Presented by:

Maryam Bandali - 1861486

Abdolhadi Rezaei- 1837982

SAPIENZA
UNIVERSITÀ DI ROMA

EuroSAT: A Novel Dataset and Deep Learning Benchmark for Land Use and Land Cover Classification

Vision & Perception

Project Definition:

Purpose of project is satellite images classification to detect areas such as : Annual Crop, Forest, Herbaceous Vegetation, Highway, Industrial, Pasture, Permanent Crop, Residential , River and SeaLake (10 classes)

In this paper, we make the following contributions:

1) We provide benchmarks for the proposed EuroSAT dataset using Convolutional Neural Networks (CNNs).
2) We evaluate the performance of each spectral band of the Sentinel-2 satellite for the task of patch- based land use and land cover classification.
3) In order to improve the performance of the network we have made use of some overfit techniques such as Dropout and Data Augmentation.
4) We have done experiments with a pretrained network(ResNet) and we have got astonishing result
5) Finally, we have tested the output of the network in the form of an application.

Objective of the project

Implementation Steps:

1) Data preparation(Train/Validation/Test)
2) Visualising
3) Creating layers and models
4) Setting training parameters(Loss & Optimization Function,...)
5) Train the model (Using fit())
6) Analyze results
7) Overfit(Dropout & Data Augmentation)
8) Transfer Learning(ResNet)
9) Using our pretrained model

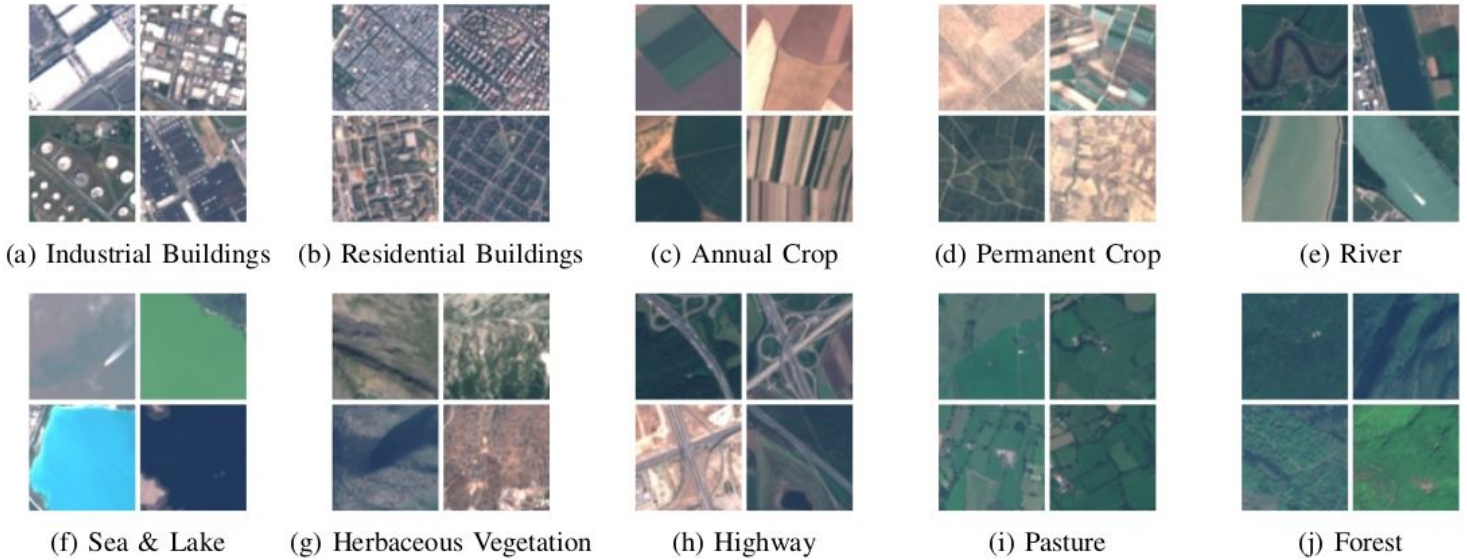
Data preparation(Train, Validation and Test)

We have made a function(make_directory()) for data preparation,once you download from internet there are just 10 folders into EuroSAT folder which are our classes, by running this function we divide our dataset into train,test and validation folders as we need for training.

- train: learning the parameters of the model.
- valid: learning hyper-parameters.
- test : evaluate the dataset

Visualising

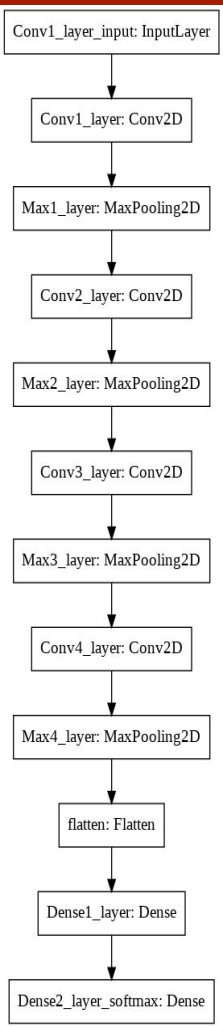
This overview shows sample image patches of all 10 classes covered in the proposed EuroSAT dataset. The images measure 64x64 pixels. Each class contains 2,000 image. In total, the dataset has 20,000 geo-referenced images.



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

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

Creating layers and models




```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Conv1 layer (Conv2D)         (None, 64, 64, 16)        1216
_________________________________________________________________
Max1 layer (MaxPooling2D)    (None, 32, 32, 16)        0
_________________________________________________________________
Conv2 layer (Conv2D)         (None, 32, 32, 32)        12832
_________________________________________________________________
Max2 layer (MaxPooling2D)    (None, 16, 16, 32)        0
_________________________________________________________________
Conv3 layer (Conv2D)         (None, 16, 16, 64)        51264
_________________________________________________________________
Max3 layer (MaxPooling2D)    (None, 8, 8, 64)          0
_________________________________________________________________
Conv4 layer (Conv2D)         (None, 8, 8, 128)         204928
_________________________________________________________________
Max4 layer (MaxPooling2D)    (None, 4, 4, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 2048)              0
_________________________________________________________________
Dense1 layer (Dense)         (None, 2048)              4196352
_________________________________________________________________
Dense2 layer softmax (Dense) (None, 10)                20490
=================================================================
Total params: 4,487,082
Trainable params: 4,487,082
Non-trainable params: 0
```

Setting training parameters(Loss & Optimization Function,...)

```
model.compile(optimizer=
        tf.keras.optimizers.RMSprop(0.001),
            loss='categorical_crossentropy',
                metrics=['acc'])
```

Setting training parameters

```
# Normalize
train_datagen = ImageDataGenerator(rescale=1./255)
validation_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
        # This is the target directory
        train_dir,
        # All images will be resized to 64*64
        target_size=(64, 64),
        batch_size=20,
        class_mode='categorical',shuffle=True)#because we have 10
classes

validation_generator =validation_datagen.flow_from_directory(
        validation_dir,target_size=(64,
64),batch_size=20,class_mode='categorical',shuffle=True)
```

There is a lot of work in building an iterator in Python,Python generators are a simple way of creating iterators.
Simply speaking, a generator is a function that returns an object (iterator) which we can iterate over (one value at a time).
we are going to show you how to generate our dataset on multiple cores in real time and feed it right away to our deep learning model.
It may seem a bit daunting, but thankfully Keras has utilities to take care of these steps automatically. Keras has a module with image
processing helper tools, located at `keras.preprocessing.image`. In particular, it contains the class `ImageDataGenerator` which allows to
quickly set up Python generators that can automatically turn image files on disk into batches of pre-processed tensors. This is what we
will use here.
`https://keras.io/preprocessing/image/`

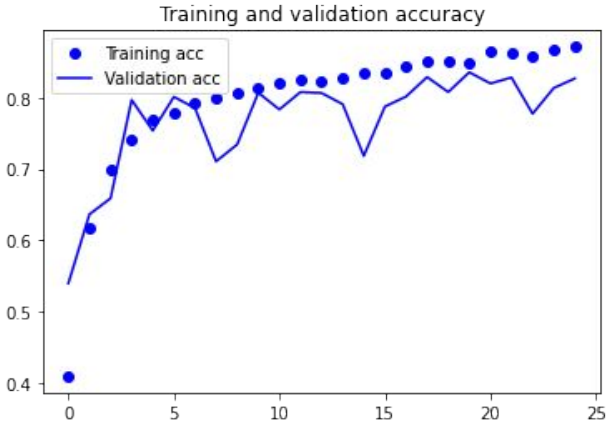
Train the model (Using fit())

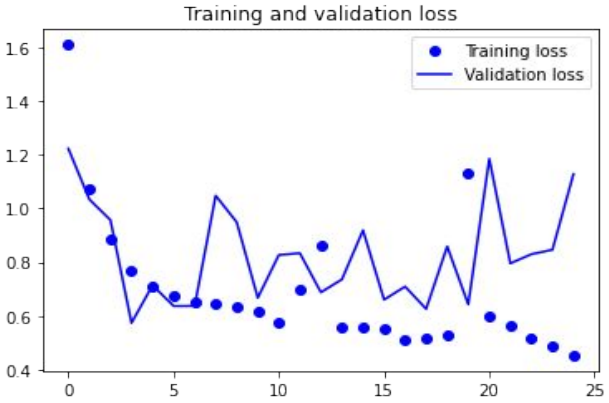
```
history = model.fit(
      train_generator,

steps_per_epoch=700,
      epochs=25,

      validation_data=
validation_generator,

validation_steps=150
      )
```

Let's fit our model to the data using the generator. We do it using the `fit` method. Because the data is being generated endlessly, the generator needs to know example how many samples to draw from the generator before declaring an epoch over. This is the role of the `steps_per_epoch` argument. i.e. after having run for `steps_per_epoch` gradient descent steps, the fitting process will go to the next epoch. In our case, batches are 20-sample large, so it will take 700 batches until we see our target of 1400 samples.

When using `fit_generator`, one may pass a `validation_data` argument, much like with the `fit` method. Importantly, this argument is allowed to be a data generator itself, but it could be a tuple of Numpy arrays as well. If you pass a generator as `validation_data`, then this generator is expected to yield batches of validation data endlessly, and thus you should also specify the `validation_steps` argument, which tells the process how many batches to draw from the validation generator for evaluation.

Analyze Results





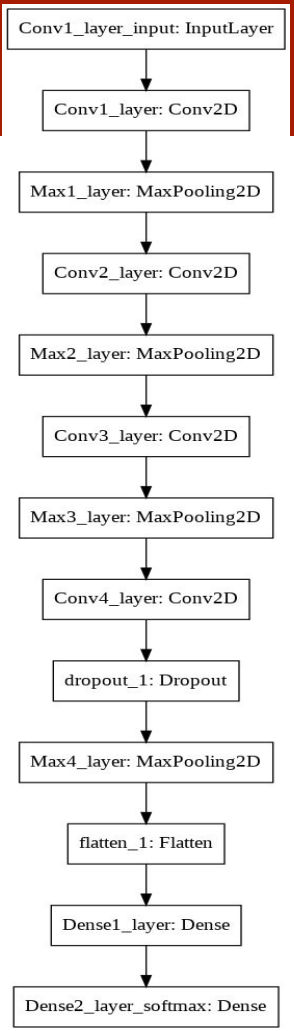

```
loss: 0.4523 - acc: 0.8711 - val_loss: 1.1284 - val_acc:
0.8267
```

As you see in the plot, we have faced overfitting in this stage, because training pattern does not follow the validation pattern.
Due to this fact, we have taken advantages of Dropout and Data Augmentation techniques.

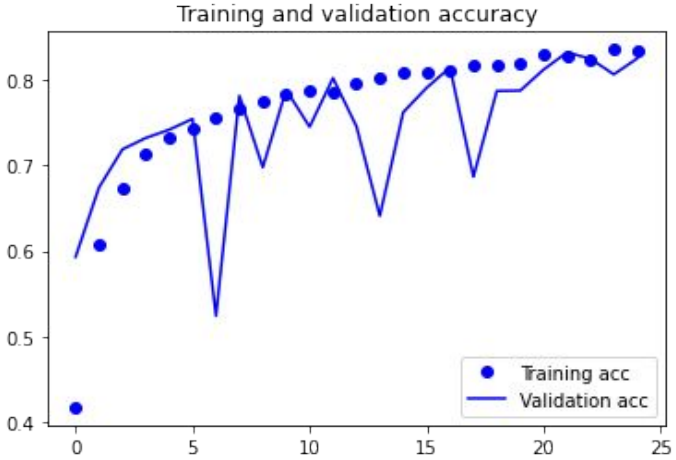
Confusion matrix

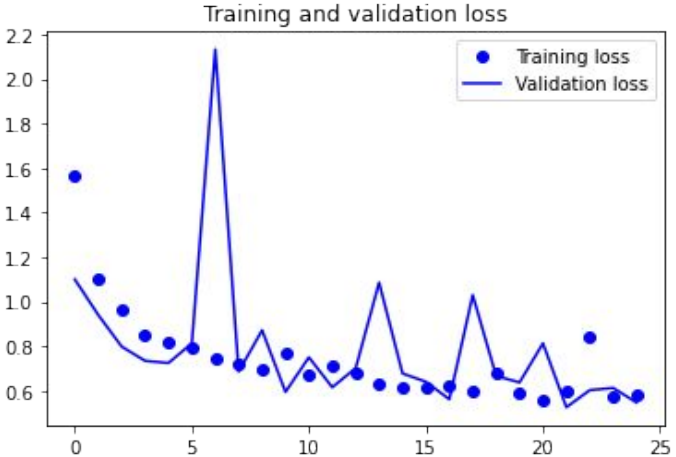
```
array([[214,   0,   8,  10,   0,   5,  45,   0,  17,   1],
       [  0, 250,   6,   0,   0,  28,   3,   1,   4,   8],
       [  3,   2, 218,   5,   2,   4,  52,  13,   1,   0],
       [  6,   0,   6, 219,   6,   3,  27,   8,  25,   0],
       [  0,   0,   5,   1, 241,   0,   7,  46,   0,   0],
       [  2,   0,   7,   6,   0, 236,  46,   1,   2,   0],
       [  3,   0,  43,   5,   1,   0, 238,   9,   1,   0],
       [  0,   0,   1,   0,   5,   0,   0, 294,   0,   0],
       [  2,   1,   7,  17,   1,   2,   9,   0, 260,   1],
       [  6,   4,   2,   0,   0,   4,   0,   0,   5, 279]])
```

Overfit(Dropout)




```
model2.add(tf.keras.layers.Dropout(0.3))

loss: 0.5821 - acc: 0.8336 - val_loss:
0.5470 - val_acc: 0.8250
```





Confusion matrix after Dropout

```
array([[232,   0,   6,   5,   0,  20,  18,   0,  17,   2],
       [  0, 278,   4,   0,   0,  18,   0,   0,   0,   0],
       [  3,   8, 217,   2,   3,   5,  49,  10,   2,   1],
       [  9,   3,   7, 216,   9,   5,  21,   4,  26,   0],
       [  0,   0,   7,   3, 257,   0,   4,  29,   0,   0],
       [  3,   1,   9,   1,   0, 277,   4,   0,   4,   1],
       [  3,   0,  46,   3,   4,  14, 224,   6,   0,   0],
       [  0,   1,   4,   1,   1,   1,   1, 291,   0,   0],
       [ 15,   1,   7,  20,   0,  21,   5,   0, 229,   2],
       [  2,  17,   1,   0,   0,   6,   0,   0,   2, 272]])
```

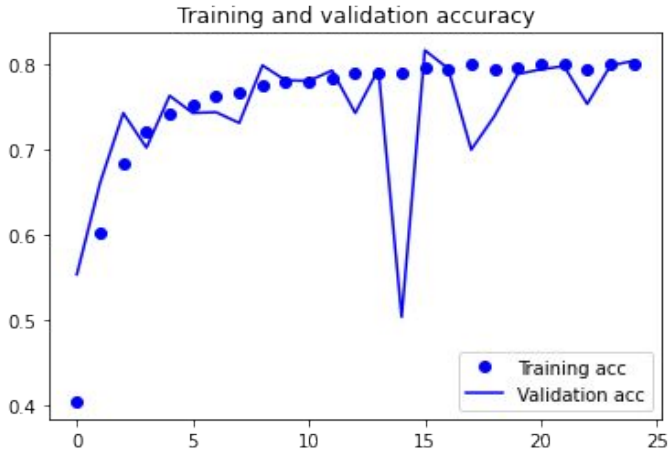
Overfit(Data Augmentation)

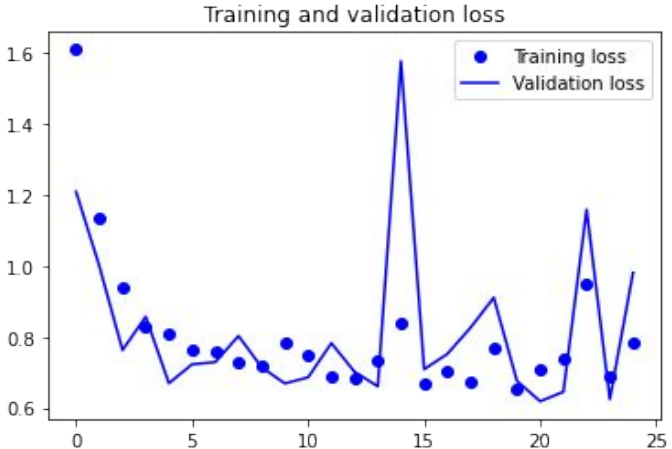
```
datagen = ImageDataGenerator(rotation_range=40,
      width_shift_range=0.2, height_shift_range=0.2,
      shear_range=0.2, zoom_range=0.2,
      horizontal_flip=True, fill_mode='nearest')
```

If we train a new network using this data augmentation configuration, our network will never see twice the same input. However, the inputs that it sees are still heavily intercorrelated, since they come from a small number of original images -- we cannot produce new information, we can only remix existing information. As such, this might not be quite enough to completely get rid of overfitting.

Rotation_range is a value in degrees (0-180), a range within which to randomly rotate pictures.
Width_shift and **height_shift** are ranges (as a fraction of total width or height) within which to randomly translate pictures
vertically or horizontally.
shear_range is for randomly applying shearing transformations.
zoom_range is for randomly zooming inside pictures.
horizontal_flip is for randomly flipping half of the images horizontally -- relevant when there are no assumptions of horizontal
asymmetry (e.g. real-world pictures).
fill_mode is the strategy used for filling in newly created pixels, which can appear after a rotation or a width/height shift.

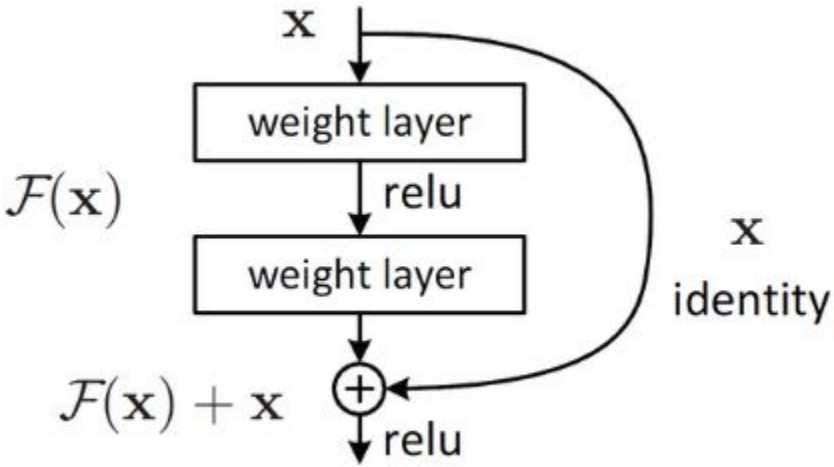
```
loss: 0.7853 - acc: 0.7991 - val_loss: 0.9803 - val_acc: 0.8037
```





Residual Learning

The residual block is consists of 2 layers, one non-linearity in the middle which they use ReLU and the x connection is the identity and they add x and F(x) representation.
This method allows us to design deeper networks in order to deal with Much complicated problems and tasks.



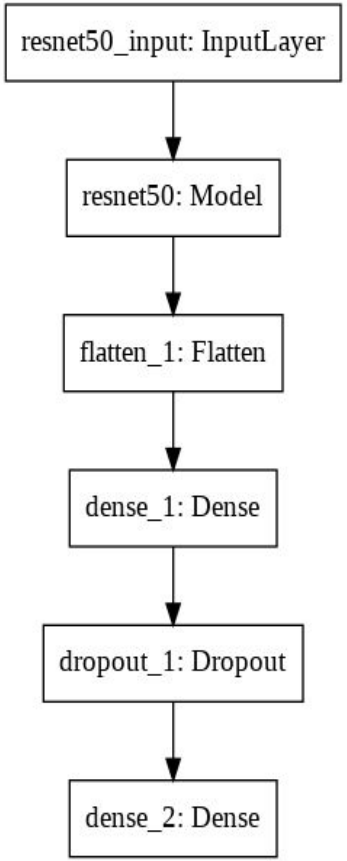
Transfer Learning(ResNet)

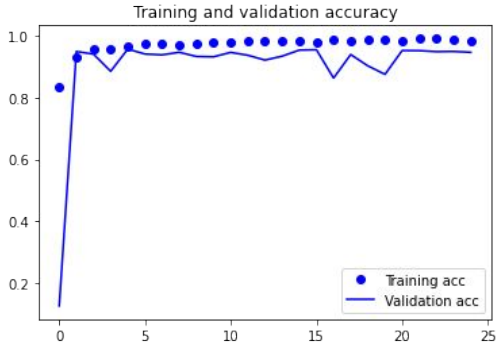
Transfer learning is a machine learning method where a model developed for a task is reused as the starting point for a model on a second task.
Use a CNN already trained on different data (ResNet)
In this project, you will discover how you can use transfer learning to speed up training and improve the performance of your deep learning model.

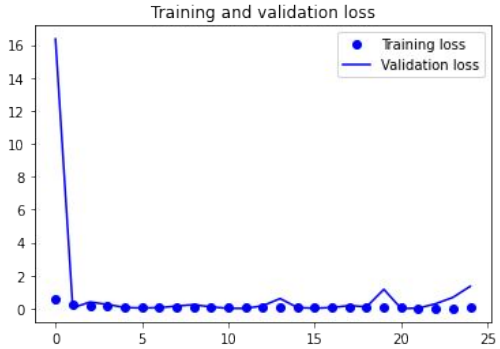
This model was the winner of ImageNet challenge in 2015. The fundamental breakthrough with ResNet was it allowed us to train extremely deep neural networks with 150 layers successfully.

```
model4 = models.Sequential()
model4.add(resnet_model)
model4.add(layers.Flatten())
model4.add(layers.Dense(1024
, activation='relu'))
model4.add(Dropout(0.4))
model4.add(layers.Dense(10,
activation='softmax'))
```

```
loss: 0.0684 - acc: 0.9840 -
val_loss: 1.3526 - val_acc:
0.9473
```







Confusion matrix after Transfer Learning

```
array([[270,   0,   0,   3,   0,   2,  10,   0,  10,   5],
       [  0, 287,   6,   0,   0,   1,   0,   0,   1,   5],
       [  0,   1, 282,   0,   1,   8,   6,   0,   0,   2],
       [  1,   1,   4, 265,   3,   2,   2,   0,  12,  10],
       [  0,   0,   1,   2, 291,   0,   4,   2,   0,   0],
       [  3,   0,   8,   0,   0, 286,   0,   0,   2,   1],
       [  9,   0,  23,   0,   2,   6, 259,   0,   1,   0],
       [  0,   0,   6,   1,   2,   0,   0, 291,   0,   0],
       [  4,   0,   1,   9,   1,   0,   0,   0, 280,   5],
       [  1,   0,   0,   0,   0,   0,   0,   0,   3, 296]])
```

Using our pretrained model

We are going to show you, how it is possible to use the pretrained model.

Load trained model

```
from keras.models import load_model
model = load_model('/content/drive/My Drive/Classroom/Vision and Perception/Final Project /saveModels
/euroSAT_transfer_learning1.h5')
img = image.load_img(img_path, target_size=(64, 64))  #Load the image
imshow(img)
x = image.img_to_array(img)
x /= 255.
x = np.expand_dims(x, axis=0)
print(model.predict(x))
myClass=model.predict_classes(x)
print(myClass)
```

Using our pretrained model

```
if myClass==0:
 print('The photo that you have uploaded is related to AnnualCrop category' )
elif myClass==1:
 print('The photo that you have uploaded is related to Forest category' )
elif myClass==2:
 print('The photo that you have uploaded is related to HerbaceousVegetation category' )
elif myClass==3:
 print('The photo that you have uploaded is related to Highway category' )
elif myClass==4:
 print('The photo that you have uploaded is related to industrial category' )
elif myClass==5:
 print('The photo that you have uploaded is related to pasture category' )
elif myClass==6:
 print('The photo that you have uploaded is related to permanentCrop category' )
elif myClass==7:
 print('The photo that you have uploaded is related to residental category' )
elif myClass==8:
 print('The photo that you have uploaded is related to river category' )
else:
 print('The photo that you have uploaded is related to seaLake category' )
```

Output:

The photo that you have uploaded is related to **residential** category

Conclusions

What we were expecting and what we have got:
After applying data augmentation, we have got accuracy of 80%, however, we were expecting a higher accuracy. We believe this result is highly related to the type of dataset. For instance, in the dataset of dog vs cat, after applying data augmentation, you get a good result, because the images are variant and applying this technique can make a huge diversity in the dataset. Conversely, in our data set, we have images which are not geometrically diverse, so the data augmentation does not make any significant effect. All in all, we have applied different kind of data augmentation such as rotation_range, width_shift_range, height_shift_range, shear_range, zoom_range, horizontal_flip, fill_mode. Consequently, another result is that we use data augmentation technique when we have a low number of data in our dataset. But, the number of data in our dataset is rather appropriate (1400 images in each class for training). As a result, we need a more complex model to improve our accuracy, So when we have used ResNet in our model, we have reached the accuracy of 98%.

Using drop out technique, we get the following result:
By using this technique, we could improve the pattern of overfitting a little bit as we have expected, however, the accuracy decreased about 2%. We believe that the reason of this phenomenon would be as above.

Thanks for your attention!