* Please Read Observations Bottom Of The Notebook *

Link Of Dataset : https://www.kaggle.com/puneet6060/intel-image-classification (https://www.kaggle.com/puneet6060/intel-image-classification)

Load Data From Kaggle and Extract From Zip

In [2]:

```
--header="Host: storage.googleapis.com" --header="User-Agent: Mozilla/5.0 (X11; Linu
```

```
--2020-05-02 09:22:45--  https://storage.googleapis.com/kaggle-data-se
ts/111880/269359/bundle/archive.zip?GoogleAccessId=web-data@kaggle-161
607.iam.gserviceaccount.com&Expires=1588426820&Signature=S7eh8bw5uxQpF
niUMdVUiJCJuigSUHszRQexSOYoWAEPOY1VSKYcZWAdZCazLBT5Eg6zX0HZwK06GPvoWGP
ZUdn9ZAXSGN%2FBiq9J3yknLwCw%2FMc9%2Blm5IAiGGimbnvVaNtVGATEYyzj6yRRI6tI
mDK8OcXgbxO3FoRQGtNL1gGKlGplCchRb44IFaTQFp2o7ByOmpCeVXgFxtkuef7C3Fv3Jb
EAdNjo8vjh7oniNVXYlBY8maSy0ZU28QTnHoVZ%2B5fs7LK6KXwd3jlM3QFABvfO7tyDYO
fjjXmc56%2Bood%2F5J%2FHZGLelBtsepBf48Zj%2FmbIH3ST1x9Mpj7yq8pkxS5A%3D%3
D&response-content-disposition=attachment%3B+filename%3Dintel-image-cl
assification.zip (https://storage.googleapis.com/kaggle-data-sets/1118
80/269359/bundle/archive.zip?GoogleAccessId=web-data@kaggle-161607.ia
m.gserviceaccount.com&Expires=1588426820&Signature=S7eh8bw5uxQpFniUMdV
UiJCJuigSUHszRQexSOYoWAEPOY1VSKYcZWAdZCazLBT5Eg6zX0HZwK06GPvoWGPZUdn9Z
AXSGN%2FBiq9J3yknLwCw%2FMc9%2Blm5IAiGGimbnvVaNtVGATEYyzj6yRRI6tImDK8Oc
XgbxO3FoRQGtNL1gGKlGplCchRb44IFaTQFp2o7ByOmpCeVXgFxtkuef7C3Fv3JbEAdNjo
8vjh7oniNVXYlBY8maSy0ZU28QTnHoVZ%2B5fs7LK6KXwd3jlM3QFABvfO7tyDYOfjjXmc
56%2Bood%2F5J%2FHZGLelBtsepBf48Zj%2FmbIH3ST1x9Mpj7yq8pkxS5A%3D%3D&resp
onse-content-disposition=attachment%3B+filename%3Dintel-image-classifi
cation.zip)
Resolving storage.googleapis.com (storage.googleapis.com)... 108.177.9
8.128, 2607:f8b0:400e:c00::80
Connecting to storage.googleapis.com (storage.googleapis.com)|108.177.
98.128|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 363152213 (346M) [application/zip]
Saving to: ‘intel-image-classification.zip’

intel-image-classif 100%[===================>] 346.33M  77.3MB/s    in
4.5s

2020-05-02 09:22:49 (77.3 MB/s) - ‘intel-image-classification.zip’ sav
ed [363152213/363152213]
```

In [3]:

```
!ls
```

```
intel-image-classification.zip  sample_data
```

In [4]:

```
!unzip intel-image-classification.zip
```

```
Streaming output truncated to the last 5000 lines.
  inflating: seg_train/seg_train/mountain/7537.jpg
  inflating: seg_train/seg_train/mountain/7539.jpg
  inflating: seg_train/seg_train/mountain/7551.jpg
  inflating: seg_train/seg_train/mountain/7560.jpg
  inflating: seg_train/seg_train/mountain/7565.jpg
  inflating: seg_train/seg_train/mountain/7578.jpg
  inflating: seg_train/seg_train/mountain/7581.jpg
  inflating: seg_train/seg_train/mountain/7586.jpg
  inflating: seg_train/seg_train/mountain/7647.jpg
  inflating: seg_train/seg_train/mountain/7652.jpg
  inflating: seg_train/seg_train/mountain/7654.jpg
  inflating: seg_train/seg_train/mountain/7662.jpg
  inflating: seg_train/seg_train/mountain/767.jpg
  inflating: seg_train/seg_train/mountain/7672.jpg
  inflating: seg_train/seg_train/mountain/7679.jpg
  inflating: seg_train/seg_train/mountain/7681.jpg
  inflating: seg_train/seg_train/mountain/7693.jpg
  inflating: seg_train/seg_train/mountain/7695.jpg
```

Import Libraries

In [1]:

```
import os
import shutil
from tqdm import tqdm

import cv2
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.utils import shuffle
from sklearn import metrics
from sklearn.metrics import confusion_matrix


%tensorflow_version 2.x
import tensorflow as tf
from tensorflow.keras import applications
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dropout, Flatten, Dense, Conv2D, MaxPooling2D,
from tensorflow.keras import optimizers
from tensorflow.keras.layers import Input
from tensorflow.keras.models import Model
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import optimizers

# tf.compat.v1.logging.set_verbosity(tf.compat.v1.logging.ERROR)
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:1
9: FutureWarning: pandas.util.testing is deprecated. Use the functions
in the public API at pandas.testing instead.
  import pandas.util.testing as tm
```

In [4]:

```
sns.set_style("whitegrid") # For Graph Background White on Colab
```

Define Image Data Generator

In [14]:

```
train_datagen = ImageDataGenerator(rescale = 1./255, rotation_range=30, width_shift
                                   brightness_range=None, shear_range=0.0, zoom_ran
                                   channel_shift_range=0.0, fill_mode='nearest', ho
                                   vertical_flip=False, validation_split=0.0, dtype
```

In [15]:

```
test_datagen = ImageDataGenerator(rescale = 1./255, rotation_range=30, width_shift_
                                  brightness_range=None, shear_range=0.0, zoom_rang
                                  channel_shift_range=0.0, fill_mode='nearest', hor
                                  vertical_flip=False, validation_split=0.0, dtype=
```

Generate Train and Test Image Data

In [20]:

```
train_directory = './seg_train/seg_train/'
test_directory = './seg_test/seg_test/'

n_batch_size = 256
n_epochs = 50
```

In [6]:

```
train_generator = train_datagen.flow_from_directory(train_directory,
                                                    target_size=(150, 150),
                                                    batch_size = n_batch_size,
                                                    class_mode = 'categorical')
```

```
Found 14034 images belonging to 6 classes.
```

In [53]:

```
test_generator = test_datagen.flow_from_directory(test_directory,
                                                  target_size=(150, 150),
                                                  batch_size = n_batch_size,
                                                  class_mode = 'categorical')
```

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

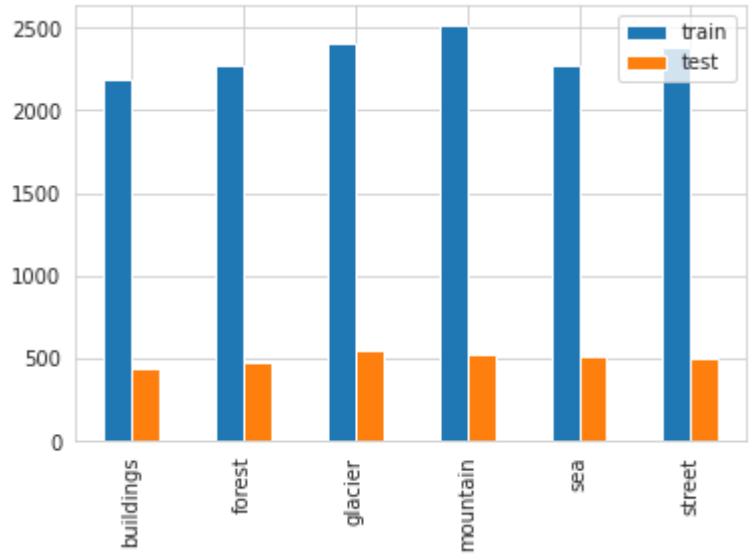
In [25]:

```
train_labels = train_generator.classes
test_labels = test_generator.classes
class_names = list(test_generator.class_indices.keys())
```

Graphical Representation Of Data

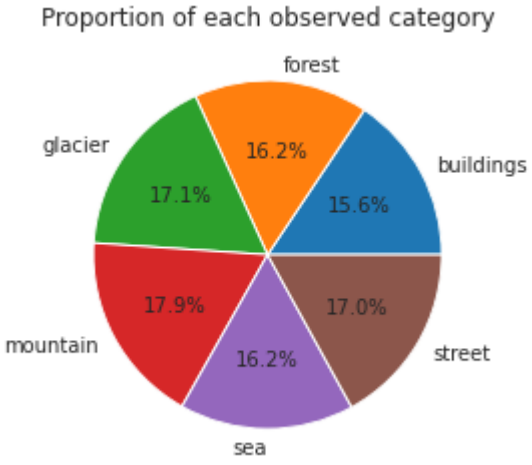
In [9]:

```
_, train_counts = np.unique(train_labels, return_counts=True)
_, test_counts = np.unique(test_labels, return_counts=True)
pd.DataFrame({'train': train_counts, 'test': test_counts}, index=class_names).plot.
plt.show()
```



In [10]:

```
plt.pie(train_counts, explode=(0, 0, 0, 0, 0, 0) , labels=class_names, autopct='%1.
#plt.axis('equal')
plt.title('Proportion of each observed category')
plt.show()
```



Display Some Train Images

In [31]:

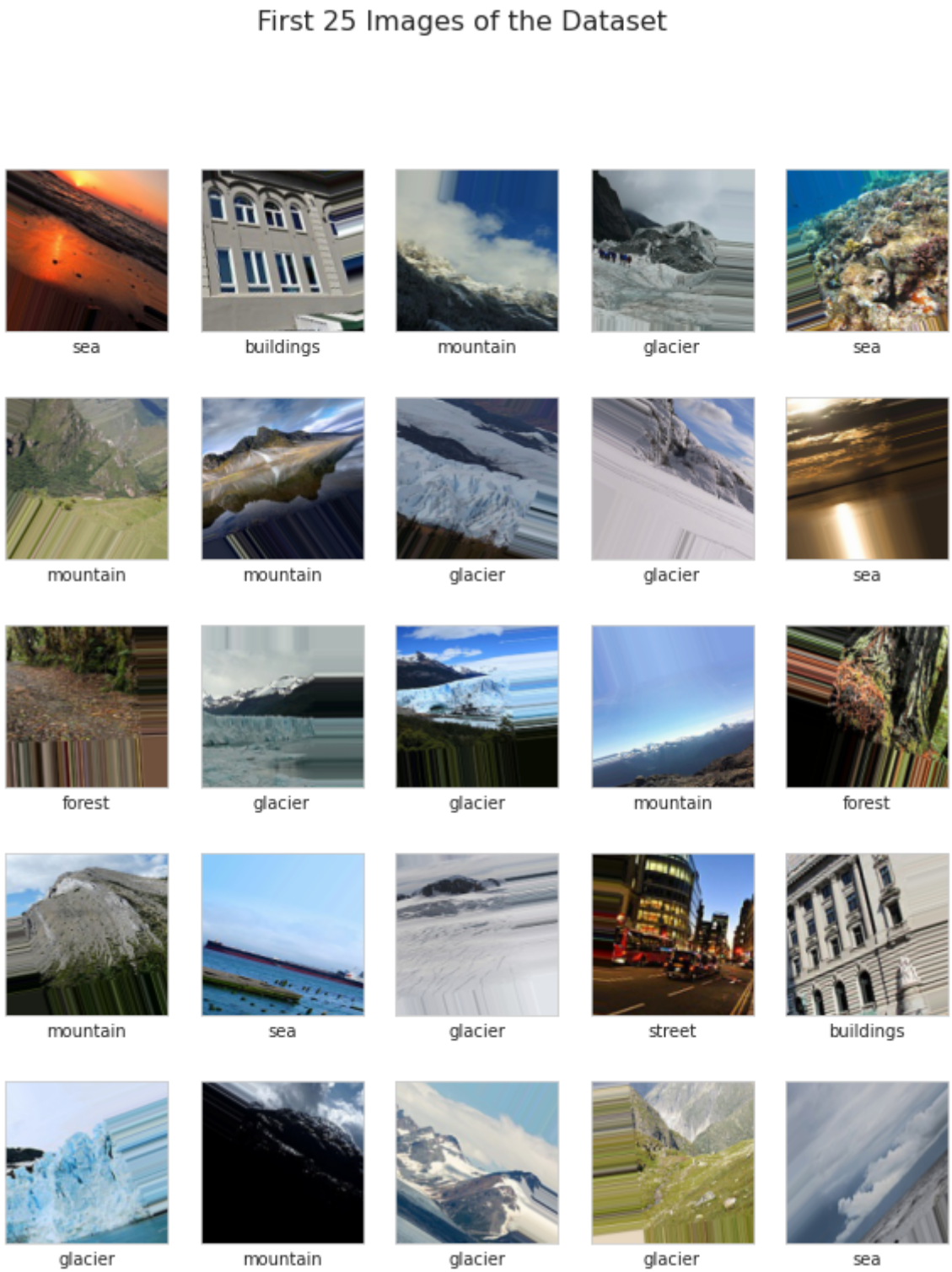
```
def display_train_data(class_names, images, labels, n_rows=5 , n_cols=5 , fig_width
    """
        Display n images from the images array with its corresponding labels
    """
    n_images = n_rows * n_cols

    fig = plt.figure(figsize=(fig_width, fig_height))
    fig.suptitle("First "+ str(n_images) +" Images of the Dataset", fontsize=16)
    for i in range(n_images):
        plt.subplot(n_rows,n_cols,i+1)
        plt.xticks([])
        plt.yticks([])
        plt.grid(False)
        plt.imshow(images[i], cmap=plt.cm.binary)
        plt.xlabel(class_names[np.where(labels[i] == 1)[0][0]])
    plt.show()
```

In [13]:

```
images, labels = train_generator.next()

display_train_data(class_names, images, labels, n_rows=5 , n_cols=5 , fig_width=10,
```



Load VGG16 as Base Model

In [18]:

```
vgg_model = applications.VGG16(weights='imagenet', include_top=False, input_shape=(

print('Model Loaded.')
```

```
Model Loaded.
```

Create Custom Layers Top of on VGG16 Model

In [40]:

```
vgg_model_output = vgg_model.output

top_model = Conv2D(filters=32, kernel_size=(3, 3), activation='relu', padding="same
top_model = Conv2D(filters=64, kernel_size=(3, 3), activation='relu', padding="same
top_model = MaxPooling2D(pool_size=(2, 2), strides=(2, 2))(top_model)
top_model = Dropout(0.25)(top_model)

top_model = Flatten()(top_model)

top_model = Dense(256, activation='relu')(top_model)
top_model = Dropout(0.5)(top_model)

top_model = Dense(512, activation='relu')(top_model)
top_model = Dropout(0.5)(top_model)

top_model = Dense(6, activation='softmax')(top_model)

custom_model = Model(inputs=vgg_model.input, outputs=top_model)
```

Active Last 14 Layers

In [20]:

```
for layer in custom_model.layers[:-14]:
    layer.trainable = False

for layer in custom_model.layers:
    print(layer, layer.trainable)
```

```
<tensorflow.python.keras.engine.input_layer.InputLayer object at 0x7f5
ae241f208> False
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0x7f5ae
24390f0> False
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0x7f5ae
2439438> False
<tensorflow.python.keras.layers.pooling.MaxPooling2D object at 0x7f5ae
2485b00> False
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0x7f5ae
2485e10> False
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0x7f5ae
246bba8> False
<tensorflow.python.keras.layers.pooling.MaxPooling2D object at 0x7f5ae
2e613c8> False
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0x7f5ae
232ac50> False
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0x7f5ae
232add8> False
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0x7f5ae
235d588> False
<tensorflow.python.keras.layers.pooling.MaxPooling2D object at 0x7f5ae
23374a8> False
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0x7f5ae
2356b70> False
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0x7f5ae
23482e8> False
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0x7f5ae
2341518> False
<tensorflow.python.keras.layers.pooling.MaxPooling2D object at 0x7f5ae
2322390> False
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0x7f5ae
2322240> True
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0x7f5ae
233c588> True
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0x7f5ae
232f748> True
<tensorflow.python.keras.layers.pooling.MaxPooling2D object at 0x7f5ae
237a668> True
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0x7f5ae
25df9e8> True
<tensorflow.python.keras.layers.convolutional.Conv2D object at 0x7f5ae
25dfcc0> True
<tensorflow.python.keras.layers.pooling.MaxPooling2D object at 0x7f5ae
25e3ba8> True
<tensorflow.python.keras.layers.core.Dropout object at 0x7f5ae25e3da0>
True
<tensorflow.python.keras.layers.core.Flatten object at 0x7f5ae25edc18>
True
<tensorflow.python.keras.layers.core.Dense object at 0x7f5ae25f3048> T
rue
<tensorflow.python.keras.layers.core.Dropout object at 0x7f5ae25edfd0>
True
<tensorflow.python.keras.layers.core.Dense object at 0x7f5ae24e86a0> T
```

```
rue
<tensorflow.python.keras.layers.core.Dropout object at 0x7f5ae24e8198>
True
<tensorflow.python.keras.layers.core.Dense object at 0x7f5ae24f43c8> T
rue
```

Compile Our Model

In [44]:

```
custom_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['a
```

Generate Directory for Save Checkpoints after every Epoch

In [0]:

```
# shutil.rmtree(checkpoint_dir)
```

In [47]:

```
# Save The Checkpoints

# Include the epoch in the file name (uses `str.format`)
cp_dir = 'saved_checkpoints/'
if not os.path.exists(cp_dir):
    os.mkdir(cp_dir)

checkpoint_path = cp_dir+"cp-{epoch:04d}.ckpt"
checkpoint_dir = os.path.dirname(checkpoint_path)
```

In [48]:

```
# Create a callback that saves the model's weights every 5 epochs
cp_callback = ModelCheckpoint(
                  filepath=checkpoint_path,
                  verbose=0,
                  save_weights_only=True,
                  save_freq=5)
# Save the weights using the `checkpoint_path` format
custom_model.save_weights(checkpoint_path.format(epoch=0))
```

Start Training

In [149]:

```
trained_history = custom_model.fit( train_generator, steps_per_epoch = len(train_ge
                          epochs = n_epochs, use_multiprocessing=False, validation_
                          workers = 1, validation_steps=len(test_generator), callba
```

```
Epoch 1/50
55/55 [==============================] - 97s 2s/step - loss: 1.2514
- accuracy: 0.4326 - val_loss: 0.7622 - val_accuracy: 0.7023
Epoch 2/50
55/55 [==============================] - 96s 2s/step - loss: 0.6823
- accuracy: 0.7586 - val_loss: 0.5522 - val_accuracy: 0.8063
Epoch 3/50
55/55 [==============================] - 96s 2s/step - loss: 0.5234
- accuracy: 0.8269 - val_loss: 0.4429 - val_accuracy: 0.8390
Epoch 4/50
55/55 [==============================] - 96s 2s/step - loss: 0.4408
- accuracy: 0.8516 - val_loss: 0.4131 - val_accuracy: 0.8590
Epoch 5/50
55/55 [==============================] - 96s 2s/step - loss: 0.4461
- accuracy: 0.8508 - val_loss: 0.4140 - val_accuracy: 0.8570
Epoch 6/50
55/55 [==============================] - 96s 2s/step - loss: 0.3999
- accuracy: 0.8684 - val_loss: 0.4361 - val_accuracy: 0.8567
Epoch 7/50
```

In [0]:

```
```

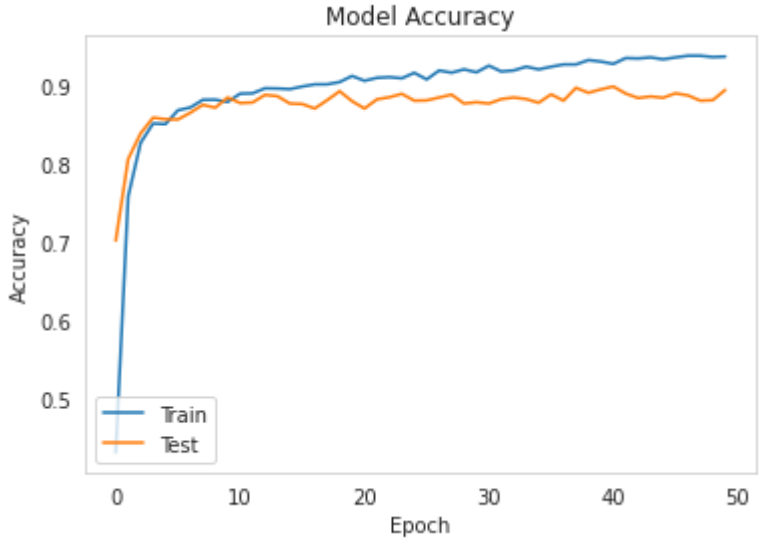
Plot Graph to Compair Train & Test Accuracy and Loss

In [51]:

```
def plot_graph(train_history, train_acc, test_acc, title, x_label, y_label):
    plt.plot(train_history.history[train_acc], label='Train')
    plt.plot(train_history.history[test_acc], label='Test')
    plt.title(title)
    plt.ylabel(y_label)
    plt.xlabel(x_label)
    plt.legend(loc='lower left')
    plt.grid()
    plt.show()
```

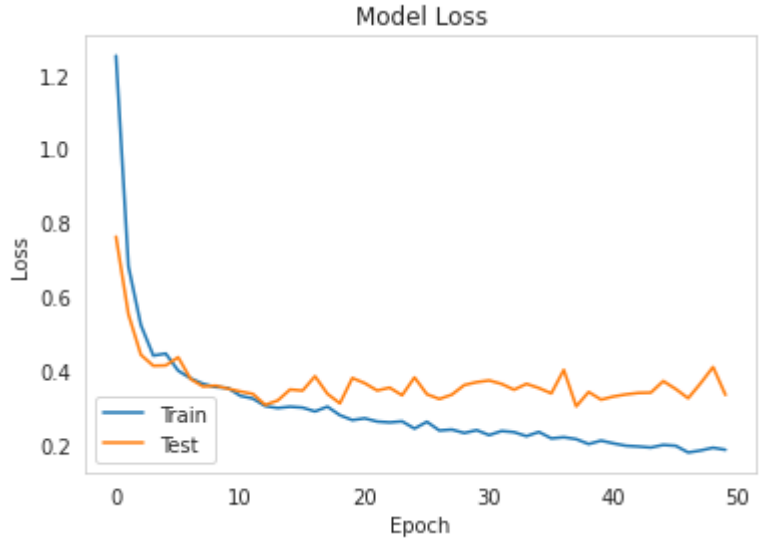
In [150]:

```
title = 'Model Accuracy'
x_label = 'Epoch'
y_label = 'Accuracy'
plot_graph(trained_history, 'accuracy', 'val_accuracy', title, x_label, y_label)
```



In [151]:

```
title = 'Model Loss'
x_label = 'Epoch'
y_label = 'Loss'
plot_graph(trained_history, 'loss', 'val_loss', title, x_label, y_label)
```



In [0]:

```

```

Load Weights which has Maximum Accuracy and Minimum Loss

In [24]:

```
# Get Latest Checkpoint from Saved Directory
# latest_cp = tf.train.latest_checkpoint(checkpoint_dir)
# print(latest_cp)

cp = checkpoint_dir+'/cp-0019.ckpt'

# Create a new model instance
# custom_model = create_model()

# Load the previously saved weights
custom_model.load_weights(cp)

# Re-evaluate the model
loss, acc = custom_model.evaluate(test_generator, steps=len(test_generator), verbos
print("Restored Model, Accuracy: {:5.2f}%".format(100*acc))
```

```
12/12 [==============================] - 15s 1s/step - loss: 0.3190 -
accuracy: 0.8907
Restored Model, Accuracy: 89.07%
```

In [0]:

```

```

Generate Test Data For Evaluation

In [201]:

```
test_generator = test_datagen.flow_from_directory(test_directory,
                                                  target_size=(150, 150),
                                                  batch_size = n_batch_size,
                                                  class_mode = 'categorical',
                                                  shuffle = False)
# set 'shuffle = False' in test_data_generator when you use 'flow_from_directory'
```

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

Prediction For Test Data

In [35]:

```
test_generator.reset()
predictions = custom_model.predict(test_generator, steps=len(test_generator), verbo
```

```
12/12 [==============================] - 15s 1s/step
```

In [0]:

```

```

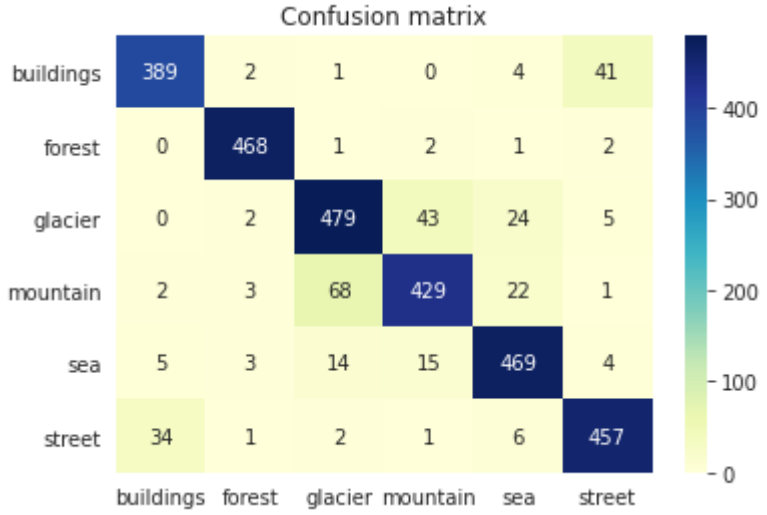
Represent The Match Labels and Mismatch Labels in the form of Confusion Matrix

In [203]:

```
pred_labels = np.argmax(predictions, axis = 1)
test_labels = test_generator.classes
class_names = list(test_generator.class_indices.keys())

CM = confusion_matrix(test_labels, pred_labels)

ax = plt.axes()
sns.heatmap(CM, annot=True, xticklabels=class_names, yticklabels=class_names, cmap=
ax.set_title('Confusion matrix')
plt.show()
```



Detaile Report Of Accuracy

In [169]:

```
report = metrics.classification_report(test_labels, pred_labels, target_names=class
print(report)
```

```
              precision    recall  f1-score   support

   buildings       0.90      0.89      0.89       437
      forest       0.97      0.98      0.98       474
     glacier       0.84      0.86      0.85       553
    mountain       0.87      0.82      0.84       525
         sea       0.89      0.93      0.91       510
      street       0.89      0.91      0.90       501

    accuracy                           0.89      3000
   macro avg       0.89      0.89      0.89      3000
weighted avg       0.89      0.89      0.89      3000
```

Trained Model Apply on New (or Unseen) Data

Load New Data

In [53]:

```
IMAGE_SIZE = (150, 150)
```

In [54]:

```
def load_new_data(datasets):

    images = []

    print("\nLoading {}".format(datasets))
        # Iterate through each image in our folder
    for file in tqdm(os.listdir(datasets)):

                # Get the path name of the image
        img_path = os.path.join(datasets, file)

                # Open and resize the img
        image = cv2.imread(img_path)
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        image = cv2.resize(image, IMAGE_SIZE)

        # Append the image and its corresponding label to the output
        images.append(image)

    images = np.array(images, dtype = 'float32')

    return images
```

In [65]:

```
new_images = load_new_data('./seg_pred/seg_pred/')
```

```
  3%|▌          | 202/7301 [00:00<00:03, 2018.44it/s]

Loading ./seg_pred/seg_pred/
100%|██████████| 7301/7301 [00:03<00:00, 2014.12it/s]
```

In [55]:

```
# Scale the Data
new_images = new_images / 255.0
```

Predict New Data

In [56]:

```
predictions_new_data = custom_model.predict(new_images)
pred_labels_new_data = np.argmax(predictions_new_data, axis = 1)
```

Display Some Predicted Images with Labels

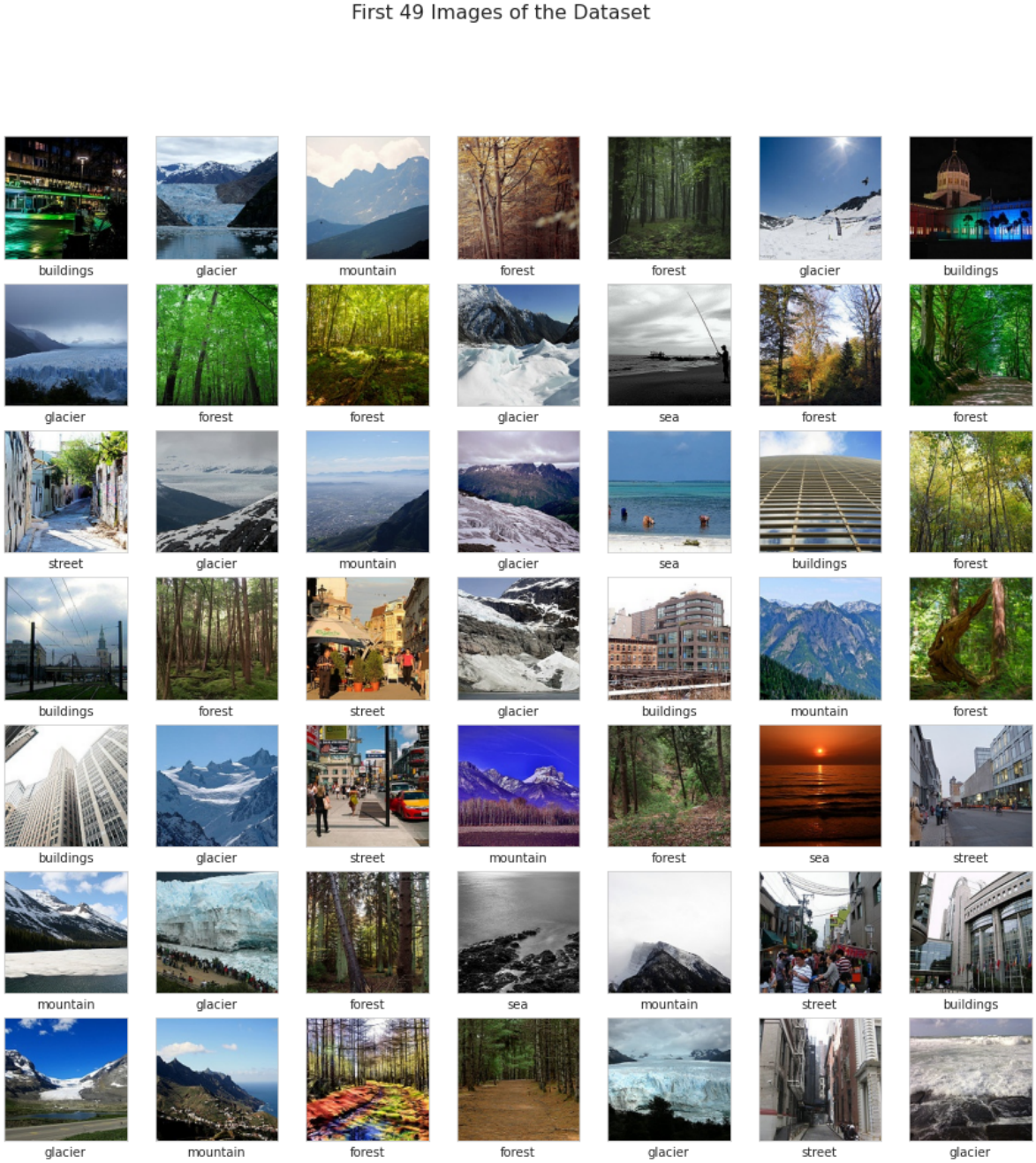
In [60]:

```
def display_data(class_names, images, labels, n_rows=5 , n_cols=5 , fig_width=10, f
    """
        Display n images from the images array with its corresponding labels
    """
    n_images = n_rows * n_cols

    fig = plt.figure(figsize=(fig_width, fig_height))
    fig.suptitle("First "+ str(n_images) +" Images of the Dataset", fontsize=16)
    for i in range(n_images):
        plt.subplot(n_rows,n_cols,i+1)
        plt.xticks([])
        plt.yticks([])
        plt.grid(False)
        plt.imshow(images[i], cmap=plt.cm.binary)
        plt.xlabel(class_names[labels[i]])
    plt.show()
```

In [82]:

```
display_data(class_names, new_images, pred_labels_new_data, n_rows=7, n_cols=7 , fi
```



Save the Predicted Images in their Class Folders

In [65]:

```
def classify_new_images(class_names, images, labels):

    dirs = 'new_data_output'
    if not os.path.exists(dirs):
        os.mkdir(dirs)

    for i in range(len(images)):

        class_nm = class_names[labels[i]]

        class_path = os.path.join(dirs, class_nm)

        if not os.path.exists(class_path):
            os.mkdir(class_path)


        filename = class_path+'/'+class_nm+str(f"{i:04}")+'.jpg'
        cv2.imwrite(filename, images[i])
```

In [66]:

```
classify_new_images(class_names, new_images*255.0 , pred_labels_new_data)
```

In [0]:

```
# shutil.rmtree('new_data_output')
```

Link of Trained Model

Download Trained Model: https://drive.google.com/open?id=1-2nL_Tt8mWlKZRv-u8P7tLjLYVeT0Lis (https://drive.google.com/open?id=1-2nL_Tt8mWlKZRv-u8P7tLjLYVeT0Lis)

In [ ]:

```
# Load Model After Download
from tensorflow.keras.models import load_model

custom_model = load_model('custom_model_89.h5')
```

* Observations *

1. The data is very complecated with very small size images.
2. I tried various pre-processing steps but because of complicated data and small size images I got 89% accuracy.
3. Also I tried with unseen data given by the kaggle compitation. Then I got good results.
4. The mismatch class images are also similer to match class images. (Fore example, "glacier & sea", "glacier & mountain", "street & buildings" images are more similar to each other.)

5. If data have high quality images then I think accuracy increase by 5% to 7% of current accuracy. (i.e, accuracy would be 94% to 96%).

In [ ]: