In [2]:

```
## Import neccessary packages
import os
import cv2
import random
import warnings
import argparse
import itertools
import numpy as np
from imutils import paths
import matplotlib.pyplot as plt
warnings.filterwarnings("ignore")
from tqdm import tqdm_notebook as tqdm

# import the tensorflow.keras packages
from tensorflow.keras.layers import Dense
from tensorflow.keras import backend as K
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import Flatten
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Activation
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.preprocessing.image import ImageDataGenerator

SEED = 50
```

In [3]:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [4]:

```
# create CNN Model

class LeNet:
    @staticmethod
    def build(width, height, depth, classes):
        # initialize the model
        model = Sequential()                ## siamese networks
        inputShape = (height, width, depth)

        # if we are using "channels first", update the input shape
        print(K.image_data_format())
        if K.image_data_format() == "channels_first":
            inputShape = (depth, height, width)

        # first set of CONV => RELU => POOL layers
        model.add(Conv2D(20, (5, 5), padding="same",input_shape=inputShape))
        model.add(Activation("relu"))
        model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

        # second set of CONV => RELU => POOL layers
        model.add(Conv2D(50, (5, 5), padding="same"))
        model.add(Activation("relu"))
        model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

        # first (and only) set of FC => RELU layers
        model.add(Flatten())
        model.add(Dense(500))
        model.add(Activation("relu"))

        # softmax classifier
        model.add(Dense(classes))
        model.add(Activation("softmax"))

        # return the constructed network architecture
        return model
```

In [5]:
```
DATASET = "/content/drive/MyDrive/Data Science/Deep Learning/CNN/Dataset/CNN_Train"  # this folder must cont
MODEL = "/content/drive/MyDrive/Data Science/Deep Learning/CNN/Dataset/Scene.model" # name to store the mode
PLOT = "plot.png" # plot name
```

In [6]:
```
imagePaths = sorted(list(paths.list_images(DATASET)))
random.seed(SEED)
random.shuffle(imagePaths)
imagePaths[:5]
```

Out[6]:
```
['/content/drive/MyDrive/Data Science/Deep Learning/CNN/Dataset/CNN_Train/Sea/3840.jpg',
 '/content/drive/MyDrive/Data Science/Deep Learning/CNN/Dataset/CNN_Train/Forest/1369.jpg',
 '/content/drive/MyDrive/Data Science/Deep Learning/CNN/Dataset/CNN_Train/Sea/3691.jpg',
 '/content/drive/MyDrive/Data Science/Deep Learning/CNN/Dataset/CNN_Train/Buildings/1891.jpg',
 '/content/drive/MyDrive/Data Science/Deep Learning/CNN/Dataset/CNN_Train/Sea/1741.jpg']
```

In [7]:
```
## This is the shape of one image
image=cv2.imread(imagePaths[5])
print("Shape of one image=>",image.shape)

##Resize it => deep learning models train faster on small images
image = cv2.resize(image, (28, 28))
print("Resize Shape of one image=>",image.shape)
print("Type",type(image))

##convert it to array
image = img_to_array(image)
print("Type",type(image))
```

```
Shape of one image=> (150, 150, 3)
Resize Shape of one image=> (28, 28, 3)
Type <class 'numpy.ndarray'>
Type <class 'numpy.ndarray'>
```

In [8]:
```
from google.colab.patches import cv2_imshow
image=cv2.imread(imagePaths[5])
cv2_imshow(image)
image = cv2.resize(image,(100,100))
cv2_imshow(image)
```

In [9]:
```
## Extract The Label From Image Path
label = imagePaths[5].split(os.path.sep)[-2]
label
```

Out[9]:
```
'Buildings'
```

In [10]:

```
# initialize the data and labels
print("[INFO] loading images...")
data = []
labels = []

# grab the image paths and randomly shuffle them
imagePaths = sorted(list(paths.list_images(DATASET)))
random.seed(SEED)
random.shuffle(imagePaths)

# progress bar
with tqdm(total=len(imagePaths)) as pbar:
    # loop over the input images
    for idx, imagePath in enumerate(imagePaths):
        # load the image, pre-process it, and store it in the data list
        image = cv2.imread(imagePath)
        image = cv2.resize(image, (28, 28))
        image = img_to_array(image)
        data.append(image)

        # extract the class label from the image path and update the
        # labels list
        label = imagePath.split(os.path.sep)[-2]

        if label == "Buildings":
            label = 0
        elif label == "Forest":
            label = 1
        elif label == "Sea":
            label = 2

        # print("pr: ", label)

        labels.append(label)

        # update the progressbar
        pbar.update(1)
```

```
[INFO] loading images...

  0%|          | 0/1326 [00:00<?, ?it/s]
```

In [11]:
```
#pixel values are integers that range from 0 (black) to 255 (white).
data = np.array(data, dtype="float") / 255.0
labels = np.array(labels)
```

In [12]:
```
data.shape
```

Out[12]:
```
(1326, 28, 28, 3)
```

In [13]:
```
data[1][27]
```

Out[13]:
```
array([[0., 0., 0.],
       [0., 0., 0.],
       [0., 0., 0.],
       [0., 0., 0.],
       [0., 0., 0.],
       [0., 0., 0.],
       [0., 0., 0.],
       [0., 0., 0.],
       [0., 0., 0.],
       [0., 0., 0.],
       [0., 0., 0.],
       [0., 0., 0.],
       [0., 0., 0.],
       [0., 0., 0.],
       [0., 0., 0.],
       [0., 0., 0.],
       [0., 0., 0.],
       [0., 0., 0.],
       [0., 0., 0.],
       [0., 0., 0.],
       [0., 0., 0.],
       [0., 0., 0.],
       [0., 0., 0.],
       [0., 0., 0.],
       [0., 0., 0.],
       [0., 0., 0.],
       [0., 0., 0.],
       [0., 0., 0.]])
```

```
In [14]: labels

Out[14]: array([2, 1, 2, ..., 1, 1, 2])

In [15]: (trainX, testX, trainY, testY) = train_test_split(data, labels, test_size=0.25, random_state=SEED)

In [16]: trainX.shape

Out[16]: (994, 28, 28, 3)

In [17]: trainY.shape

Out[17]: (994,)

In [18]: trainY[1]

Out[18]: 0

In [19]: trainY[2]

Out[19]: 1

In [20]: trainY[4]

Out[20]: 2

In [21]: trainY = to_categorical(trainY, num_classes=3)
         testY = to_categorical(testY, num_classes=3)

In [22]: trainY[1]

Out[22]: array([1., 0., 0.], dtype=float32)

In [23]: trainY[2]

Out[23]: array([0., 1., 0.], dtype=float32)
```

In [24]:

```
trainY[4]
```

Out[24]:

```
array([0., 0., 1.], dtype=float32)
```

Data Preprocessing (Augumentation)

In [25]:

```
# construct the image generator for data augmentation
aug = ImageDataGenerator(rotation_range=30,
                         width_shift_range=0.1,
                         height_shift_range=0.1,
                         shear_range=0.2,
                         zoom_range=0.2,
                         horizontal_flip=True,
                         fill_mode="nearest")
```

In [38]:

```
EPOCHS = 200
INIT_LR = 1e-3
BS = 32
```

In [28]:

```
# initialize the model
print("[INFO] compiling model...")
model = LeNet.build(width=28, height=28, depth=3, classes=3)
opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
model.compile(loss="categorical_crossentropy", optimizer=opt, metrics=["accuracy"])
print("[INFO] model complied...")
```

```
[INFO] compiling model...
channels_last
[INFO] model complied...
```

In [29]:
```
print(model.summary())
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_2 (Conv2D)           (None, 28, 28, 20)        1520      
                                                                 
 activation_4 (Activation)   (None, 28, 28, 20)        0         
                                                                 
 max_pooling2d_2 (MaxPooling  (None, 14, 14, 20)       0         
 2D)                                                             
                                                                 
 conv2d_3 (Conv2D)           (None, 14, 14, 50)        25050     
                                                                 
 activation_5 (Activation)   (None, 14, 14, 50)        0         
                                                                 
 max_pooling2d_3 (MaxPooling  (None, 7, 7, 50)         0         
 2D)                                                             
                                                                 
 flatten_1 (Flatten)         (None, 2450)              0         
                                                                 
 dense_2 (Dense)             (None, 500)               1225500   
                                                                 
 activation_6 (Activation)   (None, 500)               0         
                                                                 
 dense_3 (Dense)             (None, 3)                 1503      
                                                                 
 activation_7 (Activation)   (None, 3)                 0         
                                                                 
=================================================================
Total params: 1,253,573
Trainable params: 1,253,573
Non-trainable params: 0
_________________________________________________________________
None
```

In [ ]:
```

```

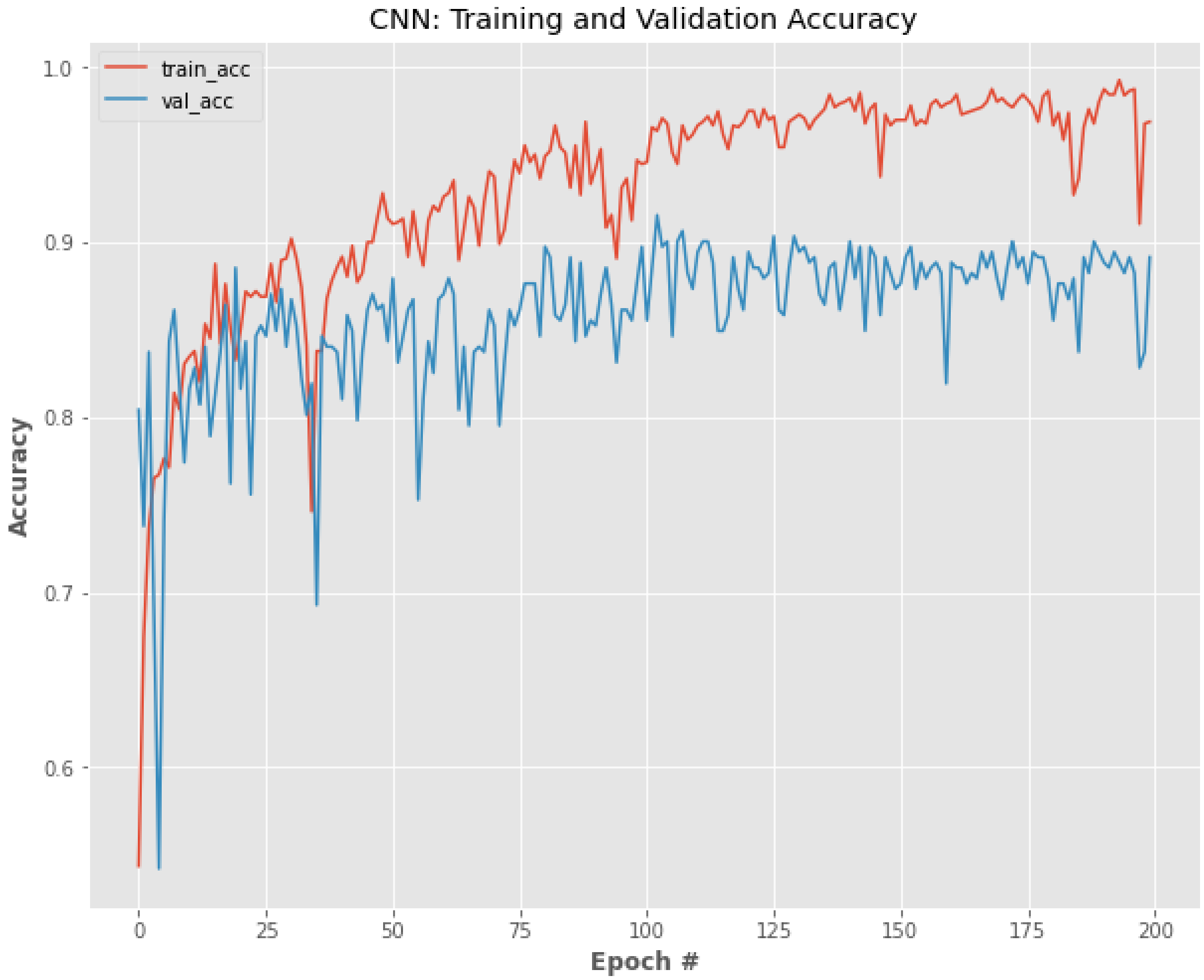
In [30]:

```
# train the network
print("[INFO] training network...")
H = model.fit(x=aug.flow(trainX, trainY, batch_size=BS),
              validation_data=(testX, testY),
              steps_per_epoch=len(trainX) // BS,
              epochs=EPOCHS,
              verbose=1)
```

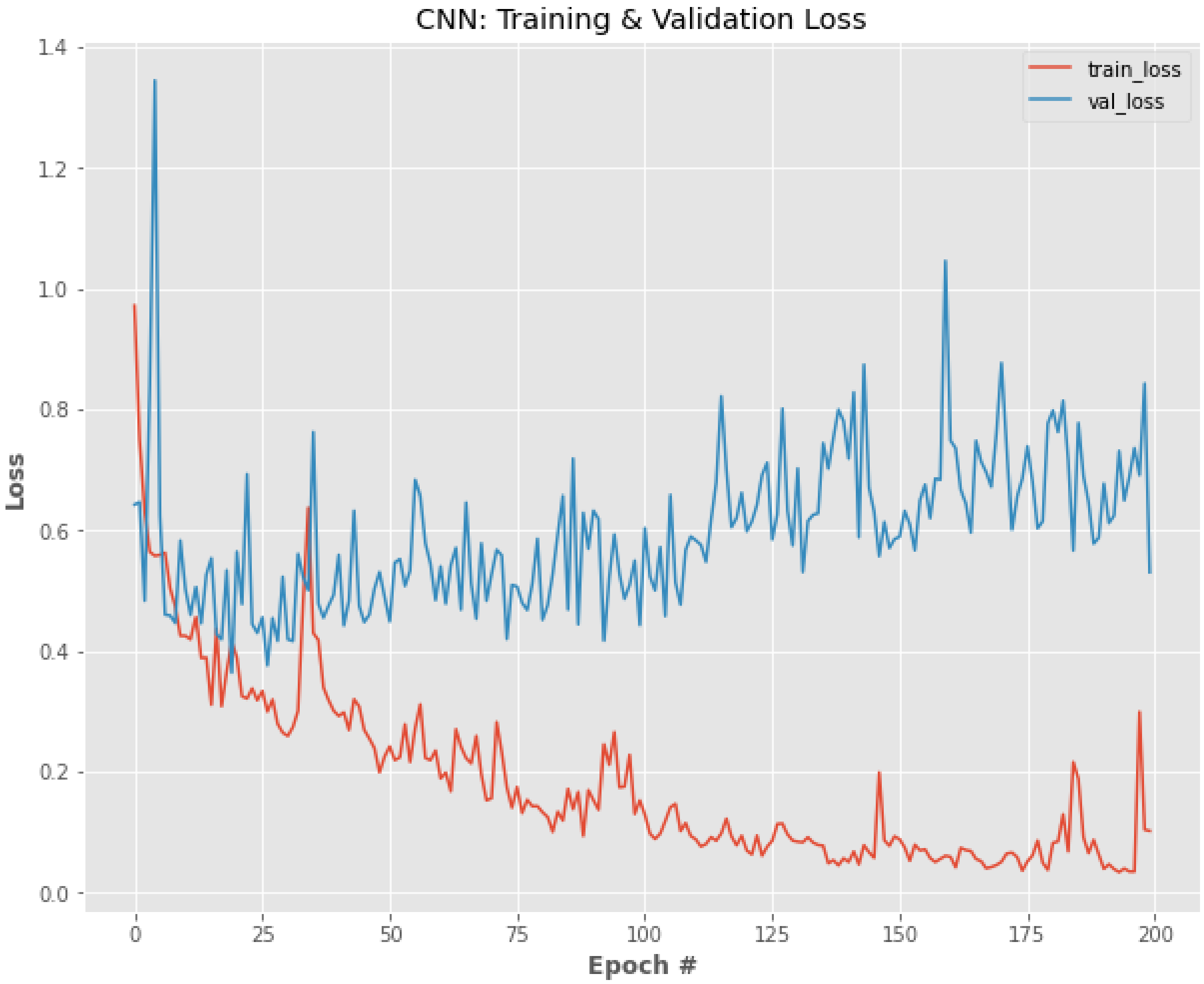
```
[INFO] training network...
Epoch 1/200
31/31 [==============================] - 4s 104ms/step - loss: 0.9725 - accuracy: 0.5437 - val_loss: 0.64
32 - val_accuracy: 0.8042
Epoch 2/200
31/31 [==============================] - 3s 94ms/step - loss: 0.7489 - accuracy: 0.6746 - val_loss: 0.647
4 - val_accuracy: 0.7380
Epoch 3/200
31/31 [==============================] - 3s 98ms/step - loss: 0.6314 - accuracy: 0.7370 - val_loss: 0.483
7 - val_accuracy: 0.8373
Epoch 4/200
31/31 [==============================] - 3s 95ms/step - loss: 0.5648 - accuracy: 0.7651 - val_loss: 0.840
5 - val_accuracy: 0.6898
Epoch 5/200
31/31 [==============================] - 3s 94ms/step - loss: 0.5583 - accuracy: 0.7672 - val_loss: 1.345
2 - val_accuracy: 0.5422
Epoch 6/200
31/31 [==============================] - 4s 124ms/step - loss: 0.5600 - accuracy: 0.7765 - val_loss: 0.62
46 - val_accuracy: 0.7410
Epoch 7/200
```

In [31]:

```
# plot the training and validation accuracy
N = np.arange(0, EPOCHS)
plt.style.use("ggplot")
plt.figure(figsize = [10,8])
plt.plot(N, H.history["accuracy"], label="train_acc")
plt.plot(N, H.history["val_accuracy"], label="val_acc")
plt.title("CNN: Training and Validation Accuracy")
plt.xlabel("Epoch #", weight="bold")
plt.ylabel("Accuracy", weight="bold")
plt.legend()
plt.show()
```



```
In [32]: # plot the training and validation loss
         N = np.arange(0, EPOCHS)
         plt.style.use("ggplot")
         plt.figure(figsize = [10,8])
         plt.plot(N, H.history["loss"], label="train_loss")
         plt.plot(N, H.history["val_loss"], label="val_loss")
         plt.title("CNN: Training & Validation Loss")
         plt.xlabel("Epoch #", weight="bold")
         plt.ylabel("Loss", weight="bold")
         plt.legend()
         plt.show()
```



In [33]:

```
model.save("/content/drive/MyDrive/Data Science/Deep Learning/CNN/Dataset/Scene.model")
```

```
INFO:tensorflow:Assets written to: /content/drive/MyDrive/Data Science/Deep Learning/CNN/Dataset/Scene.mode
l/assets
```

In [34]:

```
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.models import load_model
import numpy as np
import argparse
import imutils
import cv2
import matplotlib.pyplot as plt
%matplotlib inline
```

In [35]:

```
def display_img(img):
    fig = plt.figure(figsize=(12,10))
    plt.grid(b=None)
    ax = fig.add_subplot(111)
    ax.imshow(img)
```

In [39]:

```
# import the necessary packages
from tensorflow.keras.models import load_model
import pickle
import cv2

# # load the model
print("[INFO] loading network and...")
#model = load_model(MODEL)
model = load_model("/content/drive/MyDrive/Data Science/Deep Learning/CNN/Dataset/Scene.model")
# grab the image paths and randomly shuffle themt
testImagePaths = sorted(list(paths.list_images('/content/drive/MyDrive/Data Science/Deep Learning/CNN/Datase

all_class = ["Buildings", "Forest", "Sea"]


# progress bar
with tqdm(total=len(testImagePaths)) as pbar:

    for imagePath in testImagePaths:

        # load the image
        image = cv2.imread(imagePath)
        orig = image.copy()

        # pre-process the image for classification
        image = cv2.resize(image, (28, 28))
        image = image.astype("float") / 255.0
        image = img_to_array(image)
        image = np.expand_dims(image, axis=0)

        # classify the input image
        prd_conf = model.predict(image)[0]

        # build the label
        label = all_class[np.argmax(prd_conf)]
        proba = prd_conf[np.argmax(prd_conf)]

        label = "{}: {:.2f}%".format(label, proba * 100)

        # draw the label on the image
        output = imutils.resize(orig, width=400)
        cv2.putText(output, label, (10, 25),  cv2.FONT_HERSHEY_SIMPLEX,
            0.7, (255, 0, 0), 2)

        # convert img to rgb format and display in notebook
        img = cv2.cvtColor(output, cv2.COLOR_BGR2RGB)
        display_img(img)

        pbar.update(1)
```

```
In [ ]: pip install gradio
```

```
In [41]: import gradio as gr
```

In [42]:

```
def predict_image(image):
    # load the image

    # pre-process the image for classification
    image = cv2.resize(image, (28, 28))
    image = image.astype("float") / 255.0
    image = img_to_array(image)
    image = np.expand_dims(image, axis=0)


    preds = model.predict(image)[0]
    result = dict()
    result["Buildings"] = round(float(list(preds)[0]), 3)
    result["Forest"] = round(float(list(preds)[1]), 3)
    result["Sea"] = round(float(list(preds)[2]), 3)

    print(result)

    return result
```

In [43]:

```
im = gr.inputs.Image(shape=(32,32))
label = gr.outputs.Label(num_top_classes=3)

gr.Interface(fn=predict_image, inputs=im, outputs=label, capture_session=True, title="CNN Demo").launch(shar
```

```
Colab notebook detected. To show errors in colab notebook, set `debug=True` in `launch()`
Running on public URL: https://53401.gradio.app (https://53401.gradio.app)

This share link expires in 72 hours. For free permanent hosting, check out Spaces (https://huggingface.co/s
paces)
```

Out[43]:

```
(<gradio.routes.App at 0x7f585af31510>,
 'http://127.0.0.1:7860/',
 'https://53401.gradio.app')
```

In [44]:

```
import tensorflow as tf
print(tf.__version__)
```

```
2.8.2
```