CIS 731: ANN - Homework 3

by: **Prathma Rastogi (409745940)**

Image Classification using Convolutional Neural Network

Required libraries

In [177]:

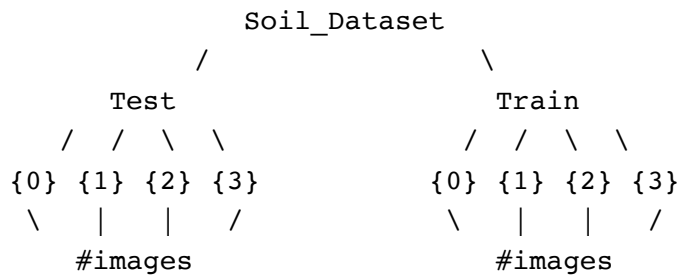
```
import tensorflow as tf
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
from tensorflow.keras.layers import Dropout, MaxPooling2D, AveragePooling2D, Den
se, Flatten, Input, Conv2D, add, Activation
from tensorflow.keras.layers import (Dense, Dropout, Activation, Flatten, Reshap
e, Layer, BatchNormalization, LocallyConnected2D, ZeroPadding2D, Conv2D, MaxPool
ing2D, Conv2DTranspose,AveragePooling2D, GaussianNoise, UpSampling2D, Input)
from tensorflow.keras.utils import plot_model
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.models import Sequential , Model , load_model
from tensorflow.keras.preprocessing.image import load_img , img_to_array , Image
DataGenerator
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.optimizers import Adam, SGD, RMSprop
from tensorflow.keras.callbacks import ModelCheckpoint, ReduceLROnPlateau, Early
Stopping
from keras import applications
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelBinarizer
from sklearn.metrics import classification_report, confusion_matrix
from mlxtend.plotting import plot_confusion_matrix

import time
import numpy as np
import warnings
import matplotlib.pyplot as plt
from PIL import Image
import os
import pandas as pd
from skimage import io
```

Dataset Preprocessing

Dataset source: [https://www.kaggle.com/omkargurav/soil-classification-image-data (https://www.kaggle.com/omkargurav/soil-classification-image-data)](https://www.kaggle.com/omkargurav/soil-classification-image-data)

This dataset comprises of different types of soils classified into categories: *{0} Alluvial Soil, {1} Black Soil, {2} Clay Soil, and {3} Red Soil*

Structure of dataset folder:



Soils can vary by textures, color, etc. This prediction on different soils can help in boosting agriculture and study information on soils.

In [127]:

```
test_dir = "Soil_Dataset/Test"
train_dir = "Soil_Dataset/Train"
```

Keras class ***ImageDataGenerator*** is used for generating batches of tensor images with random transformations and augmentations for training.\ Here I rescaled images for normalization.\ I didn't use any other augmentation as soil images if augmented using shift or rotations will remain same.

In [128]:

```
train_datagen = ImageDataGenerator(rescale = 1./255)
test_datagen = ImageDataGenerator(rescale = 1./255)
```

Created separated iterators for train and test data.\ As the dataset comprises of 4 classes, it belongs to *class_mode*: **categorical** .\ Batch size of 64 is taken.\ Images are resized to 150 * 150.

There are total 715 images belong to training data and 188 images belong to test data, i.e., dataset is divided into 80% training and 20% test set.

In [129]:

```
train_it = train_datagen.flow_from_directory(train_dir, target_size=(150,150), c
lass_mode='categorical', batch_size=64, shuffle=True)

test_it = test_datagen.flow_from_directory(test_dir, target_size=(150,150), clas
s_mode='categorical', batch_size=64, shuffle=False)
```

```
Found 715 images belonging to 4 classes.
Found 188 images belonging to 4 classes.
```

Simple (basemodel) CNN

Implementing simple CNN model:

- Creating 5 layers with input and activation function.
- ReLu is added to each layer so that any negative values will not be passed to next layer.
- Max Pooling is done after every convolutional layer to reduce the dimension of feature maps.
- On top of CNN, creating flatten layer to prepare the vectors for fully connected layers.
- Then, Dense layers are added.
- Last layer is 4 units dense because, there are 4 classes to predict from in the end.

In [81]:

```
simple_model = Sequential()
simple_model.add(Conv2D(64, (3, 3), activation='relu', input_shape=(150, 150, 3
)))
simple_model.add(MaxPooling2D((2, 2)))
simple_model.add(Conv2D(64, (3, 3), activation='relu'))
simple_model.add(MaxPooling2D((2, 2)))
simple_model.add(Conv2D(64, (3, 3), activation='relu'))

simple_model.add(Flatten())
simple_model.add(Dense(64, activation='relu'))
simple_model.add(Dense(4, activation='softmax'))
```

Summary of model

Trainable Parameters: 4,810,948

In [82]:

```
simple_model.summary()
```

```
Model: "sequential_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_33 (Conv2D)           (None, 148, 148, 64)      1792
_________________________________________________________________
max_pooling2d_14 (MaxPooling (None, 74, 74, 64)        0
_________________________________________________________________
conv2d_34 (Conv2D)           (None, 72, 72, 64)        36928
_________________________________________________________________
max_pooling2d_15 (MaxPooling (None, 36, 36, 64)        0
_________________________________________________________________
conv2d_35 (Conv2D)           (None, 34, 34, 64)        36928
_________________________________________________________________
flatten_8 (Flatten)          (None, 73984)             0
_________________________________________________________________
dense_16 (Dense)             (None, 64)                4735040
_________________________________________________________________
dense_17 (Dense)             (None, 4)                 260
=================================================================
Total params: 4,810,948
Trainable params: 4,810,948
Non-trainable params: 0
_________________________________________________________________
```

Model is compiled and fit

In [83]:

```
simple_model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

In [84]:

```
start = time.time()

simple_history = simple_model.fit_generator(train_it, epochs=20, validation_data
= test_it)
end = time.time()
print("Total train time: ",(end-start)/60," mins")
```

```
Epoch 1/20
12/12 [==============================] - 18s 2s/step - loss: 1.1847
- accuracy: 0.5245 - val_loss: 0.5895 - val_accuracy: 0.7660
Epoch 2/20
12/12 [==============================] - 17s 1s/step - loss: 0.5101
- accuracy: 0.8000 - val_loss: 0.9707 - val_accuracy: 0.6543
Epoch 3/20
12/12 [==============================] - 17s 1s/step - loss: 0.3770
- accuracy: 0.8238 - val_loss: 0.3680 - val_accuracy: 0.8777
Epoch 4/20
12/12 [==============================] - 17s 1s/step - loss: 0.2642
- accuracy: 0.8937 - val_loss: 0.2772 - val_accuracy: 0.8936
Epoch 5/20
12/12 [==============================] - 18s 1s/step - loss: 0.1912
- accuracy: 0.9287 - val_loss: 0.2239 - val_accuracy: 0.9149
Epoch 6/20
12/12 [==============================] - 17s 1s/step - loss: 0.1598
- accuracy: 0.9413 - val_loss: 0.2846 - val_accuracy: 0.9043
Epoch 7/20
12/12 [==============================] - 17s 1s/step - loss: 0.1228
- accuracy: 0.9469 - val_loss: 0.2219 - val_accuracy: 0.9096
Epoch 8/20
12/12 [==============================] - 17s 1s/step - loss: 0.0688
- accuracy: 0.9762 - val_loss: 0.2336 - val_accuracy: 0.9096
Epoch 9/20
12/12 [==============================] - 17s 1s/step - loss: 0.0454
- accuracy: 0.9874 - val_loss: 0.4087 - val_accuracy: 0.8883
Epoch 10/20
12/12 [==============================] - 18s 2s/step - loss: 0.0686
- accuracy: 0.9734 - val_loss: 0.2960 - val_accuracy: 0.9096
Epoch 11/20
12/12 [==============================] - 17s 1s/step - loss: 0.0527
- accuracy: 0.9790 - val_loss: 0.6706 - val_accuracy: 0.8617
Epoch 12/20
12/12 [==============================] - 17s 1s/step - loss: 0.0964
- accuracy: 0.9678 - val_loss: 0.5082 - val_accuracy: 0.8830
Epoch 13/20
12/12 [==============================] - 17s 1s/step - loss: 0.0959
- accuracy: 0.9650 - val_loss: 0.4072 - val_accuracy: 0.8883
Epoch 14/20
12/12 [==============================] - 19s 2s/step - loss: 0.0495
- accuracy: 0.9832 - val_loss: 0.4334 - val_accuracy: 0.8883
Epoch 15/20
12/12 [==============================] - 18s 1s/step - loss: 0.0258
- accuracy: 0.9958 - val_loss: 0.6313 - val_accuracy: 0.8723
Epoch 16/20
12/12 [==============================] - 18s 2s/step - loss: 0.1470
- accuracy: 0.9413 - val_loss: 0.4589 - val_accuracy: 0.8936
Epoch 17/20
12/12 [==============================] - 18s 2s/step - loss: 0.0934
- accuracy: 0.9706 - val_loss: 0.2216 - val_accuracy: 0.9255
Epoch 18/20
12/12 [==============================] - 19s 2s/step - loss: 0.0471
- accuracy: 0.9846 - val_loss: 0.3139 - val_accuracy: 0.9202
Epoch 19/20
12/12 [==============================] - 18s 2s/step - loss: 0.0326
- accuracy: 0.9888 - val_loss: 0.3030 - val_accuracy: 0.9096
Epoch 20/20
12/12 [==============================] - 17s 1s/step - loss: 0.0235
- accuracy: 0.9902 - val_loss: 0.4273 - val_accuracy: 0.8883
Total train time:  6.517240349451701  mins
```

Performance

Performance of the model is evaluated on the basis of Accuracy and Cross Entropy results on test dataset.\
Performance output is shown using confusion matrix.\ Also, comparison of Accuracy and Cross Entropy results shown between train and test data using graph plots.

Accuracy on test data ~ 86%

In [107]:

```
score = model.evaluate_generator(test_it)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
WARNING:tensorflow:From <ipython-input-107-3ae0b742d680>:1: Model.ev
aluate_generator (from tensorflow.python.keras.engine.training) is d
eprecated and will be removed in a future version.
Instructions for updating:
Please use Model.evaluate, which supports generators.
Test loss: 4.162484645843506
Test accuracy: 0.8617021441459656
```

Confusion Matrix & Classification report

Precision, Recall, f1-score results.

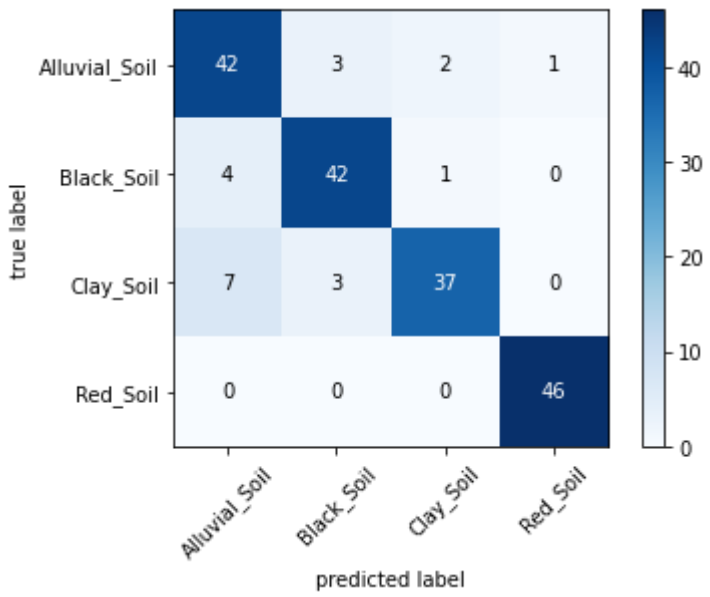
In [320]:

```
#Confusion Matrix and Classification Report
num_of_test_samples = 188
Y_pred = simple_model.predict_generator(test_it, num_of_test_samples)
y_pred = np.argmax(Y_pred, axis=1)
simple_cm = confusion_matrix(test_it.classes, y_pred)
print('Confusion Matrix')
print(simple_cm)
print('Classification Report')
target_names = ['Alluvial_Soil', 'Black_Soil', 'Clay_Soil', 'Red_Soil']
print(classification_report(test_it.classes, y_pred, target_names=target_names))
```

```
WARNING:tensorflow:Your input ran out of data; interrupting trainin
g. Make sure that your dataset or generator can generate at least `s
teps_per_epoch * epochs` batches (in this case, 188 batches). You ma
y need to use the repeat() function when building your dataset.
Confusion Matrix
[[42  3  2  1]
 [ 4 42  1  0]
 [ 7  3 37  0]
 [ 0  0  0 46]]
Classification Report
              precision    recall  f1-score   support

Alluvial_Soil       0.79      0.88      0.83        48
   Black_Soil       0.88      0.89      0.88        47
    Clay_Soil       0.93      0.79      0.85        47
     Red_Soil       0.98      1.00      0.99        46

     accuracy                           0.89       188
    macro avg       0.89      0.89      0.89       188
 weighted avg       0.89      0.89      0.89       188
```

In [326]:

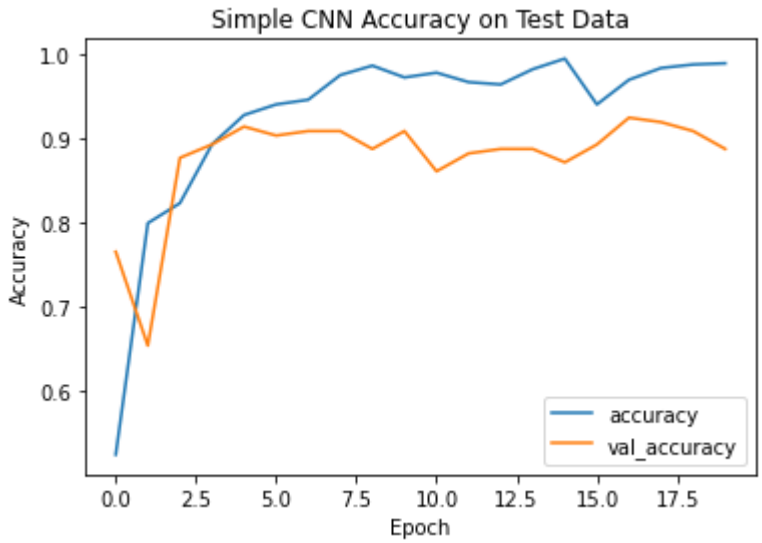
```
plot_confusion_matrix(simple_cm, colorbar=True, class_names=target_names)
plt.show()
```



Accuracy & Cross Entropy Loss result plots of Train vs Test Data

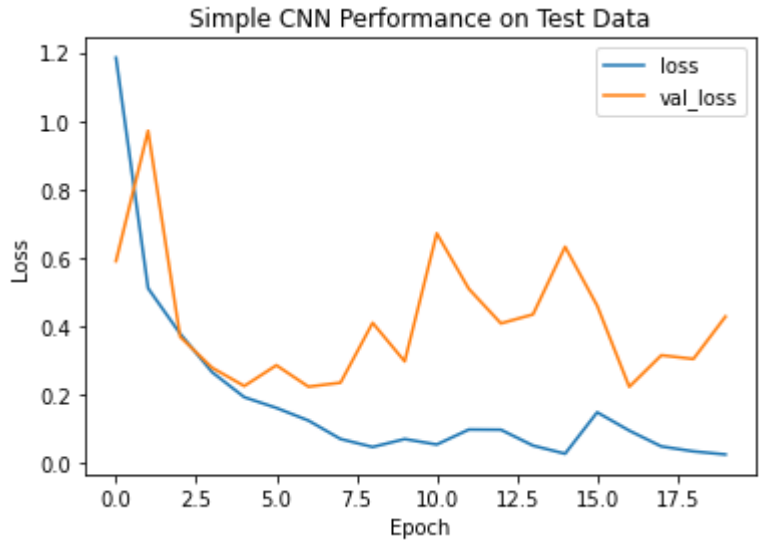
In [281]:

```
plt.plot(simple_history.history["accuracy"])
plt.plot(simple_history.history['val_accuracy'])
plt.title("Simple CNN Accuracy on Test Data")
plt.ylabel("Accuracy")
plt.xlabel("Epoch")
plt.legend(["accuracy","val_accuracy"], loc='lower right')
plt.show()
```



In [282]:

```
plt.plot(simple_history.history["loss"])
plt.plot(simple_history.history['val_loss'])
plt.title("Simple CNN Performance on Test Data")
plt.ylabel("Loss")
plt.xlabel("Epoch")
plt.legend(["loss","val_loss"], loc='upper right')
plt.show()
```



Prediction Output

In [297]:

```
simple_predSet = simple_model.predict_generator(test_it,verbose=0)
simple_predicted_class_indices=np.argmax(simple_predSet,axis=1)
labels = (test_it.class_indices)
labels = dict((v,k) for k,v in labels.items())
simple_predictions = [labels[k] for k in simple_predicted_class_indices]
filenames=test_it.filenames
results=pd.DataFrame({"Filename":filenames,
                      "Predictions":simple_predictions})
print(labels)
results[:50]
```

```
{0: 'Alluvial_Soil', 1: 'Black_Soil', 2: 'Clay_Soil', 3: 'Red_Soil'}
```

Out[297]:

	Filename	Predictions
0	Alluvial_Soil/Alluvial_1.jpg	Alluvial_Soil
1	Alluvial_Soil/Alluvial_10.jpg	Alluvial_Soil
2	Alluvial_Soil/Alluvial_11.jpg	Clay_Soil
3	Alluvial_Soil/Alluvial_12.jpg	Alluvial_Soil
4	Alluvial_Soil/Alluvial_13.jpg	Alluvial_Soil
5	Alluvial_Soil/Alluvial_14.jpg	Alluvial_Soil
6	Alluvial_Soil/Alluvial_15.jpg	Alluvial_Soil
7	Alluvial_Soil/Alluvial_16.jpg	Alluvial_Soil
8	Alluvial_Soil/Alluvial_17.jpg	Alluvial_Soil
9	Alluvial_Soil/Alluvial_18.jpg	Black_Soil
10	Alluvial_Soil/Alluvial_19.jpg	Alluvial_Soil
11	Alluvial_Soil/Alluvial_2.jpg	Alluvial_Soil
12	Alluvial_Soil/Alluvial_20.jpg	Alluvial_Soil
13	Alluvial_Soil/Alluvial_21.jpg	Alluvial_Soil
14	Alluvial_Soil/Alluvial_22.jpg	Alluvial_Soil
15	Alluvial_Soil/Alluvial_23.jpg	Alluvial_Soil
16	Alluvial_Soil/Alluvial_24.jpg	Alluvial_Soil
17	Alluvial_Soil/Alluvial_25.jpg	Alluvial_Soil
18	Alluvial_Soil/Alluvial_26.jpg	Black_Soil
19	Alluvial_Soil/Alluvial_27.jpg	Alluvial_Soil
20	Alluvial_Soil/Alluvial_28.jpg	Alluvial_Soil
21	Alluvial_Soil/Alluvial_29.jpg	Alluvial_Soil
22	Alluvial_Soil/Alluvial_3.jpg	Alluvial_Soil
23	Alluvial_Soil/Alluvial_30.jpg	Black_Soil
24	Alluvial_Soil/Alluvial_31.jpg	Alluvial_Soil
25	Alluvial_Soil/Alluvial_32.jpg	Alluvial_Soil
26	Alluvial_Soil/Alluvial_33.jpg	Alluvial_Soil
27	Alluvial_Soil/Alluvial_34.jpg	Alluvial_Soil
28	Alluvial_Soil/Alluvial_35.jpg	Clay_Soil
29	Alluvial_Soil/Alluvial_36.jpg	Alluvial_Soil
30	Alluvial_Soil/Alluvial_37.jpg	Alluvial_Soil
31	Alluvial_Soil/Alluvial_38.jpg	Alluvial_Soil
32	Alluvial_Soil/Alluvial_39.jpg	Alluvial_Soil
33	Alluvial_Soil/Alluvial_4.jpg	Alluvial_Soil
34	Alluvial_Soil/Alluvial_40.jpg	Alluvial_Soil
35	Alluvial_Soil/Alluvial_41.jpg	Alluvial_Soil

	Filename	Predictions
36	Alluvial_Soil/Alluvial_42.jpg	Alluvial_Soil
37	Alluvial_Soil/Alluvial_43.jpg	Alluvial_Soil
38	Alluvial_Soil/Alluvial_44.jpg	Alluvial_Soil
39	Alluvial_Soil/Alluvial_45.jpg	Alluvial_Soil
40	Alluvial_Soil/Alluvial_46.jpg	Alluvial_Soil
41	Alluvial_Soil/Alluvial_47.jpg	Alluvial_Soil
42	Alluvial_Soil/Alluvial_48.jpg	Alluvial_Soil
43	Alluvial_Soil/Alluvial_5.jpg	Alluvial_Soil
44	Alluvial_Soil/Alluvial_6.jpg	Alluvial_Soil
45	Alluvial_Soil/Alluvial_7.jpg	Red_Soil
46	Alluvial_Soil/Alluvial_8.jpg	Alluvial_Soil
47	Alluvial_Soil/Alluvial_9.jpg	Alluvial_Soil
48	Black_Soil/Black_1.jpg	Black_Soil
49	Black_Soil/Black_10.jpg	Black_Soil

Second batch of 64 images is taken from test data created using ImageDataGenerator and validated predictions of those images with model output.

In [300]:

```
x,y = test_it.next()
simple_inputSet=[]
simple_actualLabelsSet=[]
simple_predictionsSet = []

for i in range(0,64):
    simple_inputSet.append(test_it[1][0][i])
    simple_actualLabelsSet.append(test_it[1][1][i])
    simple_predictionsSet.append(vgg_predictions[64+i])

plt.figure(figsize=(16,16))
for n in range(16):
    ax = plt.subplot(4,4,n+1)
    plt.imshow(simple_inputSet[n])
    plt.title("Actual:"+labels[simple_actualLabelsSet[n].argmax()]+"\n Predicte
d:"+str(simple_predictions[n]))
    plt.axis('off')
```

Actual:Black_Soil
Predicted:Alluvial_Soil

Actual:Black_Soil
Predicted:Alluvial_Soil

Actual:Black_Soil
Predicted:Clay_Soil

Actual:Black_Soil
Predicted:Alluvial_Soil

Actual:Black_Soil
Predicted:Alluvial_Soil

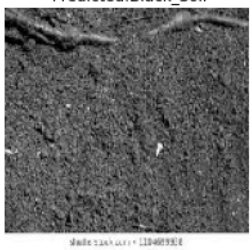
Actual:Black_Soil
Predicted:Alluvial_Soil

Actual:Black_Soil
Predicted:Alluvial_Soil

Actual:Black_Soil
Predicted:Alluvial_Soil

Actual:Black_Soil
Predicted:Alluvial_Soil

Actual:Black_Soil
Predicted:Black_Soil



Actual:Black_Soil
Predicted:Alluvial_Soil

Actual:Black_Soil
Predicted:Alluvial_Soil

Actual:Black_Soil
Predicted:Alluvial_Soil

Actual:Black_Soil
Predicted:Alluvial_Soil

Actual:Black_Soil
Predicted:Alluvial_Soil

Actual:Black_Soil
Predicted:Alluvial_Soil

Complex CNN with 8 layers

Implementing CNN of 8 layers with inputs:

- Creating 8 layers with input and activation function.
- ReLu is added to each layer so that any negative values will not be passed to next layer.
- Padding is added on the edges so that important features information will not be missed.
- Batch Normalization is applied after every Conv2D layers to normalize the input passing to next layer.
- Average Pooling is done after every convolutional layer to reduce the dimension of feature maps.
- Dropout is added for regularization and reduce overfitting of model.
- On top of CNN, creating flatten layer to prepare the vectors for fully connected layers.
- Then, Dense layers are added.
- Last layer is 4 units dense with softmax activation because, there are 4 classes to predict from in the end.

In [114]:

```
model = Sequential()
model.add(Conv2D(64,(3,3),activation = "relu",padding ="same",kernel_initializer
="he_normal", input_shape=(150,150,3)))
model.add(Conv2D(64,(3,3),activation = "relu",padding ="same",kernel_initializer
="he_normal"))
model.add(BatchNormalization())


model.add(AveragePooling2D(pool_size = (2,2), strides=2))
model.add(Dropout(0.5))
model.add(BatchNormalization())


model.add(Conv2D(128,(3,3),activation = "relu",padding ="same",kernel_initialize
r="he_normal"))
model.add(Conv2D(128,(3,3),activation = "relu",padding ="same",kernel_initialize
r="he_normal"))
model.add(BatchNormalization())

model.add(AveragePooling2D(pool_size = (2,2), strides=2))
model.add(Dropout(0.5))
model.add(BatchNormalization())


model.add(Conv2D(256,(3,3),activation = "relu", padding ="same",kernel_initializ
er="he_normal"))
model.add(Conv2D(256,(3,3),activation = "relu",padding ="same",kernel_initialize
r="he_normal"))
model.add(BatchNormalization())

model.add(AveragePooling2D(pool_size = (2,2), strides=2))
model.add(Dropout(0.5))
model.add(BatchNormalization())

model.add(Flatten())
model.add(Dense(512,activation="relu"))
model.add(Dropout(0.7))
model.add(Dense(4,activation="softmax"))
```

Summary of model

Trainable Parameters: 43,617,092

In [115]:

```
model.summary()
```

```
Model: "sequential_10"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_36 (Conv2D)           (None, 150, 150, 64)      1792
_________________________________________________________________
conv2d_37 (Conv2D)           (None, 150, 150, 64)      36928
_________________________________________________________________
batch_normalization_12 (Batc (None, 150, 150, 64)      256
_________________________________________________________________
average_pooling2d_6 (Average (None, 75, 75, 64)        0
_________________________________________________________________
dropout_8 (Dropout)          (None, 75, 75, 64)        0
_________________________________________________________________
batch_normalization_13 (Batc (None, 75, 75, 64)        256
_________________________________________________________________
conv2d_38 (Conv2D)           (None, 75, 75, 128)       73856
_________________________________________________________________
conv2d_39 (Conv2D)           (None, 75, 75, 128)       147584
_________________________________________________________________
batch_normalization_14 (Batc (None, 75, 75, 128)       512
_________________________________________________________________
average_pooling2d_7 (Average (None, 37, 37, 128)       0
_________________________________________________________________
dropout_9 (Dropout)          (None, 37, 37, 128)       0
_________________________________________________________________
batch_normalization_15 (Batc (None, 37, 37, 128)       512
_________________________________________________________________
conv2d_40 (Conv2D)           (None, 37, 37, 256)       295168
_________________________________________________________________
conv2d_41 (Conv2D)           (None, 37, 37, 256)       590080
_________________________________________________________________
batch_normalization_16 (Batc (None, 37, 37, 256)       1024
_________________________________________________________________
average_pooling2d_8 (Average (None, 18, 18, 256)       0
_________________________________________________________________
dropout_10 (Dropout)         (None, 18, 18, 256)       0
_________________________________________________________________
batch_normalization_17 (Batc (None, 18, 18, 256)       1024
_________________________________________________________________
flatten_9 (Flatten)          (None, 82944)             0
_________________________________________________________________
dense_18 (Dense)             (None, 512)               42467840
_________________________________________________________________
dropout_11 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_19 (Dense)             (None, 4)                 2052
=================================================================
Total params: 43,618,884
Trainable params: 43,617,092
Non-trainable params: 1,792
_________________________________________________________________
```

Model is compiled and fit

In [117]:

```
model.compile(optimizer='adam',loss="categorical_crossentropy",metrics=["accurac
y"])
```

In [118]:

```
start = time.time()

history = model.fit_generator(train_it, epochs=20, validation_data = test_it)
end = time.time()
print("Total train time: ",(end-start)/60," mins")
```

```
Epoch 1/20
12/12 [==============================] - 176s 15s/step - loss: 4.473
7 - accuracy: 0.7497 - val_loss: 29.5991 - val_accuracy: 0.4681
Epoch 2/20
12/12 [==============================] - 158s 13s/step - loss: 3.729
9 - accuracy: 0.8308 - val_loss: 69.1792 - val_accuracy: 0.4840
Epoch 3/20
12/12 [==============================] - 145s 12s/step - loss: 3.063
2 - accuracy: 0.8783 - val_loss: 21.2583 - val_accuracy: 0.5745
Epoch 4/20
12/12 [==============================] - 144s 12s/step - loss: 2.875
9 - accuracy: 0.8699 - val_loss: 27.9526 - val_accuracy: 0.5957
Epoch 5/20
12/12 [==============================] - 146s 12s/step - loss: 2.758
4 - accuracy: 0.8503 - val_loss: 43.8700 - val_accuracy: 0.4149
Epoch 6/20
12/12 [==============================] - 147s 12s/step - loss: 1.500
1 - accuracy: 0.8713 - val_loss: 48.2972 - val_accuracy: 0.4043
Epoch 7/20
12/12 [==============================] - 155s 13s/step - loss: 1.040
0 - accuracy: 0.9245 - val_loss: 14.5683 - val_accuracy: 0.5851
Epoch 8/20
12/12 [==============================] - 144s 12s/step - loss: 1.034
7 - accuracy: 0.8923 - val_loss: 6.3664 - val_accuracy: 0.6862
Epoch 9/20
12/12 [==============================] - 180s 15s/step - loss: 0.921
9 - accuracy: 0.9245 - val_loss: 6.0004 - val_accuracy: 0.6702
Epoch 10/20
12/12 [==============================] - 181s 15s/step - loss: 0.469
2 - accuracy: 0.9287 - val_loss: 4.7636 - val_accuracy: 0.7128
Epoch 11/20
12/12 [==============================] - 157s 13s/step - loss: 2.462
1 - accuracy: 0.8420 - val_loss: 16.7752 - val_accuracy: 0.4043
Epoch 12/20
12/12 [==============================] - 203s 17s/step - loss: 1.520
1 - accuracy: 0.8112 - val_loss: 10.0474 - val_accuracy: 0.4894
Epoch 13/20
12/12 [==============================] - 173s 14s/step - loss: 1.533
5 - accuracy: 0.8685 - val_loss: 22.7445 - val_accuracy: 0.5585
Epoch 14/20
12/12 [==============================] - 169s 14s/step - loss: 0.830
1 - accuracy: 0.8797 - val_loss: 36.7583 - val_accuracy: 0.5585
Epoch 15/20
12/12 [==============================] - 214s 18s/step - loss: 0.910
9 - accuracy: 0.9007 - val_loss: 7.2518 - val_accuracy: 0.7447
Epoch 16/20
12/12 [==============================] - 233s 19s/step - loss: 0.798
6 - accuracy: 0.8937 - val_loss: 14.7363 - val_accuracy: 0.6755
Epoch 17/20
12/12 [==============================] - 196s 16s/step - loss: 0.662
7 - accuracy: 0.8853 - val_loss: 20.6452 - val_accuracy: 0.6277
Epoch 18/20
12/12 [==============================] - 184s 15s/step - loss: 0.407
4 - accuracy: 0.9245 - val_loss: 13.2161 - val_accuracy: 0.6862
Epoch 19/20
12/12 [==============================] - 162s 14s/step - loss: 0.324
6 - accuracy: 0.9441 - val_loss: 12.4079 - val_accuracy: 0.7021
Epoch 20/20
12/12 [==============================] - 154s 13s/step - loss: 0.309
4 - accuracy: 0.9315 - val_loss: 3.4216 - val_accuracy: 0.7394
Total train time:  62.02657202084859  mins
```

Performance

Performance of the model is evaluated on the basis of Accuracy and Cross Entropy results on test dataset.\
Performance output is shown using confusion matrix.\ Also, comparison of Accuracy and Cross Entropy results shown between train and test data using graph plots.

Accuracy on test data ~ 74%

In [119]:

```
score = model.evaluate_generator(test_it)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
Test loss: 3.4215548038482666
Test accuracy: 0.7393617033958435
```

Confusion Matrix & Classification report

Precision, Recall, f1-score results.

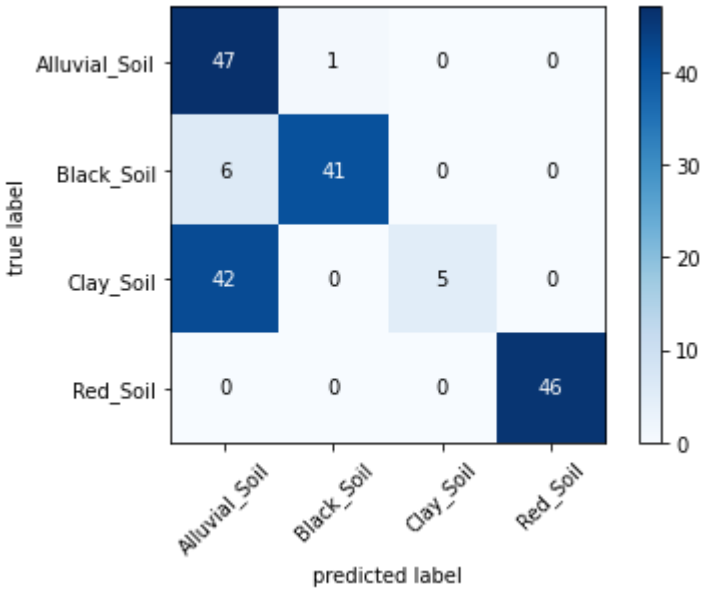
In [334]:

```
#Confution Matrix and Classification Report
num_of_test_samples = 188
Y_pred = model.predict_generator(test_it, num_of_test_samples)
y_pred = np.argmax(Y_pred, axis=1)
cm = confusion_matrix(test_it.classes, y_pred)
print('Confusion Matrix')
print(cm)
print('Classification Report')
target_names = ['Alluvial_Soil', 'Black_Soil', 'Clay_Soil', 'Red_Soil']
print(classification_report(test_it.classes, y_pred, target_names=target_names))
```

```
WARNING:tensorflow:Your input ran out of data; interrupting trainin
g. Make sure that your dataset or generator can generate at least `s
teps_per_epoch * epochs` batches (in this case, 188 batches). You ma
y need to use the repeat() function when building your dataset.
Confusion Matrix
[[47  1  0  0]
 [ 6 41  0  0]
 [42  0  5  0]
 [ 0  0  0 46]]
Classification Report
              precision    recall  f1-score   support

Alluvial_Soil       0.49      0.98      0.66        48
   Black_Soil       0.98      0.87      0.92        47
    Clay_Soil       1.00      0.11      0.19        47
     Red_Soil       1.00      1.00      1.00        46

     accuracy                           0.74       188
    macro avg       0.87      0.74      0.69       188
 weighted avg       0.87      0.74      0.69       188
```

In [335]:

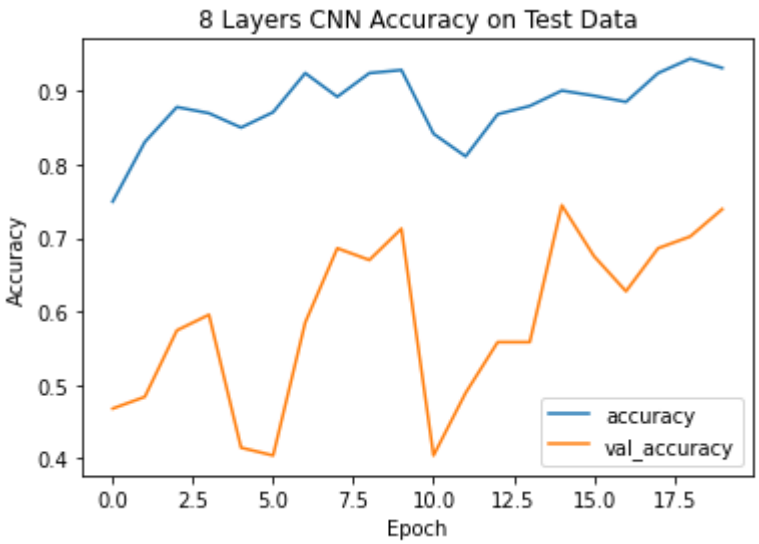
```
plot_confusion_matrix(cm, colorbar=True, class_names=target_names)
plt.show()
```



Accuracy & Cross Entropy result plots of Train vs Test Data

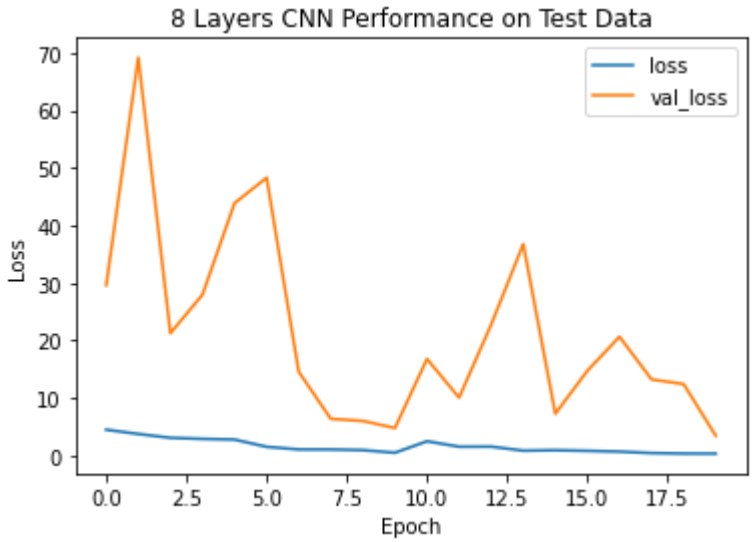
In [338]:

```
plt.plot(history.history["accuracy"])
plt.plot(history.history['val_accuracy'])
plt.title("8 Layers CNN Accuracy on Test Data")
plt.ylabel("Accuracy")
plt.xlabel("Epoch")
plt.legend(["accuracy","val_accuracy"], loc='lower right')
plt.show()
```



In [337]:

```
plt.plot(history.history["loss"])
plt.plot(history.history['val_loss'])
plt.title("8 Layers CNN Performance on Test Data")
plt.ylabel("Loss")
plt.xlabel("Epoch")
plt.legend(["loss","val_loss"], loc='upper right')
plt.show()
```



Prediction Output

In [305]:

```
predSet = model.predict_generator(test_it,verbose=0)
predicted_class_indices=np.argmax(predSet,axis=1)
labels = (test_it.class_indices)
labels = dict((v,k) for k,v in labels.items())
predictions = [labels[k] for k in predicted_class_indices]
filenames=test_it.filenames
results=pd.DataFrame({"Filename":filenames,
                      "Predictions":predictions})
print(labels)
results[0:50]
```

```
{0: 'Alluvial_Soil', 1: 'Black_Soil', 2: 'Clay_Soil', 3: 'Red_Soil'}
```

Out[305]:

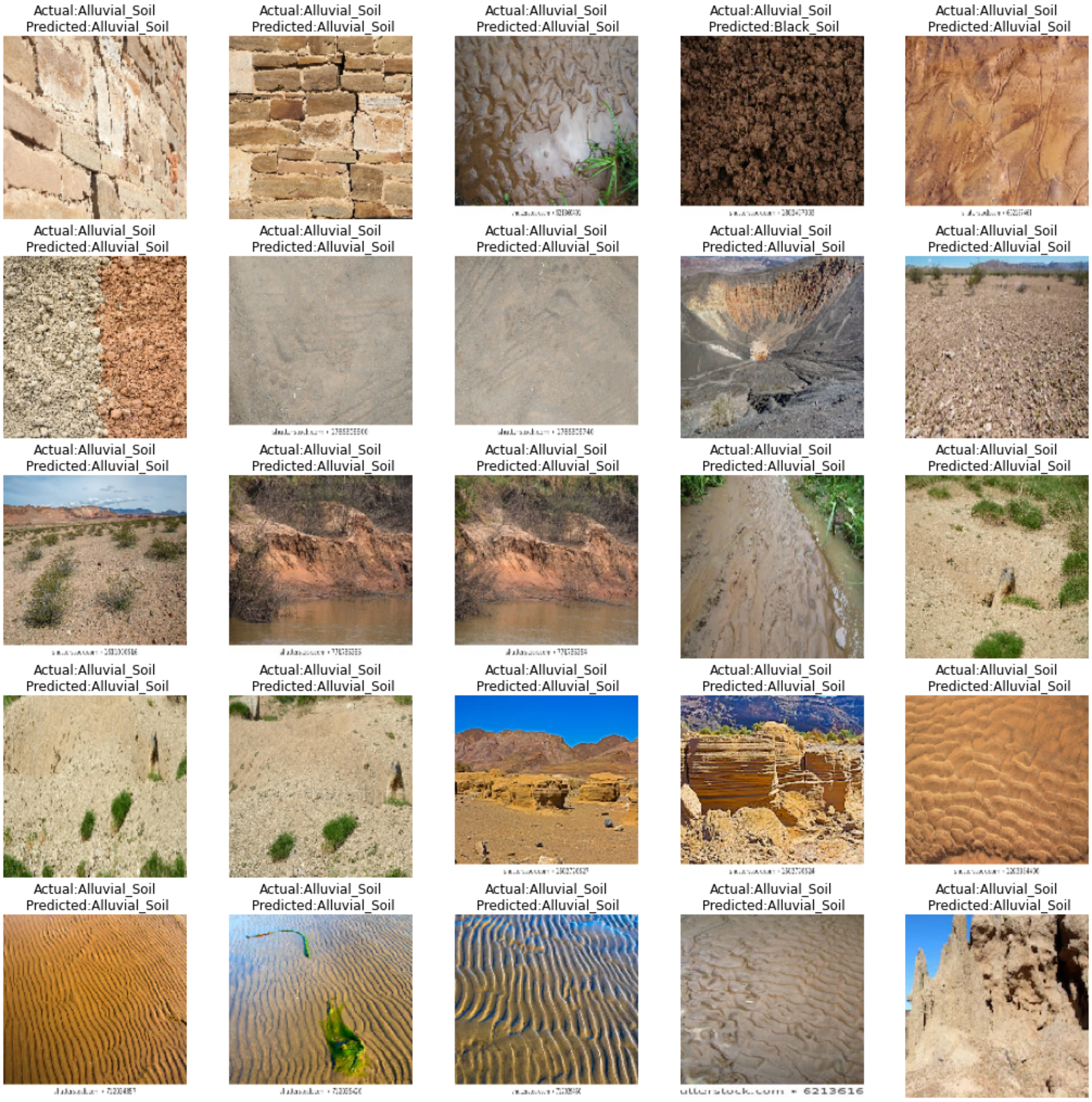
	Filename	Predictions
0	Alluvial_Soil/Alluvial_1.jpg	Alluvial_Soil
1	Alluvial_Soil/Alluvial_10.jpg	Alluvial_Soil
2	Alluvial_Soil/Alluvial_11.jpg	Alluvial_Soil
3	Alluvial_Soil/Alluvial_12.jpg	Alluvial_Soil
4	Alluvial_Soil/Alluvial_13.jpg	Alluvial_Soil
5	Alluvial_Soil/Alluvial_14.jpg	Alluvial_Soil
6	Alluvial_Soil/Alluvial_15.jpg	Alluvial_Soil
7	Alluvial_Soil/Alluvial_16.jpg	Alluvial_Soil
8	Alluvial_Soil/Alluvial_17.jpg	Alluvial_Soil
9	Alluvial_Soil/Alluvial_18.jpg	Alluvial_Soil
10	Alluvial_Soil/Alluvial_19.jpg	Alluvial_Soil
11	Alluvial_Soil/Alluvial_2.jpg	Alluvial_Soil
12	Alluvial_Soil/Alluvial_20.jpg	Alluvial_Soil
13	Alluvial_Soil/Alluvial_21.jpg	Alluvial_Soil
14	Alluvial_Soil/Alluvial_22.jpg	Alluvial_Soil
15	Alluvial_Soil/Alluvial_23.jpg	Alluvial_Soil
16	Alluvial_Soil/Alluvial_24.jpg	Alluvial_Soil
17	Alluvial_Soil/Alluvial_25.jpg	Alluvial_Soil
18	Alluvial_Soil/Alluvial_26.jpg	Alluvial_Soil
19	Alluvial_Soil/Alluvial_27.jpg	Alluvial_Soil
20	Alluvial_Soil/Alluvial_28.jpg	Alluvial_Soil
21	Alluvial_Soil/Alluvial_29.jpg	Alluvial_Soil
22	Alluvial_Soil/Alluvial_3.jpg	Alluvial_Soil
23	Alluvial_Soil/Alluvial_30.jpg	Black_Soil
24	Alluvial_Soil/Alluvial_31.jpg	Alluvial_Soil
25	Alluvial_Soil/Alluvial_32.jpg	Alluvial_Soil
26	Alluvial_Soil/Alluvial_33.jpg	Alluvial_Soil
27	Alluvial_Soil/Alluvial_34.jpg	Alluvial_Soil
28	Alluvial_Soil/Alluvial_35.jpg	Alluvial_Soil
29	Alluvial_Soil/Alluvial_36.jpg	Alluvial_Soil
30	Alluvial_Soil/Alluvial_37.jpg	Alluvial_Soil
31	Alluvial_Soil/Alluvial_38.jpg	Alluvial_Soil
32	Alluvial_Soil/Alluvial_39.jpg	Alluvial_Soil
33	Alluvial_Soil/Alluvial_4.jpg	Alluvial_Soil
34	Alluvial_Soil/Alluvial_40.jpg	Alluvial_Soil
35	Alluvial_Soil/Alluvial_41.jpg	Alluvial_Soil

	Filename	Predictions
36	Alluvial_Soil/Alluvial_42.jpg	Alluvial_Soil
37	Alluvial_Soil/Alluvial_43.jpg	Alluvial_Soil
38	Alluvial_Soil/Alluvial_44.jpg	Alluvial_Soil
39	Alluvial_Soil/Alluvial_45.jpg	Alluvial_Soil
40	Alluvial_Soil/Alluvial_46.jpg	Alluvial_Soil
41	Alluvial_Soil/Alluvial_47.jpg	Alluvial_Soil
42	Alluvial_Soil/Alluvial_48.jpg	Alluvial_Soil
43	Alluvial_Soil/Alluvial_5.jpg	Alluvial_Soil
44	Alluvial_Soil/Alluvial_6.jpg	Alluvial_Soil
45	Alluvial_Soil/Alluvial_7.jpg	Alluvial_Soil
46	Alluvial_Soil/Alluvial_8.jpg	Alluvial_Soil
47	Alluvial_Soil/Alluvial_9.jpg	Alluvial_Soil
48	Black_Soil/Black_1.jpg	Black_Soil
49	Black_Soil/Black_10.jpg	Black_Soil

First batch of 64 images is taken from test data created using ImageDataGenerator and validated predictions of those images with model output.

In [319]:

```
x,y = test_it.next()
inputSet=[]
actualLabelsSet=[]
predictionsSet = []

for i in range(0,64):
    inputSet.append(test_it[0][0][i])
    actualLabelsSet.append(test_it[0][1][i])
    predictionsSet.append(predictions[i])

plt.figure(figsize=(18,18))
for n in range(20,45):
    ax = plt.subplot(5,5,n-20+1)
    plt.imshow(inputSet[n])
    plt.title("Actual:"+labels[actualLabelsSet[n].argmax()]+"\n Predicted:"+str(
predictionsSet[n]))
    plt.axis('off')
```


Actual:Alluvial_Soil
Predicted:Alluvial_Soil
Actual:Alluvial_Soil
Predicted:Alluvial_Soil
Actual:Alluvial_Soil
Predicted:Alluvial_Soil
Actual:Alluvial_Soil
Predicted:Black_Soil
Actual:Alluvial_Soil
Predicted:Alluvial_Soil
Actual:Alluvial_Soil
Predicted:Alluvial_Soil
Actual:Alluvial_Soil
Predicted:Alluvial_Soil
Actual:Alluvial_Soil
Predicted:Alluvial_Soil
Actual:Alluvial_Soil
Predicted:Alluvial_Soil
Actual:Alluvial_Soil
Predicted:Alluvial_Soil
Actual:Alluvial_Soil
Predicted:Alluvial_Soil
Actual:Alluvial_Soil
Predicted:Alluvial_Soil
Actual:Alluvial_Soil
Predicted:Alluvial_Soil
Actual:Alluvial_Soil
Predicted:Alluvial_Soil
Actual:Alluvial_Soil
Predicted:Alluvial_Soil
Actual:Alluvial_Soil
Predicted:Alluvial_Soil
Actual:Alluvial_Soil
Predicted:Alluvial_Soil
Actual:Alluvial_Soil
Predicted:Alluvial_Soil
Actual:Alluvial_Soil
Predicted:Alluvial_Soil
Actual:Alluvial_Soil
Predicted:Alluvial_Soil
Actual:Alluvial_Soil
Predicted:Alluvial_Soil
Actual:Alluvial_Soil
Predicted:Alluvial_Soil
Actual:Alluvial_Soil
Predicted:Alluvial_Soil
Actual:Alluvial_Soil
Predicted:Alluvial_Soil
Actual:Alluvial_Soil
Predicted:Alluvial_Soil

VGG16

VGG16 is CNN architecture build in 2014. Instead of using large number of hyper parameters, VGG16 uses convolutional layer with 3x3 filter with stride 1 and uses max pool layer of 2x2 filter with strides 2.

Implementing VGG16 CNN:

- Creating 16 layers with weights and activation function.
- ReLu is added to each layer so that any negative values will not be passed to next layer.
- Padding is added on the edges so that important features information will not be missed.
- Max Pooling is done after every convolutional layer with 2x2 pool size of 2 strides to reduce the dimension of feature maps.
- On top of CNN, creating flatten layer to prepare the vectors for fully connected layers.
- Then, Dense layers are added.
- Last layer is 4 units dense with softmax activation because, there are 4 classes to predict from in the end.

In [151]:

```
vgg_model = Sequential()
vgg_model.add(Conv2D(input_shape=(150,150,3),filters=64,kernel_size=(3,3),paddin
g="same", activation="relu"))
vgg_model.add(Conv2D(filters=64,kernel_size=(3,3),padding="same", activation="re
lu"))
vgg_model.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))

vgg_model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same", activation=
"relu"))
vgg_model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same", activation=
"relu"))
vgg_model.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))

vgg_model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation=
"relu"))
vgg_model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation=
"relu"))
vgg_model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation=
"relu"))
vgg_model.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))

vgg_model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation=
"relu"))
vgg_model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation=
"relu"))
vgg_model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation=
"relu"))
vgg_model.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))

vgg_model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation=
"relu"))
vgg_model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation=
"relu"))
vgg_model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation=
"relu"))
vgg_model.add(MaxPooling2D(pool_size=(2,2),strides=(2,2)))

vgg_model.add(Flatten())
vgg_model.add(Dense(units=4096,activation="relu"))
vgg_model.add(Dense(units=4096,activation="relu"))
vgg_model.add(Dense(units=4, activation="softmax"))
```

Summary of model

Trainable Parameters: 65,070,916

In [153]:

```
vgg_model.summary()
```

```
Model: "sequential_15"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_83 (Conv2D)           (None, 150, 150, 64)      1792
_________________________________________________________________
conv2d_84 (Conv2D)           (None, 150, 150, 64)      36928
_________________________________________________________________
max_pooling2d_31 (MaxPooling (None, 75, 75, 64)        0
_________________________________________________________________
conv2d_85 (Conv2D)           (None, 75, 75, 128)       73856
_________________________________________________________________
conv2d_86 (Conv2D)           (None, 75, 75, 128)       147584
_________________________________________________________________
max_pooling2d_32 (MaxPooling (None, 37, 37, 128)       0
_________________________________________________________________
conv2d_87 (Conv2D)           (None, 37, 37, 256)       295168
_________________________________________________________________
conv2d_88 (Conv2D)           (None, 37, 37, 256)       590080
_________________________________________________________________
conv2d_89 (Conv2D)           (None, 37, 37, 256)       590080
_________________________________________________________________
max_pooling2d_33 (MaxPooling (None, 18, 18, 256)       0
_________________________________________________________________
conv2d_90 (Conv2D)           (None, 18, 18, 512)       1180160
_________________________________________________________________
conv2d_91 (Conv2D)           (None, 18, 18, 512)       2359808
_________________________________________________________________
conv2d_92 (Conv2D)           (None, 18, 18, 512)       2359808
_________________________________________________________________
max_pooling2d_34 (MaxPooling (None, 9, 9, 512)         0
_________________________________________________________________
conv2d_93 (Conv2D)           (None, 9, 9, 512)         2359808
_________________________________________________________________
conv2d_94 (Conv2D)           (None, 9, 9, 512)         2359808
_________________________________________________________________
conv2d_95 (Conv2D)           (None, 9, 9, 512)         2359808
_________________________________________________________________
max_pooling2d_35 (MaxPooling (None, 4, 4, 512)         0
_________________________________________________________________
flatten_13 (Flatten)         (None, 8192)              0
_________________________________________________________________
dense_29 (Dense)             (None, 4096)              33558528
_________________________________________________________________
dense_30 (Dense)             (None, 4096)              16781312
_________________________________________________________________
dense_31 (Dense)             (None, 4)                 16388
=================================================================
Total params: 65,070,916
Trainable params: 65,070,916
Non-trainable params: 0
_________________________________________________________________
```

Model is compiled and fit

In [152]:

```
vgg_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

In [155]:

```
# checkpoint = ModelCheckpoint("vgg16_1.h5", monitor='val_accuracy', verbose=1,
 save_best_only=True, save_weights_only=False, mode='auto', period=1)

early = EarlyStopping(monitor='val_accuracy', min_delta=0, patience=20, verbose=
1, mode='auto')

start = time.time()

vgg_history = vgg_model.fit_generator(generator=train_it, validation_data= test_
it, epochs=20, callbacks=[early])

end = time.time()

print("Total train time: ",(end-start)/60," mins")
```

```
Epoch 1/20
12/12 [==============================] - 271s 23s/step - loss: 1.414
9 - accuracy: 0.2923 - val_loss: 1.3820 - val_accuracy: 0.3989
Epoch 2/20
12/12 [==============================] - 243s 20s/step - loss: 1.431
0 - accuracy: 0.3343 - val_loss: 1.3959 - val_accuracy: 0.2500
Epoch 3/20
12/12 [==============================] - 237s 20s/step - loss: 1.379
1 - accuracy: 0.2965 - val_loss: 1.3950 - val_accuracy: 0.2500
Epoch 4/20
12/12 [==============================] - 238s 20s/step - loss: 1.343
9 - accuracy: 0.3427 - val_loss: 1.3061 - val_accuracy: 0.4202
Epoch 5/20
12/12 [==============================] - 256s 21s/step - loss: 1.365
6 - accuracy: 0.3231 - val_loss: 1.3915 - val_accuracy: 0.2500
Epoch 6/20
12/12 [==============================] - 262s 22s/step - loss: 1.378
0 - accuracy: 0.2965 - val_loss: 1.3946 - val_accuracy: 0.2500
Epoch 7/20
12/12 [==============================] - 268s 22s/step - loss: 1.381
6 - accuracy: 0.2965 - val_loss: 1.3958 - val_accuracy: 0.2500
Epoch 8/20
12/12 [==============================] - 283s 24s/step - loss: 1.378
4 - accuracy: 0.2965 - val_loss: 1.3927 - val_accuracy: 0.2500
Epoch 9/20
12/12 [==============================] - 278s 23s/step - loss: 1.378
4 - accuracy: 0.2965 - val_loss: 1.3931 - val_accuracy: 0.2500
Epoch 10/20
12/12 [==============================] - 284s 24s/step - loss: 1.378
1 - accuracy: 0.2965 - val_loss: 1.3959 - val_accuracy: 0.2500
Epoch 11/20
12/12 [==============================] - 286s 24s/step - loss: 1.377
3 - accuracy: 0.2965 - val_loss: 1.3963 - val_accuracy: 0.2500
Epoch 12/20
12/12 [==============================] - 299s 25s/step - loss: 1.378
1 - accuracy: 0.2965 - val_loss: 1.3926 - val_accuracy: 0.2500
Epoch 13/20
12/12 [==============================] - 294s 25s/step - loss: 1.377
8 - accuracy: 0.2965 - val_loss: 1.3936 - val_accuracy: 0.2500
Epoch 14/20
12/12 [==============================] - 298s 25s/step - loss: 1.377
6 - accuracy: 0.2965 - val_loss: 1.3948 - val_accuracy: 0.2500
Epoch 15/20
12/12 [==============================] - 269s 22s/step - loss: 1.377
8 - accuracy: 0.2965 - val_loss: 1.3964 - val_accuracy: 0.2500
Epoch 16/20
12/12 [==============================] - 282s 23s/step - loss: 1.377
9 - accuracy: 0.2965 - val_loss: 1.3927 - val_accuracy: 0.2500
Epoch 17/20
12/12 [==============================] - 293s 24s/step - loss: 1.377
8 - accuracy: 0.2965 - val_loss: 1.3947 - val_accuracy: 0.2500
Epoch 18/20
12/12 [==============================] - 267s 22s/step - loss: 1.377
6 - accuracy: 0.2965 - val_loss: 1.3954 - val_accuracy: 0.2500
Epoch 19/20
12/12 [==============================] - 223s 19s/step - loss: 1.377
2 - accuracy: 0.2965 - val_loss: 1.3956 - val_accuracy: 0.2500
Epoch 20/20
12/12 [==============================] - 279s 23s/step - loss: 1.377
4 - accuracy: 0.2965 - val_loss: 1.3951 - val_accuracy: 0.2500
Total train time:  98.5236760656039  mins
```

Performance

Performance of the model is evaluated on the basis of Accuracy and Cross Entropy results on test dataset.\
Performance output is shown using confusion matrix.\ Also, comparison of Accuracy and Cross Entropy results shown between train and test data using graph plots.

Accuracy on test data ~ 25%

In [341]:

```
score = vgg_model.evaluate_generator(test_it)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
Test loss: 1.395052194595337
Test accuracy: 0.25
```

Confusion Matrix & Classification report

Precision, Recall, f1-score results.

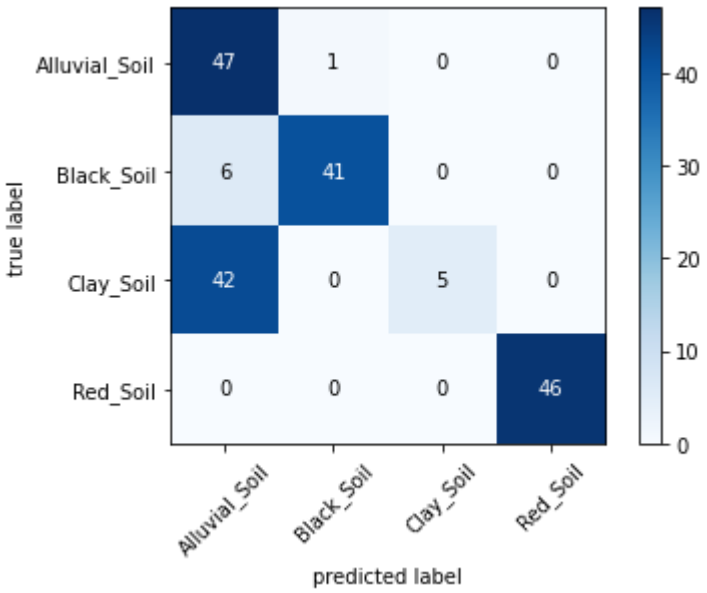
In [339]:

```
#Confusion Matrix and Classification Report
num_of_test_samples = 188
Y_pred = vgg_model.predict_generator(test_it, num_of_test_samples)
vgg_cm = confusion_matrix(test_it.classes, y_pred)
y_pred = np.argmax(Y_pred, axis=1)
print('Confusion Matrix')
print(vgg_cm)
print('Classification Report')
target_names = ['Alluvial_Soil', 'Black_Soil', 'Clay_Soil', 'Red_Soil']
print(classification_report(test_it.classes, y_pred, target_names=target_names))
```

```
WARNING:tensorflow:Your input ran out of data; interrupting trainin
g. Make sure that your dataset or generator can generate at least `s
teps_per_epoch * epochs` batches (in this case, 188 batches). You ma
y need to use the repeat() function when building your dataset.
Confusion Matrix
[[47  1  0  0]
 [ 6 41  0  0]
 [42  0  5  0]
 [ 0  0  0 46]]
Classification Report
              precision    recall  f1-score   support

Alluvial_Soil       0.00      0.00      0.00        48
   Black_Soil       0.25      1.00      0.40        47
    Clay_Soil       0.00      0.00      0.00        47
     Red_Soil       0.00      0.00      0.00        46

     accuracy                           0.25       188
    macro avg       0.06      0.25      0.10       188
 weighted avg       0.06      0.25      0.10       188


/usr/local/lib/python3.8/site-packages/sklearn/metrics/_classificati
on.py:1221: UndefinedMetricWarning: Precision and F-score are ill-de
fined and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [340]:

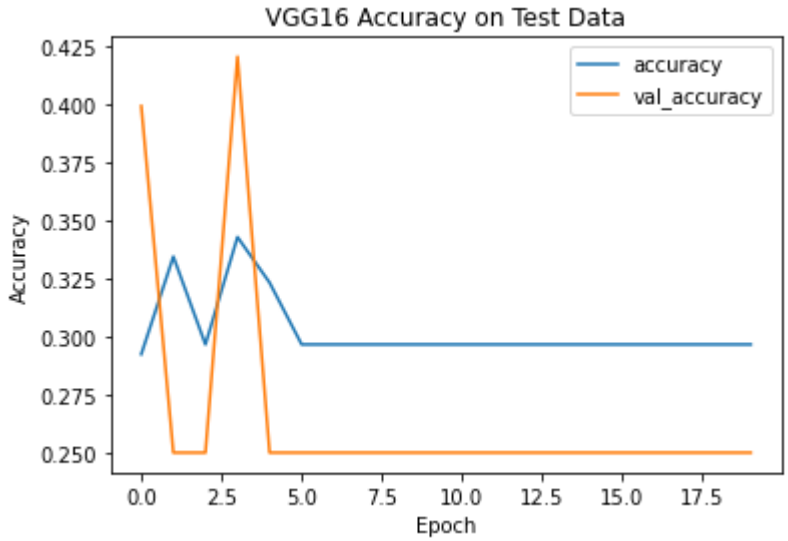
```
plot_confusion_matrix(vgg_cm, colorbar=True, class_names=target_names)
plt.show()
```



Accuracy & Cross Entropy result plots of Train vs Test Data

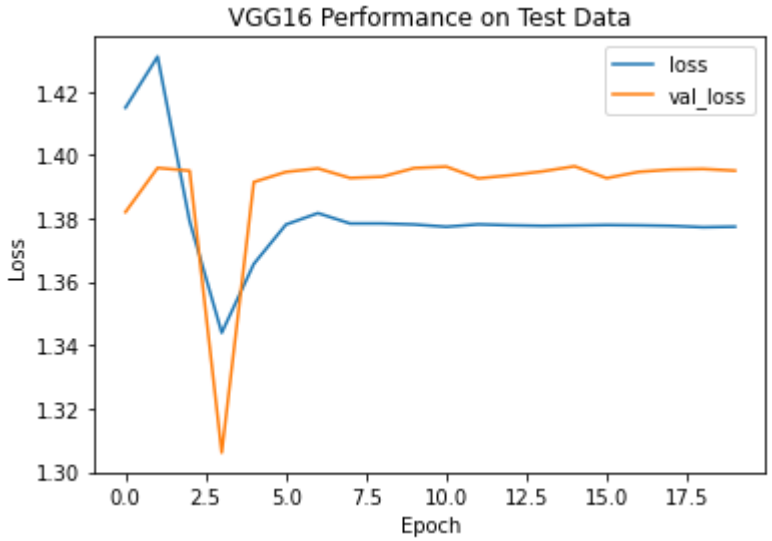
In [270]:

```
plt.plot(vgg_history.history["accuracy"])
plt.plot(vgg_history.history['val_accuracy'])
plt.title("VGG16 Accuracy on Test Data")
plt.ylabel("Accuracy")
plt.xlabel("Epoch")
plt.legend(["accuracy","val_accuracy"])
plt.show()
```



In [269]:

```
plt.plot(vgg_history.history["loss"])
plt.plot(vgg_history.history['val_loss'])
plt.title("VGG16 Performance on Test Data")
plt.ylabel("Loss")
plt.xlabel("Epoch")
plt.legend(["loss","val_loss"])
plt.show()
```



Prediction Output

In [253]:

```
vgg_predSet = vgg_model.predict_generator(test_it,verbose=0)
vgg_predicted_class_indices=np.argmax(vgg_predSet,axis=1)
labels = (test_it.class_indices)
labels = dict((v,k) for k,v in labels.items())
vgg_predictions = [labels[k] for k in vgg_predicted_class_indices]
filenames=test_it.filenames
results=pd.DataFrame({"Filename":filenames,
                      "Predictions":vgg_predictions})
results[0:50]
```

Out[253]:

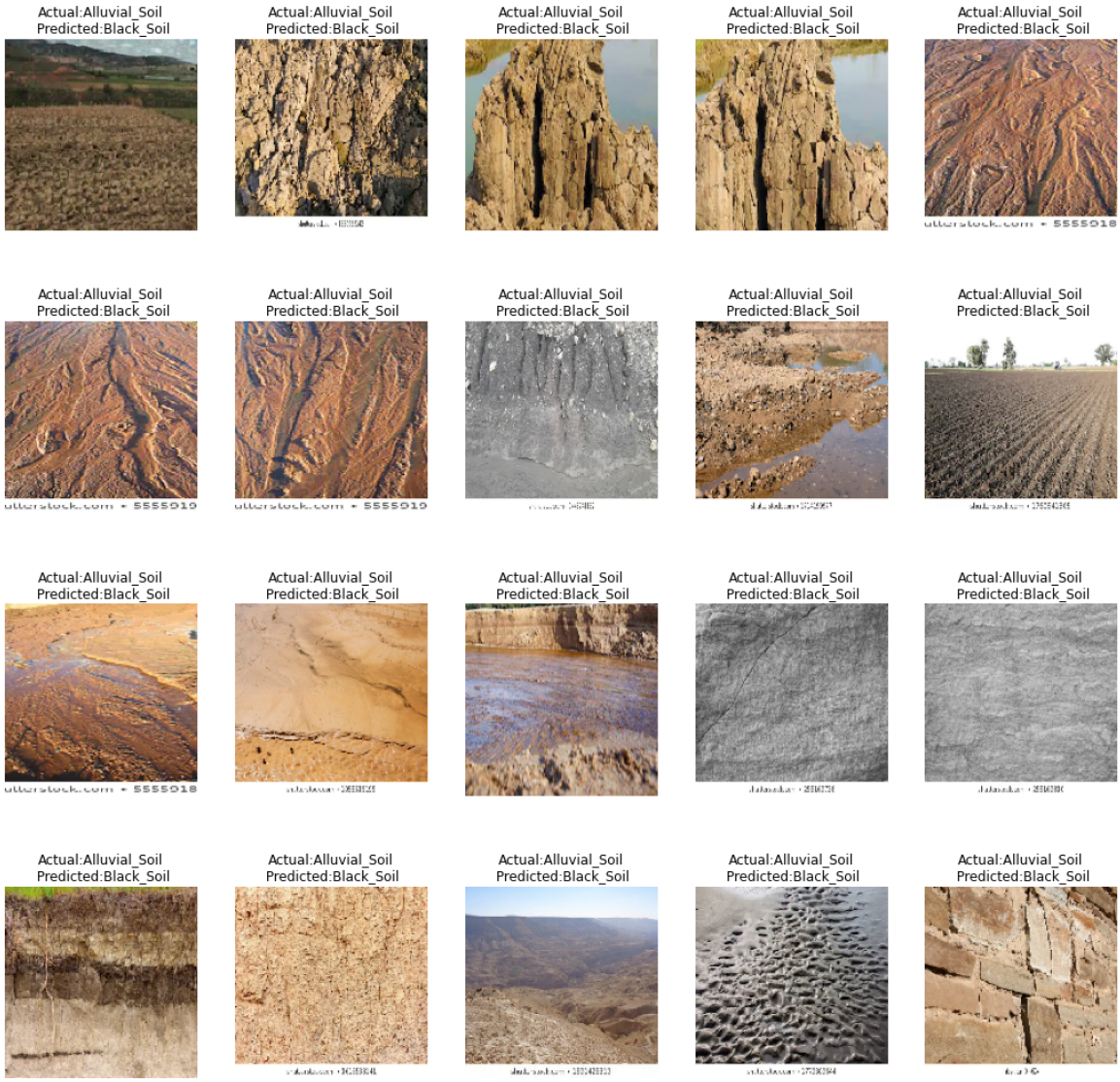
	Filename	Predictions
0	Alluvial_Soil/Alluvial_1.jpg	Black_Soil
1	Alluvial_Soil/Alluvial_10.jpg	Black_Soil
2	Alluvial_Soil/Alluvial_11.jpg	Black_Soil
3	Alluvial_Soil/Alluvial_12.jpg	Black_Soil
4	Alluvial_Soil/Alluvial_13.jpg	Black_Soil
5	Alluvial_Soil/Alluvial_14.jpg	Black_Soil
6	Alluvial_Soil/Alluvial_15.jpg	Black_Soil
7	Alluvial_Soil/Alluvial_16.jpg	Black_Soil
8	Alluvial_Soil/Alluvial_17.jpg	Black_Soil
9	Alluvial_Soil/Alluvial_18.jpg	Black_Soil
10	Alluvial_Soil/Alluvial_19.jpg	Black_Soil
11	Alluvial_Soil/Alluvial_2.jpg	Black_Soil
12	Alluvial_Soil/Alluvial_20.jpg	Black_Soil
13	Alluvial_Soil/Alluvial_21.jpg	Black_Soil
14	Alluvial_Soil/Alluvial_22.jpg	Black_Soil
15	Alluvial_Soil/Alluvial_23.jpg	Black_Soil
16	Alluvial_Soil/Alluvial_24.jpg	Black_Soil
17	Alluvial_Soil/Alluvial_25.jpg	Black_Soil
18	Alluvial_Soil/Alluvial_26.jpg	Black_Soil
19	Alluvial_Soil/Alluvial_27.jpg	Black_Soil
20	Alluvial_Soil/Alluvial_28.jpg	Black_Soil
21	Alluvial_Soil/Alluvial_29.jpg	Black_Soil
22	Alluvial_Soil/Alluvial_3.jpg	Black_Soil
23	Alluvial_Soil/Alluvial_30.jpg	Black_Soil
24	Alluvial_Soil/Alluvial_31.jpg	Black_Soil
25	Alluvial_Soil/Alluvial_32.jpg	Black_Soil
26	Alluvial_Soil/Alluvial_33.jpg	Black_Soil
27	Alluvial_Soil/Alluvial_34.jpg	Black_Soil
28	Alluvial_Soil/Alluvial_35.jpg	Black_Soil
29	Alluvial_Soil/Alluvial_36.jpg	Black_Soil
30	Alluvial_Soil/Alluvial_37.jpg	Black_Soil
31	Alluvial_Soil/Alluvial_38.jpg	Black_Soil
32	Alluvial_Soil/Alluvial_39.jpg	Black_Soil
33	Alluvial_Soil/Alluvial_4.jpg	Black_Soil
34	Alluvial_Soil/Alluvial_40.jpg	Black_Soil
35	Alluvial_Soil/Alluvial_41.jpg	Black_Soil

	Filename	Predictions
36	Alluvial_Soil/Alluvial_42.jpg	Black_Soil
37	Alluvial_Soil/Alluvial_43.jpg	Black_Soil
38	Alluvial_Soil/Alluvial_44.jpg	Black_Soil
39	Alluvial_Soil/Alluvial_45.jpg	Black_Soil
40	Alluvial_Soil/Alluvial_46.jpg	Black_Soil
41	Alluvial_Soil/Alluvial_47.jpg	Black_Soil
42	Alluvial_Soil/Alluvial_48.jpg	Black_Soil
43	Alluvial_Soil/Alluvial_5.jpg	Black_Soil
44	Alluvial_Soil/Alluvial_6.jpg	Black_Soil
45	Alluvial_Soil/Alluvial_7.jpg	Black_Soil
46	Alluvial_Soil/Alluvial_8.jpg	Black_Soil
47	Alluvial_Soil/Alluvial_9.jpg	Black_Soil
48	Black_Soil/Black_1.jpg	Black_Soil
49	Black_Soil/Black_10.jpg	Black_Soil

First batch of 64 images is taken from test data created using ImageDataGenerator and validated predictions of those images with model output.

In [295]:

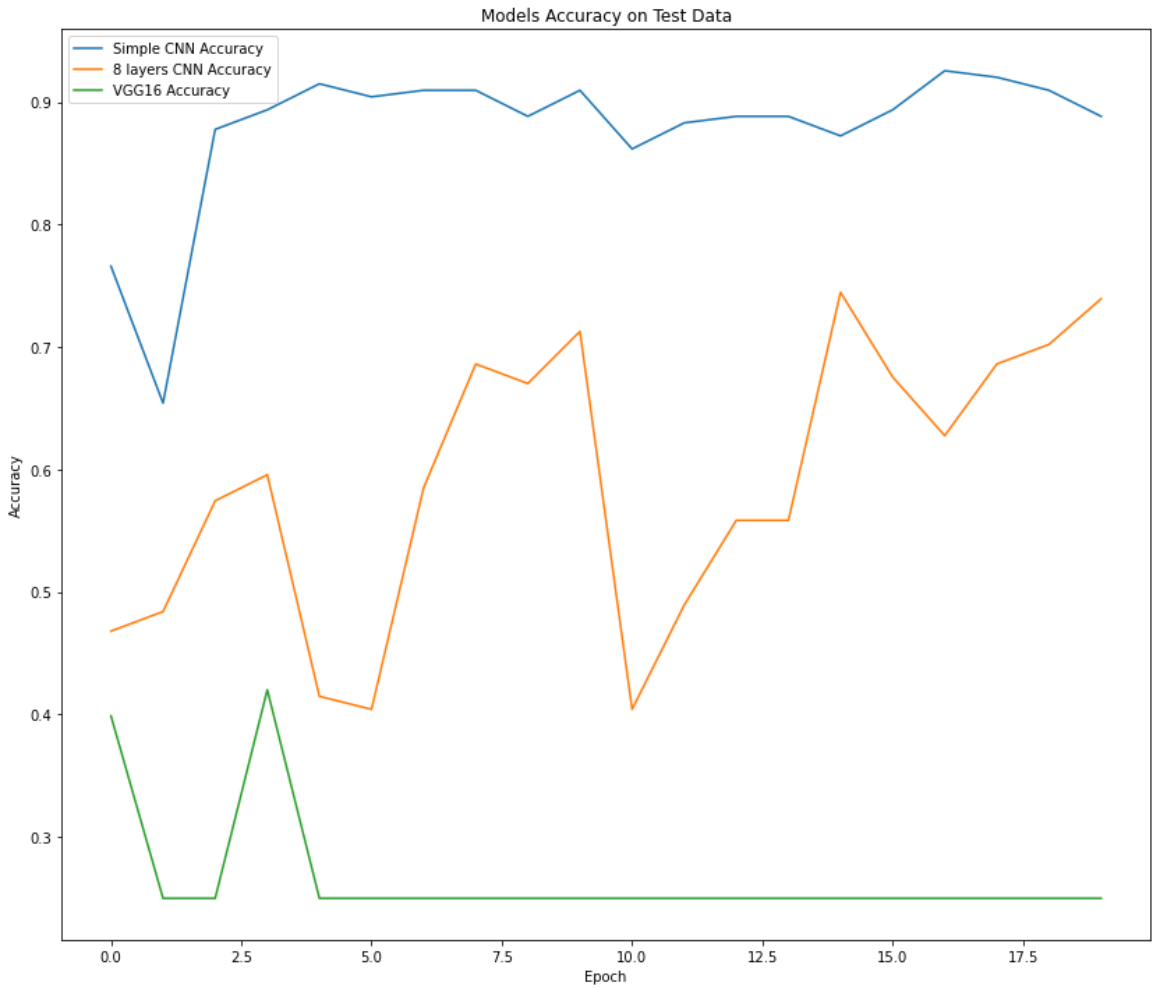
```
# x,y = test_it.next()
vgg_inputSet=[]
vgg_actualLabelsSet=[]
vgg_predictionsSet = []

for i in range(0,64):
    vgg_inputSet.append(test_it[0][0][i])
    vgg_actualLabelsSet.append(test_it[0][1][i])
    vgg_predictionsSet.append(vgg_predictions[i])

plt.figure(figsize=(18,18))
for n in range(0,20):
    ax = plt.subplot(4,5,n+1)
    plt.imshow(vgg_inputSet[n])
    plt.title("Actual:"+labels[vgg_actualLabelsSet[n].argmax()]+"\n Predicted:"+
str(vgg_predictions[n]))
    plt.axis('off')
```



Models Comparison

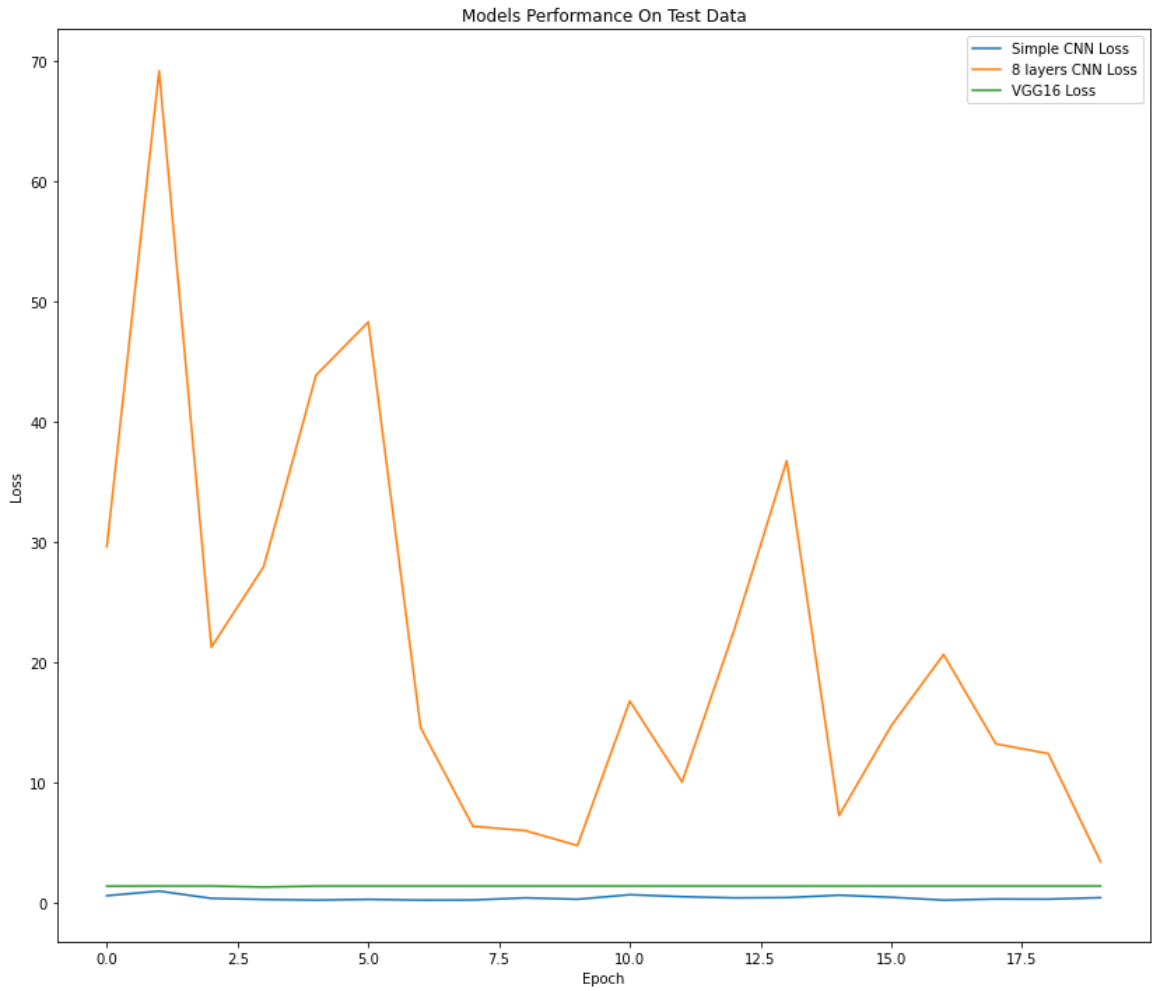
In [342]:

```
plt.figure(figsize=(14,12))
plt.plot(simple_history.history['val_accuracy'])
plt.plot(history.history['val_accuracy'])
plt.plot(vgg_history.history['val_accuracy'])
plt.title("Models Accuracy on Test Data")
plt.ylabel("Accuracy")
plt.xlabel("Epoch")
plt.legend(["Simple CNN Accuracy ","8 layers CNN Accuracy", "VGG16 Accuracy"], l
oc='upper left')
plt.show()
```



In [343]:

```
plt.figure(figsize=(14,12))
plt.plot(simple_history.history['val_loss'])
plt.plot(history.history['val_loss'])
plt.plot(vgg_history.history['val_loss'])
plt.title("Models Performance On Test Data")
plt.ylabel("Loss")
plt.xlabel("Epoch")
plt.legend(["Simple CNN Loss ","8 layers CNN Loss", "VGG16 Loss"], loc='upper right')
plt.show()
```



Conclusion

From the above experiments it can be concluded, training complex models on simple dataset doesn't result in better performance. As observed from above, if we compare accuracies, simple CNN results in better accuracy of approximately 86% as compared to complex CNN and VGG16 CNN models. Also, if we have a look at VGG16, it performs much worse. In VGG16, accuracy becomes stagnant after 3 epochs only. Also if we observe cross entropy, simple CNN performs better with lowest cross entropy compared to complex CNN and VGG16 CNN architecture. As seen in above cross entropy plot, complex CNN is behaving much arbitrary and fluctuating in every epoch which is not adequate for better results.

References

- https://machinelearningmastery.com/how-to-configure-image-data-augmentation-when-training-deep-learning-neural-networks/ (https://machinelearningmastery.com/how-to-configure-image-data-augmentation-when-training-deep-learning-neural-networks/)
- https://machinelearningmastery.com/how-to-load-large-datasets-from-directories-for-deep-learning-with-keras/ (https://machinelearningmastery.com/how-to-load-large-datasets-from-directories-for-deep-learning-with-keras/)
- https://www.tensorflow.org/tutorials/images/cnn (https://www.tensorflow.org/tutorials/images/cnn)
- https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html (https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html)
- https://machinelearningmastery.com/how-to-reduce-overfitting-with-dropout-regularization-in-keras/ (https://machinelearningmastery.com/how-to-reduce-overfitting-with-dropout-regularization-in-keras/)
- https://towardsdatascience.com/step-by-step-vgg16-implementation-in-keras-for-beginners-a833c686ae6c (https://towardsdatascience.com/step-by-step-vgg16-implementation-in-keras-for-beginners-a833c686ae6c)
- https://keras.io/api/applications/vgg/ (https://keras.io/api/applications/vgg/)

In [ ]: