Objective

Main objective of the current project is to build a image classifier using Convoluted neural networks & through traditional fully connected DNNs. To understand what makes CNNs special. This model can be able to separate out multiple classes of images and may be useful in image gallery sorting or searching of images etc.

Importing libraries

In [58]:
```python
from tensorflow import keras
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt
import os
from tensorflow.keras import layers
from tensorflow.keras import Sequential
import cv2
import numpy as np
import warnings
from sklearn.metrics import classification_report
import gc
from numba import cuda

warnings.filterwarnings("ignore")

%matplotlib inline
```

Loading data

In [2]:
```python
train_folder = os.path.abspath('./data/seg_train/seg_train/')
test_folder = os.path.abspath('./data/seg_test/seg_test/')
```

Helper functions

In [3]:
```python
def read_image(folder,file_name):
    return cv2.imread(os.path.join(folder,file_name),cv2.IMREAD_UNCHANGED)

def join_directories(folders):
    current_path = None
    for folder in folders:
        if current_path:
            current_path = os.path.join(current_path,folder)
        else:
            current_path = folder
            continue
    return current_path
```

Description

- **The data chosen for this project is a publicly available data set provided by Intel corporation and available at https://www.kaggle.com/puneet6060/intel-image-classification**
- **This Data contains around 25k images of size 150x150 distributed under 6 categories.**
  - Buildings
  - Forest
  - Glacier
  - Mountain
  - Sea
  - Street

In [4]:
```python
folder_sizes = dict()
for folder in os.listdir(train_folder):
    folder_sizes[folder] = len(os.listdir(os.path.join(train_folder,folder)))

print("Label distribution is as follows")
display(folder_sizes)
```

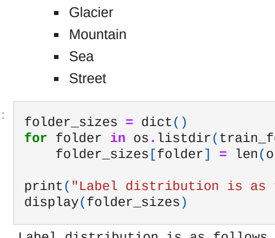
```
Label distribution is as follows
{'buildings': 2191,
 'street': 2382,
 'mountain': 2512,
 'glacier': 2404,
 'sea': 2274,
 'forest': 2271}
```

In [5]:
```python
label_files = dict()
for folder in os.listdir(train_folder):
    label_files[folder] = [file for file in os.listdir(os.path.join(train_folder,folder))]
```

Sample Images

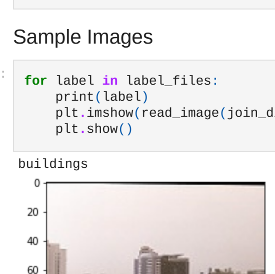
In [6]:
```python
for label in label_files:
    print(label)
    plt.imshow(read_image(join_directories([train_folder,label]),label_files[label][0]))
    plt.show()
```

buildings

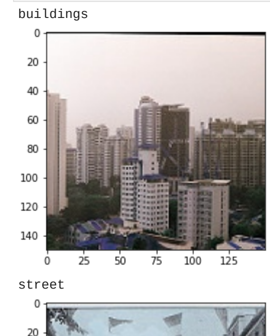


street

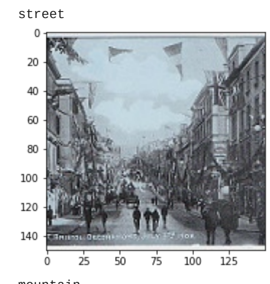
mountain



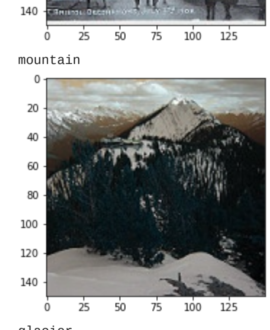
glacier



sea



forest



Data cleaning & Feature engineering

- We will be focusing on CNN architectures to derive relevant features for us.
- Since data contains RGB values we will be Normalizing them.
- For loading data We use a Keras reading library which can convert our data to batches of Tensors
- We shuffle the dataset for every iteration

For sharpening we apply tensorflow's contrast adjustment module

In [16]:
```python
train_data = keras.preprocessing.image_dataset_from_directory(train_folder,
                                                              labels='inferred',
                                                              image_size = (150,150),shuffle=True,
                                                              validation_split=0.3,
                                                              subset='training',seed=8,batch_size=16)

val_data = keras.preprocessing.image_dataset_from_directory(train_folder,
                                                              labels='inferred',
                                                              image_size = (150,150),shuffle=True,
                                                              validation_split=0.3,
                                                              subset='validation',seed=8,batch_size=16)
```

```
Found 14034 files belonging to 6 classes.
Using 9824 files for training.
Found 14034 files belonging to 6 classes.
Using 4210 files for validation.
```

In [7]:
```python
class_names = train_data.class_names
print(class_names)
```

```
['buildings', 'forest', 'glacier', 'mountain', 'sea', 'street']
```

Before Normalization

In [55]:
```python
plt.figure(figsize=(10, 10))
for images, labels in train_data.take(1):
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))
        plt.title(class_names[labels[i]])
        plt.axis("off")
```

Training

In [6]:
```python
num_classes = 6
epochs=10
```

Model - 1

- In this model we will be using Convolution 2D layers with Max pooling. With a single Dense layer before output layer

In [60]:
```python
model = Sequential([
  layers.experimental.preprocessing.Rescaling(1./255, input_shape=(150, 150, 3)),
  layers.Conv2D(16, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
#   layers.Conv2D(64, 3, padding='same', activation='relu'),
#   layers.MaxPooling2D(),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(num_classes)
])
```

In [61]:
```python
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

In [62]:
```python
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
rescaling_4 (Rescaling)      (None, 150, 150, 3)       0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 150, 150, 16)      448
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 75, 75, 16)        0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 75, 75, 32)        4640
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 37, 37, 32)        0
_________________________________________________________________
flatten_3 (Flatten)          (None, 43808)             0
_________________________________________________________________
dense_6 (Dense)              (None, 128)               5607552
_________________________________________________________________
dense_7 (Dense)              (None, 6)                 774
=================================================================
Total params: 5,613,414
Trainable params: 5,613,414
Non-trainable params: 0
_________________________________________________________________
```

In [63]:
```python
history = model.fit(
  train_data,
  validation_data=val_data,
  epochs=epochs
)
```

```
Epoch 1/10
614/614 [==============================] - 131s 212ms/step - loss: 1.3344 - accuracy: 0.5427 - val_loss: 0.7855 - val_accuracy: 0.7102
Epoch 2/10
614/614 [==============================] - 24s 39ms/step - loss: 0.6529 - accuracy: 0.7600 - val_loss: 0.6660 - val_accuracy: 0.7615
Epoch 3/10
614/614 [==============================] - 24s 39ms/step - loss: 0.4266 - accuracy: 0.8455 - val_loss: 0.7290 - val_accuracy: 0.7561
Epoch 4/10
614/614 [==============================] - 24s 40ms/step - loss: 0.2230 - accuracy: 0.9264 - val_loss: 0.8428 - val_accuracy: 0.7587
Epoch 5/10
614/614 [==============================] - 25s 40ms/step - loss: 0.1311 - accuracy: 0.9564 - val_loss: 1.0445 - val_accuracy: 0.7458
Epoch 6/10
614/614 [==============================] - 25s 40ms/step - loss: 0.0997 - accuracy: 0.9689 - val_loss: 1.1834 - val_accuracy: 0.7461
Epoch 7/10
614/614 [==============================] - 24s 39ms/step - loss: 0.0702 - accuracy: 0.9804 - val_loss: 1.2481 - val_accuracy: 0.7599
Epoch 8/10
614/614 [==============================] - 24s 38ms/step - loss: 0.0635 - accuracy: 0.9784 - val_loss: 1.2040 - val_accuracy: 0.7708
Epoch 9/10
614/614 [==============================] - 24s 39ms/step - loss: 0.0334 - accuracy: 0.9927 - val_loss: 1.3832 - val_accuracy: 0.7570
Epoch 10/10
614/614 [==============================] - 24s 40ms/step - loss: 0.0407 - accuracy: 0.9904 - val_loss: 1.3672 - val_accuracy: 0.7606
```

In [65]:
```python
model.save('./model_1_with_16_32_64_d128_d6')
```

```
INFO:tensorflow:Assets written to: ./model_1_with_16_32_64_d128_d6/assets
```

In [67]:
```python
del model
gc.collect()
```

Out[67]: 37558

Model 1 Results

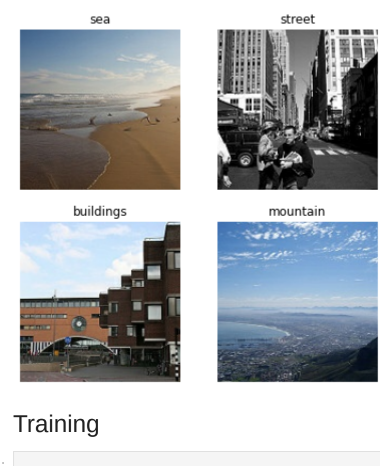
In [64]:
```python
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

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

epochs_range = range(epochs)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
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')
plt.show()
```



In [69]:
```python
device = cuda.get_current_device()
device.reset()
```

Model 2

- In this model we will try the same network but this time to reduce the overfitting we can try doing a random dropout

In [8]:
```python
model_2 = Sequential([
  layers.experimental.preprocessing.Rescaling(1./255, input_shape=(150, 150, 3)),
  layers.Conv2D(16, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Dropout(0.2),
#   layers.Conv2D(64, 3, padding='same', activation='relu'),
#   layers.MaxPooling2D(),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(num_classes)
])
```

In [10]:
```python
model_2.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

In [11]:
```python
model_2.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
rescaling_1 (Rescaling)      (None, 150, 150, 3)       0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 150, 150, 16)      448
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 75, 75, 16)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 75, 75, 32)        4640
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 37, 37, 32)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 37, 37, 32)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 43808)             0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               5607552
_________________________________________________________________
dense_2 (Dense)              (None, 6)                 774
=================================================================
Total params: 5,613,414
Trainable params: 5,613,414
Non-trainable params: 0
_________________________________________________________________
```

In [13]:
```python
history_2 = model_2.fit(
  train_data,
  validation_data=val_data,
  epochs=epochs
)
```

```
Epoch 1/10
614/614 [==============================] - 31s 43ms/step - loss: 1.3312 - accuracy: 0.5199 - val_loss: 0.7631 - val_accuracy: 0.7176
Epoch 2/10
614/614 [==============================] - 24s 39ms/step - loss: 0.6887 - accuracy: 0.7440 - val_loss: 0.8325 - val_accuracy: 0.6922
Epoch 3/10
614/614 [==============================] - 24s 39ms/step - loss: 0.4740 - accuracy: 0.8287 - val_loss: 0.6932 - val_accuracy: 0.7753
Epoch 4/10
614/614 [==============================] - 24s 40ms/step - loss: 0.2813 - accuracy: 0.9044 - val_loss: 0.9063 - val_accuracy: 0.7572
Epoch 5/10
614/614 [==============================] - 24s 40ms/step - loss: 0.1742 - accuracy: 0.9435 - val_loss: 1.0556 - val_accuracy: 0.7705
Epoch 6/10
614/614 [==============================] - 24s 39ms/step - loss: 0.1215 - accuracy: 0.9625 - val_loss: 1.1314 - val_accuracy: 0.7732
Epoch 7/10
614/614 [==============================] - 24s 40ms/step - loss: 0.0917 - accuracy: 0.9714 - val_loss: 1.1769 - val_accuracy: 0.7720
Epoch 8/10
614/614 [==============================] - 24s 39ms/step - loss: 0.0566 - accuracy: 0.9831 - val_loss: 1.3447 - val_accuracy: 0.7553
Epoch 9/10
614/614 [==============================] - 25s 40ms/step - loss: 0.0681 - accuracy: 0.9806 - val_loss: 1.5231 - val_accuracy: 0.7245
Epoch 10/10
614/614 [==============================] - 24s 39ms/step - loss: 0.0587 - accuracy: 0.9815 - val_loss: 1.4486 - val_accuracy: 0.7722
```

In [14]:
```python
model_2.save('./model_2_with_16_32_64_drop_d128_d6')
```

```
INFO:tensorflow:Assets written to: ./model_2_with_16_32_64_drop_d128_d6/assets
```

In [15]:
```python
del model_2
gc.collect()
```

Out[15]: 6814

Model 2 results

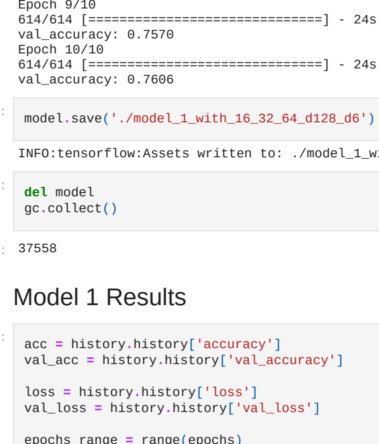
In [16]:
```python
acc = history_2.history['accuracy']
val_acc = history_2.history['val_accuracy']

loss = history_2.history['loss']
val_loss = history_2.history['val_loss']

epochs_range = range(epochs)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
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')
plt.show()
```



In [33]:
```python
device = cuda.get_current_device()
device.reset()
```

Model 3

- As Overfitting is not getting resolved we can try Increasing the dataset by Data augmentation techniques such as random rotation, translation etc.

In [7]:
```python
datagen = ImageDataGenerator(
                            shear_range=0.2,
                            zoom_range=0.2,
                            horizontal_flip=True,
                            validation_split=0.3,
                            )
```

In [8]:
```python
train_data = datagen.flow_from_directory(
                                        train_folder,
                                        target_size = (150,150),shuffle=True,
                                        subset='training',seed=8,batch_size=16,
                                        class_mode='binary'
                                        )
val_data =  datagen.flow_from_directory(
                                        train_folder,
                                        target_size = (150,150),shuffle=True,
                                        subset='validation',seed=8,batch_size=16,
                                        class_mode='binary'
                                        )
```

```
Found 9826 images belonging to 6 classes.
Found 4208 images belonging to 6 classes.
```

In [9]:
```python
model_3 = Sequential([
  layers.experimental.preprocessing.Rescaling(1./255, input_shape=(150, 150, 3)),
  layers.Conv2D(16, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Dropout(0.2),
#   layers.Conv2D(64, 3, padding='same', activation='relu'),
#   layers.MaxPooling2D(),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(num_classes)
])
```

In [10]:
```python
model_3.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

In [11]:
```python
model_3.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
rescaling (Rescaling)        (None, 150, 150, 3)       0
_________________________________________________________________
conv2d (Conv2D)              (None, 150, 150, 16)      448
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 75, 75, 16)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 75, 75, 32)        4640
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 37, 37, 32)        0
_________________________________________________________________
dropout (Dropout)            (None, 37, 37, 32)        0
_________________________________________________________________
flatten (Flatten)            (None, 43808)             0
_________________________________________________________________
dense (Dense)                (None, 128)               5607552
_________________________________________________________________
dense_1 (Dense)              (None, 6)                 774
=================================================================
Total params: 5,613,414
Trainable params: 5,613,414
Non-trainable params: 0
_________________________________________________________________
```

In [12]:
```python
history_3 = model_3.fit(
  train_data,
  validation_data=val_data,
  epochs=epochs
)
```

```
Epoch 1/10
615/615 [==============================] - 106s 166ms/step - loss: 1.3872 - accuracy: 0.4813 - val_loss: 0.9106 - val_accuracy: 0.6419
Epoch 2/10
615/615 [==============================] - 98s 160ms/step - loss: 0.8223 - accuracy: 0.7010 - val_loss: 0.7232 - val_accuracy: 0.7395
Epoch 3/10
615/615 [==============================] - 100s 163ms/step - loss: 0.7225 - accuracy: 0.7378 - val_loss: 0.7170 - val_accuracy: 0.7293
Epoch 4/10
615/615 [==============================] - 101s 164ms/step - loss: 0.6361 - accuracy: 0.7703 - val_loss: 0.5943 - val_accuracy: 0.7866
Epoch 5/10
615/615 [==============================] - 101s 164ms/step - loss: 0.5743 - accuracy: 0.7937 - val_loss: 0.5638 - val_accuracy: 0.7997
Epoch 6/10
615/615 [==============================] - 100s 163ms/step - loss: 0.5325 - accuracy: 0.8060 - val_loss: 0.6303 - val_accuracy: 0.7745
Epoch 7/10
615/615 [==============================] - 100s 163ms/step - loss: 0.4954 - accuracy: 0.8223 - val_loss: 0.5485 - val_accuracy: 0.7997
Epoch 8/10
615/615 [==============================] - 100s 162ms/step - loss: 0.4677 - accuracy: 0.8279 - val_loss: 0.5345 - val_accuracy: 0.8082
Epoch 9/10
615/615 [==============================] - 101s 164ms/step - loss: 0.4311 - accuracy: 0.8419 - val_loss: 0.5385 - val_accuracy: 0.8032
Epoch 10/10
615/615 [==============================] - 100s 163ms/step - loss: 0.4311 - accuracy: 0.8415 - val_loss: 0.5283 - val_accuracy: 0.8158
```

In [13]:
```python
model_3.save('./model_2_with_horflip_shear2_zoom_2_16_32_64_drop_d128_d6')
```

```
INFO:tensorflow:Assets written to: ./model_2_with_horflip_shear2_zoom_2_16_32_64_drop_d128_d6/assets
```

In [14]:
```python
del model_3
gc.collect()
```

Out[14]: 6795

Model 3 results

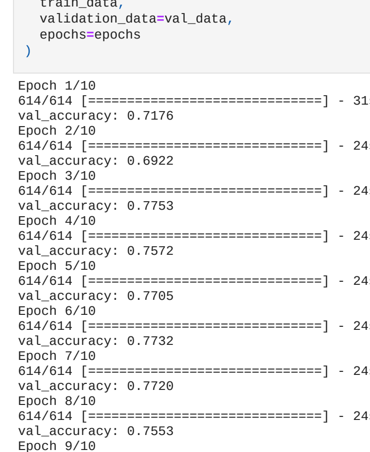
In [15]:
```python
acc = history_3.history['accuracy']
val_acc = history_3.history['val_accuracy']

loss = history_3.history['loss']
val_loss = history_3.history['val_loss']

epochs_range = range(epochs)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
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')
plt.show()
```



In [17]:
```python
test_data = keras.preprocessing.image_dataset_from_directory(test_folder,
                                                              labels='inferred',
                                                              image_size = (150,150),shuffle=False,
                                                              seed=8,batch_size=16)
```

```
Found 3000 files belonging to 6 classes.
```

In [49]:
```python
labels = list()
for x,y in test_data:
    labels.extend(y.numpy())
```

In [54]:
```python
model_1 = tf.keras.models.load_model('./model_1_with_16_32_64_d128_d6/')
model_1_predictions = model_1.predict_classes(test_data)
del model_1
gc.collect()
```

Out[54]: 10707

In [55]:
```python
model_2 = tf.keras.models.load_model('./model_2_with_16_32_64_drop_d128_d6/')
model_2_predictions = model_2.predict_classes(test_data)
del model_2
gc.collect()
```

Out[55]: 11713

In [56]:
```python
model_3 = tf.keras.models.load_model('./model_2_with_horflip_shear2_zoom_2_16_32_64_drop_d128_d6/')
model_3_predictions = model_3.predict_classes(test_data)
del model_3
gc.collect()
```

Out[56]: 11713

In [66]:
```python
print("Model 1 Classification report")
print()
print(classification_report(labels,model_1_predictions))
```

Model 1 Classification report

	precision	recall	f1-score	support
0	0.76	0.68	0.72	437
1	0.90	0.93	0.91	474
2	0.79	0.59	0.68	553
3	0.63	0.78	0.70	525
4	0.70	0.74	0.72	510
5	0.78	0.80	0.79	501
accuracy			0.75	3000
macro avg	0.76	0.75	0.75	3000
weighted avg	0.76	0.75	0.75	3000

In [67]:
```python
print("Model 2 Classification report")
print()
print(classification_report(labels,model_2_predictions))
```

Model 2 Classification report

	precision	recall	f1-score	support
0	0.75	0.72	0.74	437
1	0.93	0.93	0.93	474
2	0.75	0.71	0.73	553
3	0.69	0.76	0.73	525
4	0.74	0.71	0.73	510
5	0.78	0.80	0.79	501
accuracy			0.77	3000
macro avg	0.77	0.77	0.77	3000
weighted avg	0.77	0.77	0.77	3000

In [68]:
```python
print("Model 3 Classification report")
print()
print(classification_report(labels,model_3_predictions))
```

Model 3 Classification report

	precision	recall	f1-score	support
0	0.87	0.72	0.79	437
1	0.87	0.98	0.92	474
2	0.77	0.72	0.74	553
3	0.70	0.82	0.75	525
4	0.81	0.76	0.78	510
5	0.84	0.85	0.84	501
accuracy			0.80	3000
macro avg	0.81	0.81	0.81	3000
weighted avg	0.81	0.80	0.80	3000

Summary & Key findings

- Based on the training curves obtained from the models, it is evident that a lot of overfitting is happening very early in the model itself.
- Adding dropout increased max acquired Val accuray slightly 0.77 with drop out & 0.75 without
- Augmenting the data improved the test accuracy further to 0.80 and validation loss was decreasing(More scope for better results).
- Based on these observations & the classification reports shown above we should go with the Model 3 - with Data augmentation.

Further steps

- As Augmenting the data is improving the Validation set accuracy, adding more data will definitely result in better performance of the model.
- Augmenting data also improved the training a lot, as the training curves indicate both traning accuracy & validation accuracy are improving. We can train for more epochs which will definitely result in better accuracy.
- As overfitting is happening quiet early in the model, we can decrease the complexity of the model. Reducing no. of convolution layers will help.
- Hyper parameter tuning such as changing the optimizer, changing batch size might help better generalize. Due to GPU restrictions of My laptop I am not able to test with higher batch size than 16. And any model above 10 Lakh trainable parameters is becoming unweildy for the GPU.

In [ ]:

```
```