Capstone Project 2 Final Report

1. Updated Problem Statement:

Image classification is one of the most popular areas in the last decade. There are many developing areas that needs AI implementations in order to recognize the environmental images. For instance; unmanned vehicles need maximum possible accuracy to render their environmental obstacles. In my project I want to classify nature pictures with high accuracy and find ways to improve the accuracy. That will contribute future models as well as training of future data scientists. The objective of this project is as defined below.

- According to given dataset, to build neural network model to predict randomly chosen nature pictures under six classes.
- To find possible ways to improve the accuracy of model.
- To give a general picture of accuracy for the test set.

2. Obtaining Data:

The context of the data includes image data of Natural Scenes around the world. Photographs were collected from Jan Bottinger who is a German photographer and traveller. He publishes his photographs publicly on Unsplash web portal. The Train, Test and Prediction data is separated in each zip files. There are around 14k images in Train, 3k in Test and 7k in Prediction.

This data was initially published on https://datahack.analyticsvidhya.com by Intel to host an Image Classification Challenge. This data contains around 25k images of size 150x150 distributed under 6 categories. {'buildings' -> 0, 'forest' -> 1, 'glacier' -> 2, 'mountain' -> 3, 'sea' -> 4, 'street' -> 5}

The data downloaded from web portal under the folder classification for train and test set. Both train and test folders have subfolders that previously classified in six categories. There is a third folder for prediction which has no classification in it.

In order to import data “os” module was imported, class names were defined, a new function was created. This steps are reflected below:

```
1 import numpy as np
2 import os
```

Categories

In [46]:
```
class_names = ['mountain', 'street', 'glacier', 'buildings', 'sea', 'forest']
class_names_label = {'mountain': 0,
                     'street' : 1,
                     'glacier' : 2,
                     'buildings' : 3,
                     'sea' : 4,
                     'forest' : 5
                     }
nb_classes = 6
```

Data loading

In [3]:
```
def load_data():
    """
        Load the data:
            - images to train the network.
            - images to evaluate how accurately the network learned to classify images.
    """

    datasets = ['seg_train/seg_train', 'seg_test/seg_test']
    size = (150,150)
    output = []
    for dataset in datasets:
        directory =  dataset
        images = []
        labels = []
        for folder in os.listdir(directory):
            curr_label = class_names_label[folder]
            for file in os.listdir(directory + "/" + folder):
                img_path = directory + "/" + folder + "/" + file
                curr_img = cv2.imread(img_path)
                curr_img = cv2.resize(curr_img, size)
                images.append(curr_img)
                labels.append(curr_label)
        images, labels = shuffle(images, labels)
        images = np.array(images, dtype = 'float32')
        labels = np.array(labels, dtype = 'int32')

        output.append((images, labels))

    return output
```

In [4]:
```
(train_images, train_labels), (test_images, test_labels) = load_data()
```

3. Data Cleaning & Data Wrangling:

The data was pretty clean and ready to use. After importing the data, the information for the data was analyzed as it is seen below:

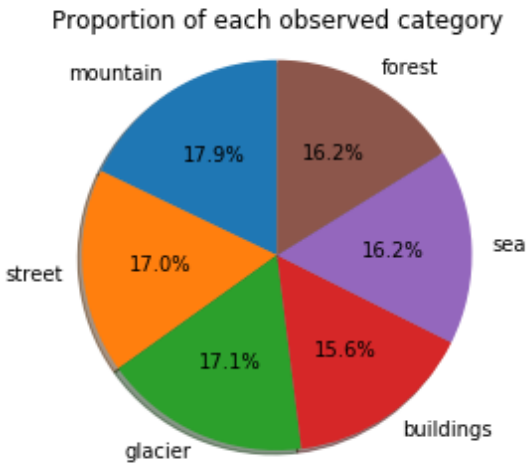
In [5]:
```
print ("Number of training examples: " + str(train_labels.shape[0]))
print ("Number of testing examples: " + str(test_labels.shape[0]))
print ("Each image is of size: " + str(train_images.shape[1:]))
```

```
Number of training examples: 14034
Number of testing examples: 3000
Each image is of size: (150, 150, 3)
```

Distribution of Data

In [20]:
```
sizes = np.bincount(train_labels)
explode = (0, 0, 0, 0, 0, 0)
plt.pie(sizes, explode=explode, labels=class_names,
autopct='%1.1f%%', shadow=True, startangle=90)
plt.axis('equal')
plt.title('Proportion of each observed category')

plt.show()
```



4. Initial Findings:

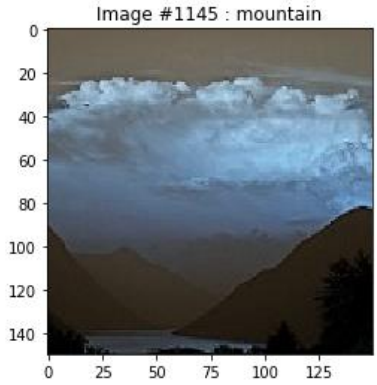
- After preparing the dataset the images are scaled as below:

In [21]:
```
train_images = train_images / 255.0
test_images = test_images / 255.0
```

- Then to display a random image in the training set, the code below conducted. The labels were checked by this for both one single picture and multiple pictures.

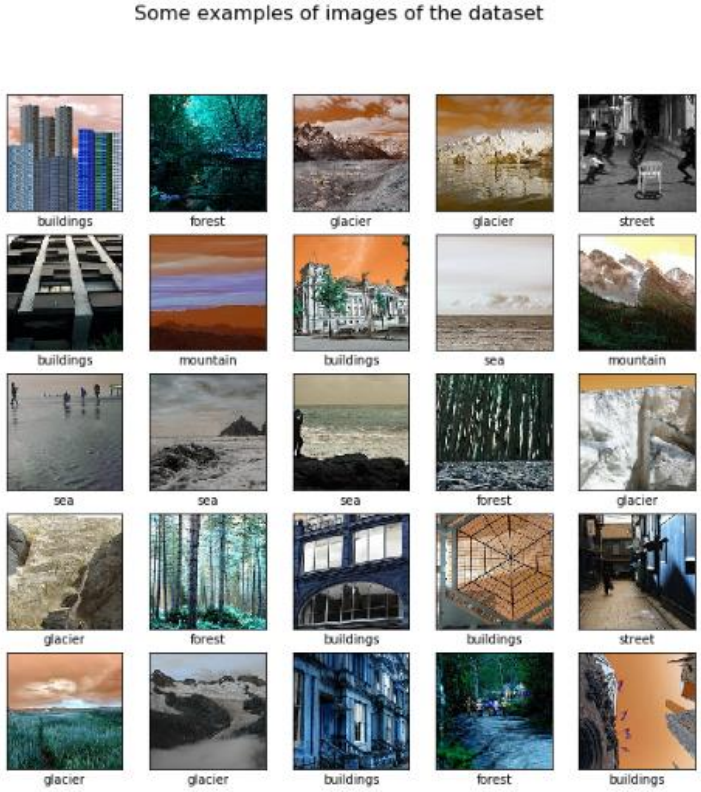
In [22]:

```
index = np.random.randint(train_images.shape[0])
plt.figure()
plt.imshow(train_images[index])
plt.grid(False)
plt.title('Image #{} : '.format(index) + class_names[train_labels[index]])
plt.show()
```



In [23]:

```
fig = plt.figure(figsize=(10,10))
fig.suptitle("Some examples of images of the dataset", fontsize=16)
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i], cmap=plt.cm.binary)
    plt.xlabel(class_names[train_labels[i]])
plt.show()
```



After this stage we will start building the deep learning model by the help of tensorflow Keras.

5.CNN Model:

At this stage, we are ready to create our model. First, we will built the CNN model. And then train and fit the model. After this we will evaluate the model on preprepared test set. At the end we will carry out an error analysis for the model.

We can build an easy model composed of different layers such as:

Conv2D: (32 filters of size 3 by 3) The features will be "extracted" from the image.

MaxPooling2D: The images get half sized.

Flatten: Transforms the format of the images from a 2d-array to a 1d-array of 150 150 3 pixel values.

Relu : given a value x, returns max(x, 0).

Softmax: 6 neurons, probability that the image belongs to one of the classes.

In [47]:
```
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation = 'relu', input_shape = (150, 150, 3)),
    # the nn will learn the good filter to use
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(32, (3, 3), activation = 'relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation=tf.nn.relu),
    tf.keras.layers.Dense(6, activation=tf.nn.softmax)
])
```

Then, we can compile it with some parameters such as:

- **Optimizer**: adam = RMSProp + Momentum. What is Momentum and RMSProp ?
- Momentum = takes into account past gradient to have a better update.
- RMSProp = exponentially weighted average of the squares of past gradients.
- **Loss function**: we use sparse categorical crossentropy for classification, each images belongs to one class only

In [48]:
```
model.compile(optimizer = 'adam', loss = 'sparse_categorical_crossentropy', metrics=['accuracy'])
```

6. Findings:

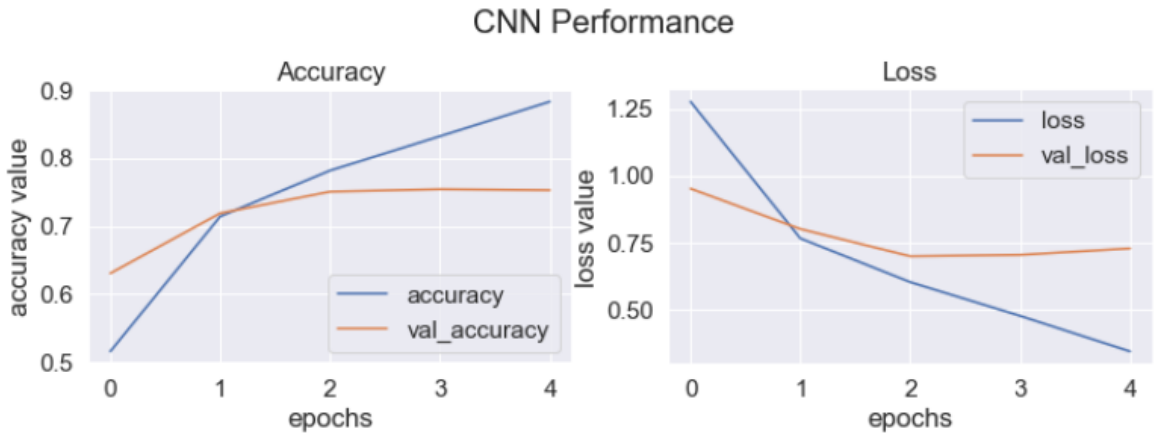
We fit the model to the data from the training set. The neural network will learn by itself the pattern in order to distinguish each category.

```
In [49]: 1 history = model.fit(train_images, train_labels, batch_size=128, epochs=5, validation_split = 0.2)

Train on 11227 samples, validate on 2807 samples
Epoch 1/5
11227/11227 [==============================] - 878s 78ms/sample - loss: 1.2660 - accuracy: 0.5379 - val_loss: 0.9270 - val_accu
racy: 0.6555
Epoch 2/5
11227/11227 [==============================] - 598s 53ms/sample - loss: 0.7819 - accuracy: 0.7066 - val_loss: 0.7659 - val_accu
racy: 0.7086
Epoch 3/5
11227/11227 [==============================] - 440s 39ms/sample - loss: 0.5581 - accuracy: 0.7984 - val_loss: 0.6914 - val_accu
racy: 0.7538
Epoch 4/5
11227/11227 [==============================] - 405s 36ms/sample - loss: 0.4093 - accuracy: 0.8555 - val_loss: 0.6181 - val_accu
racy: 0.7870
Epoch 5/5
11227/11227 [==============================] - 448s 40ms/sample - loss: 0.2754 - accuracy: 0.9066 - val_loss: 0.6704 - val_accu
racy: 0.7898
```

Firstly, accuracy reached to 0.9066 from 0.5379 and the loss end up with 0.2754. We achieved a fairly good accuracy levels.

7. CNN Performance:




```
Accuracy Scores for the Training set : 0.8058524874693831  and Test Set : 0.79218075845886
True
```

```
Accuracy Scores for the Training set : 0.790215575272916  and Test Set : 0.7801074861857543
True
```

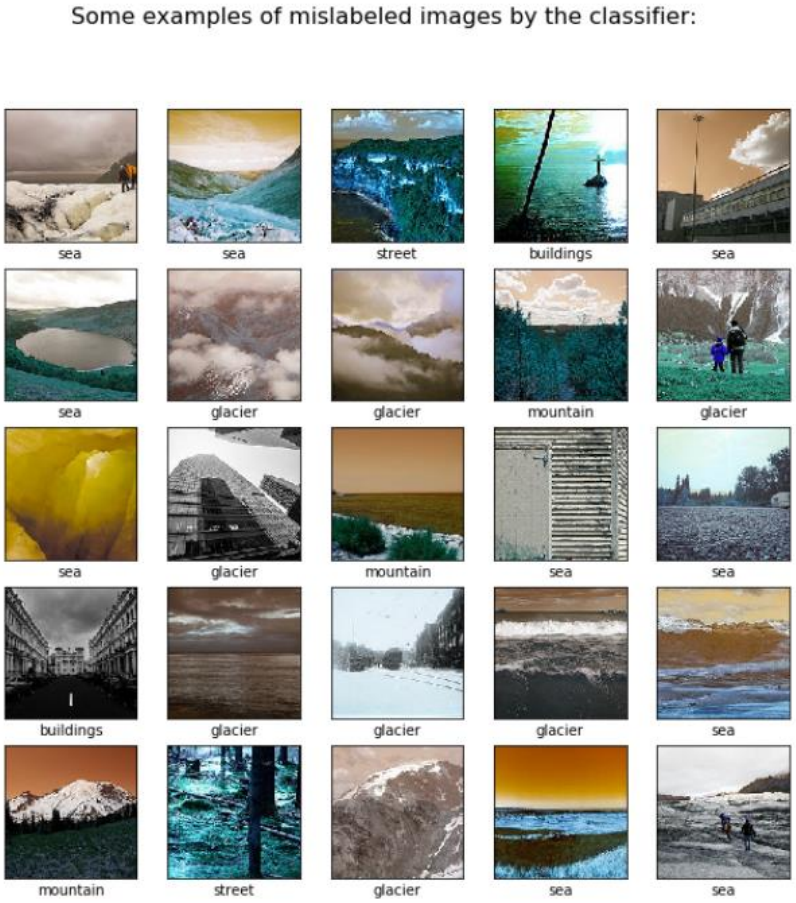
8. Model Performance:

```
In [50]:  1  test_loss = model.evaluate(test_images, test_labels)
```

```
==============================================================
==============================================================
==============================================================
==============================================================
==============================================================
==============================================================
==============================================================
==============================================================
==============================================================
==============================================================
==============================================================
==============================================================
==============================================================
==============================================================
==============================================================
==============================================================
==============================================================
========] - 84s 28ms/sample - loss: 0.6499 - accuracy: 0.7823
```

As a result, the model is a success. The performance of the model on the test set is almost 80%.

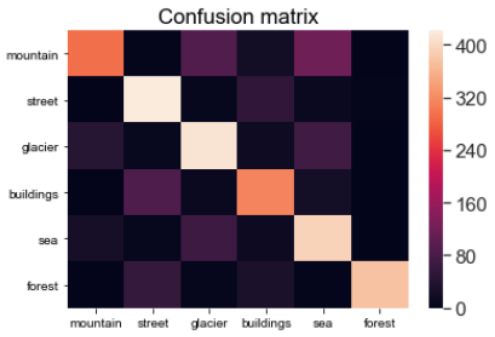
9. Random Mislabelled Images:



10. Confusion Matrix:

Confusion Matrix

In [33]:
```
CM = confusion_matrix(test_labels, pred_labels)
ax = plt.axes()
sn.set(font_scale=1.4)
sn.heatmap(CM, annot=False,annot_kws={"size": 16},  xticklabels=class_names, yticklabels=class_names, ax = ax)
ax.set_title('Confusion matrix')
plt.show()
```



As a result; our model created by a deep learning model module. Tensorflow, Keras used to build the model. An image data set under the classification of 6 used in the model. After training the model, test set successfully predicted with a rate of 80%. At the end, we gave a set of mislabelled data and a confusion matrix where the models performance can be assessed visually by classifications.

In order to enhance this model, image augmentation and transfer learning can be used and we can think about adding more layers to the model.