```
SATELLITE IMAGE CLASSIFICATION:
```

Pre-Processing the Data

```
import numpy as np
import pandas as pd
import io
import os
import matplotlib.pyplot as plt
import tensorflow as tf
from matplotlib import image as mpimg
```

```
     /opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this version of SciPy (detected version 1.23.5
       warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

```
import sys
print(sys.executable)
```

```
     /opt/conda/bin/python
```

```
#Loading the dataset and checking the class labels

data_dir = '/kaggle/input/satellite-image-classification/data'
labels = os.listdir(data_dir)
labels
```

```
     ['cloudy', 'desert', 'green_area', 'water']
```

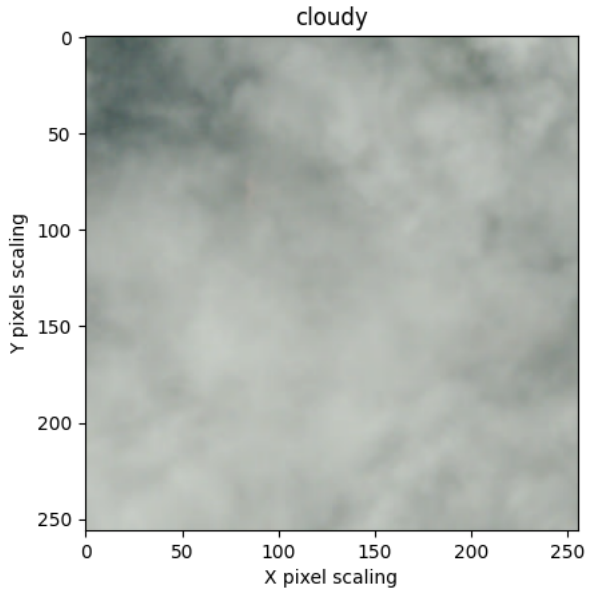
```
# Number of Images for each class label
for label in labels:
    print(label, len(os.listdir(data_dir+'/'+label)))
```

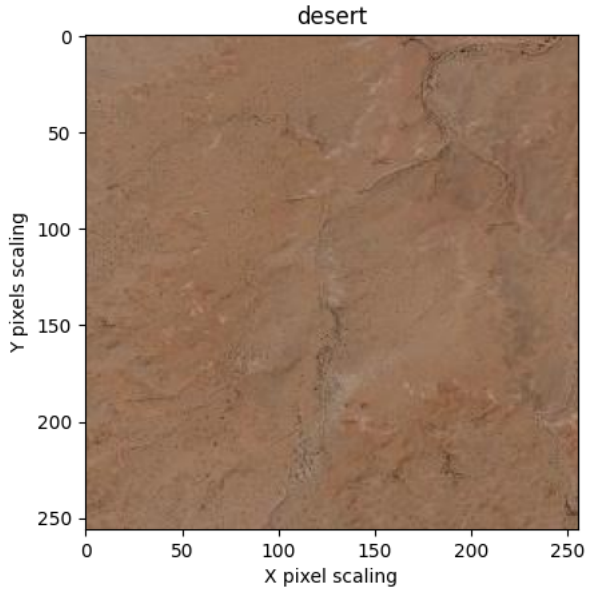
```
     cloudy 1500
     desert 1131
     green_area 1500
     water 1500
```

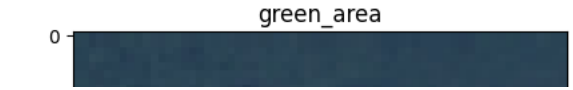
```
label_counts = {}

for label in labels:
    label_dir = os.path.join(data_dir, label)
    num_images = len(os.listdir(label_dir))
    label_counts[label] = num_images

total_count = sum(label_counts.values())
print("Total image count:", total_count)
```

```
     Total image count: 5631
```

```
#Visualizing one image for every class label

for label in labels:
    path = os.listdir(data_dir + '/' + label)
    img = data_dir + '/' + label + '/' + path[1]
    plt.title(label)
    plt.xlabel("X pixel scaling")
    plt.ylabel("Y pixels scaling")
    image = mpimg.imread(img)
    plt.imshow(image)
    plt.show()
```








```
img_width, img_height = 224, 224
target_size = (img_width,img_height)
batch_size = 32
input_shape = (img_height, img_height, 3)
```

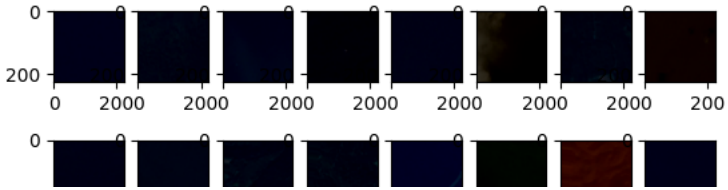
```
num_label = 4
epoch = 10
```

```
#Create a data generator for image augmentation
from tensorflow.keras.preprocessing.image import ImageDataGenerator
datagen = ImageDataGenerator( samplewise_center=True,  # set each sample mean to 0
                              rotation_range=10,  # randomly rotate images in the range (degrees, 0 to 180)
                              zoom_range = 0.1, # Randomly zoom imag
                              width_shift_range=0.1,  # randomly shift images horizontally (fraction of total width)
                              height_shift_range=0.1,  # randomly shift images vertically (fraction of total height)
                              horizontal_flip=True,  # randomly flip images
                              vertical_flip=False,
                              rescale=1./255,#scale images
                              validation_split=0.2) #split data, 80% for training and 20% for testing

#create training set from folders
train_data=datagen.flow_from_directory(data_dir,
                                       target_size=(target_size),
                                       batch_size=batch_size,
                                       class_mode='categorical',
                                       shuffle=True,subset='training')

#create test set
test_data=datagen.flow_from_directory(data_dir,
                                       target_size=(target_size),
                                       batch_size=batch_size,
                                       shuffle=False,subset='validation')
```

```
Found 4505 images belonging to 4 classes.
Found 1126 images belonging to 4 classes.
```

```
#sample images
img_iter = datagen.flow_from_directory(data_dir,
                                       target_size=(target_size),
                                       batch_size=32)
x, y = img_iter.next()
fig, ax = plt.subplots(nrows=4, ncols=8)
for i in range(32):
    image = x[i]
    ax.flatten()[i].imshow(np.squeeze(image))
plt.show()
```

```
Found 5631 images belonging to 4 classes.
```




```
# Importing all nececcesary libraries for the models
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import (
    Dense,
    Conv2D,
    MaxPool2D,
    Flatten,
    Dropout,
    BatchNormalization,
)
from sklearn.metrics import confusion_matrix , classification_report, ConfusionMatrixDisplay
import seaborn as sn


from tensorflow.keras.layers import (
    Input, Conv2D, MaxPooling2D, ZeroPadding2D,
    Flatten, BatchNormalization, AveragePooling2D, Dense,
    Activation, Add, GlobalAveragePooling2D )
from tensorflow.keras.models import Model
from tensorflow.keras import activations
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.regularizers import l2


import tensorflow as tf
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.applications.vgg16 import preprocess_input
```

CNN Model

```
model_CNN = Sequential()
model_CNN.add(Conv2D(75, (3, 3), strides=1, padding="same", activation="relu",
                 input_shape=input_shape))
model_CNN.add(MaxPool2D((2, 2), strides=2, padding="same"))
model_CNN.add(Conv2D(50, (3, 3), strides=1, padding="same", activation="relu"))
model_CNN.add(MaxPool2D((2, 2), strides=2, padding="same"))
model_CNN.add(Conv2D(25, (3, 3), strides=1, padding="same", activation="relu"))
model_CNN.add(MaxPool2D((2, 2), strides=2, padding="same"))
model_CNN.add(Flatten())
model_CNN.add(Dense(units=512, activation="relu"))
model_CNN.add(Dropout(0.3))
model_CNN.add(Dense(units=4, activation="softmax"))
```

```
model_CNN.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 224, 224, 75)      2100

 max_pooling2d (MaxPooling2D  (None, 112, 112, 75)     0
 )

 conv2d_1 (Conv2D)           (None, 112, 112, 50)      33800

 max_pooling2d_1 (MaxPooling  (None, 56, 56, 50)       0
 2D)

 conv2d_2 (Conv2D)           (None, 56, 56, 25)        11275

 max_pooling2d_2 (MaxPooling  (None, 28, 28, 25)       0
 2D)

 flatten (Flatten)           (None, 19600)             0

 dense (Dense)               (None, 512)               10035712

 dropout (Dropout)           (None, 512)               0

 dense_1 (Dense)             (None, 4)                 2052

=================================================================
Total params: 10,084,939
Trainable params: 10,084,939
Non-trainable params: 0
_________________________________________________________________
```

```
# Set the loss function and metrics
model_CNN.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

```
history_CNN = model_CNN.fit(train_data,
                    validation_data=test_data,
                    epochs=epoch)
```

```
Epoch 1/10
141/141 [==============================] - 128s 792ms/step - loss: 0.4874 - accuracy: 0.7658 - val_loss: 0.3661 - val_accuracy: 0.8135
Epoch 2/10
141/141 [==============================] - 71s 501ms/step - loss: 0.2946 - accuracy: 0.8846 - val_loss: 0.2340 - val_accuracy: 0.9103
Epoch 3/10
141/141 [==============================] - 70s 497ms/step - loss: 0.1536 - accuracy: 0.9472 - val_loss: 0.1331 - val_accuracy: 0.9538
Epoch 4/10
141/141 [==============================] - 72s 507ms/step - loss: 0.1458 - accuracy: 0.9512 - val_loss: 0.1501 - val_accuracy: 0.9494
Epoch 5/10
141/141 [==============================] - 71s 501ms/step - loss: 0.1239 - accuracy: 0.9567 - val_loss: 0.2214 - val_accuracy: 0.9245
Epoch 6/10
141/141 [==============================] - 70s 500ms/step - loss: 0.1187 - accuracy: 0.9567 - val_loss: 0.1562 - val_accuracy: 0.9414
Epoch 7/10
141/141 [==============================] - 70s 500ms/step - loss: 0.1010 - accuracy: 0.9643 - val_loss: 0.1594 - val_accuracy: 0.9369
Epoch 8/10
```

```
141/141 [==============================] - 71s 501ms/step - loss: 0.1156 - accuracy: 0.9589 - val_loss: 0.1583 - val_accuracy: 0.9361
Epoch 9/10
141/141 [==============================] - 70s 499ms/step - loss: 0.1030 - accuracy: 0.9629 - val_loss: 0.1481 - val_accuracy: 0.9449
Epoch 10/10
141/141 [==============================] - 71s 502ms/step - loss: 0.1045 - accuracy: 0.9620 - val_loss: 0.0968 - val_accuracy: 0.9583
```

```
%%capture
CNN_train_loss, CNN_train_accuracy = model_CNN.evaluate(train_data)
CNN_validation_loss, CNN_validation_accuracy = model_CNN.evaluate(test_data)
```

```
print('Training loss: ', CNN_train_loss)
print('Training accuracy: ', CNN_train_accuracy)

print('Validation loss: ', CNN_validation_loss)
print('Validation accuracy: ', CNN_validation_accuracy)
```

```
Training loss:  0.06249570474028587
Training accuracy:  0.9782463908195496
Validation loss:  0.11180952936410904
Validation accuracy:  0.9564831256866455
```

VGG16 Model

```
model_VGG16=Sequential()
model_VGG16.add(Conv2D(input_shape=input_shape,filters=64,kernel_size=(3,3),padding="same", activation="relu"))
model_VGG16.add(Conv2D(filters=64,kernel_size=(3,3),padding="same", activation="relu"))

model_VGG16.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))

model_VGG16.add(Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu"))
model_VGG16.add(Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu"))

model_VGG16.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))

model_VGG16.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model_VGG16.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))
model_VGG16.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu"))

model_VGG16.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))

model_VGG16.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model_VGG16.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model_VGG16.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))

model_VGG16.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))

model_VGG16.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model_VGG16.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model_VGG16.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))

model_VGG16.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))

model_VGG16.add(Flatten())
```

```
model_VGG16.add(Dense(units=4096,activation="relu"))
model_VGG16.add(Dense(units=4096,activation="relu"))
model_VGG16.add(Dense(units=4, activation="softmax"))


model_VGG16.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_3 (Conv2D)           (None, 224, 224, 64)      1792

 conv2d_4 (Conv2D)           (None, 224, 224, 64)      36928

 max_pooling2d_3 (MaxPooling  (None, 112, 112, 64)     0
 2D)

 conv2d_5 (Conv2D)           (None, 112, 112, 128)     73856

 conv2d_6 (Conv2D)           (None, 112, 112, 128)     147584

 max_pooling2d_4 (MaxPooling  (None, 56, 56, 128)      0
 2D)

 conv2d_7 (Conv2D)           (None, 56, 56, 256)       295168

 conv2d_8 (Conv2D)           (None, 56, 56, 256)       590080

 conv2d_9 (Conv2D)           (None, 56, 56, 256)       590080

 max_pooling2d_5 (MaxPooling  (None, 28, 28, 256)      0
 2D)

 conv2d_10 (Conv2D)          (None, 28, 28, 512)       1180160

 conv2d_11 (Conv2D)          (None, 28, 28, 512)       2359808

 conv2d_12 (Conv2D)          (None, 28, 28, 512)       2359808

 max_pooling2d_6 (MaxPooling  (None, 14, 14, 512)      0
 2D)

 conv2d_13 (Conv2D)          (None, 14, 14, 512)       2359808

 conv2d_14 (Conv2D)          (None, 14, 14, 512)       2359808

 conv2d_15 (Conv2D)          (None, 14, 14, 512)       2359808

 max_pooling2d_7 (MaxPooling  (None, 7, 7, 512)        0
 2D)

 flatten_1 (Flatten)         (None, 25088)             0

 dense_2 (Dense)             (None, 4096)              102764544

 dense_3 (Dense)             (None, 4096)              16781312

 dense_4 (Dense)             (None, 4)                 16388

=================================================================
```

```
Total params: 134,276,932
Trainable params: 134,276,932
Non-trainable params: 0
```

```
# Set the loss function and metrics
model_VGG16.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

```
history_VGG16 = model_VGG16.fit(
    train_data,
    epochs=epoch,
    validation_data=test_data,
)
```

```
Epoch 1/10
141/141 [==============================] - 117s 713ms/step - loss: 1.5091 - accuracy: 0.2737 - val_loss: 1.3801 - val_accuracy: 0.2664
Epoch 2/10
141/141 [==============================] - 90s 638ms/step - loss: 1.3805 - accuracy: 0.2575 - val_loss: 1.3796 - val_accuracy: 0.2664
Epoch 3/10
141/141 [==============================] - 91s 642ms/step - loss: 1.3800 - accuracy: 0.2604 - val_loss: 1.3811 - val_accuracy: 0.2664
Epoch 4/10
141/141 [==============================] - 91s 643ms/step - loss: 1.3798 - accuracy: 0.2599 - val_loss: 1.3805 - val_accuracy: 0.2664
Epoch 5/10
141/141 [==============================] - 90s 637ms/step - loss: 1.3804 - accuracy: 0.2648 - val_loss: 1.3801 - val_accuracy: 0.2664
Epoch 6/10
141/141 [==============================] - 90s 637ms/step - loss: 1.3805 - accuracy: 0.2595 - val_loss: 1.3798 - val_accuracy: 0.2664
Epoch 7/10
141/141 [==============================] - 91s 642ms/step - loss: 1.3804 - accuracy: 0.2646 - val_loss: 1.3797 - val_accuracy: 0.2664
Epoch 8/10
141/141 [==============================] - 91s 641ms/step - loss: 1.3803 - accuracy: 0.2590 - val_loss: 1.3796 - val_accuracy: 0.2664
Epoch 9/10
141/141 [==============================] - 91s 641ms/step - loss: 1.3802 - accuracy: 0.2615 - val_loss: 1.3798 - val_accuracy: 0.2664
Epoch 10/10
141/141 [==============================] - 90s 634ms/step - loss: 1.3801 - accuracy: 0.2568 - val_loss: 1.3797 - val_accuracy: 0.2664
```

```
%%capture
VGG16_train_loss, VGG16_train_accuracy = model_VGG16.evaluate(train_data)
VGG16_validation_loss, VGG16_validation_accuracy = model_VGG16.evaluate(test_data)
```

```
print('Training loss: ', VGG16_train_loss)
print('Training accuracy: ', VGG16_train_accuracy)

print('Validation loss: ', VGG16_validation_loss)
print('Validation accuracy: ', VGG16_validation_accuracy)
```

```
Training loss:  1.379721999168396
Training accuracy:  0.2663707137107849
Validation loss:  1.3796777725219727
Validation accuracy:  0.2664298415184021
```

ResNet34 Model

```
def res_identity(x, filters):
    x_skip = x
    f1, f2 = filters

    # First block
    x = Conv2D(f1, kernel_size=(3, 3), strides=(1, 1), padding='same', kernel_regularizer=l2(0.001))(x)
    x = BatchNormalization()(x)
    x = Activation(activations.relu)(x)

    # Second block
    x = Conv2D(f2, kernel_size=(3, 3), strides=(1, 1), padding='same', kernel_regularizer=l2(0.001))(x)
    x = BatchNormalization()(x)

    # Add the input
    x = Add()([x, x_skip])  # Skip connection
    x = Activation(activations.relu)(x)

    return x


def res_conv(x, s, filters):

    x_skip = x
    f1, f2 = filters

    # First block
    x = Conv2D(f1, kernel_size=(3, 3), strides=(s, s), padding='same', kernel_regularizer=l2(0.001))(x)
    x = BatchNormalization()(x)
    x = Activation(activations.relu)(x)

    # Second block
    x = Conv2D(f2, kernel_size=(3, 3), strides=(1, 1), padding='same', kernel_regularizer=l2(0.001))(x)
    x = BatchNormalization()(x)

    #(skip connection)
    if s > 1:
        x_skip = Conv2D(f2, kernel_size=(1, 1), strides=(s, s), padding='valid', kernel_regularizer=l2(0.001))(x_skip)
        x_skip = BatchNormalization()(x_skip)

    # Add
    x = Add()([x, x_skip])
    x = Activation(activations.relu)(x)

    return x


def resnet34():
    input_im = Input(shape=input_shape)  # Adjust the input shape for your dataset
    x = Conv2D(64, kernel_size=(7, 7), strides=(2, 2), padding='same', kernel_regularizer=l2(0.001))(input_im)
    x = BatchNormalization()(x)
    x = Activation(tf.keras.activations.relu)(x)

    x = MaxPooling2D((3, 3), strides=(2, 2))(x)

    x = res_conv(x, s=1, filters=(64, 64))
    x = res_identity(x, filters=(64, 64))
```

```
        x = res_identity(x, filters=(64, 64))

        x = res_conv(x, s=2, filters=(128, 128))
        x = res_identity(x, filters=(128, 128))
        x = res_identity(x, filters=(128, 128))
        x = res_identity(x, filters=(128, 128))

        x = res_conv(x, s=2, filters=(256, 256))
        x = res_identity(x, filters=(256, 256))
        x = res_identity(x, filters=(256, 256))
        x = res_identity(x, filters=(256, 256))
        x = res_identity(x, filters=(256, 256))
        x = res_identity(x, filters=(256, 256))

        x = res_conv(x, s=2, filters=(512, 512))
        x = res_identity(x, filters=(512, 512))
        x = res_identity(x, filters=(512, 512))

        x = GlobalAveragePooling2D()(x)
        x = Dense(4, activation='softmax', kernel_initializer='he_normal')(x)

        model = Model(inputs=input_im, outputs=x, name='ResNet34')
        return model

    #ResNet-34 model
    model_ResNet = resnet34()



    model_ResNet.summary()
```

```
 activation_30 (Activation)     (None, 7, 7, 512)    0           ['add_14[0][0]']

 conv2d_50 (Conv2D)             (None, 7, 7, 512)    2359808     ['activation_30[0][0]']

 batch_normalization_34 (BatchN  (None, 7, 7, 512)   2048        ['conv2d_50[0][0]']
 ormalization)

 activation_31 (Activation)     (None, 7, 7, 512)    0           ['batch_normalization_34[0][0]']

 conv2d_51 (Conv2D)             (None, 7, 7, 512)    2359808     ['activation_31[0][0]']

 batch_normalization_35 (BatchN  (None, 7, 7, 512)   2048        ['conv2d_51[0][0]']
 ormalization)

 add_15 (Add)                   (None, 7, 7, 512)    0           ['batch_normalization_35[0][0]',
                                                                  'activation_30[0][0]']

 activation_32 (Activation)     (None, 7, 7, 512)    0           ['add_15[0][0]']

 global_average_pooling2d (Glob  (None, 512)         0           ['activation_32[0][0]']
 alAveragePooling2D)

 dense_5 (Dense)                (None, 4)            2052        ['global_average_pooling2d[0][0]'
                                                                 ]

==================================================================================================
Total params: 21,312,260
Trainable params: 21,295,236
Non-trainable params: 17,024
__________________________________________________________________________________________________
```

```
# Set the loss function and metrics
model_ResNet.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
              loss='categorical_crossentropy',
              metrics=['accuracy'])


history_ResNet = model_ResNet.fit(
    x = train_data,
    validation_data = test_data,
    epochs = epoch)
```

```
 Epoch 1/10
 141/141 [==============================] - 104s 564ms/step - loss: 7.4921 - accuracy: 0.7314 - val_loss: 7.5170 - val_accuracy: 0.2922
 Epoch 2/10
 141/141 [==============================] - 77s 545ms/step - loss: 3.9502 - accuracy: 0.8666 - val_loss: 4.2296 - val_accuracy: 0.5400
 Epoch 3/10
 141/141 [==============================] - 78s 551ms/step - loss: 2.5320 - accuracy: 0.8624 - val_loss: 3.2131 - val_accuracy: 0.5924
 Epoch 4/10
 141/141 [==============================] - 77s 544ms/step - loss: 2.3438 - accuracy: 0.8382 - val_loss: 264.3384 - val_accuracy: 0.1723
 Epoch 5/10
 141/141 [==============================] - 78s 551ms/step - loss: 1.8235 - accuracy: 0.8908 - val_loss: 1.8099 - val_accuracy: 0.7345
 Epoch 6/10
 141/141 [==============================] - 77s 544ms/step - loss: 1.2735 - accuracy: 0.9077 - val_loss: 1.2489 - val_accuracy: 0.8233
 Epoch 7/10
 141/141 [==============================] - 84s 597ms/step - loss: 1.0384 - accuracy: 0.9154 - val_loss: 1.0734 - val_accuracy: 0.8650
 Epoch 8/10
 141/141 [==============================] - 78s 552ms/step - loss: 0.9304 - accuracy: 0.9119 - val_loss: 2.8826 - val_accuracy: 0.6892
 Epoch 9/10
 141/141 [==============================] - 78s 550ms/step - loss: 0.8167 - accuracy: 0.9221 - val_loss: 1.7923 - val_accuracy: 0.6989
```

```
Epoch 10/10
141/141 [==============================] - 77s 546ms/step - loss: 0.7626 - accuracy: 0.9205 - val_loss: 8.7169 - val_accuracy: 0.7513
```

```
%%capture
ResNet_train_loss, ResNet_train_accuracy = model_ResNet.evaluate(train_data)
ResNet_validation_loss, ResNet_validation_accuracy = model_ResNet.evaluate(test_data)
```

```
print('Training loss: ', ResNet_train_loss)
print('Training accuracy: ', ResNet_train_accuracy)

print('Validation loss: ', ResNet_validation_loss)
print('Validation accuracy: ', ResNet_validation_accuracy)
```

```
Training loss:  8.126148223876953
Training accuracy:  0.7751387357711792
Validation loss:  8.43387222290039
Validation accuracy:  0.7468916773796082
```

Transfer Learning

```
# Load the VGG16 model with pre-trained weights
TL_base_model = VGG16(weights="imagenet", include_top=False, input_shape=input_shape)  # Adjust input shape

# Freeze the pre-trained layers
TL_base_model.trainable = False

# Get the images and labels separately from the generator output
train_data_x, train_data_y = train_data.next()  # Assuming train_data is a data generator
test_data_x, test_data_y = test_data.next()    # Assuming test_data is a data generator

# Preprocess the image data
train_data_x = preprocess_input(train_data_x)
test_data_x = preprocess_input(test_data_x)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58889256/58889256 [==============================] - 0s 0us/step
```

```
# Add custom layers for your classification task
flatten_layer = Flatten()
dense_layer_1 = Dense(50, activation='relu')
dense_layer_2 = Dense(20, activation='relu')
prediction_layer = Dense(4, activation='softmax')  # Use 4 units for 4 classes
```

```
# Create the final model by stacking the base model and custom layers
model_TL = tf.keras.Sequential([
    TL_base_model,
    flatten_layer,
    dense_layer_1,
    dense_layer_2,
```

```
    prediction_layer
])


model_TL.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 7, 7, 512)         14714688

 flatten_2 (Flatten)         (None, 25088)             0

 dense_6 (Dense)             (None, 50)                1254450

 dense_7 (Dense)             (None, 20)                1020

 dense_8 (Dense)             (None, 4)                 84

=================================================================
Total params: 15,970,242
Trainable params: 1,255,554
Non-trainable params: 14,714,688
_________________________________________________________________
```

```
# Compile the model with an appropriate optimizer, loss function, and metrics
model_TL.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Train the model using your train_data generator and validate on test_data generator



history_TL = model_TL.fit(
    x=train_data_x,  # Training data
    y=train_data_y,  # Training labels
    epochs=epoch,        # Adjust the number of epochs as needed
    validation_data=(test_data_x, test_data_y)  # Validation data as a tuple (images, labels)
)
```

```
Epoch 1/10
1/1 [==============================] - 2s 2s/step - loss: 3.3614 - accuracy: 0.2188 - val_loss: 3.0837 - val_accuracy: 0.0000e+00
Epoch 2/10
1/1 [==============================] - 0s 265ms/step - loss: 1.6930 - accuracy: 0.3125 - val_loss: 2.1590 - val_accuracy: 0.0000e+00
Epoch 3/10
1/1 [==============================] - 0s 304ms/step - loss: 1.4344 - accuracy: 0.2812 - val_loss: 1.2759 - val_accuracy: 0.0000e+00
Epoch 4/10
1/1 [==============================] - 0s 262ms/step - loss: 1.5295 - accuracy: 0.2812 - val_loss: 1.0334 - val_accuracy: 1.0000
Epoch 5/10
1/1 [==============================] - 0s 261ms/step - loss: 1.5092 - accuracy: 0.1875 - val_loss: 1.2372 - val_accuracy: 0.0000e+00
Epoch 6/10
1/1 [==============================] - 0s 268ms/step - loss: 1.4391 - accuracy: 0.2812 - val_loss: 1.7061 - val_accuracy: 0.0000e+00
Epoch 7/10
1/1 [==============================] - 0s 263ms/step - loss: 1.4317 - accuracy: 0.3125 - val_loss: 2.0908 - val_accuracy: 0.0000e+00
Epoch 8/10
1/1 [==============================] - 0s 267ms/step - loss: 1.4279 - accuracy: 0.3125 - val_loss: 2.2597 - val_accuracy: 0.0000e+00
Epoch 9/10
```

```
1/1 [==============================] - 0s 263ms/step - loss: 1.3984 - accuracy: 0.3125 - val_loss: 2.2779 - val_accuracy: 0.0000e+00
Epoch 10/10
1/1 [==============================] - 0s 261ms/step - loss: 1.4327 - accuracy: 0.2188 - val_loss: 2.0611 - val_accuracy: 0.0000e+00
```

```
%%capture
TL_train_loss, TL_train_accuracy = model_TL.evaluate(train_data)
TL_validation_loss, TL_validation_accuracy = model_TL.evaluate(test_data)
```

```
print('Training loss: ', TL_train_loss)
print('Training accuracy: ', TL_train_accuracy)

print('Validation loss: ', TL_validation_loss)
print('Validation accuracy: ', TL_validation_accuracy)
```

```
Training loss:  1.4397505521774292
Training accuracy:  0.2011098712682724
Validation loss:  1.4411168098449707
Validation accuracy:  0.2007104754447937
```

InceptionV3

```
from tensorflow.keras.applications import InceptionV3
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelBinarizer
```

```
# # Set random seed for reproducibility
# np.random.seed(42)
# tf.random.set_seed(42)
```

```
# Load InceptionV3 pre-trained model without the top (fully connected) layers
Inception_base_model = InceptionV3(weights='imagenet', include_top=False, input_shape=input_shape)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/inception_v3/inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5
87910968/87910968 [==============================] - 1s 0us/step
```

```
# Add custom fully connected layers for classification
z = Inception_base_model.output
z = GlobalAveragePooling2D()(z)
z = Dense(1024, activation='relu')(z)
predictions = Dense(num_label, activation='softmax')(z)
```

```
# Create the final model
model_Inception = Model(inputs=Inception_base_model.input, outputs=predictions)



# Compile the model
model_Inception.compile(
    optimizer=tf.keras.optimizers.legacy.Adam(lr=0.0001),
    loss='categorical_crossentropy', metrics=['accuracy'])
```

```
 /opt/conda/lib/python3.10/site-packages/keras/optimizers/legacy/adam.py:117: UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
   super().__init__(name, **kwargs)
```

```
model_Inception.fit(
    train_data,
    steps_per_epoch=len(train_data),
    epochs=epoch,
    validation_data=test_data,
    validation_steps=len(test_data))
```

```
 Epoch 1/10
 141/141 [==============================] - 99s 626ms/step - loss: 0.1350 - accuracy: 0.9547 - val_loss: 5.1899 - val_accuracy: 0.4529
 Epoch 2/10
 141/141 [==============================] - 82s 577ms/step - loss: 0.0245 - accuracy: 0.9925 - val_loss: 0.3939 - val_accuracy: 0.8899
 Epoch 3/10
 141/141 [==============================] - 81s 575ms/step - loss: 0.0132 - accuracy: 0.9958 - val_loss: 0.0109 - val_accuracy: 0.9956
 Epoch 4/10
 141/141 [==============================] - 82s 578ms/step - loss: 0.0098 - accuracy: 0.9978 - val_loss: 0.0258 - val_accuracy: 0.9929
 Epoch 5/10
 141/141 [==============================] - 87s 617ms/step - loss: 0.0056 - accuracy: 0.9980 - val_loss: 0.0139 - val_accuracy: 0.9956
 Epoch 6/10
 141/141 [==============================] - 81s 575ms/step - loss: 0.0188 - accuracy: 0.9945 - val_loss: 0.0109 - val_accuracy: 0.9964
 Epoch 7/10
 141/141 [==============================] - 81s 574ms/step - loss: 0.0032 - accuracy: 0.9987 - val_loss: 0.0394 - val_accuracy: 0.9902
 Epoch 8/10
 141/141 [==============================] - 81s 570ms/step - loss: 0.0174 - accuracy: 0.9980 - val_loss: 0.9328 - val_accuracy: 0.8419
 Epoch 9/10
 141/141 [==============================] - 82s 578ms/step - loss: 0.0265 - accuracy: 0.9918 - val_loss: 0.0026 - val_accuracy: 0.9991
 Epoch 10/10
 141/141 [==============================] - 82s 582ms/step - loss: 8.5671e-04 - accuracy: 1.0000 - val_loss: 8.3545e-04 - val_accuracy: 1.0000
 <keras.callbacks.History at 0x7aa32b11fc10>
```

```
Inception_train_loss, Inception_train_accuracy = model_Inception.evaluate(train_data, verbose=0)
Inception_validation_loss, Inception_validation_accuracy = model_Inception.evaluate(test_data, verbose=0)
```

```
print('Validation loss: ', Inception_validation_loss)
print('Validation accuracy: ', Inception_validation_accuracy)
```

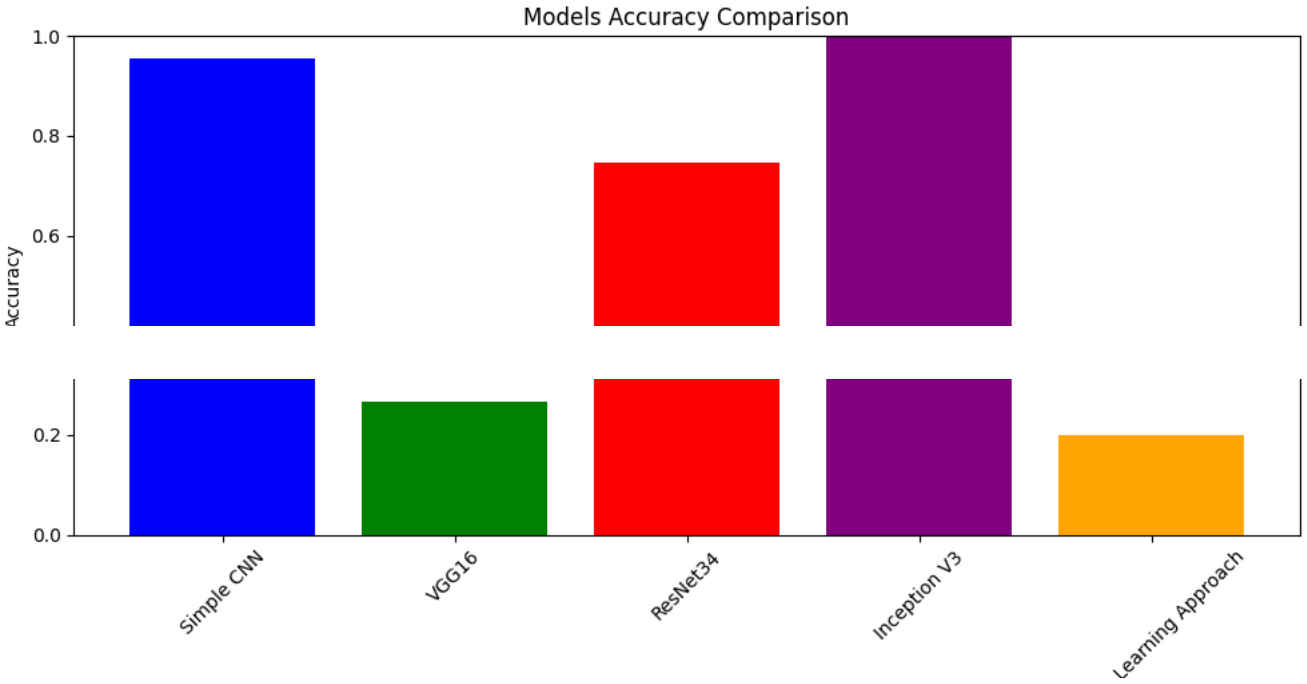
```
 Validation loss:  0.005212030373513699
 Validation accuracy:  0.99733567237854
```

Comparision of Models

```
import matplotlib.pyplot as plt
import numpy as np

# Replace these with your actual model names and accuracy values
model_names = ['Simple CNN', 'VGG16', 'ResNet34', 'Inception V3', 'Transfer Learning Approach']
accuracies = [CNN_validation_accuracy, VGG16_validation_accuracy, ResNet_validation_accuracy,
                Inception_validation_accuracy, TL_validation_accuracy] #all are calculated previously

accuracies_train = [CNN_train_accuracy, VGG16_train_accuracy, ResNet_train_accuracy,
                Inception_train_accuracy, TL_train_accuracy]
# Create a DataFrame
data = {'Model': model_names, 'Train Accuracy': accuracies_train, 'Test Accuracy': accuracies}
df = pd.DataFrame(data)

# Print the DataFrame
print(df)
```

```
                         Model  Train Accuracy  Test Accuracy
0                   Simple CNN        0.978246       0.956483
1                        VGG16        0.266371       0.266430
2                     ResNet34        0.775139       0.746892
3                  Inception V3        0.999334       0.997336
4  Transfer Learning Approach        0.201110       0.200710
```

```
# Create a barplot
plt.figure(figsize=(10, 6))
plt.bar(model_names, accuracies, color=['blue', 'green', 'red', 'purple', 'orange'])
plt.xlabel('Models')
plt.ylabel('Accuracy')
plt.title('Models Accuracy Comparison')
plt.ylim(0, 1.0)  # Set the y-axis limit if needed
plt.xticks(rotation=45)  # Rotate x-axis labels for readability
plt.tight_layout()

# Show the plot
plt.show()
```


Models Accuracy Comparison
Accuracy
1.0
0.8
0.6
0.2
0.0
Simple CNN
VGG16
ResNet34
Inception V3
Learning Approach