Remote Sensing Scene Classification using ResNet50, MobileNetV2, VGG16 and DenseNet121

-A comparative study

Samsthidhaa Sree S

Abstract:

Remote sensing scene classification (RSSC) is a hotspot and plays a key role in remote sensing image interpretation in recent years. A substantial advancement in the classification of remote sensing scenes has been accomplished with the recent creation of convolutional neural networks. It is challenging to categorize remote sensing image sceneries because many things combine and associate spatially to create complex and varied settings. It divides the images into various categories such as forests, airports, and rivers according to their contents.

There are totally four networks used in this project namely ResNet50**,** MobileNetV2, VGG16 and DenseNet121. Deep residual networks like the popular ResNet-50 model is a convolutional neural network (CNN) that is 50 layers deep. A Residual Neural Network (ResNet) is an Artificial Neural Network (ANN) of a kind that stacks residual blocks on top of each other to form a network. MobileNet-v2 is a convolutional neural network that is 53 layers deep. DenseNet-121 has 120 Convolutions and 4 Average Pool layers. VGG16 is a variant of VGG model with 16 convolution layers. We intend to train and test data on these distinct pre-trained CNN models from the TensorFlow library and assess their accuracy in comparison. We wish to determine which model fits the given dataset the best and give an unknown image as input. It classifies and returns the class of the given image.

Introduction:

Before moving on to Remote Sensing Scene Classification, let us get to know about Image classification. Basically, Image Classification is the process of feeding an image into a model built using specific algorithms that output the class or likelihood of the image belonging to that class. We humans, distinguish images based on their colour and size, but a machine see them as pixels which has values ranging from 0 to 255. There are numerous models through which the images can be trained and tested but we shift our focus on the below mentioned models.

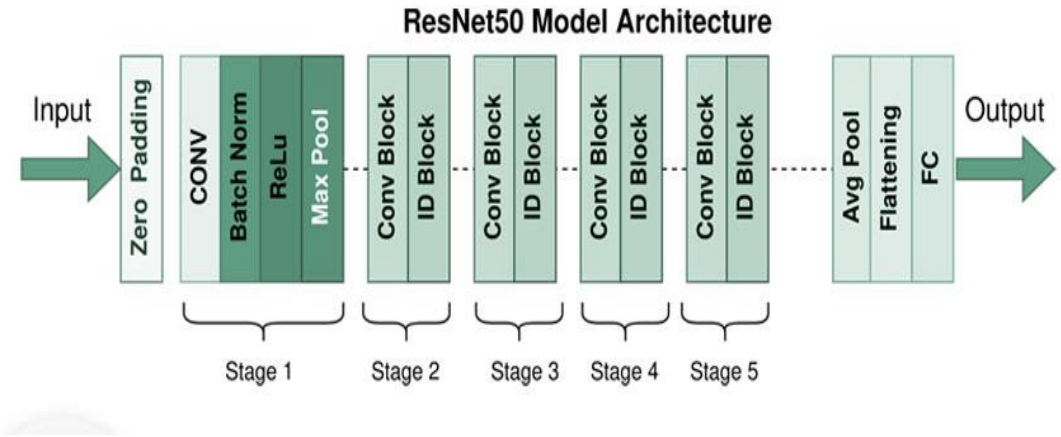
Resnet50

ResNet stands for Residual Network and is a specific type of convolutional neural network (CNN). The ResNet architecture follows two basic design rules. First, the number of filters in each layer is the same depending on the size of the output feature map. Second, if the feature map's size is halved, it has double the number of filters to maintain the time complexity of each layer.

The 50-layer ResNet architecture includes the following elements, as shown in the below:

- A 7×7 kernel convolution alongside 64 other kernels with a 2-sized stride.
- A max pooling layer with a 2-sized stride.
- 9 more layers—3×3,64 kernel convolution, another with 1×1,64 kernels, and a third with 1×1,256 kernels. These 3 layers are repeated 3 times.
- 12 more layers with 1×1,128 kernels, 3×3,128 kernels, and 1×1,512 kernels, iterated 4 times.

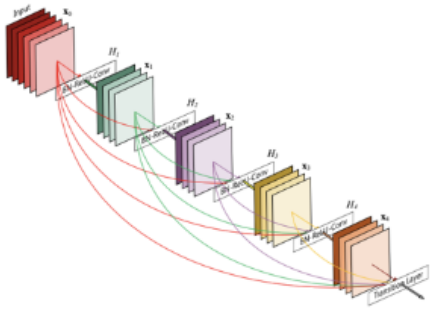
- 18 more layers with 1×1,256 cores, and 2 cores 3×3,256 and 1×1,1024, iterated 6 times.
- 9 more layers with 1×1,512 cores, 3×3,512 cores, and 1×1,2048 cores iterated 3 times.



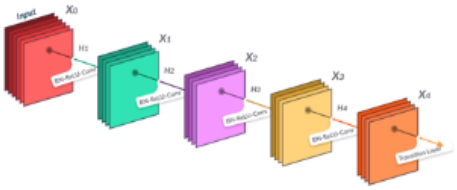
DenseNet121

A **DenseNet** is a type of convolutional neural network that utilises dense connections between layers, through Dense Blocks, where we connect all layers (with matching feature-map sizes) directly with each other.

DenseNet is quite similar to ResNet with some fundamental differences. ResNet uses an additive method (+) that merges the previous layer (identity) with the future layer, whereas DenseNet concatenates (.) the output of the previous layer with the future layer.



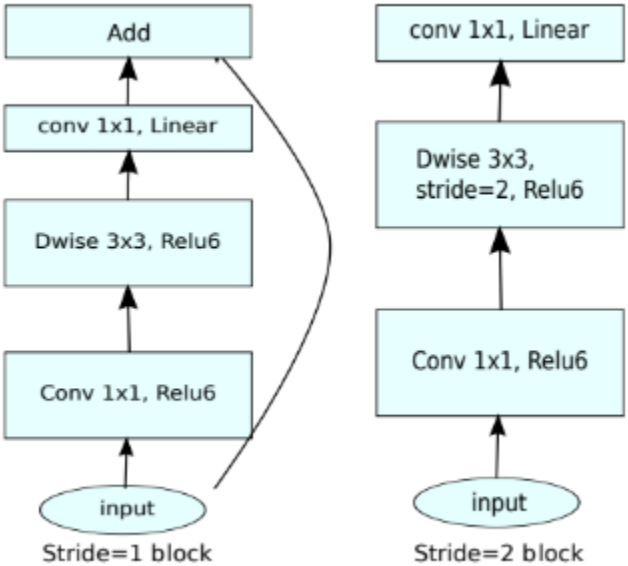
DenseNet Structure



ResNet Structure

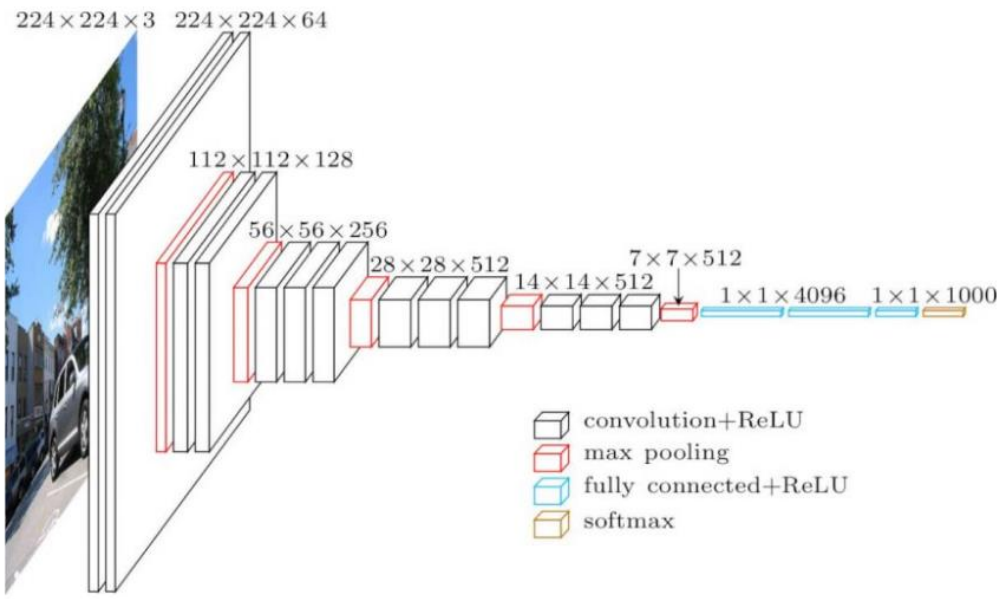
MobileNetV2

MobileNetV2 is a convolutional neural network architecture that seeks to perform well on mobile devices. It is based on an inverted residual structure where the residual connections are between the bottleneck layers. The intermediate expansion layer uses lightweight depth wise convolutions to filter features as a source of non-linearity.



VGG16

The input to the network is an image of dimensions (224, 224, 3). The first two layers have 64 channels of a 3*3 filter size and the same padding. Then after a max pool layer of stride (2, 2), two layers have convolution layers of 128 filter size and filter size (3, 3). This is followed by a max-pooling layer of stride (2, 2) which is the same as the previous layer. Then there are 2 convolution layers of filter size (3, 3) and 256 filters. After that, there are 2 sets of 3 convolution layers and a max pool layer. Each has 512 filters of (3, 3) size with the same padding. This image is then passed to the stack of two convolution layers. In these convolution and max-pooling layers, the filters we use are of the size 3*3. In some of the layers, it also uses 1*1 pixel which is used to manipulate the number of input channels. There is a padding of 1-pixel (same padding) done after each convolution layer to prevent the spatial feature of the image.



Methodology:

- Four different pretrained models are chosen to train the NWPU-RESISC45 dataset which contains 45 class with 700 images in each class.
- The datasets are separated into 2 out of which 90% of it was for training the rest 10% are used for testing.
- The program is built in such a way that the images were trained using Resnet50, MobileNetV2, VGG 16 and DenseNet121 models and both the accuracy and loss of training and testing images were calculated.
- The obtained results of accuracy in each neural network are taken and plotted to see which algorithm is more efficient.

Code: ResNet50

```
import matplotlib.pyplot as plt
import numpy as np
import PIL
import pathlib
import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras import utils
from tensorflow.keras.utils import image_dataset_from_directory
from tensorflow.keras.layers import Dense,Flatten
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam
import keras
from keras import models
from keras.models import Sequential, Model,load_model

train_path=r"C:/Users/sams.sridharan/Downloads/SAM/SAM/train"
train_ds= tf.keras.utils.image_dataset_from_directory('C:/Users/sams.sridharan/Downloads/SAM/SAM/train')
valid_path=r"C:/Users/sams.sridharan/Downloads/SAM/SAM/valid"
val_ds= tf.keras.utils.image_dataset_from_directory('C:/Users/sams.sridharan/Downloads/SAM/SAM/valid')

class_names = train_ds.class_names
print(class_names)

resnet_model = Sequential()

pretrained_model= tf.keras.applications.ResNet50(include_top=False,
                   input_shape=(256,256,3),
                   pooling='avg',classes=3,
                   weights='imagenet')
for layer in pretrained_model.layers:
        layer.trainable=False
```

```
resnet_model.add(pretrained_model)
resnet_model.add(Flatten())
resnet_model.add(Dense(512, activation='relu'))
resnet_model.add(Dense(256, activation='relu'))
resnet_model.add(Dense(128, activation='relu'))
resnet_model.add(Dense(64, activation='relu'))
resnet_model.add(Dense(32, activation='relu'))
resnet_model.add(Dense(45, activation='softmax'))

resnet_model.summary()

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

import scipy
history = resnet_model.fit(train_ds.repeat(),
                           batch_size=16,
                           steps_per_epoch=15,
                           epochs=400,
                           validation_data=val_ds.repeat(),
                           validation_steps=15,
                           verbose=1)
```

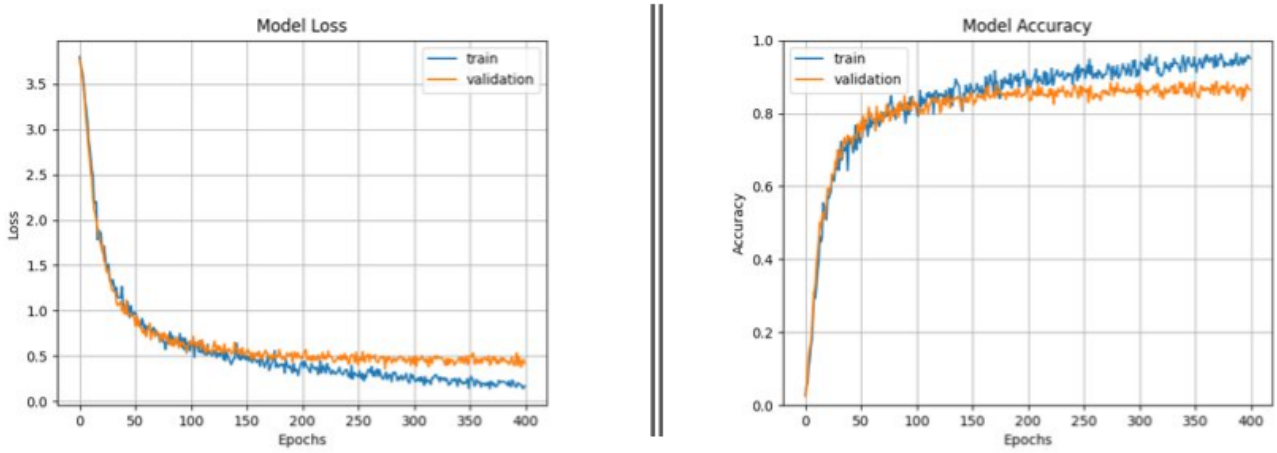
```
pic="C:/Users/sams.sridharan/Downloads/UCMerced_LandUse/Images/beach/beach09.tif"
image=cv2.imread(pic)
image_resized= cv2.resize(image, (256,256))
image=np.expand_dims(image_resized,axis=0)
pred=resnet_model.predict(image)
print(pred)
output_class=class_names[np.argmax(pred)]
print("image is",output_class)


file_name = input("Enter the file name to be saved :)")
resnet_model.save("%s.h5" % file_name)

fig1 = plt.gcf()
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.axis(ymin=0,ymax=1)
plt.grid()
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epochs')
plt.legend(['train', 'validation'])
plt.savefig('resnet 15spe 400epoch acc')
plt.show()


plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.grid()
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epochs')
plt.legend(['train', 'validation'])
plt.savefig('resnet 15spe 400epoch loss')
plt.show()
```

Resnet50 model Accuracy and loss



An image from beach class was given as input and the classification was monitored.

Input Image:

Output:

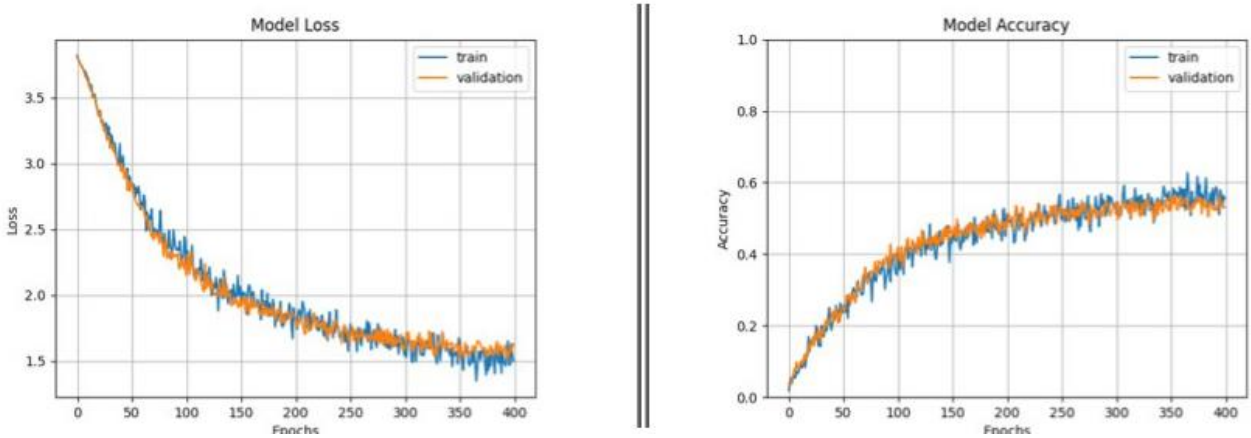
```
[[1.54666865e-04 1.78500137e-03 9.57155949e-04 4.71570669e-03
  2.37442821e-01 1.80526394e-02 1.95719651e-03 1.22715684e-03
  4.93239537e-02 5.65051008e-03 2.04394906e-04 1.15228899e-01
  2.92511415e-02 1.33855632e-02 1.21462808e-05 1.81719903e-02
  1.00298377e-03 5.22702874e-04 1.12382965e-02 2.33022310e-02
  1.97619852e-02 2.21549645e-01 9.21643898e-03 8.88276991e-05
  1.76556199e-03 3.28281941e-03 1.22082548e-03 1.74818898e-03
  5.41983638e-03 7.00428864e-05 2.08924362e-03 2.70748918e-04
  4.25448315e-03 7.48785445e-04 3.48770879e-02 1.63678825e-02
  4.93059633e-03 6.94878697e-02 4.57040928e-02 3.59758246e-03
  7.81439652e-04 1.99974072e-03 1.45951088e-03 1.88321201e-03
  1.38364350e-02]]
image is beach
```

The updated code snippets are attached for the rest three models, disregarding the lines that were duplicated.

MobileNetV2

```
pretrained_model= tf.keras.applications.MobileNetV2(include_top=False,
                    input_shape=(256,256,3),
                    pooling='avg',classes=45,
                    weights='imagenet')
for layer in pretrained_model.layers:
        layer.trainable=False
```

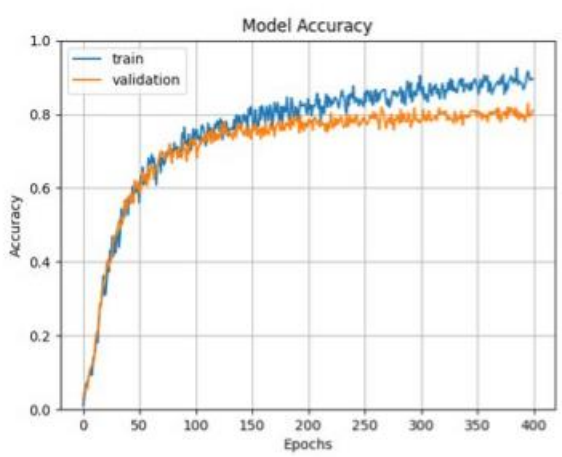
MobileNetV2 model Accuracy and loss

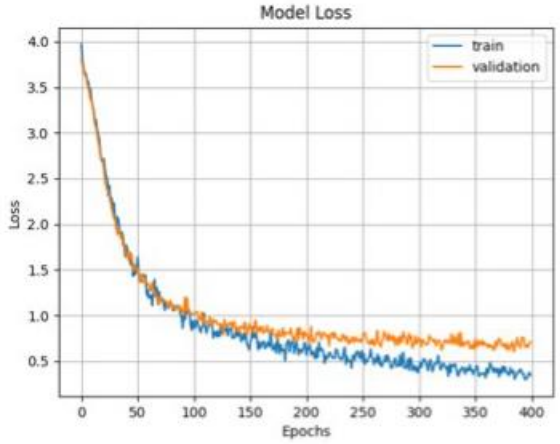


VGG16

```
pretrained_model= tf.keras.applications.VGG16(include_top=False,
                    input_shape=(256,256,3),
                    pooling='avg',classes=3,
                    weights='imagenet')
for layer in pretrained_model.layers:
        layer.trainable=False
```

VGG16 model Accuracy and loss

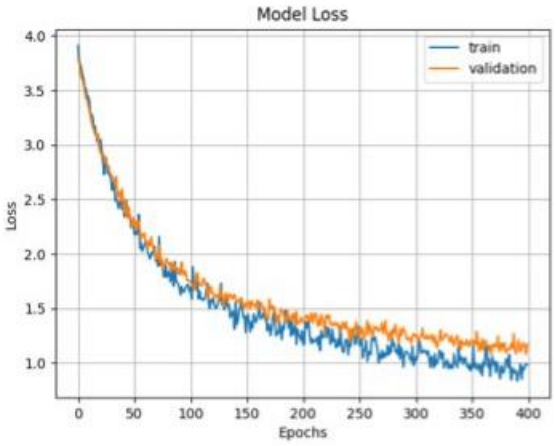


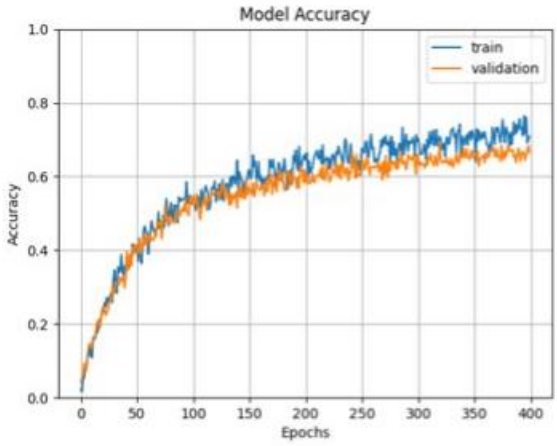


DenseNet121

```
pretrained_model= tf.keras.applications.DenseNet121(include_top=False,
                    input_shape=(256,256,3),
                    pooling='avg',classes=3,
                    weights='imagenet')
for layer in pretrained_model.layers:
        layer.trainable=False
```

DenseNet121 model Accuracy and loss





Accuracy Comparison:

Model	Accuracy
ResNet50	**90%**
MobileNetV2	**56%**
VGG16	**82%**
DenseNet121	**66%**

Conclusion:

We have conducted a series of testing and training on the NWPU-RESISC45 dataset which was created by a research team of the Northwestern Polytechnic University using Google Earth Imagery in 2017. It contains 45 classes. Every class includes 700 RGB images and the size of each image is 256×256 pixels. As mentioned, the dataset is trained on four different models to predict the loss and accuracy of each model thereby concluding the efficient model among these. Among these 4 models, the Resnet50 model has attained a higher accuracy rate and minimum loss compared to the other 3. Thus, we conclude that Resnet50 is the efficient network among these 4 for classifying Remote Sensing images. We used ResNet50 for testing an image and thus gave an image belonging to beach class. The image was successfully classified to that particular class by the ResNet50 model.