plot.bar(color = "green")
```

Out[7]: <AxesSubplot:title={'center':'CELL TYPE'}, ylabel='Count'>

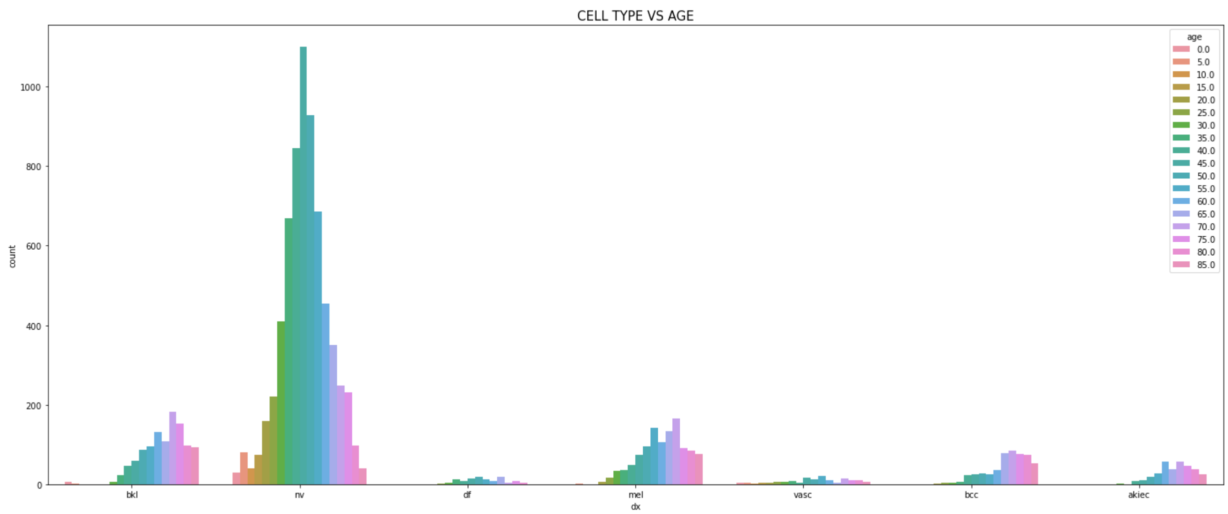


In [8]:
```
# Pie plot of dx type
plt.figure(figsize=(6,6))
df_metadata['dx_type'].value_counts().plot.pie(autopct="%1.1f%%", shadow=True)
plt.ylabel("Dx type")
plt.show()
```

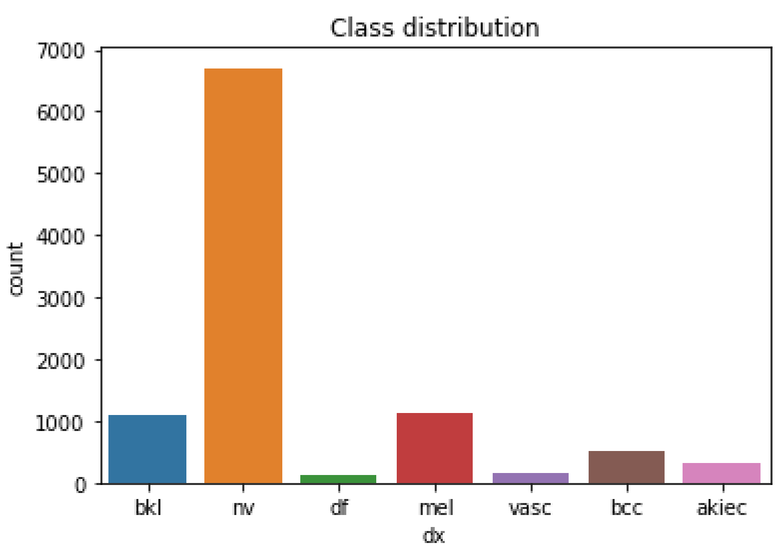


In [9]:
```
# Plot of count vs age and cell type
plt.figure(figsize=(25,10))
plt.title('CELL TYPE VS AGE',fontsize = 15)
sns.countplot(x='dx', hue='age',data=df_metadata)
```

Out[9]: <AxesSubplot:title={'center':'CELL TYPE VS AGE'}, xlabel='dx', ylabel='count'>

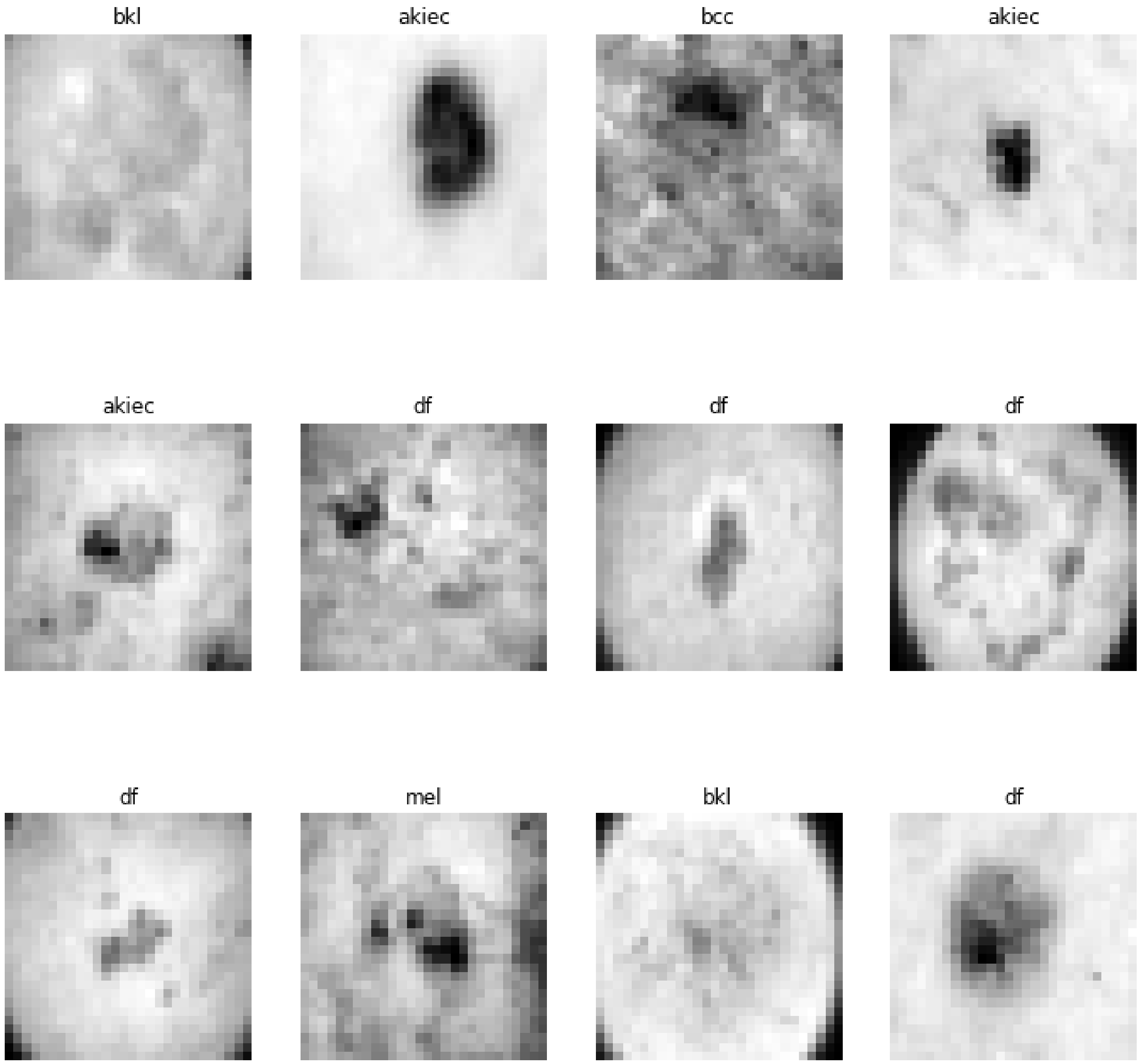


In [10]:
```
# Plot of class distribution
sns.countplot(x='dx', data = df_metadata)
plt.title('Class distribution')
plt.show()
```



In [11]:
```
import imblearn
from imblearn.over_sampling import RandomOverSampler

# Plot some examples of the images
# Set y as the label and x as the pixels
y = dataset_images['label']
x = dataset_images.drop(columns = ['label'])
#get x_train ,y_train
x.shape

# Oversample the data to make them balanced
oversample = RandomOverSampler()
x,y  = oversample.fit_resample(x,y)
x = np.array(x).reshape(-1,28,28,1)
print('Shape of Data :',x.shape)


# Map the labels to the name of the cancer
label_mapping = {
    0: 'nv',
    1: 'mel',
    2: 'bkl',
    3: 'bcc',
    4: 'akiec',
    5: 'vasc',
    6: 'df'
}

# Sample 12 pairs
sample_data = pd.Series(list(zip(x, y))).sample(12)

sample_X = np.stack(np.array(sample_data.apply(lambda x: x[0])))
sample_y = np.array(sample_data.apply(lambda x: x[1]))

# Plot the images
plt.figure(figsize=(12, 12))

for i in range(12):
    plt.subplot(3, 4, i + 1)
    plt.imshow((sample_X[i]), cmap = "gray")
    img_label = label_mapping[sample_y[i]]
    plt.title(img_label)
    plt.axis("off")
plt.show()
```

```
Shape of Data : (46935, 28, 28, 1)
```



In [12]:

```
# Importing the packages needed for CNN
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras.utils import np_utils
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE
import imblearn
```

In [16]:
```
# define the number of classes
num_classes = 7

# drop the column of 'label' in dataset to get the pure images dataset
x = dataset_images.drop(['label'], axis = 1)
y = dataset_images['label']

# Oversample to make the data balanced
oversample = SMOTE()
x, y = oversample.fit_resample(x, y)

# Split the data in train and test set
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_s

# change the data to np array
x_train = np.array(x_train)
x_test = np.array(x_test)
np.array(y_train)
np.array(y_test)

# Reshape the data to the appropriate shape
img_rows = 28
img_cols = 28
x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)


x_train = x_train.astype('float32')
x_test = x_test.astype('float32')

# Normalise the data
x_train /= 255
x_test /= 255
print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')

# convert class vectors to binary class matrices
y_train = keras.utils.np_utils.to_categorical(y_train, num_classes)
y_test = keras.utils.np_utils.to_categorical(y_test, num_classes)
```

```
x_train shape: (37548, 28, 28, 1)
37548 train samples
9387 test samples
```

In [17]:
```
t0= time.time()

# Fit the CNN model
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=(img_rows, img_cols, 1)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss=keras.losses.categorical_crossentropy, optimizer='adam', metri
model.summary()

# Set parameters for history
batch_size = 128
epochs = 20


history = model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs,
              verbose=1, shuffle=True,
              validation_split = 0.2)

t1 = time.time() - t0
print("Time elapsed: ", t1) # CPU seconds elapsed (floating point)
score = model.evaluate(x_test, y_test, verbose=0)
print('Summary: Loss over the test dataset: %.2f, Accuracy: %.2f' % (score[0], sc
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_2 (Conv2D)           (None, 26, 26, 32)        320

 max_pooling2d_2 (MaxPooling  (None, 13, 13, 32)       0
 2D)

 conv2d_3 (Conv2D)           (None, 11, 11, 64)        18496

 max_pooling2d_3 (MaxPooling  (None, 5, 5, 64)         0
 2D)

 dropout_2 (Dropout)         (None, 5, 5, 64)          0

 flatten_1 (Flatten)         (None, 1600)              0

 dense_2 (Dense)             (None, 128)               204928

 dropout_3 (Dropout)         (None, 128)               0
```

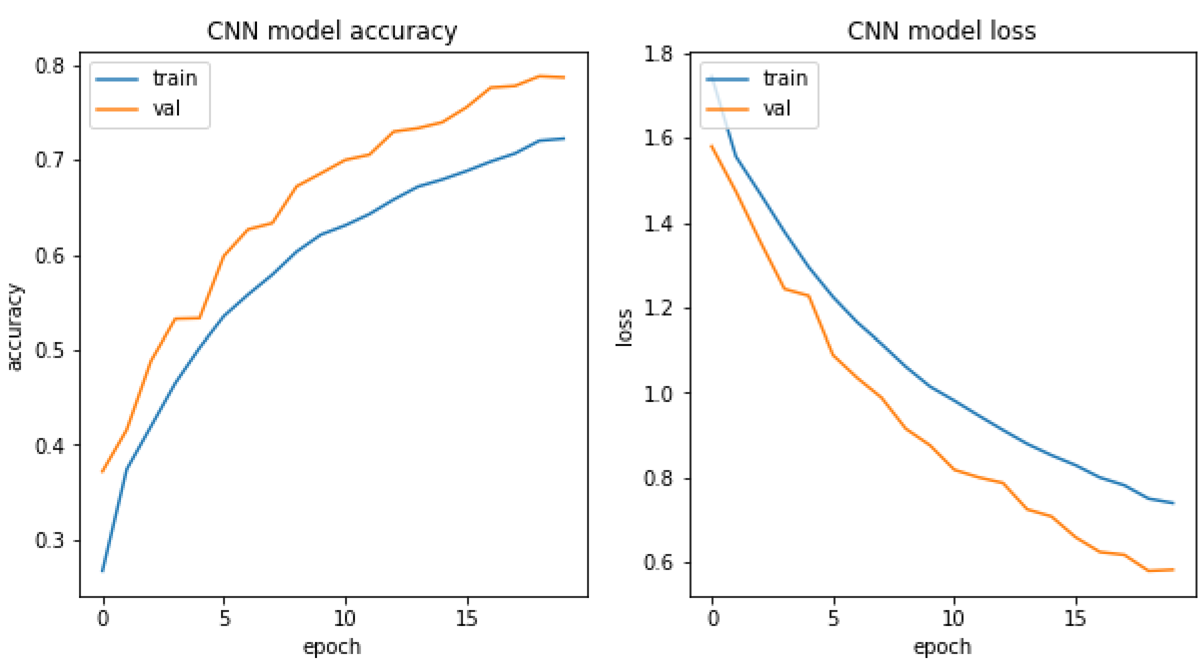
```
 dense_3 (Dense)             (None, 7)                 903

=================================================================
Total params: 224,647
Trainable params: 224,647
Non-trainable params: 0
_________________________________________________________________
Epoch 1/20
235/235 [==============================] - 9s 38ms/step - loss: 1.7452 - acc
uracy: 0.2669 - val_loss: 1.5800 - val_accuracy: 0.3719
Epoch 2/20
235/235 [==============================] - 9s 37ms/step - loss: 1.5557 - acc
uracy: 0.3741 - val_loss: 1.4733 - val_accuracy: 0.4158
Epoch 3/20
235/235 [==============================] - 9s 37ms/step - loss: 1.4683 - acc
uracy: 0.4194 - val_loss: 1.3557 - val_accuracy: 0.4884
Epoch 4/20
235/235 [==============================] - 9s 37ms/step - loss: 1.3784 - acc
uracy: 0.4649 - val_loss: 1.2438 - val_accuracy: 0.5325
Epoch 5/20
235/235 [==============================] - 9s 37ms/step - loss: 1.2952 - acc
uracy: 0.5022 - val_loss: 1.2276 - val_accuracy: 0.5332
Epoch 6/20
235/235 [==============================] - 9s 37ms/step - loss: 1.2250 - acc
uracy: 0.5353 - val_loss: 1.0887 - val_accuracy: 0.5987
Epoch 7/20
235/235 [==============================] - 9s 37ms/step - loss: 1.1650 - acc
uracy: 0.5581 - val_loss: 1.0351 - val_accuracy: 0.6268
Epoch 8/20
235/235 [==============================] - 9s 39ms/step - loss: 1.1149 - acc
uracy: 0.5790 - val_loss: 0.9884 - val_accuracy: 0.6334
Epoch 9/20
235/235 [==============================] - 9s 38ms/step - loss: 1.0614 - acc
uracy: 0.6033 - val_loss: 0.9153 - val_accuracy: 0.6722
Epoch 10/20
235/235 [==============================] - 9s 38ms/step - loss: 1.0146 - acc
uracy: 0.6212 - val_loss: 0.8760 - val_accuracy: 0.6859
Epoch 11/20
235/235 [==============================] - 9s 38ms/step - loss: 0.9815 - acc
uracy: 0.6310 - val_loss: 0.8185 - val_accuracy: 0.7000
Epoch 12/20
235/235 [==============================] - 9s 38ms/step - loss: 0.9464 - acc
uracy: 0.6430 - val_loss: 0.8007 - val_accuracy: 0.7056
Epoch 13/20
235/235 [==============================] - 9s 37ms/step - loss: 0.9123 - acc
uracy: 0.6583 - val_loss: 0.7875 - val_accuracy: 0.7300
Epoch 14/20
235/235 [==============================] - 9s 37ms/step - loss: 0.8797 - acc
uracy: 0.6719 - val_loss: 0.7252 - val_accuracy: 0.7336
Epoch 15/20
235/235 [==============================] - 9s 38ms/step - loss: 0.8528 - acc
uracy: 0.6794 - val_loss: 0.7086 - val_accuracy: 0.7398
Epoch 16/20
235/235 [==============================] - 9s 38ms/step - loss: 0.8292 - acc
uracy: 0.6884 - val_loss: 0.6593 - val_accuracy: 0.7557
Epoch 17/20
235/235 [==============================] - 9s 38ms/step - loss: 0.8002 - acc
```

```
uracy: 0.6984 - val_loss: 0.6246 - val_accuracy: 0.7764
Epoch 18/20
235/235 [==============================] - 9s 38ms/step - loss: 0.7821 - acc
uracy: 0.7069 - val_loss: 0.6180 - val_accuracy: 0.7782
Epoch 19/20
235/235 [==============================] - 9s 38ms/step - loss: 0.7505 - acc
uracy: 0.7204 - val_loss: 0.5802 - val_accuracy: 0.7884
Epoch 20/20
235/235 [==============================] - 9s 37ms/step - loss: 0.7402 - acc
uracy: 0.7225 - val_loss: 0.5829 - val_accuracy: 0.7872
Time elapsed:  177.51641273498535
Summary: Loss over the test dataset: 0.58, Accuracy: 0.79
```

In [18]:

```
# Plot the training and validation accuracy
plt.figure(figsize=(10,5))

plt.subplot(1,2,1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('CNN model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')

plt.subplot(1,2,2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('CNN model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
```

Out[18]: <matplotlib.legend.Legend at 0x165a8b26520>

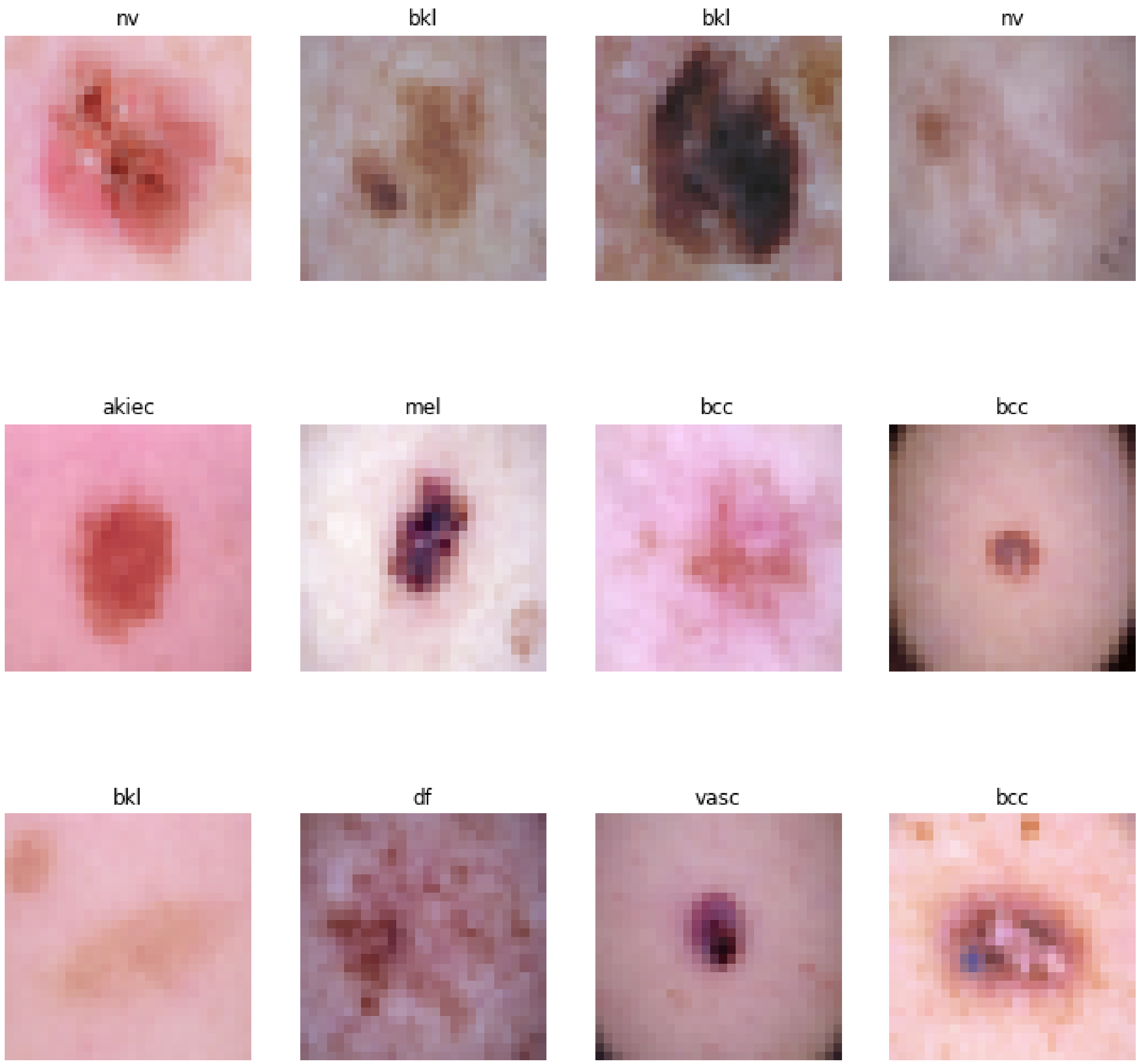


In [19]:
```
from imblearn.over_sampling import SMOTE
# Now we are going to work with the RGB images
# import the data of colored images
dataset_images = pd.read_csv("C:\\Users\\irrro\\OneDrive\\Documents\\Documents\\i
y = dataset_images['label']
x = dataset_images.drop(columns = ['label'])
#get x shape
print(x.shape)

# Over sample the data and reshape them into the right format
oversample = SMOTE()
x, y = oversample.fit_resample(x, y)
x = np.array(x).reshape(-1,28,28,3)
print('Shape of Data :',x.shape)
```

```
(10015, 2352)
Shape of Data : (46935, 28, 28, 3)
```

In [21]:

```
# First lets plot some examples
sample_data = pd.Series(list(zip(x, y))).sample(12)

sample_X = np.stack(np.array(sample_data.apply(lambda x: x[0])))
sample_y = np.array(sample_data.apply(lambda x: x[1]))

plt.figure(figsize=(12, 12))

for i in range(12):
    plt.subplot(3, 4, i + 1)
    plt.imshow((sample_X[i]))
    img_label = label_mapping[sample_y[i]]
    plt.title(img_label)
    plt.axis("off")
plt.show()
```



In [20]:
```
from tensorflow.keras.layers import Conv2D, Flatten, Dense, MaxPool2D

# Split into training and test set
X_train, X_test, Y_train, Y_test = train_test_split(x,y, test_size=0.2, random_st

# Reshape the data
X_train = X_train.reshape(X_train.shape[0], 28, 28, 3)
X_test = X_test.reshape(X_test.shape[0], 28, 28, 3)

X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
print('x_train shape:', X_train.shape)

# Normalise the data
X_train /= 255
X_test /= 255

# Fit the model
model = Sequential()
model.add(Conv2D(16, kernel_size = (3,3), input_shape = (28, 28, 3), activation =
model.add(Conv2D(32, kernel_size = (3,3), activation = 'relu'))
model.add(MaxPool2D(pool_size = (2,2)))
model.add(Conv2D(32, kernel_size = (3,3), activation = 'relu', padding = 'same'))
model.add(Conv2D(64, kernel_size = (3,3), activation = 'relu'))
model.add(MaxPool2D(pool_size = (2,2), padding = 'same'))
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(7, activation='softmax'))
model.summary()
```

```
x_train shape: (37548, 28, 28, 3)
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_4 (Conv2D)           (None, 28, 28, 16)        448

 conv2d_5 (Conv2D)           (None, 26, 26, 32)        4640

 max_pooling2d_4 (MaxPooling  (None, 13, 13, 32)       0
 2D)

 conv2d_6 (Conv2D)           (None, 13, 13, 32)        9248

 conv2d_7 (Conv2D)           (None, 11, 11, 64)        18496

 max_pooling2d_5 (MaxPooling  (None, 6, 6, 64)         0
 2D)

 flatten_2 (Flatten)         (None, 2304)              0

 dense_4 (Dense)             (None, 64)                147520

 dense_5 (Dense)             (None, 32)                2080
```

```
 dense_6 (Dense)             (None, 7)                 231

=================================================================
Total params: 182,663
Trainable params: 182,663
Non-trainable params: 0
_________________________________________________________________
```

In [21]:
```
# Compile the model
import time
t0= time.time()
import tensorflow as tf
model.compile(loss=keras.losses.categorical_crossentropy, optimizer='adam', metri
callback = tf.keras.callbacks.ModelCheckpoint(filepath='best_model.h5',
                                                  monitor='val_acc', mode='max',
                                                 verbose=1)
model.compile(loss = 'sparse_categorical_crossentropy',
             optimizer = 'adam',
              metrics = ['accuracy'])
history = model.fit(X_train,
                    Y_train,
                    validation_split=0.2,
                    batch_size = 128,
                    epochs = 20,
                    callbacks=[callback])
t1 = time.time() - t0
print("Time elapsed: ", t1) # CPU seconds elapsed (floating point)
score = model.evaluate(X_test, Y_test, verbose=0)
print('Summary: Loss over the test dataset: %.2f, Accuracy: %.2f' % (score[0], sc
```

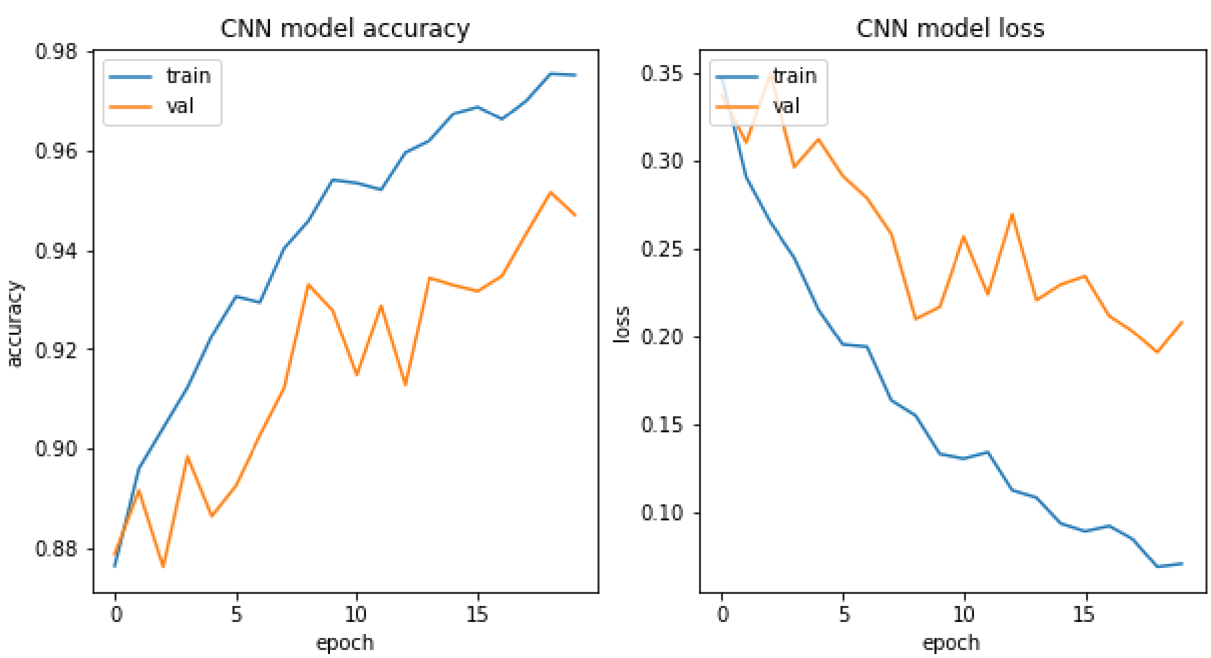
```
Epoch 1/20
235/235 [==============================] - ETA: 0s - loss: 1.4559 - accurac
y: 0.4162
Epoch 00001: saving model to best_model.h5
235/235 [==============================] - 20s 85ms/step - loss: 1.4559 - ac
curacy: 0.4162 - val_loss: 1.0790 - val_accuracy: 0.5920
Epoch 2/20
235/235 [==============================] - ETA: 0s - loss: 1.0171 - accurac
y: 0.6099
Epoch 00002: saving model to best_model.h5
235/235 [==============================] - 20s 84ms/step - loss: 1.0171 - ac
curacy: 0.6099 - val_loss: 0.9542 - val_accuracy: 0.6282
Epoch 3/20
235/235 [==============================] - ETA: 0s - loss: 0.8530 - accurac
y: 0.6826
Epoch 00003: saving model to best_model.h5
235/235 [==============================] - 19s 82ms/step - loss: 0.8530 - ac
curacy: 0.6826 - val_loss: 0.8875 - val_accuracy: 0.6565
Epoch 4/20
235/235 [==============================] - ETA: 0s - loss: 0.7319 - accurac
y: 0.7321
Epoch 00004: saving model to best_model.h5
235/235 [==============================] - 19s 81ms/step - loss: 0.7319 - ac
curacy: 0.7321 - val_loss: 0.6345 - val_accuracy: 0.7636
Epoch 5/20
235/235 [==============================] - ETA: 0s - loss: 0.6332 - accurac
y: 0.7700
Epoch 00005: saving model to best_model.h5
235/235 [==============================] - 19s 82ms/step - loss: 0.6332 - ac
curacy: 0.7700 - val_loss: 0.5843 - val_accuracy: 0.7844
Epoch 6/20
235/235 [==============================] - ETA: 0s - loss: 0.5704 - accurac
y: 0.7938
```

```
Epoch 00006: saving model to best_model.h5
235/235 [==============================] - 19s 83ms/step - loss: 0.5704 - ac
curacy: 0.7938 - val_loss: 0.5330 - val_accuracy: 0.8060
Epoch 7/20
235/235 [==============================] - ETA: 0s - loss: 0.4968 - accurac
y: 0.8200
Epoch 00007: saving model to best_model.h5
235/235 [==============================] - 19s 82ms/step - loss: 0.4968 - ac
curacy: 0.8200 - val_loss: 0.5147 - val_accuracy: 0.8067
Epoch 8/20
235/235 [==============================] - ETA: 0s - loss: 0.4374 - accurac
y: 0.8433
Epoch 00008: saving model to best_model.h5
235/235 [==============================] - 19s 81ms/step - loss: 0.4374 - ac
curacy: 0.8433 - val_loss: 0.4269 - val_accuracy: 0.8403
Epoch 9/20
234/235 [============================>.] - ETA: 0s - loss: 0.3955 - accurac
y: 0.8577
Epoch 00009: saving model to best_model.h5
235/235 [==============================] - 19s 81ms/step - loss: 0.3955 - ac
curacy: 0.8577 - val_loss: 0.3866 - val_accuracy: 0.8654
Epoch 10/20
234/235 [============================>.] - ETA: 0s - loss: 0.3574 - accurac
y: 0.8703
Epoch 00010: saving model to best_model.h5
235/235 [==============================] - 19s 82ms/step - loss: 0.3571 - ac
curacy: 0.8704 - val_loss: 0.3886 - val_accuracy: 0.8570
Epoch 11/20
235/235 [==============================] - ETA: 0s - loss: 0.3304 - accurac
y: 0.8799
Epoch 00011: saving model to best_model.h5
235/235 [==============================] - 20s 84ms/step - loss: 0.3304 - ac
curacy: 0.8799 - val_loss: 0.3235 - val_accuracy: 0.8819
Epoch 12/20
235/235 [==============================] - ETA: 0s - loss: 0.2987 - accurac
y: 0.8928
Epoch 00012: saving model to best_model.h5
235/235 [==============================] - 19s 82ms/step - loss: 0.2987 - ac
curacy: 0.8928 - val_loss: 0.3017 - val_accuracy: 0.8913
Epoch 13/20
235/235 [==============================] - ETA: 0s - loss: 0.2576 - accurac
y: 0.9082
Epoch 00013: saving model to best_model.h5
235/235 [==============================] - 19s 82ms/step - loss: 0.2576 - ac
curacy: 0.9082 - val_loss: 0.2933 - val_accuracy: 0.8976
Epoch 14/20
234/235 [============================>.] - ETA: 0s - loss: 0.2414 - accurac
y: 0.9133
Epoch 00014: saving model to best_model.h5
235/235 [==============================] - 19s 82ms/step - loss: 0.2413 - ac
curacy: 0.9134 - val_loss: 0.3110 - val_accuracy: 0.8840
Epoch 15/20
235/235 [==============================] - ETA: 0s - loss: 0.2280 - accurac
y: 0.9174
Epoch 00015: saving model to best_model.h5
235/235 [==============================] - 19s 81ms/step - loss: 0.2280 - ac
curacy: 0.9174 - val_loss: 0.2758 - val_accuracy: 0.9000
```

```
Epoch 16/20
234/235 [============================>.] - ETA: 0s - loss: 0.2008 - accurac
y: 0.9285
Epoch 00016: saving model to best_model.h5
235/235 [==============================] - 19s 82ms/step - loss: 0.2007 - ac
curacy: 0.9285 - val_loss: 0.2580 - val_accuracy: 0.9057
Epoch 17/20
234/235 [============================>.] - ETA: 0s - loss: 0.1899 - accurac
y: 0.9319
Epoch 00017: saving model to best_model.h5
235/235 [==============================] - 19s 82ms/step - loss: 0.1898 - ac
curacy: 0.9319 - val_loss: 0.2674 - val_accuracy: 0.9088
Epoch 18/20
235/235 [==============================] - ETA: 0s - loss: 0.1775 - accurac
y: 0.9375
Epoch 00018: saving model to best_model.h5
235/235 [==============================] - 19s 82ms/step - loss: 0.1775 - ac
curacy: 0.9375 - val_loss: 0.2274 - val_accuracy: 0.9213
Epoch 19/20
235/235 [==============================] - ETA: 0s - loss: 0.1613 - accurac
y: 0.9416
Epoch 00019: saving model to best_model.h5
235/235 [==============================] - 19s 82ms/step - loss: 0.1613 - ac
curacy: 0.9416 - val_loss: 0.2474 - val_accuracy: 0.9178
Epoch 20/20
235/235 [==============================] - ETA: 0s - loss: 0.1448 - accurac
y: 0.9476
Epoch 00020: saving model to best_model.h5
235/235 [==============================] - 19s 82ms/step - loss: 0.1448 - ac
curacy: 0.9476 - val_loss: 0.3131 - val_accuracy: 0.8921
Time elapsed:  387.1178209781647
Summary: Loss over the test dataset: 0.33, Accuracy: 0.89
```

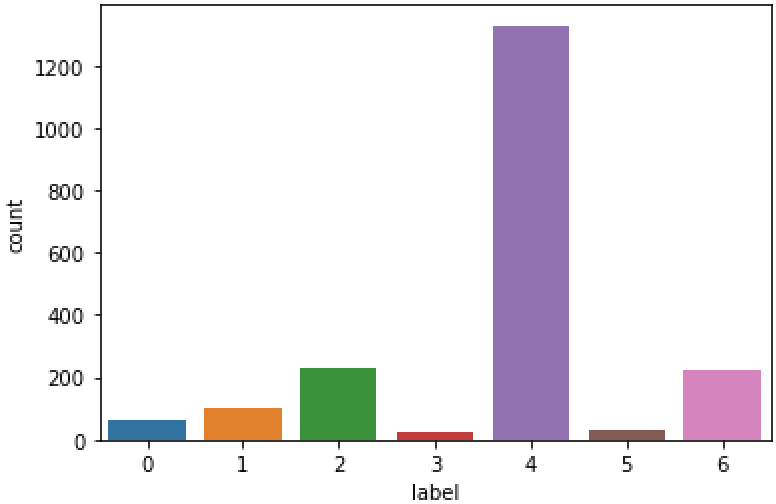
In [55]:
```
# Plot the training and validation accuracy
plt.figure(figsize=(10,5))

plt.subplot(1,2,1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('CNN model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')

plt.subplot(1,2,2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('CNN model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
```

Out[55]: <matplotlib.legend.Legend at 0x2e5b6752250>



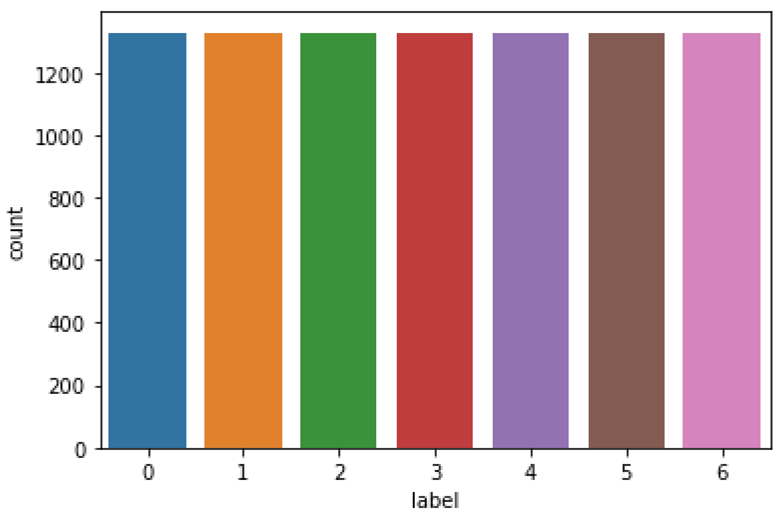
In [22]:
```
# SVM for grayscale images
# Import the right packages
from sklearn.svm import SVC
import imblearn
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import GridSearchCV

# read the data
dataset_images_full = pd.read_csv("C:\\Users\\irrro\\OneDrive\\Documents\\Documer
# Sample 2000 data
dataset_images = dataset_images_full.sample(n = 2000)

print(dataset_images.shape)

# Set x and y
y = dataset_images['label']
x = dataset_images.drop(columns = ['label'])

# Verify that there are data from every class
sns.countplot(y)
plt.show()


print(x.shape)

# Oversample the data
oversample = SMOTE()
x, y = oversample.fit_resample(x, y)

# Split into tarin and test set
X_train, X_test, Y_train, Y_test = train_test_split(x,y, test_size=0.2, random_st
print('x_train shape:', X_train.shape)

sns.countplot(y)
plt.show()
```

```
(2000, 785)

C:\Users\irrro\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWar
ning: Pass the following variable as a keyword arg: x. From version 0.12, the o
nly valid positional argument will be `data`, and passing other arguments witho
ut an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```




```
(2000, 784)
x_train shape: (7436, 784)
```

```
C:\Users\irrro\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWar
ning: Pass the following variable as a keyword arg: x. From version 0.12, the o
nly valid positional argument will be `data`, and passing other arguments witho
ut an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



In [23]:
```
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
print('x_train shape:', X_train.shape)

X_train /= 255
X_test /= 255


# Set the parameters by cross-validation and perform cross validation
parameters = [{'kernel': ['rbf'],
               'gamma': [0.01, 0.1, 0.5],
               'C': [10, 100, 1000]}]
print("# Tuning hyper-parameters")
t0= time.time()
clf = GridSearchCV(SVC(), parameters, cv=5)
clf.fit(X_train, Y_train)

score = clf.score(X_test,Y_test)
print("The score is", score)
print("best parameters from train data: ", clf.best_params_)

print('--------------------------------------')
means = clf.cv_results_['mean_test_score']
stds = clf.cv_results_['std_test_score']
for mean, std, params in zip(means, stds, clf.cv_results_['params']):
    print("%0.3f (+/-%0.03f) for %r"
          % (mean, std * 2, params))
t1 = time.time() - t0
print("Time elapsed: ", t1) # CPU seconds elapsed (floating point)
```

```
x_train shape: (7436, 784)
# Tuning hyper-parameters
The score is 0.9881720430107527
best parameters from train data:  {'C': 10, 'gamma': 0.5, 'kernel': 'rbf'}
--------------------------------------
0.853 (+/-0.017) for {'C': 10, 'gamma': 0.01, 'kernel': 'rbf'}
0.970 (+/-0.006) for {'C': 10, 'gamma': 0.1, 'kernel': 'rbf'}
0.979 (+/-0.006) for {'C': 10, 'gamma': 0.5, 'kernel': 'rbf'}
0.946 (+/-0.006) for {'C': 100, 'gamma': 0.01, 'kernel': 'rbf'}
0.973 (+/-0.005) for {'C': 100, 'gamma': 0.1, 'kernel': 'rbf'}
0.979 (+/-0.006) for {'C': 100, 'gamma': 0.5, 'kernel': 'rbf'}
0.963 (+/-0.007) for {'C': 1000, 'gamma': 0.01, 'kernel': 'rbf'}
0.973 (+/-0.005) for {'C': 1000, 'gamma': 0.1, 'kernel': 'rbf'}
0.979 (+/-0.006) for {'C': 1000, 'gamma': 0.5, 'kernel': 'rbf'}
Time elapsed:  633.8257734775543
```

In [24]:
```
# Fit the model with the optimal parameters
optimal_C = 10
optimal_gamma = 0.5
t0= time.time()

clf_final = SVC(kernel="rbf", gamma=optimal_gamma, C=optimal_C)
clf_final.fit(X_train, Y_train)
score = clf.score(X_test,Y_test)
print("The score is", score)

t1 = time.time() - t0
print("Time elapsed: ", t1) # CPU seconds elapsed (floating point)
```

```
The score is 0.9881720430107527
Time elapsed:  24.44465732574463
```

In [25]:
```
# SVM and LDA for colored images
# Import the right packages
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
import matplotlib.pyplot as plt
from sklearn.svm import SVC
import imblearn
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import GridSearchCV

# Read in the data
dataset_images = pd.read_csv("C:\\Users\\irrro\\OneDrive\\Documents\\Documents\\i

# Set x and y
y = dataset_images['label']
x = dataset_images.drop(columns = ['label'])
print(x.shape)

# Over sample the data
oversample = SMOTE()
x, y = oversample.fit_resample(x, y)
print('Shape of Data :',x.shape)

# split into train and test set and normalise
X_train, X_test, Y_train, Y_test = train_test_split(x,y, test_size=0.2, random_st


X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
print('x_train shape:', X_train.shape)

X_train /= 255
X_test /= 255


t0 = time.time()

# Perform LDA
lda = LDA()
X_train = lda.fit_transform(X_train, Y_train)
print('x_train shape:', X_train.shape)
print(lda.score(X_test,Y_test))

X_test = lda.transform(X_test)

# Set the parameters by cross-validation and perform cross validation
parameters = [{'kernel': ['rbf'],
               'gamma': [0.01, 0.1, 0.5],
               'C': [10, 100, 1000]}]
print("# Tuning hyper-parameters")
clf = GridSearchCV(SVC(), parameters, cv=5)
# clf = SVC(C=100, gamma=0.1, kernel="rbf")
clf.fit(X_train, Y_train)

print("best parameters from train data: ", clf.best_params_)
```

```
print('-------------------------------------')
means = clf.cv_results_['mean_test_score']
stds = clf.cv_results_['std_test_score']
for mean, std, params in zip(means, stds, clf.cv_results_['params']):
    print("%0.3f (+/-%0.03f) for %r"
          % (mean, std * 2, params))

t1 = time.time() - t0
print("Time elapsed: ", t1)
```

```
(10015, 2352)
Shape of Data : (46935, 2352)
x_train shape: (37548, 2352)
x_train shape: (37548, 6)
0.9079578139980825
# Tuning hyper-parameters
best parameters from train data:  {'C': 10, 'gamma': 0.1, 'kernel': 'rbf'}
-------------------------------------
0.949 (+/-0.003) for {'C': 10, 'gamma': 0.01, 'kernel': 'rbf'}
0.950 (+/-0.003) for {'C': 10, 'gamma': 0.1, 'kernel': 'rbf'}
0.945 (+/-0.003) for {'C': 10, 'gamma': 0.5, 'kernel': 'rbf'}
0.949 (+/-0.002) for {'C': 100, 'gamma': 0.01, 'kernel': 'rbf'}
0.947 (+/-0.004) for {'C': 100, 'gamma': 0.1, 'kernel': 'rbf'}
0.936 (+/-0.004) for {'C': 100, 'gamma': 0.5, 'kernel': 'rbf'}
0.948 (+/-0.003) for {'C': 1000, 'gamma': 0.01, 'kernel': 'rbf'}
0.940 (+/-0.004) for {'C': 1000, 'gamma': 0.1, 'kernel': 'rbf'}
0.934 (+/-0.005) for {'C': 1000, 'gamma': 0.5, 'kernel': 'rbf'}
Time elapsed:  641.5680644512177
```

In [26]:

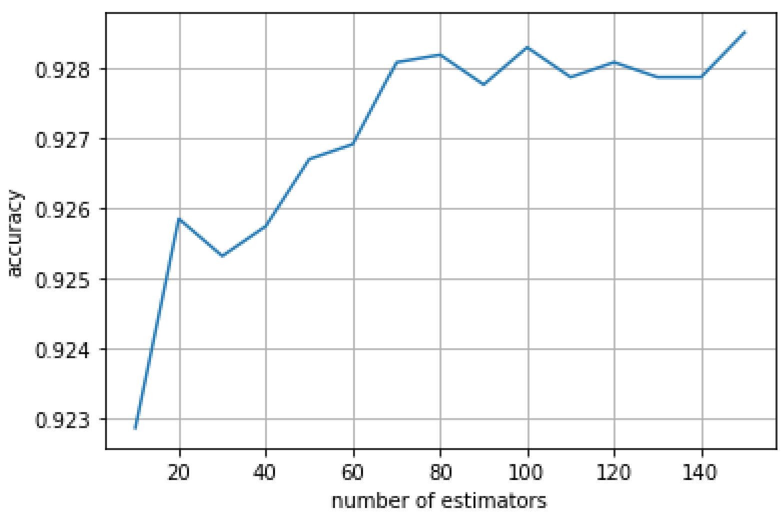
```
# Fit the model with the optimal parameters
optimal_C = 10
optimal_gamma = 0.5

clf_final = SVC(kernel="rbf", gamma=optimal_gamma, C=optimal_C)
clf_final.fit(X_train, Y_train)
score = clf.score(X_test,Y_test)
print("The score is", score)
```

```
The score is 0.9301161180355811
```

In [27]:
```
# Import the packages
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
from sklearn.model_selection import train_test_split, GridSearchCV
import matplotlib.pyplot as plt
from imblearn.over_sampling import SMOTE
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.ensemble import RandomForestClassifier

# Read in the data
dataset_images = pd.read_csv("C:\\Users\\irrro\\OneDrive\\Documents\\Documents\\i

#Set x and y
y = dataset_images['label']
x = dataset_images.drop(columns = ['label'])
print(x.shape)

# Oversample the data
oversample = SMOTE()
x, y = oversample.fit_resample(x, y)
print('Shape of Data :',x.shape)

# Split into training and test set and normalise
X_train, X_test, Y_train, Y_test = train_test_split(x,y, test_size=0.2, random_st


X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
print('x_train shape:', X_train.shape)

X_train /= 255
X_test /= 255

# Perform LDA
t0 = time.time()
lda = LDA()
X_train = lda.fit_transform(X_train, Y_train)
print(lda.score(X_test,Y_test))

X_test = lda.transform(X_test)

# Now lets implement the random forest algorithm
# Create a list of possible number of estimators
n_estimators = [10,20,30,40,50,60,70,80,90,100,110,120,130,140,150]

# Lets see how accuracy improves as number of estimators gets larger
accuracy_rf = []
for est in n_estimators:

    clf= RandomForestClassifier(n_estimators=est, random_state=54)
    clf.fit(X_train, Y_train)
    score = clf.score(X_test, Y_test)
    accuracy_rf.append(score)
print(accuracy_rf)

# Create a plot of accuracy vs number of estimators
```

```
plt.plot(n_estimators, accuracy_rf) #adds the line
plt.grid() #adds a grid to the plot
plt.ylabel('accuracy') #xlabel
plt.xlabel('number of estimators') #ylabel
plt.show()

# Fit the model with 20 number of estimators
clf_rf=RandomForestClassifier(n_estimators=20)
clf_rf.fit(X_train,Y_train)
y_pred_rf = clf_rf.predict(X_test)
print('Random forest model accuracy: {0:0.4f}'. format(accuracy_score(Y_test, y_p
cm_dt = confusion_matrix(Y_test, y_pred_rf)
print('Confusion matrix\n\n', cm_dt)

t1 = time.time() - t0
print("Time elapsed: ", t1)
```

```
(10015, 2352)
Shape of Data : (46935, 2352)
x_train shape: (37548, 2352)
0.912751677852349
[0.922872057100245, 0.9258549057206775, 0.9253222541813145, 0.925748375412805,
0.9267071481836583, 0.9269202087994034, 0.9280920421860019, 0.9281985724938745,
0.9277724512623842, 0.928305102801747, 0.9278789815702567, 0.9280920421860019,
0.9278789815702567, 0.9278789815702567, 0.9285181634174923]
```




```
Random forest model accuracy: 0.9252
Confusion matrix

 [[1286    3    0    0    6    0    0]
 [   5 1286   13    0   19    0    0]
 [   2    9 1185    0  100    0   55]
 [   0    0    0 1391    1    0    0]
 [  13   32  107    4 1049    2  139]
 [   0    0    0    0    0 1292    0]
 [   4    4   46    0  138    0 1196]]
Time elapsed:  118.74517607688904
```

In [ ]:

```

```