Intel Image Classification

Introduction:

When machine learning and image classification get integrated, computers become capable of performing visual tasks that until recently could only be carried out by humans. Together, these technologies offer the potential for breakthroughs in automation, presenting new digital opportunities for companies in a variety of domains.

How do we, humans, recognize a forest as a forest or a mountain as a mountain? We are very good at categorizing scenes based on the semantic representation and object affinity, but we know very little about the processing and encoding of natural scene categories in the human brain. In this problem, we have a dataset of ~25k images from a wide range of natural scenes from all around the world.

Identifying natural scenes from all around the world is an interesting image classification problem. We are going to classify six different category images:

- Buildings
- Forest
- Glacier
- Mountain
- Sea
- Street

Problem at hand:

Our task is to identify which kind of scene can the image be categorized into.

Value to client:

Image recognition refers to technologies that identify places, logos, people, objects, buildings, and several other variables in images. Users are sharing vast amounts of data through apps, social media and websites. Additionally, mobile phones equipped with cameras are leading to the creation of limitless digital images and videos. The large volume of digital data is being used by companies to deliver better and smarter services to the people accessing it.

From the business perspective, major applications of image recognition are face recognition, security, surveillance, visual geo-location, object recognition, gesture recognition, code

recognition, industrial automation, image analysis in medical and driver assistance. These applications are creating growth opportunities in many fields.

By analyzing images of people, places, objects, scenes, and documents, machine learning for image classification promises new levels of automation in just about every industry. Healthcare, insurance, automotive, manufacturing, and financial services are among the industries in which automated image classification is most prolific. However, in reality it is likely that many more industries will be impacted by the technology as it matures.

Data Source:

www.kaggle.com/puneet6060/intel-image-classification

Methodology:

Building and training a Convolutional Neural Network that can classify the above mentioned categories of images correctly.

Data Wrangling

The data was downloaded from Kaggle.

The dataset contains around 25k images of size 150 x 150 distributed under 6 categories. The class names along with their indices are as follows:

- **buildings → 0**
- **forest → 1**
- **glacier → 2**
- **mountain → 3**
- **sea → 4**
- **street → 5**

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.

The directory structure is as follows:

Train set-

seg_train/
...seg_train/
.....buildings/
.....forest/
.....glacier/
.....mountain/
.....sea/
.....street/

Test set-

seg_test/
...seg_test/
.....buildings/
.....forest/
.....glacier/
.....mountain/
.....sea/
.....street/

Prediction set-

seg_pred/
...seg_pred/

Data Pre-processing:

We utilize Keras's **"ImageDataGenerator"** to perform image augmentation. **"ImageDataGenerator"** takes the path to a directory & generates batches of augmented data.

Our dataset has images placed under folders which has their respective class names. We use the **"flow_from_directory"** method that helps us to read the images from the folders and also determine the number of classes along with their names.

By performing the above method we find:

- Train set contains 14034 images belonging to 6 classes.
- Test set contains 3000 images belonging to 6 classes.

Exploratory Data Analysis

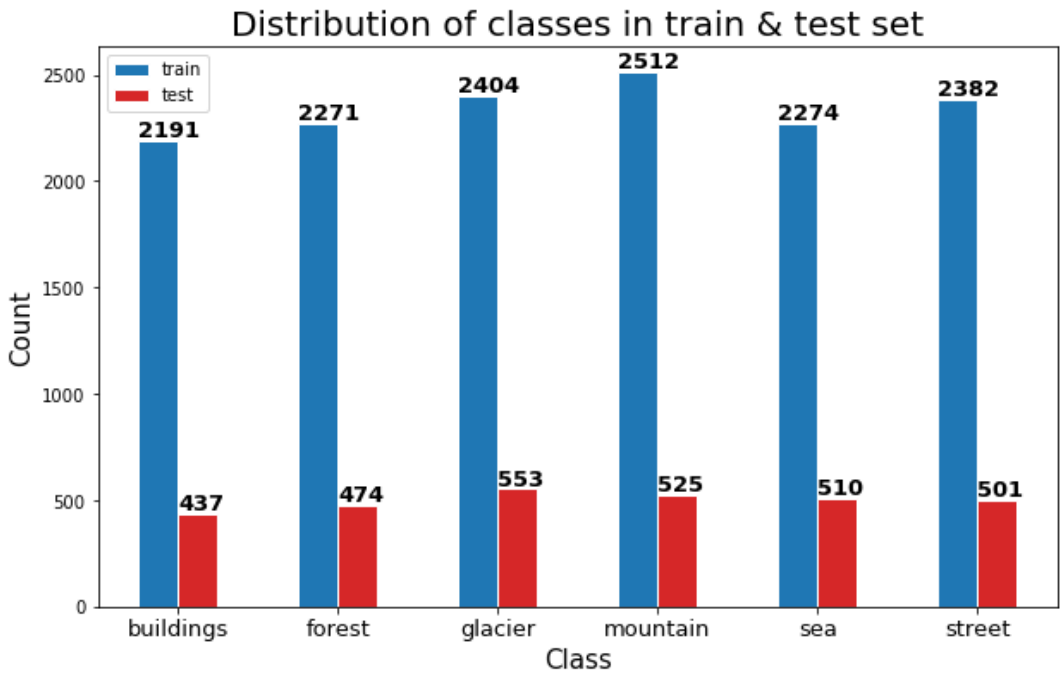
Let's explore the dataset here.

We can ask ourselves:

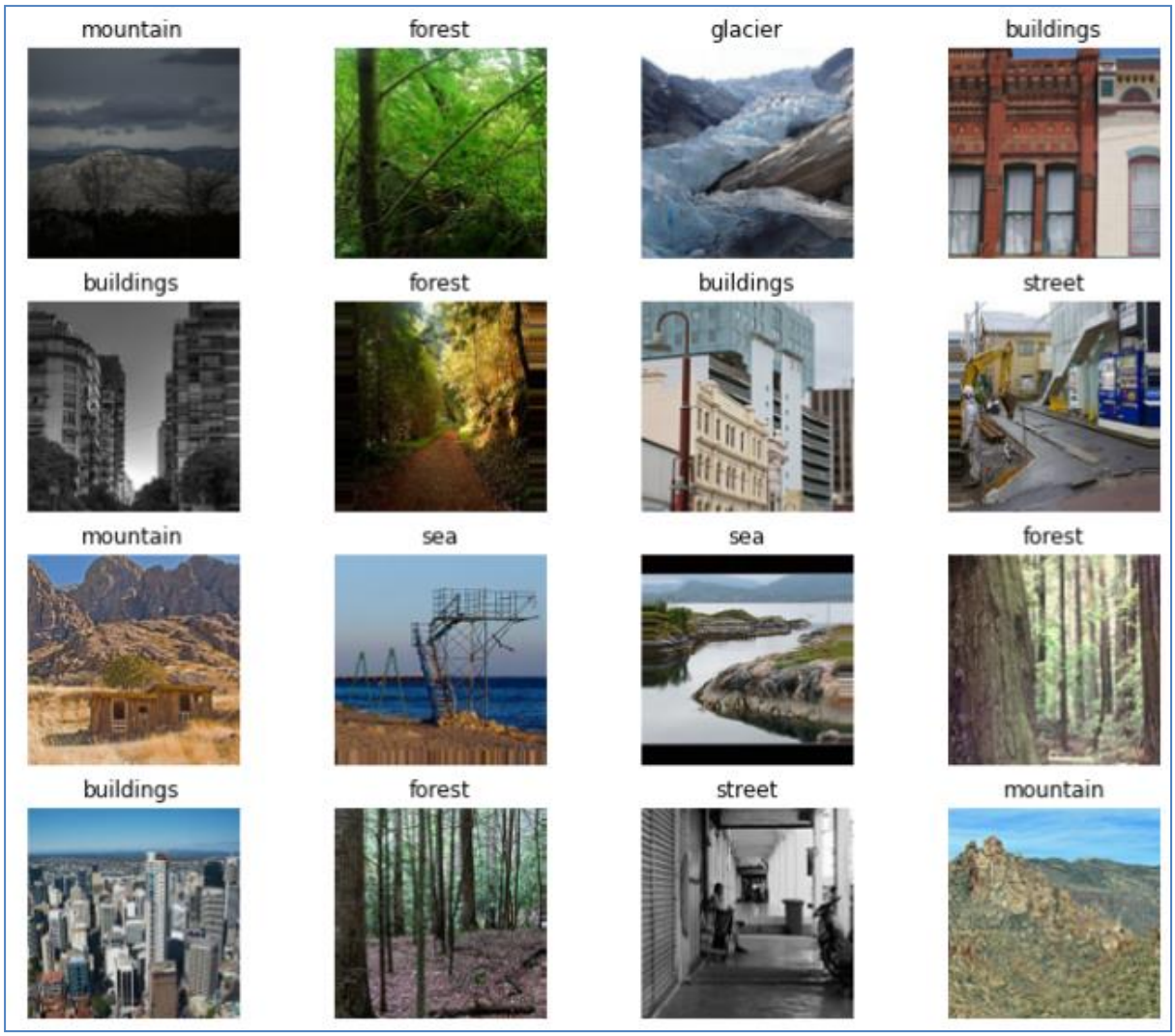
- **How many training and testing examples do we have?**
  - We already know the number of images our dataset contains.
    Number of training images: 14034
    Number of testing images: 3000

- **What is the shape of the images?**
  - Shape of training images: 150 x 150 x 3
    Shape of testing images: 150 x 150 x 3

- **What is the distribution of the images in 6 different classes?**
  - Let's visualize the distribution of the images in 6 classes in both the training and test set.



Let's see some of the images along with the class names from the train set:



Here we can see the 6 classes distinctly classified as the images are already labelled in the train set. At the end we will see how our CNN model predicts the classes for unseen and unlabelled images from the prediction set.

Convolutional Neural Network

In deep learning, a Convolutional Neural Network (CNN, or ConvNet) is a class of deep neural networks, most commonly applied to analyzing visual imagery.

A CNN takes an input image, process it and classify it under certain categories. The input image is a 3D Matrix. Computers sees an input image as array of pixels and it depends on the image resolution. Based on the image resolution, it will see h x w x d (h = Height, w = Width, d = Dimension).

In CNN each input image will pass through a series of convolution layers with filters (kernels), pooling layers, fully connected layers and apply softmax function to classify an object with probabilistic values between 0 and 1.

Model Building:

Now we create a CNN model to predict the class labels. This is the basic CNN model.

The steps are:

1. Build the model.
2. Compile the model.
3. Train / fit the data to the model.

We build a simple model composed of different layers such as:

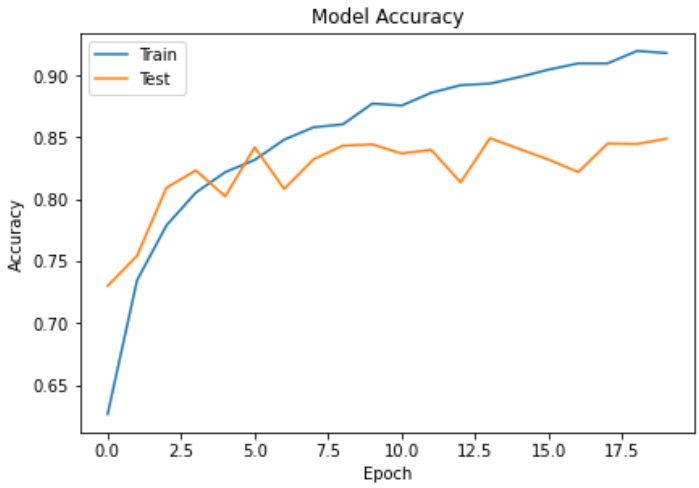
- **Conv2D:** (32 filters of size 3 by 3) The features will be extracted from the image.
- **MaxPooling2D:** The images get half sized, i.e. the images will have fewer pixels.
- **Flatten:** Transforms the format of the images from a 2D-array to a 1D-array of 150 x 150 x 3 pixel values.
- **Dense:** A.K.A. fully connected layers, take the high-level filtered images and translate them into votes. Fully Connected Layer is simply a feed forward neural network.
- **ReLU:** Given a value x, returns max(x, 0), i.e. all the negative values removed.
- **Softmax**: 6 neurons, probability that the image belongs to one of the classes.

Training:

We fit the CNN model to the data from the training set and use the test set as the validation data. We train it for 20 epochs. The neural network will learn by itself the pattern in order to distinguish each category.

Let's visualize the model accuracy and loss during the training of the neural network.

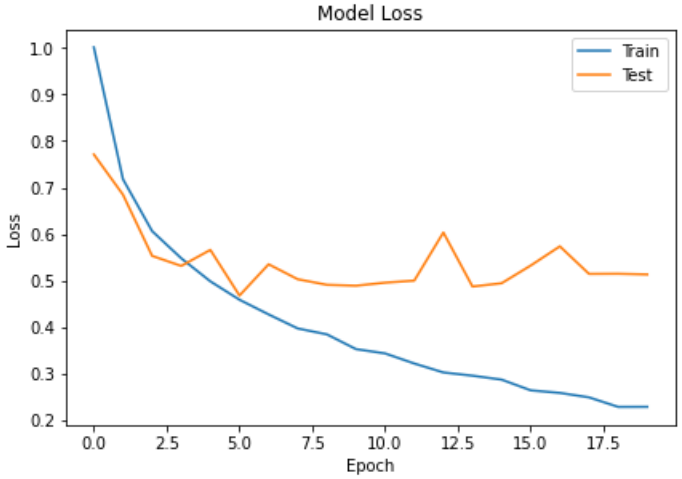
- **Model Accuracy:**



After training for 20 epochs we find:

- **Train accuracy = 92%**
- **Test accuracy = 85%**

- **Model Loss:**



After training for 20 epochs we find:

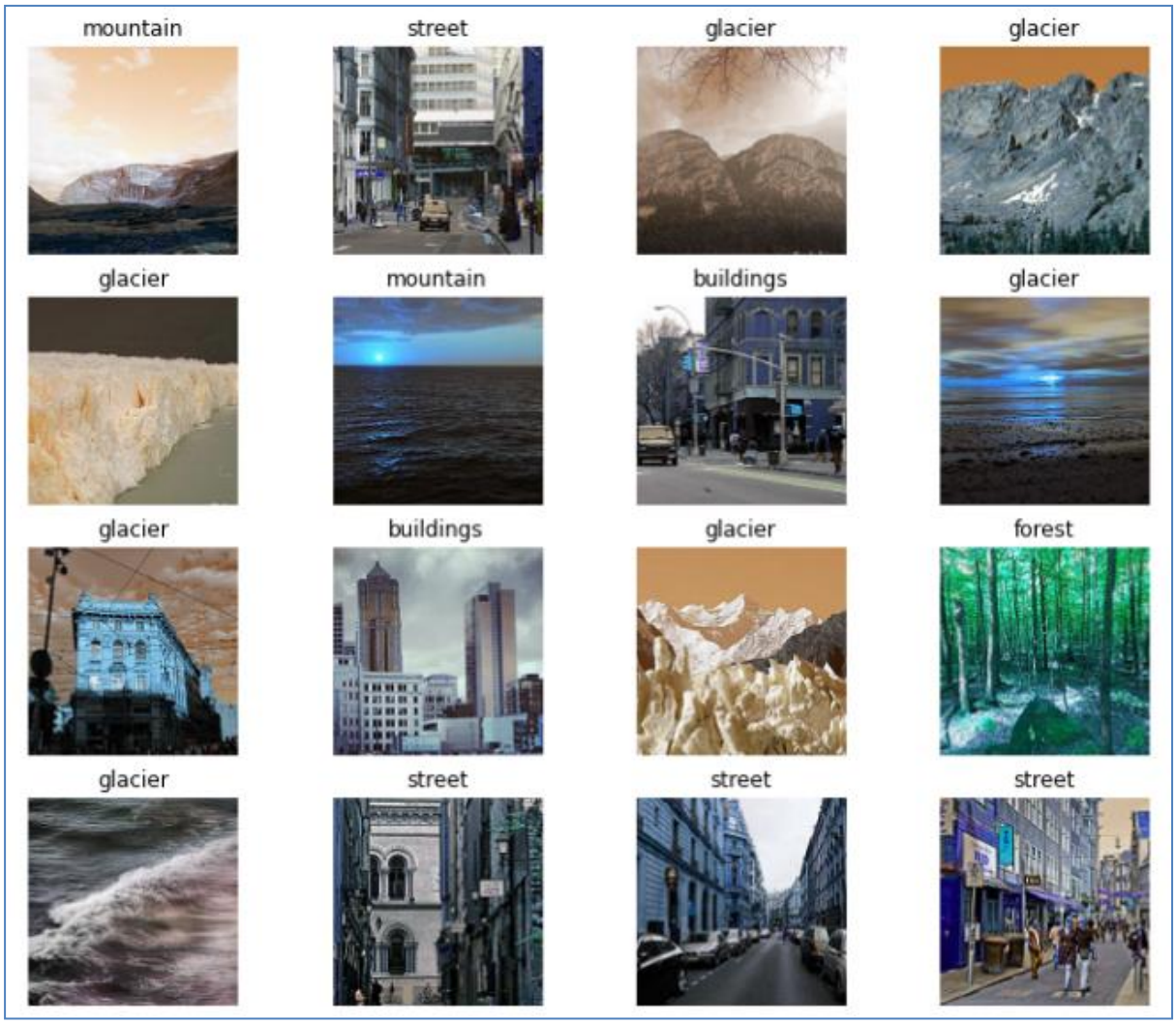
- **Train loss = 23%**
- **Test loss = 51%**

Prediction:

We use the prediction set for predicting the class of the images.

- Number of prediction images: 7301
- Shape of prediction images: 150 x 150 x 3

Let's see how our model predicts the image class by observing some of the images along with their predicted class names.



As we can observe our model predicts most of the images correctly. However, as seen above the model accuracy is 85%. This means that around 15% of the unseen images will get an incorrect prediction.

Model Improvement:

Now we'll try to improve the performance of our base model.

NOTE:

- **Due to less computational power of my machine I will not be training the following models here.**
- **However I would like to run the models on the cloud or on a machine with more computational power.**
- **It's just the summary of the models whose accuracy might improve than the base model.**

Model 2:

Let's try to improve our base model by:

- Adding a few extra convolution layers.
- Adding high-level (256 filters) convolution layer.
- Adding 2 dropout layers with a rate of 0.5.
- Adding one more fully connected layer.

Model summary:

```
model = Sequential()

model.add(Conv2D(32, kernel_size=(3,3), padding='same', activation='relu', input_shape=(150,150,3)))
model.add(Conv2D(32, kernel_size=(3,3), padding='same', activation='relu'))
model.add(MaxPool2D(pool_size=(2, 2)))

model.add(Conv2D(64, kernel_size=(3,3), padding='same', activation='relu'))
model.add(Conv2D(64, kernel_size=(3,3), padding='same', activation='relu'))
model.add(MaxPool2D(pool_size=(2, 2)))

model.add(Conv2D(128, kernel_size=(3,3), padding='same', activation='relu'))
model.add(Conv2D(128, kernel_size=(3,3), padding='same', activation='relu'))
model.add(MaxPool2D(pool_size=(2, 2)))

model.add(Conv2D(256, kernel_size=(3,3), padding='same', activation='relu'))
model.add(Conv2D(256, kernel_size=(3,3), padding='same', activation='relu'))
model.add(MaxPool2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(6, activation='softmax'))

model.compile(optimizer='Adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

VGG16:

We will use the pre-trained VGG16 model here. This model was trained on ImageNet dataset for 1000 classes. Features learned by the initial layers of VGG16 model are generic enough to apply for other image classification tasks.

We load the VGG16 model's convolutional base and make the parameter *"include_top=False"*, which lets us drop the 3 fully-connected layers at the top of the network. We also freeze all the convolutional VGG16 layers, so as to avoid destroying any of the information they contain during future training rounds.

```
vgg16 = VGG16(include_top=False, weights='imagenet', input_shape=(150, 150, 3))

for layer in vgg16.layers:
    layer.trainable = False

vgg16.summary()
```

```
Model: "vgg16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 150, 150, 3)]     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 150, 150, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 150, 150, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 75, 75, 64)        0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 75, 75, 128)       73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 75, 75, 128)       147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 37, 37, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 37, 37, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 37, 37, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 37, 37, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 18, 18, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 18, 18, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 18, 18, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 18, 18, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 9, 9, 512)         0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 9, 9, 512)         2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 9, 9, 512)         2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 9, 9, 512)         2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 4, 4, 512)         0
```

Then we define our model with its own classifier on top of VGG16 convolutional base.

```
model = Sequential()

model.add(vgg16)

model.add(Flatten())
model.add(Dense(250, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(6, activation='softmax'))

model.compile(optimizer='Adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

Conclusion

The CNN model we built here is a simple one with only a few layers - 2 convolution layers with relu activation each, 2 pooling layers, a flatten layer and 2 fully connected layers with relu and softmax activation.

We got an accuracy of 85% and a loss of 51% on the test set. An accuracy of 85% is not the best, but for a simple model it is good.

However, we can improve the accuracy of the model in different ways like mentioned in the model improvement section. But due to less computational power I choose to build this simple CNN model.

Recommendations to improve the model performance:

- Add more layers.
- Train the model with more epochs.
- Use transfer learning.
- Change kernel sizes.
- Use early stopping.

Scope for future work:

- More types of class can be added.
- Collect more data.
- Error analysis.