5505 Assignment 6: Image Classification

Shiva Chakravarthy Gollapudi
Student ID: 11468697

Image classification is known as identifying and labeling groups of pixels or vectors inside based on certain rules.

Data: https://www.kaggle.com/puneet6060/intel-image-classification/data#

Tools and Environment: Using python programming and Google Colab, I have explored the data.

Step 1: Load Libraries

Import the libraries that we use to load, explore data and build the model.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.image import imread
from pathlib import Path
import cv2
from random import randint

import os
import glob as gb
```

Unzip the data files

```
[ ] !unzip /content/seg_pred.zip
    !unzip /content/seg_test.zip
    !unzip /content/seg_train.zip
```

```
[ ] train_dir = '/content/seg_train/seg_train'
    test_dir = '/content/seg_test/seg_test'
    pred_dir = '/content/seg_pred/seg_pred'
```

Data description: This Data contains around 25k images of size 150x150 distributed under 6 categories.

Output:

Below are the number of images found in each folder

```
Test Data Directory:
 501 images are found in street folder.
 525 images are found in mountain folder.
 510 images are found in sea folder.
 553 images are found in glacier folder.
 474 images are found in forest folder.
 437 images are found in buildings folder.
```

```
----------------------------------------------------------------------
Training Data Directory:
2382 images are found in street folder.
2512 images are found in mountain folder.
2274 images are found in sea folder.
2404 images are found in glacier folder.
2271 images are found in forest folder.
2191 images are found in buildings folder.
----------------------------------------------------------------------
Prediction Data Directory:
7301 images are found in predict data.
```

Step 2: Visualizing the test and train dataset images

a) Shape of the files:

```
print('Shape of train dataset Images:', train_images.shape)
print('Shape of train dataset Classes:', train_classes.shape)
print('Shape of test dataset Images:', test_images.shape)
print('Shape of test dataset Classes:', test_classes.shape)
```

```
Shape of train dataset Images: (14034, 150, 150, 3)
Shape of train dataset Classes: (14034,)
Shape of test dataset Images: (3000, 150, 150, 3)
Shape of test dataset Classes: (3000,)
```

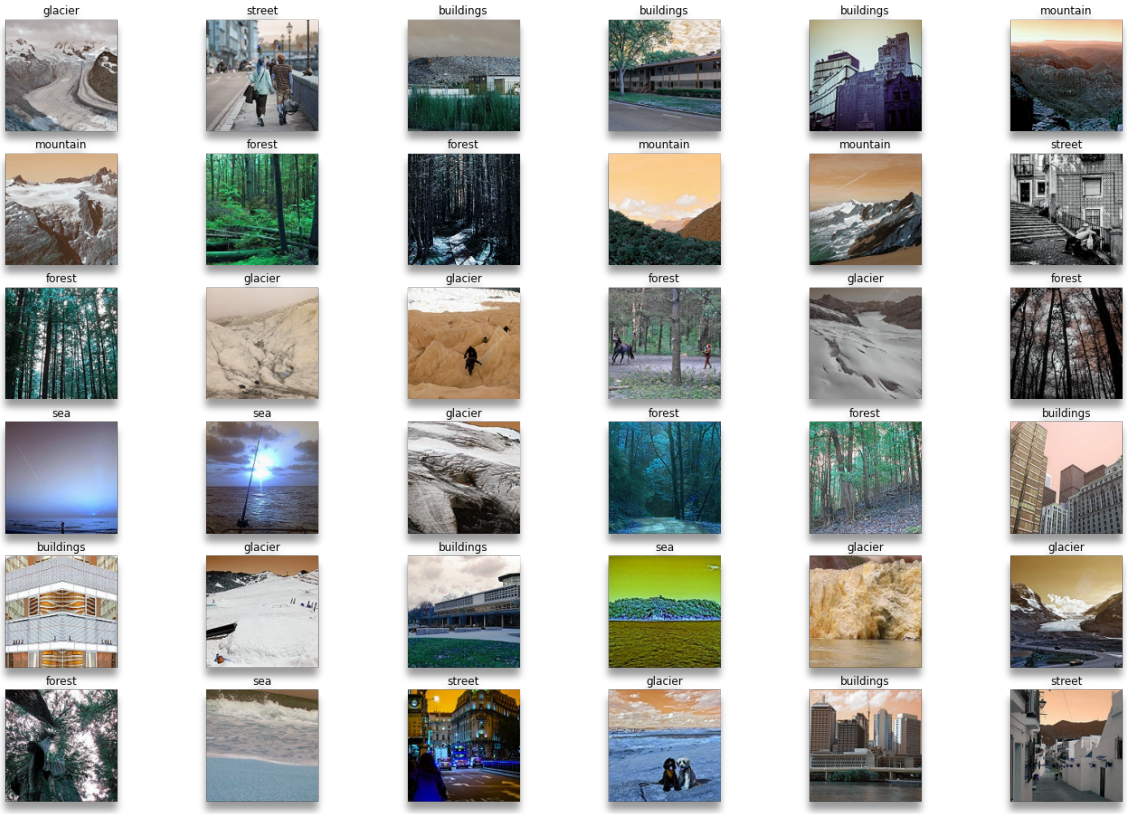
b) Displaying the train dataset images:

```
#Displaying train dataset images
plt.subplots_adjust(0,0,3,3)
for i in range(0,6*6):
  random_number = randint(0, len(train_images))
  img = train_images[random_number]
  plt.subplot(6,6,i+1)
  plt.title(train_classes[random_number])
  plt.axis('off')
  plt.imshow(img)

plt.show();
```

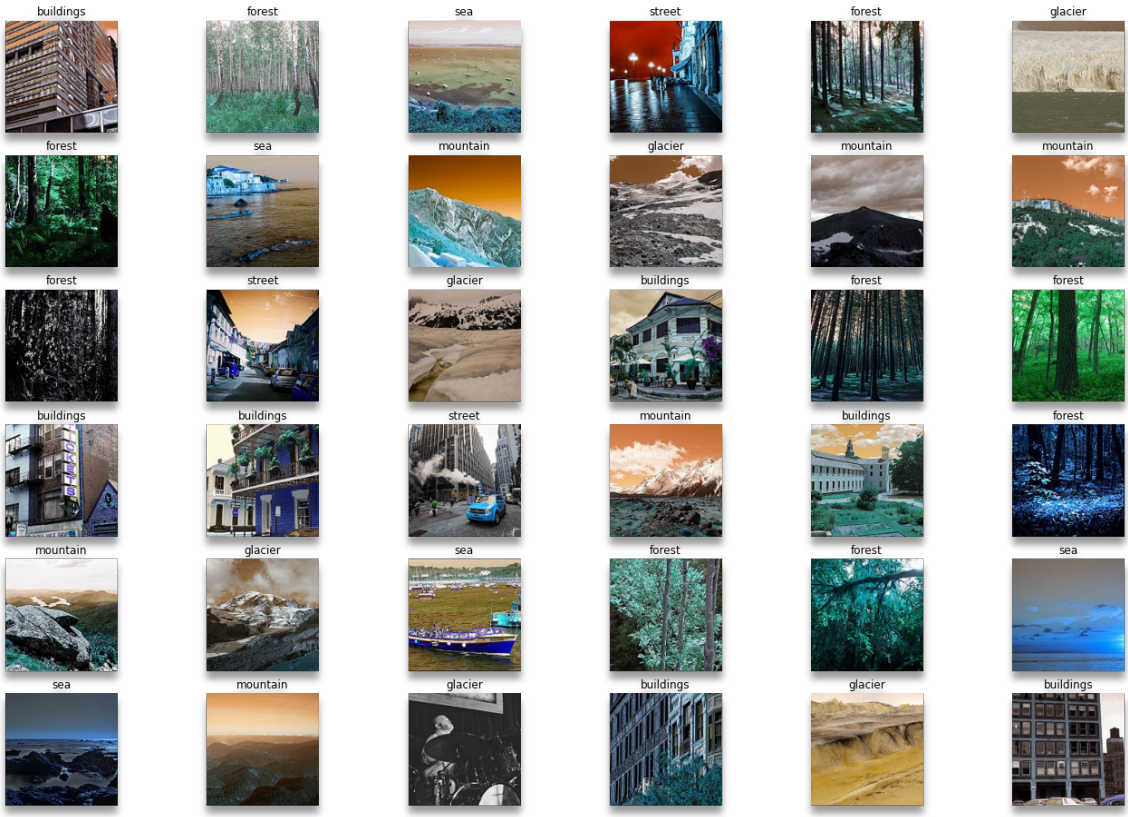
b) Displaying the test dataset images:

```
#Displaying test dataset images
plt.subplots_adjust(0,0,3,3)
for i in range(0,6*6):
  random_number = randint(0, len(test_images))
  img = test_images[random_number]
  plt.subplot(6,6,i+1)
  plt.title(test_classes[random_number])
  plt.axis('off')
  plt.imshow(img)

plt.show();
```

Train Output:



Test Output:



Step 3: Develop CNN model:

A CNN is a neural network with some convolutional layers (and some other layers). A convolutional layer has a number of filters that does convolutional operation.

A Sequential CNN model is appropriate for a plain stack of layers where each layer has exactly one input tensor and one output tensor.

In this I have used pooling function (MaxPool2D), Activation function (ReLU) and filter function (Conv2D).

Pooling function: Its function is to progressively reduce the spatial size of the representation to reduce the number of parameters and computation in the network. Pooling layer operates on each feature map independently.

```
# Developing CNN model for Image classification
model=Models.Sequential()

model.add(Layers.Conv2D(200,kernel_size=(3,3),activation='relu',input_shape=(150,150,3)))
model.add(Layers.Conv2D(180,kernel_size=(3,3),activation='relu'))
model.add(Layers.MaxPool2D(5,5))
model.add(Layers.Conv2D(180,kernel_size=(3,3),activation='relu'))
model.add(Layers.Conv2D(140,kernel_size=(3,3),activation='relu'))
model.add(Layers.Conv2D(100,kernel_size=(3,3),activation='relu'))
model.add(Layers.Conv2D(50,kernel_size=(3,3),activation='relu'))
model.add(Layers.MaxPool2D(5,5))
model.add(Layers.Flatten())
model.add(Layers.Dense(180,activation='relu'))
model.add(Layers.Dense(100,activation='relu'))
model.add(Layers.Dense(50,activation='relu'))
model.add(Layers.Dropout(rate=0.5))
model.add(Layers.Dense(6,activation='softmax'))

model.compile(optimizer=Optimizer.Adam(learning_rate=0.0001),loss='sparse_categorical_crossentropy',metrics=['accuracy'])
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 148, 148, 200)     5600

 conv2d_1 (Conv2D)           (None, 146, 146, 180)     324180

 max_pooling2d (MaxPooling2D  (None, 29, 29, 180)      0
 )

 conv2d_2 (Conv2D)           (None, 27, 27, 180)       291780

 conv2d_3 (Conv2D)           (None, 25, 25, 140)       226940

 conv2d_4 (Conv2D)           (None, 23, 23, 100)       126100

 conv2d_5 (Conv2D)           (None, 21, 21, 50)        45050

 max_pooling2d_1 (MaxPooling  (None, 4, 4, 50)         0
 2D)

 flatten (Flatten)           (None, 800)               0

 dense (Dense)               (None, 180)               144180
```

```
conv2d_5 (Conv2D)            (None, 21, 21, 50)        45050

max_pooling2d_1 (MaxPooling  (None, 4, 4, 50)          0
2D)

flatten (Flatten)            (None, 800)               0

dense (Dense)                (None, 180)               144180

dense_1 (Dense)              (None, 100)               18100

dense_2 (Dense)              (None, 50)                5050

dropout (Dropout)            (None, 50)                0

dense_3 (Dense)              (None, 6)                 306

=================================================================
Total params: 1,187,286
Trainable params: 1,187,286
Non-trainable params: 0
_________________________________________________________________
```

Step 4: Validate CNN Model with 10-Fold cross validation

In K-fold cv, training data is further split into K number of subsets, called folds, then iteratively fit the model k times, each time training the data on k-1 of the folds and evaluating on the kth fold. At the end, we average the performance on each of the folds to come up with final validation metrics for the model.

```
#set early stopping criteria
pat = 2 #this is the number of epochs with no improvment after which the training will stop
early_stopping = EarlyStopping(monitor='val_loss', patience=pat, verbose=1)

#define the model checkpoint callback -> this will keep on saving the model as a physical file
model_checkpoint = ModelCheckpoint('fas_mnist_1.h5', verbose=1, save_best_only=True)


#Function for cross validation

model_history = []
for i in range (0,10):
  print("Training on Fold: ",i+1)
  model_history.append(model.fit(train_images, train_labels, epochs=20, callbacks=[early_stopping],
              verbose=1, validation_split=0.1))
  print("=======*12, end="\n\n\n")
  print(model_history)
```

Output:

```
Training on Fold:  1
Epoch 1/20
395/395 [==============================] - 204s 515ms/step - loss: 1.0056 - accuracy: 0.6220 - val_loss: 1.3063 - val_accuracy: 0.3568
Epoch 2/20
395/395 [==============================] - 204s 516ms/step - loss: 0.8487 - accuracy: 0.7008 - val_loss: 1.1808 - val_accuracy: 0.5285
Epoch 3/20
395/395 [==============================] - 204s 516ms/step - loss: 0.7664 - accuracy: 0.7367 - val_loss: 0.7171 - val_accuracy: 0.6738
Epoch 4/20
395/395 [==============================] - 204s 516ms/step - loss: 0.6903 - accuracy: 0.7697 - val_loss: 1.3389 - val_accuracy: 0.4979
Epoch 5/20
395/395 [==============================] - 203s 515ms/step - loss: 0.6368 - accuracy: 0.7894 - val_loss: 1.0047 - val_accuracy: 0.5819
Epoch 00005: early stopping
================================================================================


[<keras.callbacks.History object at 0x7fdef2f23250>]
Training on Fold:  2
Epoch 1/20
395/395 [==============================] - 204s 516ms/step - loss: 0.5864 - accuracy: 0.8057 - val_loss: 0.9748 - val_accuracy: 0.6261
Epoch 2/20
395/395 [==============================] - 204s 516ms/step - loss: 0.5473 - accuracy: 0.8196 - val_loss: 0.9236 - val_accuracy: 0.6553
Epoch 3/20
395/395 [==============================] - 204s 516ms/step - loss: 0.5103 - accuracy: 0.8311 - val_loss: 0.9643 - val_accuracy: 0.6225
Epoch 4/20
395/395 [==============================] - 204s 516ms/step - loss: 0.4789 - accuracy: 0.8451 - val_loss: 1.0978 - val_accuracy: 0.5798
Epoch 00004: early stopping
================================================================================


[<keras.callbacks.History object at 0x7fdef2f23250>, <keras.callbacks.History object at 0x7fdef31649d0>]
Training on Fold:  3
Epoch 1/20
395/395 [==============================] - 204s 516ms/step - loss: 0.4549 - accuracy: 0.8481 - val_loss: 0.9556 - val_accuracy: 0.6232
Epoch 2/20
395/395 [==============================] - 204s 515ms/step - loss: 0.4297 - accuracy: 0.8595 - val_loss: 0.6323 - val_accuracy: 0.7578
Epoch 3/20
395/395 [==============================] - 204s 516ms/step - loss: 0.4070 - accuracy: 0.8633 - val_loss: 0.9274 - val_accuracy: 0.6895
Epoch 4/20
395/395 [==============================] - 205s 518ms/step - loss: 0.3758 - accuracy: 0.8762 - val_loss: 1.0910 - val_accuracy: 0.6567
Epoch 00004: early stopping
================================================================================


[<keras.callbacks.History object at 0x7fdef2f23250>, <keras.callbacks.History object at 0x7fdef31649d0>, <keras.callbacks.History objec
Training on Fold:  4
Epoch 1/20
395/395 [==============================] - 205s 518ms/step - loss: 0.3465 - accuracy: 0.8842 - val_loss: 0.8473 - val_accuracy: 0.7101
Epoch 2/20
395/395 [==============================] - 204s 517ms/step - loss: 0.3168 - accuracy: 0.8942 - val_loss: 0.9350 - val_accuracy: 0.7265
Epoch 3/20
395/395 [==============================] - 205s 519ms/step - loss: 0.2923 - accuracy: 0.9040 - val_loss: 0.7685 - val_accuracy: 0.7621
Epoch 4/20
395/395 [==============================] - 204s 518ms/step - loss: 0.2665 - accuracy: 0.9105 - val_loss: 0.8569 - val_accuracy: 0.7543
Epoch 5/20
395/395 [==============================] - 204s 517ms/step - loss: 0.2634 - accuracy: 0.9162 - val_loss: 1.0367 - val_accuracy: 0.7507
Epoch 00005: early stopping
================================================================================


Training on Fold:  5
Epoch 1/20
395/395 [==============================] - 205s 519ms/step - loss: 0.2196 - accuracy: 0.9251 - val_loss: 1.7588 - val_accuracy: 0.6239
Epoch 2/20
395/395 [==============================] - 204s 518ms/step - loss: 0.2059 - accuracy: 0.9317 - val_loss: 1.2195 - val_accuracy: 0.7500
Epoch 3/20
395/395 [==============================] - 204s 517ms/step - loss: 0.1876 - accuracy: 0.9407 - val_loss: 1.6411 - val_accuracy: 0.6360
Epoch 4/20
395/395 [==============================] - 204s 517ms/step - loss: 0.1888 - accuracy: 0.9382 - val_loss: 1.3042 - val_accuracy: 0.7486
Epoch 00004: early stopping
================================================================================


[<keras.callbacks.History object at 0x7fdef2f23250>, <keras.callbacks.History object at 0x7fdef31649d0>, <keras.callbacks.History objec
Training on Fold:  6
Epoch 1/20
395/395 [==============================] - 205s 518ms/step - loss: 0.1596 - accuracy: 0.9464 - val_loss: 1.3270 - val_accuracy: 0.7386
Epoch 2/20
395/395 [==============================] - 204s 517ms/step - loss: 0.1540 - accuracy: 0.9497 - val_loss: 2.5162 - val_accuracy: 0.6410
Epoch 3/20
395/395 [==============================] - 204s 517ms/step - loss: 0.1549 - accuracy: 0.9489 - val_loss: 1.3037 - val_accuracy: 0.7400
Epoch 4/20
395/395 [==============================] - 204s 517ms/step - loss: 0.1461 - accuracy: 0.9525 - val_loss: 1.2681 - val_accuracy: 0.7293
Epoch 5/20
395/395 [==============================] - 204s 518ms/step - loss: 0.1195 - accuracy: 0.9613 - val_loss: 0.9088 - val_accuracy: 0.8070
Epoch 6/20
395/395 [==============================] - 204s 517ms/step - loss: 0.1052 - accuracy: 0.9670 - val_loss: 1.7976 - val_accuracy: 0.7358
Epoch 7/20
```

```
Training on Fold:  7
Epoch 1/20
395/395 [==============================] - 372s 942ms/step - loss: 0.1080 - accuracy: 0.9641 - val_loss: 1.3601 - val_accuracy: 0.7999
Epoch 2/20
395/395 [==============================] - 204s 517ms/step - loss: 0.1008 - accuracy: 0.9700 - val_loss: 1.5843 - val_accuracy: 0.7557
Epoch 3/20
395/395 [==============================] - 204s 518ms/step - loss: 0.0847 - accuracy: 0.9748 - val_loss: 1.3901 - val_accuracy: 0.7571
Epoch 00003: early stopping
================================================================================


[<keras.callbacks.History object at 0x7fdef2f23250>, <keras.callbacks.History object at 0x7fdef31649d0>, <keras.callbacks.History objec
Training on Fold:  8
Epoch 1/20
395/395 [==============================] - 205s 518ms/step - loss: 0.0844 - accuracy: 0.9750 - val_loss: 1.5144 - val_accuracy: 0.7557
Epoch 2/20
395/395 [==============================] - 204s 518ms/step - loss: 0.1063 - accuracy: 0.9684 - val_loss: 1.1118 - val_accuracy: 0.8276
Epoch 3/20
395/395 [==============================] - 204s 518ms/step - loss: 0.0664 - accuracy: 0.9800 - val_loss: 2.7539 - val_accuracy: 0.6453
Epoch 4/20
395/395 [==============================] - 204s 517ms/step - loss: 0.1000 - accuracy: 0.9710 - val_loss: 1.6133 - val_accuracy: 0.7443
Epoch 00004: early stopping
================================================================================
Training on Fold:  9
Epoch 1/20
395/395 [==============================] - 371s 938ms/step - loss: 0.0813 - accuracy: 0.9752 - val_loss: 2.2952 - val_accuracy: 0.6624
Epoch 2/20
395/395 [==============================] - 370s 936ms/step - loss: 0.0600 - accuracy: 0.9815 - val_loss: 2.8889 - val_accuracy: 0.6510
Epoch 3/20
395/395 [==============================] - 370s 936ms/step - loss: 0.0765 - accuracy: 0.9773 - val_loss: 1.5661 - val_accuracy: 0.7721
Epoch 4/20
395/395 [==============================] - 369s 934ms/step - loss: 0.0738 - accuracy: 0.9774 - val_loss: 1.7675 - val_accuracy: 0.7514
Epoch 5/20
395/395 [==============================] - 369s 935ms/step - loss: 0.0543 - accuracy: 0.9832 - val_loss: 2.4334 - val_accuracy: 0.6852
Epoch 00005: early stopping
================================================================================


[<keras.callbacks.History object at 0x7fdef2f23250>, <keras.callbacks.History object at 0x7fdef31649d0>, <keras.callbacks.History object
Training on Fold:  10
Epoch 1/20
395/395 [==============================] - 369s 934ms/step - loss: 0.0803 - accuracy: 0.9766 - val_loss: 2.3652 - val_accuracy: 0.6688
Epoch 2/20
395/395 [==============================] - 369s 934ms/step - loss: 0.0643 - accuracy: 0.9814 - val_loss: 2.0890 - val_accuracy: 0.6788
Epoch 3/20
395/395 [==============================] - 369s 935ms/step - loss: 0.0565 - accuracy: 0.9831 - val_loss: 1.6099 - val_accuracy: 0.7557
Epoch 4/20
395/395 [==============================] - 369s 934ms/step - loss: 0.0547 - accuracy: 0.9853 - val_loss: 1.6214 - val_accuracy: 0.7949
Epoch 5/20
395/395 [==============================] - 365s 925ms/step - loss: 0.0763 - accuracy: 0.9762 - val_loss: 1.6232 - val_accuracy: 0.7087
Epoch 00005: early stopping
================================================================================
```

```
plt.figure(figsize=(15,10))
plt.title('Accuracy vs Epochs')
# plot_labels = 'Training Fold ' + i+1
for i in range(0,9):
  plt.plot(model_history[i].history['accuracy'], label='Training Fold '+str(i+1))

plt.legend()
plt.show()
```

When validated the model with test data, we got the 83.53% accuracy.

```
model.evaluate(test_images, test_labels)
model.
```

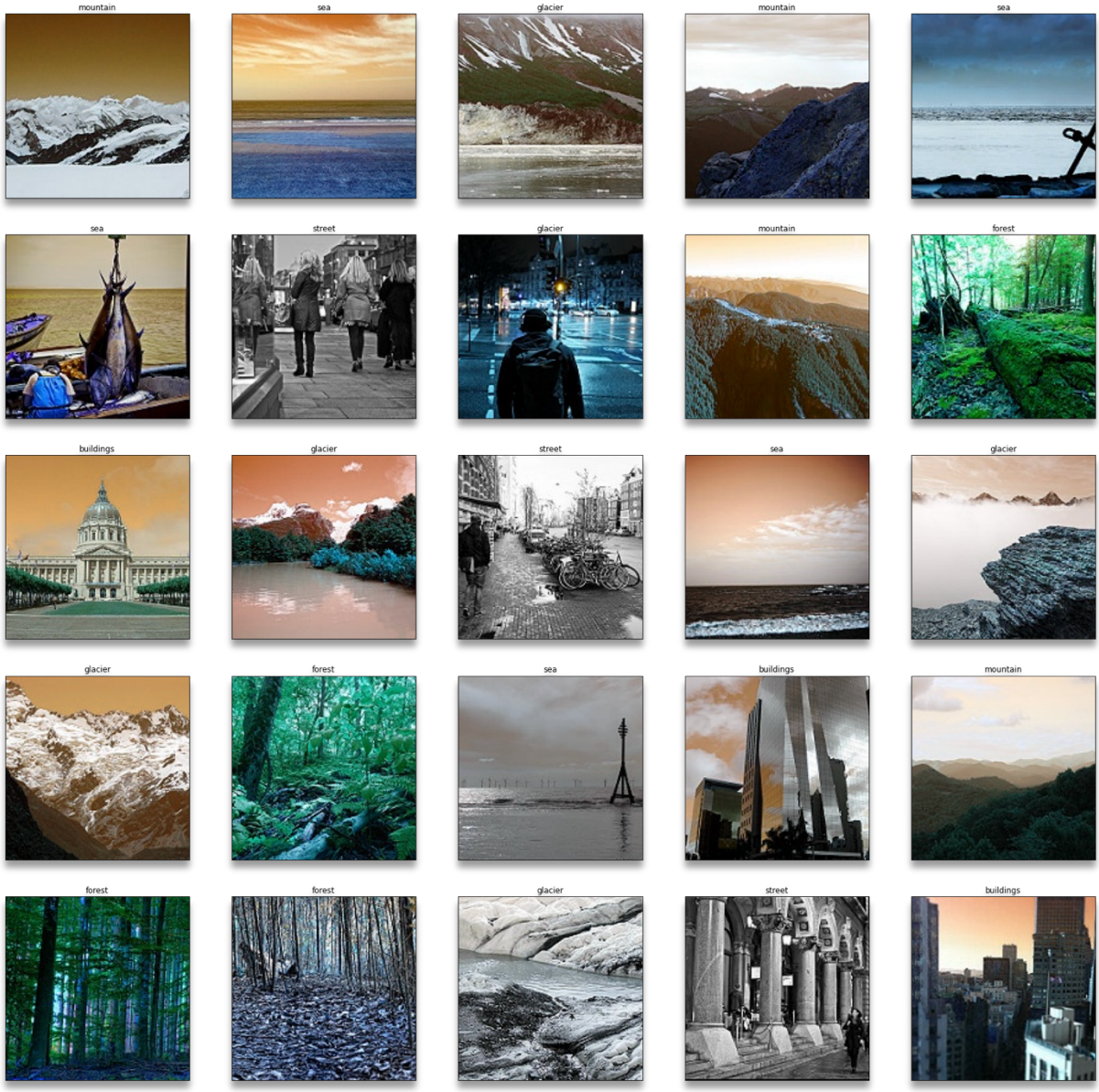
```
94/94 [==============================] - 18s 188ms/step - loss: 0.9247 - accuracy: 0.8353
[0.924651026725769, 0.8353333473205566]
```

Step 5: Predicting the labels for new images

With the model we build, i have predicted the labels for the images in the seg_pred data.

```
fig = plt.figure(figsize=(30, 30))
outer = gridspec.GridSpec(5, 5)

for i in range(25):
    inner = gridspec.GridSpecFromSubplotSpec(1, 1,subplot_spec=outer[i], wspace=0.1, hspace=0.1)
    rnd_number = randint(0,len(pred_images))
    pred_image = np.array([pred_images[rnd_number]])
    pred_class = get_classlabel(np.argmax(model.predict(pred_image), axis=-1)[0])
    # print(pred_class)
    pred_prob = model.predict(pred_image).reshape(6)
    # print(pred_prob)
    for j in range(2):
        if (j%2) == 0:
            ax = plt.Subplot(fig, inner[j])
            ax.imshow(pred_image[0])
            ax.set_title(pred_class)
            ax.set_xticks([])
            ax.set_yticks([])
            fig.add_subplot(ax)

fig.show()
```

Output:

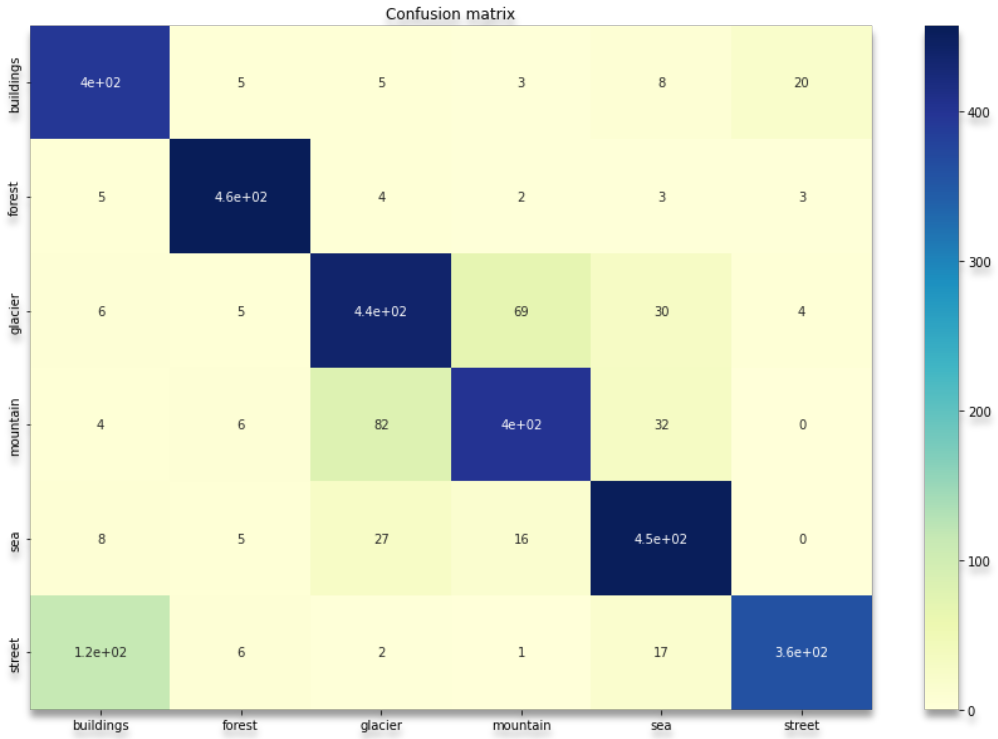


```
from stop_words import get_stop_words
stop_words = get_stop_words('en')
df['review'] = df['review'].apply( lambda x: ' '.join([x for x in str(x).split() if x not in stop_words]) ) # Removing stopwords, and numerics
df['review']=df['review'].str.replace('[^\w\s]','') # Removing punctuations and non-letter tokens
df['review'].head()
```

```
0    bought album loved title song  s great song  b...
1    misled thought buying entire cd contains one song
2    introduced many ell  high school students lois...
3    anything purchase left behind series excellent...
4    loved movies  cant wiat third one  funny  suit...
Name: review, dtype: object
```

Step 6: Confusion matrix of CNN model

```
import sklearn.metrics as metrics
import seaborn as sn
CM = confusion_matrix(test_labels, pred_labels)
fig, ax = plt.subplots(figsize =(15,10))
ax = plt.axes()
sn.heatmap(CM, annot=True,
           cmap="YlGnBu",
           annot_kws={"size": 10},
           xticklabels=image_directories,
           yticklabels=image_directories, ax = ax)
ax.set_title('Confusion matrix')
plt.show()
```



The diagonal elements represent the number of image categories predicted correctly for the test data.

Step 7: Transfer Learning model using AlexNet

Process flow-Transfer learning interaction of AlexNet

Alex-Net model straightforwardly uses ReLU non-linearity in the information designs to make the instated technique more reliable with the hypothesis and furthermore start to prepare the organization straightforwardly from the beginning stage to improve the preparation speed.

```
correct = 0
total = 0
with torch.no_grad():
    for data in test_loader:
        images, labels = data[0].to(device), data[1].to(device)
        outputs = AlexNet_model(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print('Accuracy about the 3000 test images with regard to the network: %d %%' % (
    100 * correct / total))
```

```
Accuracy about the 3000 test images with regard to the network: 90 %
```

From this model we got the accuracy for the test images as 90%.

Accuracy of each individual category is obtained from this model.

```
class_correct = list(0. for i in range(10))
class_total = list(0. for i in range(10))
with torch.no_grad():
    for data in test_loader:
        images, labels = data[0].to(device), data[1].to(device)
        outputs = AlexNet_model(images)
        _, predicted = torch.max(outputs, 1)
        c = (predicted == labels).squeeze()
        for i in range(4):
            label = labels[i]
            class_correct[label] += c[i].item()
            class_total[label] += 1

for i in range(6):
    print('Accuracy of %5s : %2d %%' % (
        classes[i], 100 * class_correct[i] / class_total[i]))
```

```
Accuracy of buildings : 94 %
Accuracy of forest : 96 %
Accuracy of glacier : 82 %
Accuracy of mountain : 88 %
Accuracy of   sea : 95 %
Accuracy of street : 77 %
```

Conclusion: Transfer learning model using AlexNet accuracy of 90% is higher than the CNN Model accuracy of 83.53%.