In [1]:

```
from numpy.random import seed
seed(1)
import tensorflow
tensorflow.random.set_seed(2)
```

In [2]:

```
#Import Stuff here

from sklearn.model_selection import train_test_split

import matplotlib.pyplot as plt

import tensorflow.keras as keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Conv2D, MaxPooling2D, Flatten, GlobalAveragePooling2D, Depthw
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.metrics import classification_report, confusion_matrix
import pandas as pd
import numpy as np
from keras.models import load_model
```

In [3]:

```
#creates a Image Generator from the directories and scales the images by dividing by 255. Also changes the images

def load_data(train_folder = 'data/train/'):

    datagen = ImageDataGenerator(rescale = 1/255)

    train_it = datagen.flow_from_directory (train_folder, target_size = (256, 256),
                                            class_mode = 'categorical', color_mode="rgb", batch_size=64, seed = 1

    val_it = datagen.flow_from_directory ('data/validation/', target_size = (256, 256),
                                            class_mode = 'categorical', color_mode="rgb", batch_size=64, seed = 1

    test_it = datagen.flow_from_directory ('data/test/', target_size = (256, 256),
                                            class_mode = 'categorical', color_mode="rgb", batch_size=64, seed = 1

    return train_it, val_it, test_it

train_it, val_it, test_it = load_data()
```

```
Found 6800 images belonging to 17 classes.
Found 1700 images belonging to 17 classes.
Found 1700 images belonging to 17 classes.
```

In [4]:

```
batchX, batchy = train_it.next()
print('Batch shape=%s, min=%.3f, max=%.3f' % (batchX.shape, batchX.min(), batchX.max()))
```

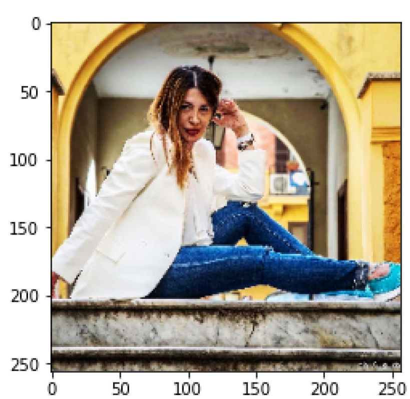
```
Batch shape=(64, 256, 256, 3), min=0.000, max=1.000
```

In [5]:

```
plt.imshow(batchX[0])
```

Out[5]:

```
<matplotlib.image.AxesImage at 0x1f04e9a1d00>
```



In [6]:
```
batchy[0]
```

Out[6]:
```
array([0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
      dtype=float32)
```

In [7]:
```
train_it.class_indices
```

Out[7]:
```
{'1977': 0,
 'Amaro': 1,
 'Brannan': 2,
 'Clarendon': 3,
 'Gingham': 4,
 'He-Fe': 5,
 'Hudson': 6,
 'Lo-Fi': 7,
 'Mayfair': 8,
 'Nashville': 9,
 'Original': 10,
 'Perpetua': 11,
 'Sutro': 12,
 'Toaster': 13,
 'Valencia': 14,
 'Willow': 15,
 'X-ProII': 16}
```

In [8]:
```
def build_model():
    # TODO: build the model,


    model = Sequential()
    model.add(BatchNormalization()) #added


    model.add(Conv2D(64, kernel_size= 3, activation = 'relu', padding='same'))
    model.add(MaxPooling2D (pool_size= 2))


    model.add(Conv2D(128, kernel_size= 3, activation = 'relu', padding='same'))
    model.add(MaxPooling2D (pool_size= 2))


    model.add(Conv2D(256, kernel_size= 3, activation = 'relu',  padding='same'))
    model.add(MaxPooling2D (pool_size= 2))


    model.add(Conv2D(512, kernel_size= 3, activation = 'relu',  padding='same'))
    model.add(MaxPooling2D (pool_size= 2))

    model.add(Conv2D(256, kernel_size= 3, activation = 'relu',  padding='same'))
    model.add(MaxPooling2D (pool_size= 2))

    model.add(Conv2D(128, kernel_size= 3, activation = 'relu',  padding='same'))
    model.add(Flatten ())


    model.add(Dense(128, activation='relu'))
    model.add(Dropout(0.2))

    model.add(Dense(64, activation='relu'))
    model.add(Dropout(0.2))

    model.add(Dense(17, activation='softmax'))

    return model

model = build_model()
```

In [9]:
```
def compile_model(model):

    model.compile(optimizer = keras.optimizers.Adam(learning_rate = 0.001),
              loss=keras.losses.CategoricalCrossentropy(),
              metrics=['accuracy'])
```

```python
    return model

def train_model(model, train_it, val_it):

    callback = keras.callbacks.EarlyStopping(monitor='val_accuracy', patience=5, restore_best_weights= True)

    history = model.fit(train_it, epochs = 50, steps_per_epoch = 107, verbose = 2,
                        validation_data= val_it, validation_steps = 27, callbacks=[callback])
    return model, history


model = compile_model(model)

model, history = train_model(model, train_it, val_it)

print (model.summary())
```

```
Epoch 1/50
107/107 - 123s - loss: 2.7395 - accuracy: 0.1003 - val_loss: 2.8032 - val_accuracy: 0.0882 - 123s/epoch - 1s/step
Epoch 2/50
107/107 - 122s - loss: 2.4679 - accuracy: 0.2046 - val_loss: 2.7037 - val_accuracy: 0.1424 - 122s/epoch - 1s/step
Epoch 3/50
107/107 - 123s - loss: 1.9587 - accuracy: 0.3634 - val_loss: 1.8111 - val_accuracy: 0.4271 - 123s/epoch - 1s/step
Epoch 4/50
107/107 - 125s - loss: 1.6036 - accuracy: 0.4769 - val_loss: 1.2894 - val_accuracy: 0.5753 - 125s/epoch - 1s/step
Epoch 5/50
107/107 - 126s - loss: 1.2015 - accuracy: 0.5896 - val_loss: 1.2089 - val_accuracy: 0.5871 - 126s/epoch - 1s/step
Epoch 6/50
107/107 - 119s - loss: 1.0837 - accuracy: 0.6241 - val_loss: 1.0240 - val_accuracy: 0.6694 - 119s/epoch - 1s/step
Epoch 7/50
107/107 - 120s - loss: 0.8838 - accuracy: 0.6899 - val_loss: 1.1885 - val_accuracy: 0.6041 - 120s/epoch - 1s/step
Epoch 8/50
107/107 - 125s - loss: 0.9515 - accuracy: 0.6740 - val_loss: 0.9193 - val_accuracy: 0.6929 - 125s/epoch - 1s/step
Epoch 9/50
107/107 - 123s - loss: 0.7793 - accuracy: 0.7328 - val_loss: 0.8196 - val_accuracy: 0.7412 - 123s/epoch - 1s/step
Epoch 10/50
107/107 - 125s - loss: 0.7432 - accuracy: 0.7449 - val_loss: 0.8336 - val_accuracy: 0.7312 - 125s/epoch - 1s/step
Epoch 11/50
107/107 - 127s - loss: 0.6002 - accuracy: 0.7951 - val_loss: 0.8017 - val_accuracy: 0.7547 - 127s/epoch - 1s/step
Epoch 12/50
107/107 - 124s - loss: 0.5218 - accuracy: 0.8162 - val_loss: 0.7692 - val_accuracy: 0.7671 - 124s/epoch - 1s/step
Epoch 13/50
107/107 - 124s - loss: 0.5847 - accuracy: 0.8068 - val_loss: 0.9529 - val_accuracy: 0.7235 - 124s/epoch - 1s/step
Epoch 14/50
107/107 - 130s - loss: 0.4438 - accuracy: 0.8457 - val_loss: 0.7793 - val_accuracy: 0.7788 - 130s/epoch - 1s/step
Epoch 15/50
107/107 - 120s - loss: 0.3885 - accuracy: 0.8599 - val_loss: 0.9106 - val_accuracy: 0.7482 - 120s/epoch - 1s/step
Epoch 16/50
107/107 - 121s - loss: 0.3561 - accuracy: 0.8719 - val_loss: 0.8379 - val_accuracy: 0.7835 - 121s/epoch - 1s/step
Epoch 17/50
107/107 - 124s - loss: 0.4789 - accuracy: 0.8474 - val_loss: 0.9283 - val_accuracy: 0.7612 - 124s/epoch - 1s/step
Epoch 18/50
107/107 - 127s - loss: 0.3187 - accuracy: 0.8940 - val_loss: 0.9713 - val_accuracy: 0.7429 - 127s/epoch - 1s/step
Epoch 19/50
107/107 - 127s - loss: 0.2462 - accuracy: 0.9150 - val_loss: 0.9253 - val_accuracy: 0.7865 - 127s/epoch - 1s/step
Epoch 20/50
107/107 - 125s - loss: 0.2998 - accuracy: 0.9034 - val_loss: 1.0524 - val_accuracy: 0.7724 - 125s/epoch - 1s/step
Epoch 21/50
107/107 - 124s - loss: 0.2843 - accuracy: 0.9031 - val_loss: 1.1794 - val_accuracy: 0.7188 - 124s/epoch - 1s/step
Epoch 22/50
107/107 - 124s - loss: 0.2978 - accuracy: 0.9041 - val_loss: 1.0150 - val_accuracy: 0.7524 - 124s/epoch - 1s/step
Epoch 23/50
107/107 - 125s - loss: 0.2003 - accuracy: 0.9347 - val_loss: 1.1027 - val_accuracy: 0.7888 - 125s/epoch - 1s/step
Epoch 24/50
107/107 - 127s - loss: 0.2536 - accuracy: 0.9184 - val_loss: 1.0813 - val_accuracy: 0.7824 - 127s/epoch - 1s/step
Epoch 25/50
107/107 - 126s - loss: 0.1612 - accuracy: 0.9449 - val_loss: 1.1489 - val_accuracy: 0.7824 - 126s/epoch - 1s/step
Epoch 26/50
107/107 - 127s - loss: 0.1710 - accuracy: 0.9409 - val_loss: 1.5154 - val_accuracy: 0.7188 - 127s/epoch - 1s/step
Epoch 27/50
107/107 - 122s - loss: 0.3498 - accuracy: 0.8962 - val_loss: 1.1630 - val_accuracy: 0.7724 - 122s/epoch - 1s/step
Epoch 28/50
107/107 - 119s - loss: 0.1974 - accuracy: 0.9349 - val_loss: 1.2314 - val_accuracy: 0.7647 - 119s/epoch - 1s/step
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 batch_normalization (BatchN  (None, None, None, 3)    12
 ormalization)

 conv2d (Conv2D)             (None, None, None, 64)    1792

 max_pooling2d (MaxPooling2D  (None, None, None, 64)   0
 )

 conv2d_1 (Conv2D)           (None, None, None, 128)   73856

 max_pooling2d_1 (MaxPooling  (None, None, None, 128)  0
 2D)

 conv2d_2 (Conv2D)           (None, None, None, 256)   295168

 max_pooling2d_2 (MaxPooling  (None, None, None, 256)  0
 2D)

 conv2d_3 (Conv2D)           (None, None, None, 512)   1180160

 max_pooling2d_3 (MaxPooling  (None, None, None, 512)  0
 2D)

 conv2d_4 (Conv2D)           (None, None, None, 256)   1179904

 max_pooling2d_4 (MaxPooling  (None, None, None, 256)  0
 2D)

 conv2d_5 (Conv2D)           (None, None, None, 128)   295040

 flatten (Flatten)           (None, None)              0

 dense (Dense)               (None, 128)               1048704

 dropout (Dropout)           (None, 128)               0

 dense_1 (Dense)             (None, 64)                8256

 dropout_1 (Dropout)         (None, 64)                0

 dense_2 (Dense)             (None, 17)                1105

=================================================================
Total params: 4,083,997
Trainable params: 4,083,991
Non-trainable params: 6
_________________________________________________________________
None
```

In [10]:

```
def eval_model(model, test_it):
    # TODO: evaluate the model

    test_loss, test_accuracy = model.evaluate (test_it, steps = 27)

    return test_loss,  test_accuracy

test_loss, test_accuracy = eval_model(model, test_it)
```

```
27/27 [==============================] - 25s 905ms/step - loss: 0.8740 - accuracy: 0.7971
```

In [11]:

```
test_it.reset()
preds = model.predict(test_it, steps = 27)
```

In [12]:

```
y_predict = np.argmax(preds,axis=1)
```

In [13]:

```
print(classification_report(test_it.classes, y_predict, target_names=test_it.class_indices))
```

```
              precision    recall  f1-score   support

        1977       0.82      0.97      0.89       100
       Amaro       0.74      0.75      0.74       100
     Brannan       0.90      0.80      0.85       100
   Clarendon       0.55      0.66      0.60       100
     Gingham       0.84      0.74      0.79       100
       He-Fe       0.83      0.95      0.88       100
      Hudson       0.79      0.90      0.84       100
       Lo-Fi       0.54      0.57      0.55       100
     Mayfair       0.79      0.58      0.67       100
   Nashville       0.96      0.99      0.98       100
    Original       0.51      0.29      0.37       100
    Perpetua       0.77      0.86      0.82       100
       Sutro       0.92      0.93      0.93       100
     Toaster       0.97      0.97      0.97       100
    Valencia       0.69      0.74      0.71       100
      Willow       1.00      1.00      1.00       100
     X-ProII       0.89      0.85      0.87       100

    accuracy                           0.80      1700
   macro avg       0.79      0.80      0.79      1700
weighted avg       0.79      0.80      0.79      1700
```

In [14]:

```
error_count = 0

for x in range (0,len(y_predict)):
    if y_predict[x] != test_it.classes[x]:
        error_count += 1

error_count
```

Out[14]:

```
345
```

In [15]:

```
model.save('CNN_Original_79_final.h5', overwrite=True,
    include_optimizer=True)
```

In [ ]: