Land Use and Land Cover detection using Satellite Image Data

National Institute of Technology Karnataka Surathkal

Date: 27-04-2021

Submitted To:
Prof. Venkatesan
CSE Department, NITK

Group Members:

Chinmay Gupta 181CO215
Shuddhatm Choudhary 181CO251

Abstract

Mapping land use/land cover (LULC) changes at regional scales is essential for a wide range of applications, including landslide, erosion, land planning, global warming etc. LULC alterations (based especially on human activities), negatively affect the patterns of climate, the patterns of natural hazard and socio-economic dynamics in global and local scale.Detecting the land cover helps in mapping of land cover for the betterment of the resources available. This process of LULC mapping is achieved through deep-learning networks pre-trained on the large-scale visual recognition competition datasets and fine-tuned on our target dataset. Resnet50, VGG15 and VGG19 architectures have performed well for image classification and these architectures were compared on this dataset.

Contents

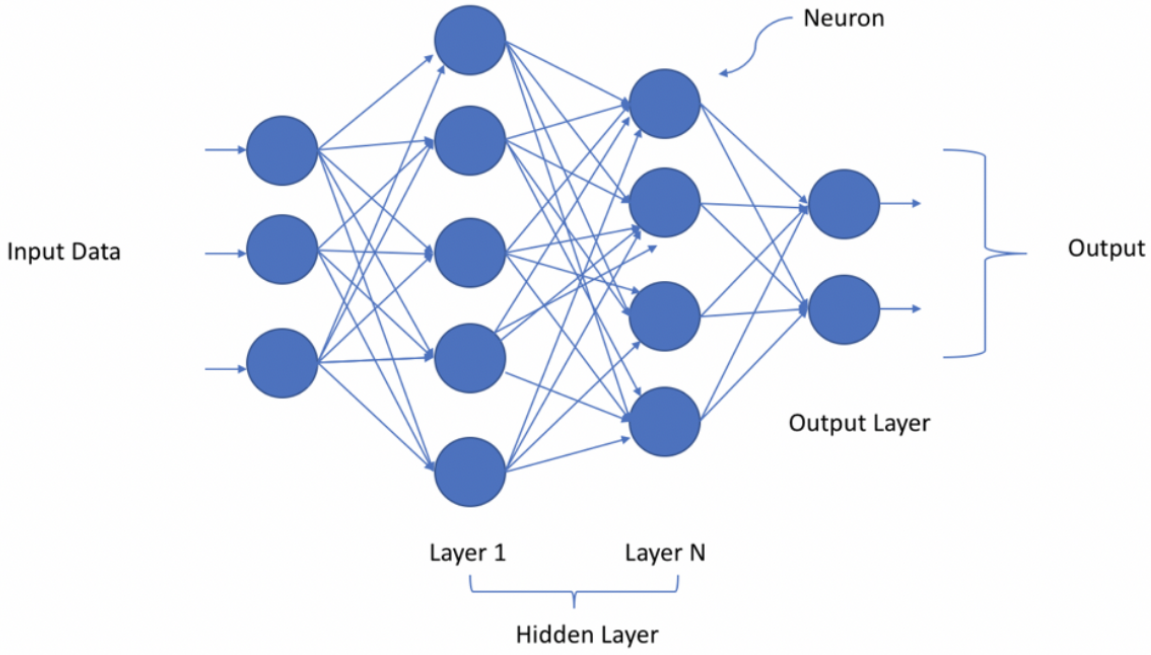
1.Introduction **3**
2.Dataset Description **4**
2.1 Dataset Description 4
2.2 Data Preprocessing 5
3. Model Description **5**
3.1 CNN 5
3.2 Architectures 6
3.2.1 Resnet-50 6
3.2.2 VGG16 7
3.2.3 VGG19 7
3.3 Callbacks 8

4. Code **9**
5. Performance,Results and Analysis **22**
6. Conclusion
7. References **25**

Introduction

Satellite imagery can help us find solutions to the growing number of environmental problems that humans face today. It allows us to not only get a bird's eye view of what's around us, but also uncovers parts of the world that are rarely seen. Tapping into the potential of categorizing land cover and land use around the world means that humans can more efficiently make use of natural resources, hopefully lowering cases of waste and deprivation. But despite its potential to be incredibly useful, satellite data is massive and confusing, and making sense of it requires complex analysis.

Classifying the images using naked eye gives high accuracy but the manpower required will be too high. With the technology available we can use deep neural networks to classify the images. Everything is made easy with the internet and deep learning is a concept that makes our work easier.



Deep learning is a branch of machine learning which refers to the process of understanding data through a certain method of training to obtain a trained model which

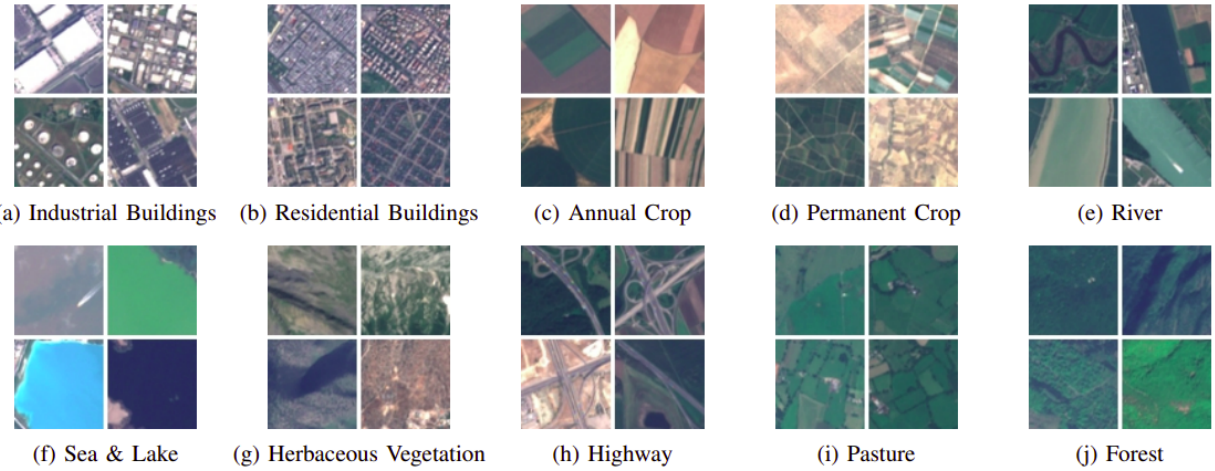
can be used multiple times. Its essence is also a neural network, but the number of hidden layers is more than one layer, which is an extension of artificial neural networks. “Neural network” is a component of deep learning. The advantage of deep learning is that it does not require artificial feature extraction but this is obtained automatically by the network. It can solve nonlinear separable problems and has strong generalization ability and robustness. Among them, the most widely used is the convolutional neural network, which is a deep neural network. Images can be directly used as input data, eliminating the complicated process of feature extraction and data reconstruction in traditional machine learning algorithms. Hence, based on previous studies and referenced papers, this project uses deep convolutional neural networks to classify land cover and land use. We have concentrated our work towards three major CNN architectures Resnet50, VGG16, and VGG19 and compared the results obtained.

Dataset Description

The dataset is a 10 class dataset , consisting of 27,000 images with 2000-3000 images belonging to each class. This dataset is taken from https://github.com/phelber/EuroSAT. Which is Sentinel-2A Satellite georeferenced labeled data. The classes contained in data are:

- Forest
- Herbaceous Vegetation
- Highway
- Industrial
- Pasture
- Permanent Crop
- Residential
- River
- Sea / Lake

This data is distributed over 34 european countries. We split the dataset in a training set and testing set in (80/20) ratio.



(a) Industrial Buildings (b) Residential Buildings (c) Annual Crop (d) Permanent Crop (e) River

(f) Sea & Lake (g) Herbaceous Vegetation (h) Highway (i) Pasture (j) Forest

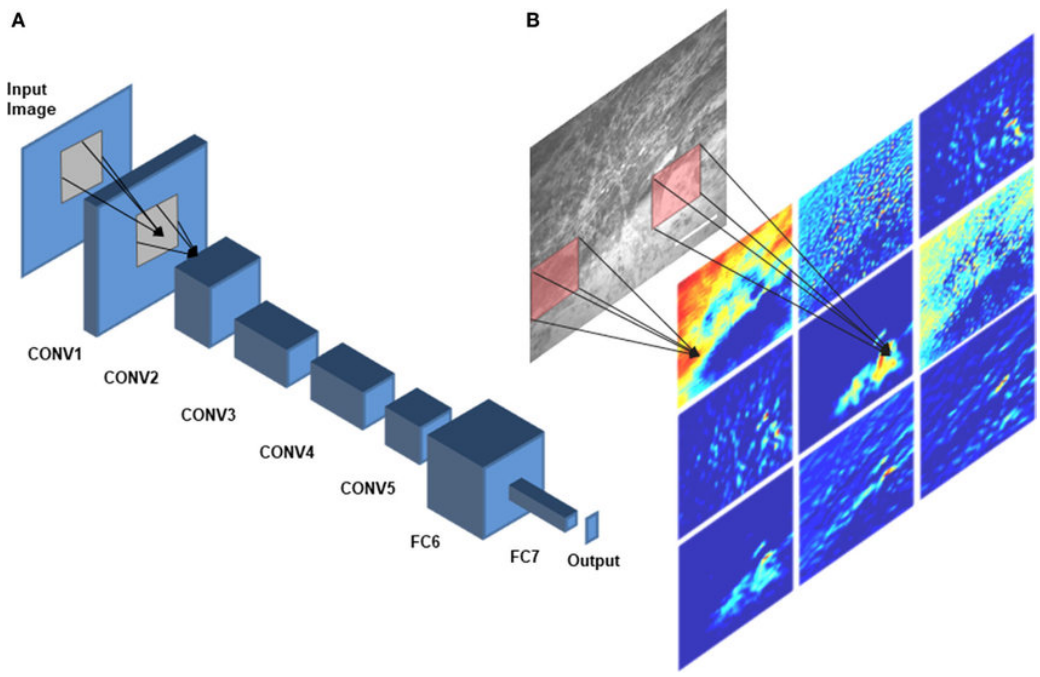
Data Preprocessing :-

Data preprocessing involves changes in the data to make the model run efficiently and smoothly. It involves labelling, image augmentation and stratified shuffle split using inbuilt libraries. **Stratified shuffle split** from sklearn library was used to split the data into training, validation and testing data. **Image data generator** from keras was used for image augmentation with batch size equal to 64 to expand the size of a training dataset by creating modified versions of images in the dataset.

MODEL DESCRIPTION:-

1.) **Convolutional Neural Networks(CNNs**) :-

Convolutional neural networks have the ability to automatically learn a large number of filters in parallel specific to a training dataset under the constraints of a specific predictive modeling problem, such as image classification. The result is highly specific features that can be detected anywhere on input images.It takes in an input image, assigns importance (learnable weights and biases) to various aspects/objects in the image, and be able to differentiate one from the other.



The rapid developments in Computer Vision, and by extension – image classification has been further accelerated by the advent of Transfer Learning.Transfer learning allows us to use a pre-existing model, trained on a huge dataset, for our own tasks. Consequently reducing the cost of training new deep learning models and since the datasets have been vetted, we can be assured of the quality.

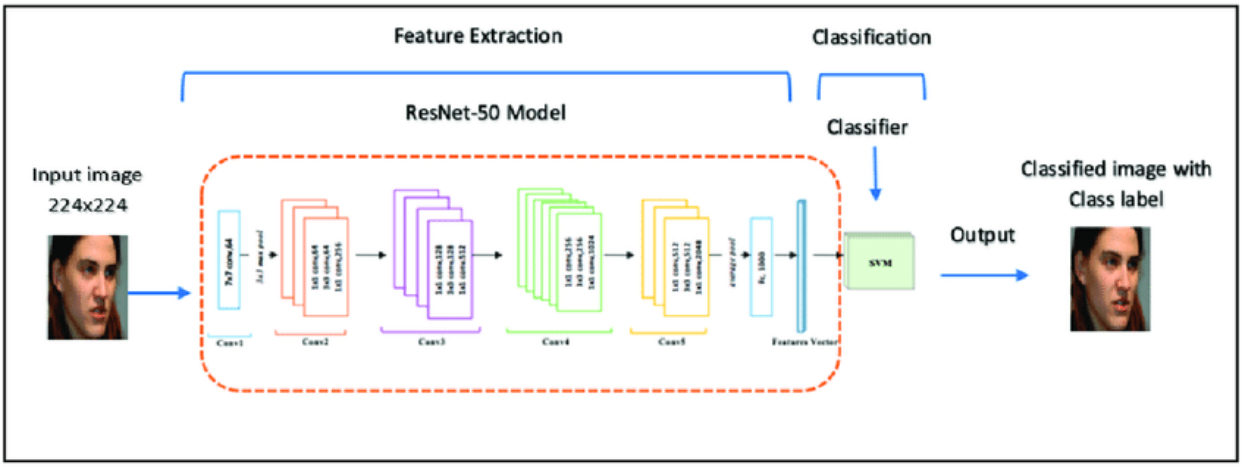
We have used 3 such CNN architectures namely Resnet50, VGG16, and VGG19 for classification. These architectures are already present in the keras library.

We used the **ReLu activation function**. The rectified linear activation function or ReLU for short is a piecewise linear function that will output the input directly if it is positive, otherwise, it will output zero. It has become the default activation function for many types of neural networks because a model that uses it is easier to train and often achieves better performance. The function is linear for values greater than zero and 0 for values less than 0.
$f(x) = max(0, z(x))$, *z is a function with predefined slop*.

Architectures:-

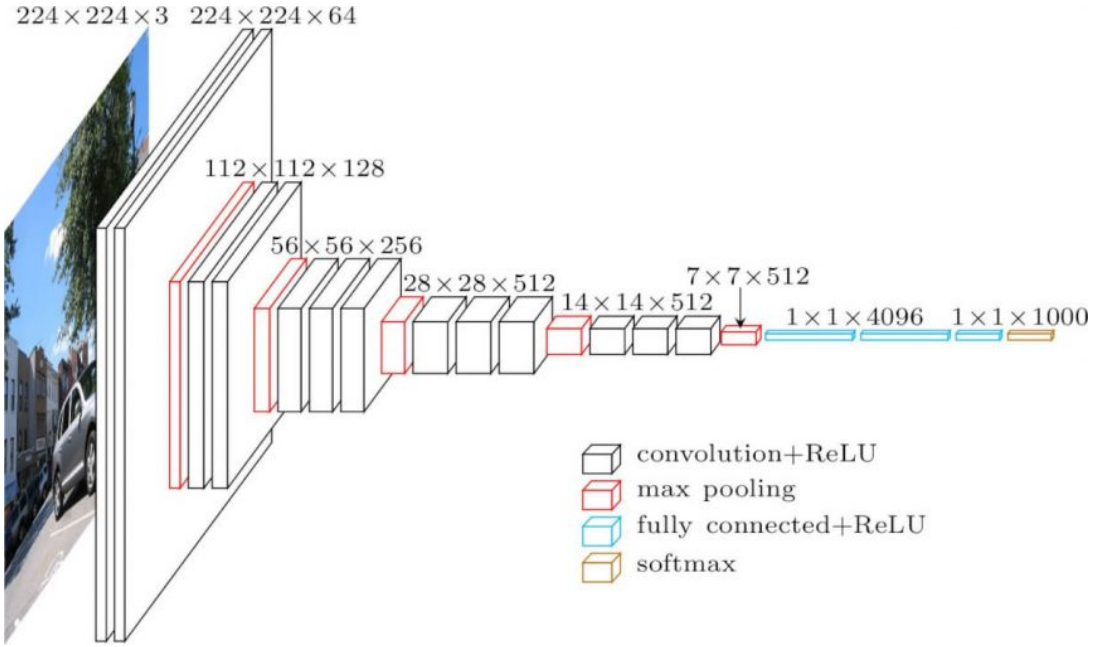
1.) **Resnet50**:-

Resnet model was proposed to solve the issue of diminishing gradient. The idea is to skip the connection and pass the residual to the next layer so that the model can continue to train. With Resnet models, CNN models can go deeper and deeper. Before applying the model we pre-trained the dense layer and used the weights learned there as initial weights for resnet50.



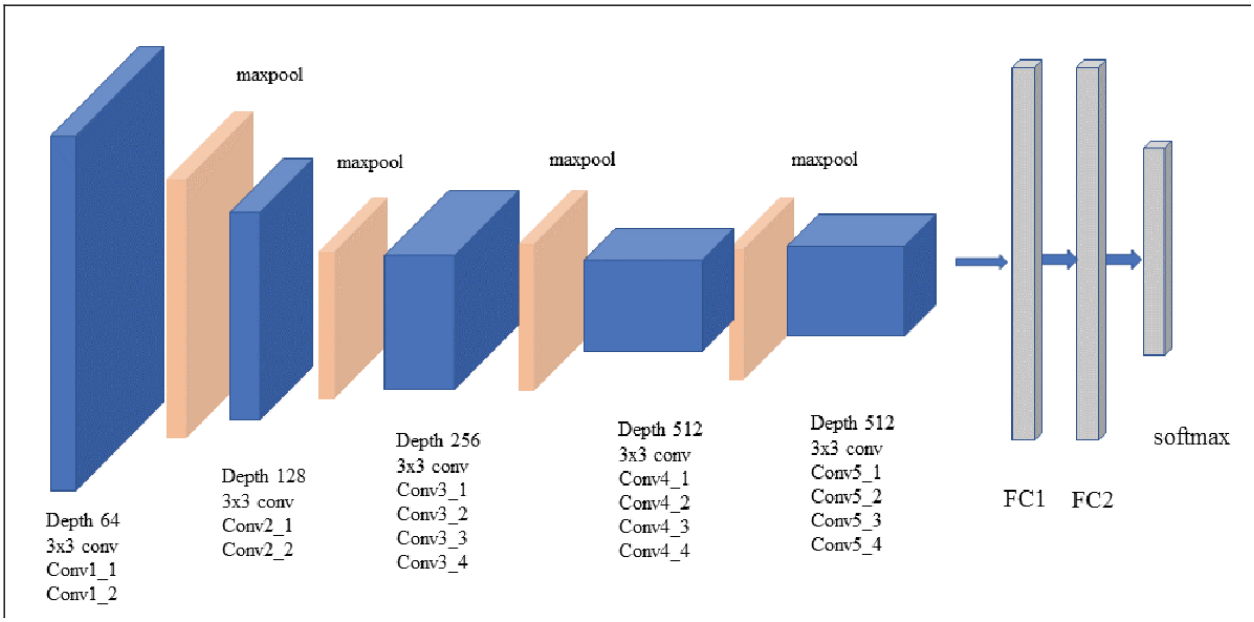
2.) **VGG16**:-

It is a CNN model that is 16 layers deep and due to the diminishing gradient the depth of the layer is set to be 16. It has 13 convolutional layers and 3 fully connected layers. We initialised the weights that we learned by learning the weights for the dense layer as it saves time and is effective in applying deep convolutional neural networks.



3.) **VGG19:-**

VGG-19 is an improvement of the model VGG-16. It is a convolution neural network model with 19 layers. It is built by stacking convolutions together but the model's depth is limited because of an issue called diminishing gradient. This issue makes deep convolutional networks difficult to train. The model was trained on ImageNet for classification of 1000 types of objects and so do the rest of the models reviewed.



As while training a deep neural network we might get the plateau after some epochs so we used three callbacks:-

Callbacks:-A callback is a set of functions to be applied at given stages of the training procedure. Callbacks can be used to get a view on internal states and statistics of the model during training.

1.) EarlyStopping:-

One way to avoid overfitting is to terminate the process early. The EarlyStoppingfunction has various metrics/arguments that you can modify to set up when the training process should stop. Here are some relevant metrics:

monitor: value being monitored, i.e: val_loss

min_delta: minimum change in the monitored value. For example, min_delta=1 means that the training process will be stopped if the absolute change of the monitored value is less than 1

patience: number of epochs with no improvement after which training will be stopped

restore_best_weights: set this metric to True if you want to keep the best weights once stopped.

2.) ModelCheckpoint:-

This callback saves the model after every epoch. Here are some relevant metrics:

filepath: the file path you want to save your model in
monitor: the value being monitored
save_best_only: set this to True if you do not want to overwrite the latest best model

3.) ReduceLRonplateau:-

This callback is used to adjust the learning rate after “patience” number of non-increasing accuracy epochs. It divides the learning rate by a number.

monitor: quantity to be monitored.

factor: factor by which the learning rate will be reduced. new_lr = lr * factor.

patience: number of epochs with no improvement after which

min_lr: lower bound on the learning rate.

Adam Optimizer:

The optimizer used was Adam Optimizer which uses the binary cross entropy loss function. Adaptive Moment Estimation (Adam) is another method that computes adaptive learning rates for each parameter. In addition to storing an exponentially decaying average of past squared gradients like Adadelta and RMSprop, Adam also keeps an exponentially decaying average of past gradients, similar to momentum. Whereas momentum can be seen as a ball running down a slope, Adam behaves like a heavy ball with friction, which thus prefers flat minima in the error surface.

Code

```
# Importing few libraries
import os
```

```
import shutil
import random
from tqdm import tqdm

import numpy as np
import pandas as pd

import PIL
import seaborn as sns
import matplotlib.pyplot as plt
```

```
DATASET = "../input/2750"

LABELS = os.listdir(DATASET)
print(LABELS)
```

```
# plot class distributions of whole dataset
counts = {}

for l in LABELS:
   counts[l] = len(os.listdir(os.path.join(DATASET, l)))


plt.figure(figsize=(12, 6))

plt.bar(range(len(counts)), list(counts.values()), align='center')
plt.xticks(range(len(counts)), list(counts.keys()), fontsize=12, rotation=40)
plt.xlabel('class label', fontsize=13)
plt.ylabel('class size', fontsize=13)
plt.title('EUROSAT Class Distribution', fontsize=15);
```

```
img_paths = [os.path.join(DATASET, l, l+'_1000.jpg') for l in LABELS]

img_paths = img_paths + [os.path.join(DATASET, l, l+'_2000.jpg') for l in
LABELS]

def plot_sat_imgs(paths):
   plt.figure(figsize=(15, 8))
   for i in range(20):
       plt.subplot(4, 5, i+1, xticks=[], yticks=[])
```

```
        img = PIL.Image.open(paths[i], 'r')
        plt.imshow(np.asarray(img))
        plt.title(paths[i].split('/')[-2])

plot_sat_imgs(img_paths)
```

```
import re
from sklearn.model_selection import StratifiedShuffleSplit
from keras.preprocessing.image import ImageDataGenerator

TRAIN_DIR = '../working/training'
TEST_DIR = '../working/testing'
BATCH_SIZE = 64
NUM_CLASSES=len(LABELS)
INPUT_SHAPE = (64, 64, 3)
CLASS_MODE = 'categorical'

# create training and testing directories
for path in (TRAIN_DIR, TEST_DIR):
    if not os.path.exists(path):
        os.mkdir(path)

# create class label subdirectories in train and test
for l in LABELS:

    if not os.path.exists(os.path.join(TRAIN_DIR, l)):
        os.mkdir(os.path.join(TRAIN_DIR, l))

    if not os.path.exists(os.path.join(TEST_DIR, l)):
        os.mkdir(os.path.join(TEST_DIR, l))
```

```
# map each image path to their class label in 'data'
data = {}

for l in LABELS:
    for img in os.listdir(DATASET+'/'+l):
        data.update({os.path.join(DATASET, l, img): l})

X = pd.Series(list(data.keys()))
y = pd.get_dummies(pd.Series(data.values()))
```

```
split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=69)

# split the list of image paths
for train_idx, test_idx in split.split(X, y):

   train_paths = X[train_idx]
   test_paths = X[test_idx]

   # define a new path for each image depending on training or testing
   new_train_paths = [re.sub('\.\.\/input\/2750', '../working/training', i) for
i in train_paths]
   new_test_paths = [re.sub('\.\.\/input\/2750', '../working/testing', i) for i
in test_paths]

   train_path_map = list((zip(train_paths, new_train_paths)))
   test_path_map = list((zip(test_paths, new_test_paths)))

   # move the files
   print("moving training files..")
   for i in tqdm(train_path_map):
       if not os.path.exists(i[1]):
           if not os.path.exists(re.sub('training', 'testing', i[1])):
               shutil.copy(i[0], i[1])

   print("moving testing files..")
   for i in tqdm(test_path_map):
       if not os.path.exists(i[1]):
           if not os.path.exists(re.sub('training', 'testing', i[1])):
               shutil.copy(i[0], i[1])
```

```
# Create a ImageDataGenerator Instance which can be used for data augmentation

train_gen = ImageDataGenerator(
   rescale=1./255,
   rotation_range=60,
   width_shift_range=0.2,
   height_shift_range=0.2,
   shear_range=0.2,
   zoom_range=0.2,
   horizontal_flip=True,
   vertical_flip = True
#   validation_split=0.2
)
```

```
train_generator = train_gen.flow_from_directory(
    directory=TRAIN_DIR,
    target_size=(64, 64),
    batch_size=BATCH_SIZE,
    class_mode=CLASS_MODE,
    #subset='training',
    color_mode='rgb',
    shuffle=True,
    seed=69
)
# test generator for evaluation purposes with no augmentations, just rescaling
test_gen = ImageDataGenerator(
    rescale=1./255,
)

test_generator = test_gen.flow_from_directory(
    directory=TEST_DIR,
    target_size=(64, 64),
    batch_size=BATCH_SIZE,
    class_mode=CLASS_MODE,
    color_mode='rgb',
    shuffle=False,
    seed=69
)
```

```
np.save('class_indices', train_generator.class_indices)
```

```
import tensorflow as tf
from keras.models import Model
from keras.layers import Dense, Dropout, Flatten
from keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau
from keras.optimizers import Adam


from keras.applications import VGG16, VGG19
from keras.applications import ResNet50

from sklearn.metrics import precision_recall_fscore_support, confusion_matrix,
fbeta_score, accuracy_score
```

```
gpus = tf.config.experimental.list_physical_devices('GPU')
if gpus:
 # Restrict TensorFlow to only use the first GPU
 try:
   tf.config.experimental.set_visible_devices(gpus[0], 'GPU')
   logical_gpus = tf.config.experimental.list_logical_devices('GPU')
   print(len(gpus), "Physical GPUs,", len(logical_gpus), "Logical GPU")
 except RuntimeError as e:
   # Visible devices must be set before GPUs have been initialized
   print(e)
```

```
tf.config.set_soft_device_placement(True)
# Note that for different CNN models we are using different setup of dense
layers
def compile_model(cnn_base, input_shape, n_classes, optimizer, fine_tune=None):

   if (cnn_base == 'ResNet50'):

       conv_base =
ResNet50(include_top=False,weights='imagenet',input_shape=input_shape)

       top_model = conv_base.output
       top_model = Flatten()(top_model)
       top_model = Dense(2048, activation='relu')(top_model)
       top_model = Dropout(0.2)(top_model)


   elif (cnn_base == 'VGG16') or (cnn_base == 'VGG19'):
       if cnn_base == 'VGG16':
           conv_base = VGG16(include_top=False,
                             weights='imagenet',
                             input_shape=input_shape)
       else:
           conv_base = VGG19(include_top=False,
                             weights='imagenet',
                             input_shape=input_shape)
       top_model = conv_base.output
       top_model = Flatten()(top_model)
       top_model = Dense(2048, activation='relu')(top_model)
       top_model = Dropout(0.2)(top_model)
       top_model = Dense(2048, activation='relu')(top_model)
       top_model = Dropout(0.2)(top_model)
```

```
    output_layer = Dense(n_classes, activation='softmax')(top_model)

    model = Model(inputs=conv_base.input, outputs=output_layer)

    if type(fine_tune) == int:
        for layer in conv_base.layers[fine_tune:]:
            layer.trainable = True
    else:
        for layer in conv_base.layers:
            layer.trainable = False

    model.compile(optimizer=optimizer, loss='categorical_crossentropy',
                  metrics=['categorical_accuracy'])

    return model

def plot_history(history):

    acc = history.history['categorical_accuracy']
    val_acc = history.history['val_categorical_accuracy']
    loss = history.history['loss']
    val_loss = history.history['val_loss']

    plt.figure(figsize=(10, 5))
    plt.subplot(1, 2, 1)
    plt.plot(acc)
    plt.plot(val_acc)
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'val'], loc='upper left')

    plt.subplot(1, 2, 2)
    plt.plot(loss)
    plt.plot(val_loss)
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'val'], loc='upper left')

    plt.show();

def display_results(y_true, y_preds, class_labels):

    results = pd.DataFrame(precision_recall_fscore_support(y_true, y_preds),
                           columns=class_labels).T
```

```
   results.rename(columns={0: 'Precision',
                           1: 'Recall',
                           2: 'F-Score',
                           3: 'Support'}, inplace=True)

   conf_mat = pd.DataFrame(confusion_matrix(y_true, y_preds),
                           columns=class_labels,
                           index=class_labels)
   f2 = fbeta_score(y_true, y_preds, beta=2, average='micro')
   accuracy = accuracy_score(y_true, y_preds)
   print(f"Accuracy: {accuracy}")
   print(f"Global F2 Score: {f2}")
   return results, conf_mat

def plot_predictions(y_true, y_preds, test_generator, class_indices):

   fig = plt.figure(figsize=(20, 10))
   for i, idx in enumerate(np.random.choice(test_generator.samples, size=20,
replace=False)):
       ax = fig.add_subplot(4, 5, i + 1, xticks=[], yticks=[])
       ax.imshow(np.squeeze(test_generator[idx]))
       pred_idx = np.argmax(y_preds[idx])
       true_idx = y_true[idx]

       plt.tight_layout()
       ax.set_title("{}\n({})".format(class_indices[pred_idx],
class_indices[true_idx]),
                    color=("green" if pred_idx == true_idx else "red"))
```

```
N_STEPS = train_generator.samples//BATCH_SIZE
N_VAL_STEPS = test_generator.samples//BATCH_SIZE
N_EPOCHS = 100

# model callbacks
checkpoint = ModelCheckpoint(filepath='../working/model.weights.best.hdf5',
                        monitor='val_categorical_accuracy',
                        save_best_only=True,
                        verbose=1)

early_stop = EarlyStopping(monitor='val_categorical_accuracy',
                           patience=10,
                           restore_best_weights=True,
                           mode='max')
```

```
reduce_lr = ReduceLROnPlateau(monitor='val_categorical_accuracy', factor=0.5,
                              patience=3, min_lr=0.00001)
```

Resnet-50 (training)

```
resnet50_model = compile_model('ResNet50', INPUT_SHAPE, NUM_CLASSES,
Adam(lr=1e-2), fine_tune=None)
resnet50_model.summary()
```

```
train_generator.reset()
test_generator.reset()

N_STEPS = train_generator.samples//BATCH_SIZE
N_VAL_STEPS = test_generator.samples//BATCH_SIZE
N_EPOCHS = 100

# model callbacks
checkpoint = ModelCheckpoint(filepath='../working/model.weights.best.hdf5',
                        monitor='val_categorical_accuracy',
                        save_best_only=True,
                        verbose=1)

early_stop = EarlyStopping(monitor='val_categorical_accuracy',
                           patience=10,
                           restore_best_weights=True,
                           mode='max')

reduce_lr = ReduceLROnPlateau(monitor='val_categorical_accuracy', factor=0.5,
                              patience=3, min_lr=0.00001)
```

```
# First Pretraining the dense layer
resnet50_history = resnet50_model.fit_generator(train_generator,
                              steps_per_epoch=N_STEPS,
                              epochs=50,
                              callbacks=[early_stop, checkpoint],
                              validation_data=test_generator,
                              validation_steps=N_VAL_STEPS)
```

```
# re-train whole network end2end
resnet50_model = compile_model('ResNet50', INPUT_SHAPE, NUM_CLASSES,
Adam(lr=1e-4), fine_tune=0)
resnet50_model.load_weights('../working/model.weights.best.hdf5')

train_generator.reset()
test_generator.reset()

resnet50_history = resnet50_model.fit_generator(train_generator,
                            steps_per_epoch=N_STEPS,
                            epochs=N_EPOCHS,
                            callbacks=[early_stop, checkpoint, reduce_lr],
                            validation_data=test_generator,
                            validation_steps=N_VAL_STEPS)
```

```
plot_history(resnet50_history)
resnet50_model.load_weights('../working/model.weights.best.hdf5')

class_indices = train_generator.class_indices
class_indices = dict((v,k) for k,v in class_indices.items())

test_generator_new = test_gen.flow_from_directory(
   directory=TEST_DIR,
   target_size=(64, 64),
   batch_size=1,
   class_mode=None,
   color_mode='rgb',
   shuffle=False,
   seed=69
)

predictions = resnet50_model.predict_generator(test_generator_new,
steps=len(test_generator_new.filenames))
predicted_classes = np.argmax(np.rint(predictions), axis=1)
true_classes = test_generator_new.classes

prf, conf_mat = display_results(true_classes, predicted_classes,
class_indices.values())
prf
# Save the model and the weights
resnet50_model.save('../working/ResNet50_eurosat.h5')
```

VGG16(training)

```
vgg16_model = compile_model('VGG16', INPUT_SHAPE, NUM_CLASSES, Adam(lr=1e-2),
fine_tune=None)
vgg16_model.summary()
```

```
train_generator.reset()
test_generator.reset()

N_STEPS = train_generator.samples//BATCH_SIZE
N_VAL_STEPS = test_generator.samples//BATCH_SIZE
N_EPOCHS = 100

# model callbacks
checkpoint = ModelCheckpoint(filepath='../working/model.weights.best.hdf5',
                      monitor='val_categorical_accuracy',
                      save_best_only=True,
                      verbose=1)

early_stop = EarlyStopping(monitor='val_categorical_accuracy',
                           patience=10,
                           restore_best_weights=True,
                           mode='max')

reduce_lr = ReduceLROnPlateau(monitor='val_categorical_accuracy', factor=0.5,
                              patience=3, min_lr=0.00001)
```

```
train_generator.reset()
# First Pretraining the dense layer
vgg16_history = vgg16_model.fit_generator(train_generator,
                           steps_per_epoch=N_STEPS,
                           epochs=50,
                           callbacks=[early_stop, checkpoint],
                           validation_data=test_generator,
                           validation_steps=N_VAL_STEPS)
```

```
# re-train whole network end2end
vgg16_model = compile_model('VGG16', INPUT_SHAPE, NUM_CLASSES, Adam(lr=1e-4),
fine_tune=0)

vgg16_model.load_weights('../working/model.weights.best.hdf5')

train_generator.reset()
```

```
test_generator.reset()

vgg16_history = vgg16_model.fit_generator(train_generator,
                            steps_per_epoch=N_STEPS,
                            epochs=N_EPOCHS,
                            callbacks=[early_stop, checkpoint, reduce_lr],
                            validation_data=test_generator,
                            validation_steps=N_VAL_STEPS)
```

```
plot_history(vgg16_history)
vgg16_model.load_weights('../working/model.weights.best.hdf5')

class_indices = train_generator.class_indices
class_indices = dict((v,k) for k,v in class_indices.items())

test_generator_new = test_gen.flow_from_directory(
   directory=TEST_DIR,
   target_size=(64, 64),
   batch_size=1,
   class_mode=None,
   color_mode='rgb',
   shuffle=False,
   seed=69
)

predictions = vgg16_model.predict_generator(test_generator_new,
steps=len(test_generator_new.filenames))
predicted_classes = np.argmax(np.rint(predictions), axis=1)
true_classes = test_generator_new.classes

prf, conf_mat = display_results(true_classes, predicted_classes,
class_indices.values())
prf
```

```
# Save the model and the weights
vgg16_model.save('../working/vgg16_eurosat.h5')
```

VGG19

```
vgg19_model = compile_model('VGG19', INPUT_SHAPE, NUM_CLASSES, Adam(lr=1e-2),
fine_tune=None)
vgg19_model.summary()
```

```
train_generator.reset()
test_generator.reset()

N_STEPS = train_generator.samples//BATCH_SIZE
N_VAL_STEPS = test_generator.samples//BATCH_SIZE
N_EPOCHS = 100

# model callbacks
checkpoint = ModelCheckpoint(filepath='../working/model.weights.best.hdf5',
                      monitor='val_categorical_accuracy',
                      save_best_only=True,
                      verbose=1)

early_stop = EarlyStopping(monitor='val_categorical_accuracy',
                           patience=10,
                           restore_best_weights=True,
                           mode='max')

reduce_lr = ReduceLROnPlateau(monitor='val_categorical_accuracy', factor=0.5,
                              patience=3, min_lr=0.00001)
```

```
train_generator.reset()
# First Pretraining the dense layer
vgg19_history = vgg19_model.fit_generator(train_generator,
                           steps_per_epoch=N_STEPS,
                           epochs=50,
                           callbacks=[early_stop, checkpoint],
                           validation_data=test_generator,
                           validation_steps=N_VAL_STEPS)
```

```
# re-train whole network end2end
vgg19_model = compile_model('VGG19', INPUT_SHAPE, NUM_CLASSES, Adam(lr=1e-4),
fine_tune=0)

vgg19_model.load_weights('../working/model.weights.best.hdf5')

train_generator.reset()
test_generator.reset()

vgg19_history = vgg19_model.fit_generator(train_generator,
                           steps_per_epoch=N_STEPS,
                           epochs=N_EPOCHS,
```

```
                            callbacks=[early_stop, checkpoint, reduce_lr],
                            validation_data=test_generator,
                            validation_steps=N_VAL_STEPS)
```

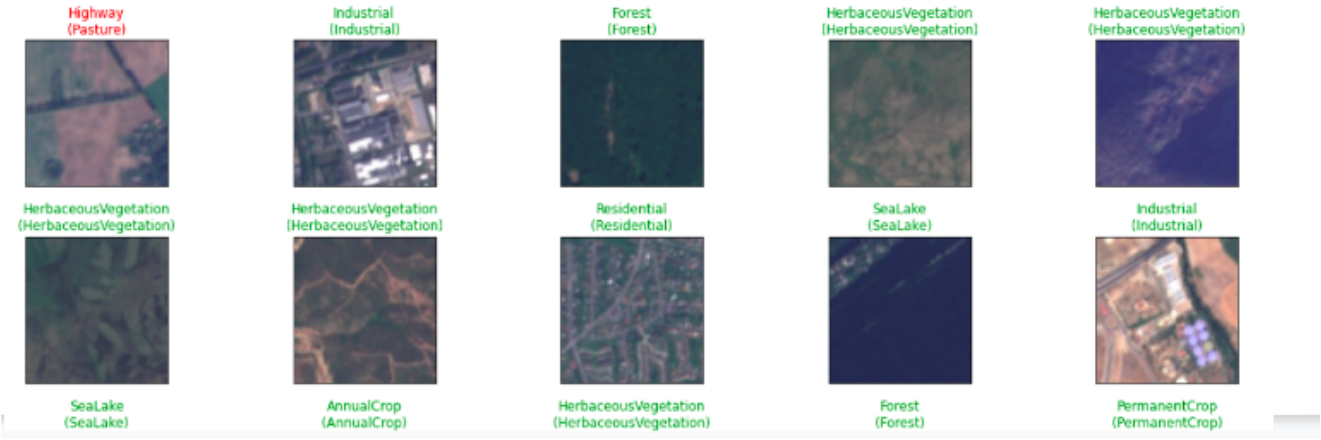
```
plot_history(vgg19_history)
vgg19_model.load_weights('../working/model.weights.best.hdf5')

class_indices = train_generator.class_indices
class_indices = dict((v,k) for k,v in class_indices.items())

test_generator_new = test_gen.flow_from_directory(
   directory=TEST_DIR,
   target_size=(64, 64),
   batch_size=1,
   class_mode=None,
   color_mode='rgb',
   shuffle=False,
   seed=69
)

predictions = vgg19_model.predict_generator(test_generator_new,
steps=len(test_generator_new.filenames))
predicted_classes = np.argmax(np.rint(predictions), axis=1)
true_classes = test_generator_new.classes

prf, conf_mat = display_results(true_classes, predicted_classes,
class_indices.values())
prf
# Save the model and the weights
vgg19_model.save('../working/vgg19_eurosat.h5')
```

Performance,Results and Analysis

Predictions

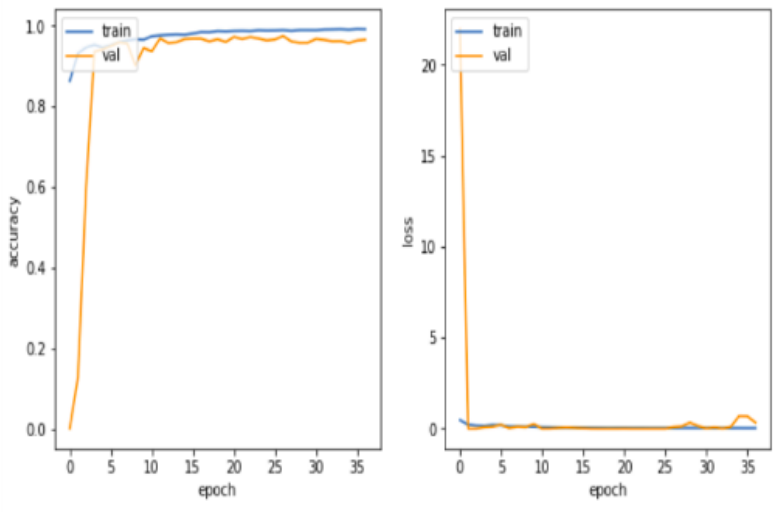
Following predictions was made on some images:

```
plot_predictions(true_classes, predictions, test_generator_new, class_indices)
```



1. Resnet50

- Inbuilt resnet50 model in keras was used.First the dense layer was pre-trained using early stopping and checkpoint.
- The pre-training model ran for 21 epochs as early stopping's patience was set for 10 epochs.
- Then loading the same weights we trained our resnet50 model and ran it for 100 epochs and reduceLronplateau callback was also used in this..
- As we used Early Stopping, the model ran for 37 epochs before the training process stopped. The final accuracy observed was **97.5%**

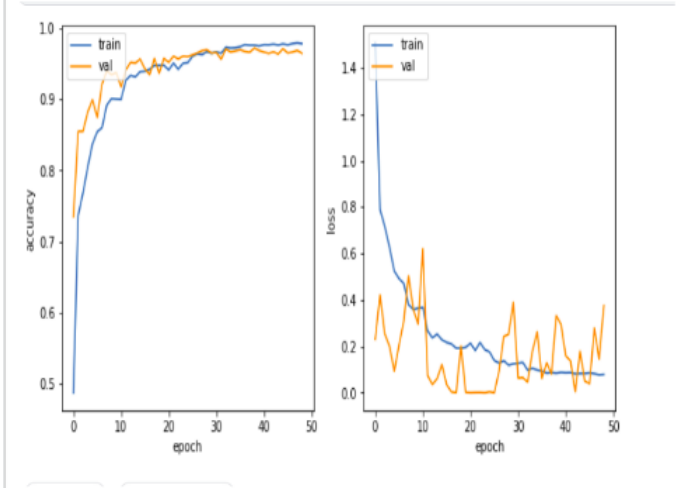



```
Found 5400 images belonging to 10 classes.
Accuracy: 0.975
Global F2 Score: 0.975
```

	Precision	Recall	F-Score	Support
AnnualCrop	0.979133	0.971338	0.975220	628.0
Forest	0.952607	1.000000	0.975728	603.0
HerbaceousVegetation	0.978984	0.960481	0.969644	582.0
Highway	0.986667	0.962825	0.974600	538.0
Industrial	0.985477	0.965447	0.975359	492.0
Pasture	0.986034	0.933862	0.959239	378.0
PermanentCrop	0.956790	0.968750	0.962733	480.0
Residential	0.965152	0.996870	0.980754	639.0
River	0.966926	0.992016	0.979310	501.0
SeaLake	1.000000	0.980322	0.990063	559.0

2. VGG16

- Inbuilt vgg-16 in keras was used starting with pre-training the dense layer for transfer learning.
- The pre-training ran for 50 epochs and the weights were saved in another model for callback.
- Then these weights were used for training the main vgg-16 model which ran for 100 epochs.
- Early stopping, checkpoint and reduceLRonplateau callbacks were used.
- Final Accuracy came to be **97.07%.**

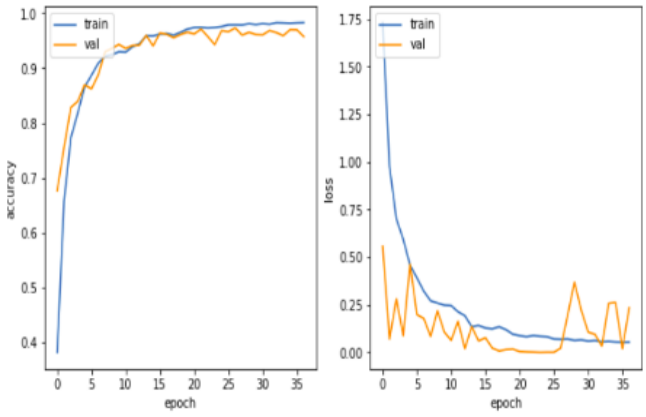



```
Found 5400 images belonging to 10 classes.
Accuracy: 0.9707407407407408
Global F2 Score: 0.9707407407407408
```

	Precision	Recall	F-Score	Support
AnnualCrop	0.950637	0.950637	0.950637	628.0
Forest	0.972358	0.991708	0.981938	603.0
HerbaceousVegetation	0.975352	0.951890	0.963478	582.0
Highway	0.984791	0.962825	0.973684	538.0
Industrial	0.977642	0.977642	0.977642	492.0
Pasture	0.978082	0.944444	0.960969	378.0
PermanentCrop	0.940695	0.958333	0.949432	480.0
Residential	0.963801	1.000000	0.981567	639.0
River	0.974104	0.976048	0.975075	501.0
SeaLake	0.994565	0.982111	0.988299	559.0

3. VGG19

- Inbuilt vgg-19 in keras was used starting with pre-training the dense layer for transfer learning.
- The pre-training ran for 50 epochs and the weights were saved in another model for callback.
- Then these weights were used for training the main vgg-16 model which ran for 100 epochs.
- Early stopping, checkpoint and reduceLRonplateau callbacks were used.
- Final accuracy came to be **97.27%**. Slightly more than vgg-16 but less than resnet-50.




```
Found 5400 images belonging to 10 classes.
Accuracy: 0.9727777777777777
Global F2 Score: 0.9727777777777779
```

	Precision	Recall	F-Score	Support
AnnualCrop	0.966346	0.960191	0.963259	628.0
Forest	0.980424	0.996683	0.988487	603.0
HerbaceousVegetation	0.975395	0.953608	0.964379	582.0
Highway	0.984820	0.964684	0.974648	538.0
Industrial	0.977131	0.955285	0.966084	492.0
Pasture	0.983871	0.968254	0.976000	378.0
PermanentCrop	0.939024	0.962500	0.950617	480.0
Residential	0.960843	0.998435	0.979279	639.0
River	0.966797	0.988024	0.977295	501.0
SeaLake	0.996337	0.973166	0.984615	559.0

Conclusion

In this project, an approach using deep learning methods was explored in order to automatically classify land use and land cover detection of Sentinel-2A satellite image data. After training using three prominent architectures (Resnet-50, VGG16 and VGG19), the best accuracy observed was **97.50% by Resnet-50.** The complete procedure was described from introduction to CNN and its architectures, to collecting data, training and predicting results and accuracy.

References

1. Eurosat: A novel dataset and deep learning benchmark for land use and land cover classification. Patrick Helber, Benjamin Bischke, Andreas Dengel, Damian Borth. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, 2019.
2. VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION Karen Simonyan & Andrew Zisserman ,Visual Geometry Group, Department of Engineering Science, University of Oxford