GRENOBLE INP - PHELMA

Machine learning and deep learning project report

Image classification

Students:
Allan Dizet
Matteo Marengo

Lecturer:
Alice Caplier

Academic year 2022/2023
3A - Biomedical Engineering

Machine learning and deep learning project

December 31, 2022

Contents

1 Introduction

1.1 Presentation of the project : objectives and challenges

This machine and deep learning project concerning Image classification is the final work of the semester. We will have to use different machine and deep learning techniques we learned in order to be able to classify images with the best results and a not a too high complexity (calculation time). With a wider scope, we should also consider how this project will help us to develop our skills and competencies; how to solve a given problem.

We will have to run a 2-class classification and then a 8-class classification. To understand this statement, let's look at the data we have to analyze and classify.

1.2 Presentation of the data at our disposal

The data at our disposal is this one : 2688 color images of size 64×64 pixels with two different labels.

- The first label is the main class of the image : [ARTIFICIAL, NATURAL].
- The second label is the secondary class of the image : [SEE SHORE, FOREST, HIGHWAY, CITY, MOUNTAIN, OPEN COUNTRY, STREET, BIG BUILDING]

With a simple script that was given to us, it was concluded that there were **1216** artificial images and **1472** natural images. And concerning the 8-class classification it is divided like that : **[360. 328. 260. 308. 374. 410. 292. 356.]**

In addition to the color images, each image is also associated with 27 handcrafted features. 24 come from different **Gabor filters** (8 different orientations, 3 different frequencies), one is the **mean Y value** of the image, one is the **mean Cb value** of the image and the last one is the **mean Cr value** of the image.

A Gabor filter is a mathematical operation used to extract features from images. It is a type of convolutional filter that is specifically designed to be sensitive to particular frequencies and orientations in an image. The Gabor filter works by convolving an image with a small kernel, which is a window of pixels with a particular shape.

This kernel is designed to have a sinusoidal pattern that is sensitive to a particular frequency and orientation. As the kernel moves over the image, it looks for pixels that are similar in intensity to the pattern in the kernel. These pixels are then highlighted, while other pixels are suppressed.

The mean Cb value of an image is a measure of the average chrominance (color) of the image in the Cb (blue-difference) channel. Cr is in the red -difference range. In the YCbCr color space, which is commonly used in image and video processing, the Y channel represents the luminance (brightness) of the image, while the Cb and Cr channels represent the chrominance (color) of the image.

All these handcrafted features are already normalized, so no preprocessing in our scripts will be needed to do the classification.

1.3 Our experimental plans for the classification

The first step is to do the 2-class classification. As the two main objectives are simplicity in terms of classification complexity (calculation time) and an 85 % test accuracy at least. Therefore, we will start with the simplest data, the already made 27 handcrafted features as the complexity will be less than extracting features from raw images. With these features, the first classification techniques that will be used would be :

- Naive Bayes classification
- K-NN classification
- Logistic regression
- Support Vector Machines (SVMs)

For the 8-class classification, if one of the mentioned classification method works we will try to implement it. If none of them work it would be then mandatory to look through the scope of deep learning and doing a neural network (NN) or a convolutional neural network (CNN).

1.4 Materials

To work in collaboration, we decided to use Google Colab as we can also use a GPU that help us to have faster results. If we had small codes to test, Spyder IDE was also chosen to run our scripts.

For the basic classification technics, **sklearn** library will be used and for neural networks it will be the **keras** one from TensorFlow.

2 2-Class Classification Problem

2.1 Naive Bayes classification

Naive Bayes is a probabilistic classifier that uses Bayes' theorem to predict the probability of each class given the input features. It assumes that the features are independent and uses the product of the individual feature probabilities to calculate the probability of the class. Naive Bayes is fast and simple to implement, and works well on high-dimensional data with many features.

The script that was used is the one in Listing 1.

```python
from sklearn.naive_bayes import GaussianNB

gnb = GaussianNB()

Y_2c = Y[:,0] # We only take the 2 classes (1: ARTIFICIAL - 2: NATURAL)
X_train, X_test, y_train, y_test = train_test_split(X, Y_2c, test_size = 0.3,
    random_state = 0)

gnb_learned = gnb.fit(X_train,y_train)

ypredtrain = gnb_learned.predict(X_train)
ypredtest = gnb_learned.predict(X_test)

# Accuracy computation

print('Train accuracy: ',metrics.accuracy_score(y_train, ypredtrain))
print('Test accuracy: ',metrics.accuracy_score(y_test,ypredtest))
```

Listing 1: Naive Bayes Classification

Here are the results obtained:

- Train accuracy (%) : 81.7
- Test accuracy (%) : 82.2
- execution time (ms) : 26

Results are not good enough in terms of accuracy and calculation time, the KNN classification will then be trained. One of the challenge with the Naive Bayes classification is that there a data independance hypothesis.

2.2 K-NN classification

K-nearest neighbors (KNN) is a non-parametric classification algorithm that makes predictions based on the majority class of the nearest neighbors in the feature space. It stores all the training data and uses a distance measure to find the K nearest neighbors for a given test sample. The predicted class is the majority class of the K nearest neighbors. KNN is simple to implement, but can be slow for large datasets.

The script that was used is the one in Listing 2.

```
from sklearn.neighbors import KNeighborsClassifier

Y_2c = Y[:,0] # We only take the 2 classes (1: ARTIFICIAL - 2: NATURAL)
X_train, X_test, y_train, y_test = train_test_split(X, Y_2c, test_size = 0.3,
    random_state = 0)

neigh = KNeighborsClassifier(n_neighbors=3)

neigh_learned = neigh.fit(X_train,y_train)

ypredtrain = gnb_learned.predict(X_train)
ypredtest = gnb_learned.predict(X_test)

# Accuracy computation

print('Train accuracy: ',metrics.accuracy_score(y_train, ypredtrain))
print('Test accuracy: ',metrics.accuracy_score(y_test,ypredtest))
```

Listing 2: K-nearest neighbors Classification

Here are the results obtained:

- Train accuracy (%) : 81.7
- Test accuracy (%) : 82.2
- execution time (ms) :79

Results are still not good enough for the requirements. One of the challenge with K-NN is how to choose K. We will train logistic regression models as it has many advantages such as interpretability, training time, scalability, or regularization.

2.3 Logistic regression

Logistic regression is a classification algorithm that predicts the probability of a binary outcome based on input features.It estimates the probability of an event occurring based on the input variables using a logistic function, which is a sigmoid curve that maps any real-valued number to a value between 0 and 1. The model is trained by minimizing the binary cross-entropy loss.

The script that was used is the one in Listing 3.

```python
from sklearn.linear_model import LogisticRegression

# Logistic regression

Y_2c = Y[:,0] # We only take the 2 classes (1: ARTIFICIAL - 2: NATURAL)
X_train, X_test, y_train, y_test = train_test_split(X, Y_2c, test_size = 0.3,
    random_state = 0)

clf = LogisticRegression(C=1, max_iter=2900)
clf.fit(X_train,y_train)

ypredtrain = clf.predict(X_train)
ypredtest = clf.predict(X_test)

# Accuracy computation

print('Train accuracy: ',metrics.accuracy_score(y_train, ypredtrain))
print('Test accuracy: ',metrics.accuracy_score(y_test,ypredtest))
```

Listing 3: Logistic Regression for 2-Classes

Results obtained were impressive, and they correspond to the requirements of the project.

- Train accuracy (%) : 90.6
- Test accuracy (%) : 90.1
- execution time (ms) : 57

However, Support Vector Machines model will still have to be implemented as it seems better if we wonder to expand our model to the 8-class classification problem. Indeed, the model provides robustness to outliers, ability to handle non-linear boundary, sparse solutions and regularization compared to Logistic Regression.

2.4 SVM without PCA

Support Vector Machines (SVMs) are a type of supervised machine learning algorithm used for classification and regression tasks. They work by finding the hyperplane in a high-dimensional space that maximally separates different classes or values.

The script for this two class classification can be found in Listing 4. We first split the dataset within a train and a test batch. We split it randomly with 30 % for the testing batch and 70 % for the training batch. The kernel we choose can be either linear or Gaussian.

```python
# We only take the 2 classes (1: ARTIFICIAL - 2: NATURAL)
Y_2c = Y[:,0]
X_train, X_test, y_train, y_test = train_test_split(X, Y_2c, test_size = 0.3,
    random_state = 0)

# SVM classification
# kernel = 'linear' or 'rbf'
svc_model = svm.SVC(kernel=kernel)
svc_model.fit(X_train, y_train)

y_pred = svc_model.predict(X_train)
y_pred_test = svc_model.predict(X_test)
```

Listing 4: SVM-2Classes

The accuracy results are in Table 1 and it is more than expected (more than 85 %) and the time cost is good.

Kernel	Training Accuracy (%)	Testing accuracy (%)	Time cost (ms)
Linear	0.91	0.90	138
Gaussian	0.91	0.88	288

Table 1: SVM-2classes accuracy and time cost

With this first good trials, we can then try an other technique in addition to SVM in order to see its impact on the accuracy and time cost. This technique is Principal Component Analysis or PCA.

2.5 SVM with PCA

Principal Component Analysis (PCA) is a statistical technique that is used to reduce the dimensionality of a dataset while retaining as much of the variation in the data as possible. It does this by projecting the data onto a lower-dimensional space, known as the principal component space, in a way that maximizes the variance of the data along the new axes.

The script we will be using is the one in Listing 5.

```python
# PCA analysis
nb_pc = 6 # number of components
pca = decomposition.PCA(nb_pc)
principalComponents = pca.fit(X).transform(X)
```

Listing 5: SVM-2Classes with PCA

Kernel	Training Accuracy (%)	Testing accuracy (%)	Time cost (ms)
Linear	0.84	0.85	362
Gaussian	0.87	0.86	424

Table 2: SVM with PCA 2 Classes accuracy and time cost

We perform a Principal Component Analysis by keeping 6 components since the components above show a variance inferior to 5%. The variance repartition among features is as following : (1) 0.35, (2) 0.11, (3) 0.10, (4) 0.07, (5) 0.06 and (6) 0.06 with a total of kept variance equal to 75%. This behavior can be observed in Fig 1.

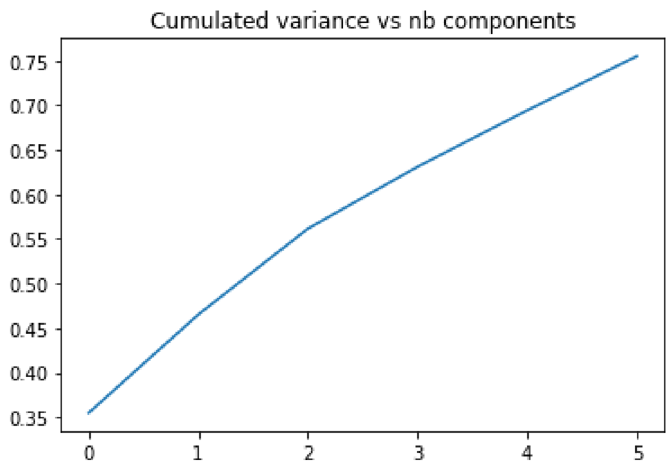

Figure 1: Cumulated variance vs nb of components

Results of the accuracy and time cost are plotted in Table 2. We also plot a visual display of the PCA in Fig 2.

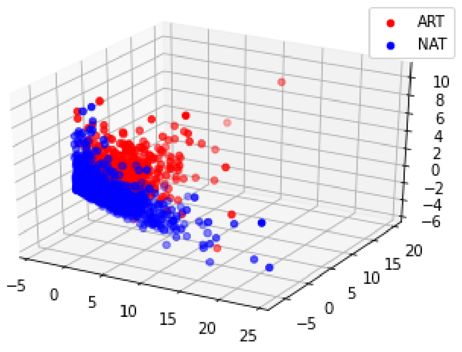

Figure 2: Visual display of the PCA

The model is still accurate but according to the time cost, the PCA shows little interest.We can then consider that we have solved the 2-class classification problem as we designed a classfifier able to produce a test accuracy higher than 85 %. Therefore, we will now be considering the 8-class classification problem. To start it simple, we will first apply the same simple method than the 2-class classification by doing a SVM.

3 8-Class Classification Problem

3.1 SVM without PCA

We select the accurate data (8-class output instead of the 2-class output). It was a bit of a challenge at first to extract these data and have them have the right dimension. We apply the same script as in Listing 4. The results that we obtain are displayed in Table 3.

Kernel	Training Accuracy (%)	Testing accuracy (%)	Time cost (ms)
Linear	0.77	0.69	268
Gaussian	0.77	0.67	533

Table 3: SVM without PCA 8 Classes accuracy and time cost

Using the previous model by taking instead of 2 classes, the 8 classes, the performance drops drastically. It is part of one of these cons that are to choose the right kernel, tune the best parameter. It does not work well on a large training set. One solution that we did not consider investigating and that could be an outlook would have been to do a multiclass classification for logistic regression (one vs all or one vs rest). Therefore, if basic classification models are not working for the 8-class classification, we might investigate into **deep learning** methods. We will start by keeping the 27 handcrafted features and applying a neural network.

3.2 NN with 27 features

A neural network is a machine learning model composed of interconnected nodes that process and transmit information. It learns to recognize patterns and make predictions based on input data through training, which involves adjusting the weights and biases of the nodes based on the error between the predicted output and the true labels.

For our Neural Network, we will use a simple one with the architecture displayed in Listing 6. The results are displayed in Table 4.

```
# Model
model = Sequential([
  Dense(64, activation='relu', input_shape=(27,)),
  Dense(64, activation='relu'),
  Dense(9, activation='softmax'),
])

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

Listing 6: Neural Network

Model	Training Accuracy (%)	Testing accuracy (%)	Time cost (s)
NN with 27 features	0.84	0.72	30.6

Table 4: NN with handcrafted features accuracy and time cost

Accuracies and loss functions can be visualized in Fig 3 and in Fig 4. We can observe both overfitting (training accuracy way higher than test accuracy), low accuracy and high loss. Therefore, a neural network with the handcrafted data seems to be not functioning, therefore we should investigate to do a Convolutional Neural Network (CNN) and extract data features from the raw images. The process will be more time consuming, and it will cost more, but it might be one of the few solutions to have a nice accuracy. One outlook might also be to do the neural network with the pixel values as entrance features to see what we can obtain of it.

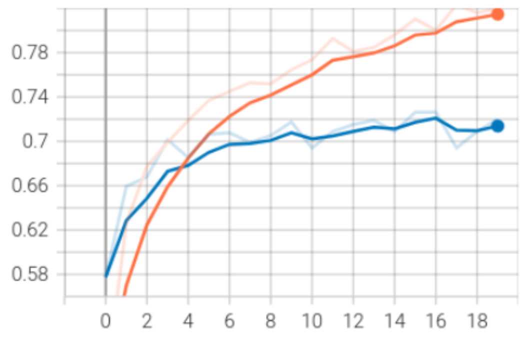

Figure 3: Accuracy functions with the epoch - Train in orange, test in blue

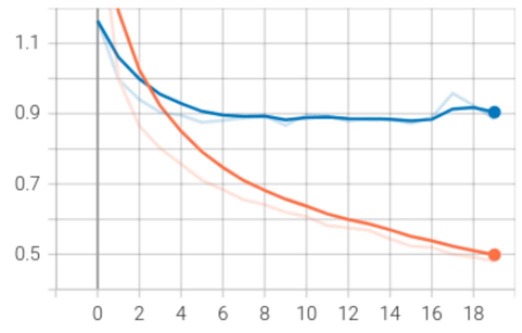

Figure 4: Loss functions with the epoch - Train in orange, test in blue

3.3 CNN - Basic implementation

A convolutional neural network (CNN) is a type of neural network specifically designed for image analysis. It is composed of layers that perform convolutions on the input data, which are mathematical operations that extract features from the data. The convolutional layers are followed by pooling layers, which down-sample the data, and fully connected layers, which perform the final classification or regression. CNNs are particularly effective for tasks such as image classification as they are able to learn and extract the most relevant features from the data.

To obtain the right accuracy and a time cost that is not too high, a first implementation was made with a basic VGG architecture written in Listing 7.

```
# First block : 2 * 32 convolution filters of size 3x3 + 2x2 MaxPooling
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                input_shape=(64, 64, 3)))
model.add(Conv2D(32, (3, 3),activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))


# Second block: 2 * 64 convolution filters of size 3x3 + 2x2 MaxPooling
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
```

Listing 7: Basic CNN with a VGG architecture

With this architecture, results were not that good. So it has been decided to add multiple regularization techniques.

3.4 CNN - Improved model with regularization technics

3.4.1 Early stopping

To increase the accuracy we add an early stopping command. Early stopping is a technique used in machine learning to prevent overfitting of a model to the training data. It involves monitoring the performance of the model on a separate validation set during the training process, and stopping the training early if the performance on the validation set stops improving or starts to degrade.

3.4.2 Dropout

We also add Dropout. Dropout is a regularization technique used in machine learning to prevent overfitting of a model to the training data. It works by randomly dropping out, or setting to zero, a certain number of activations (outputs of hidden units) in the model during training. This has the effect of forcing the model to learn multiple different ways to combine the remaining activations, which can improve the generalization of the model to unseen data.

3.4.3 Data Augmentation

Finally, the third regularization technique is Data Augmentation. Data augmentation is a technique used in machine learning to increase the size and diversity of a dataset by generating new data samples from the existing ones. Data augmentation works by applying a set of transformations to the existing data samples to generate new, augmented samples. These transformations can include operations such as cropping, rotating, flipping, scaling, and adding noise.

3.4.4 Summary

We add these technics one by one, and we looked at their effects. We will only plot the results with all the technics combined to see the final results.The final script is the one in Listing 8.

```
  # Third block: 2 * 128 convolution filters of size 3x3 + 2x2 MaxPooling
  model.add(Conv2D(128, (3, 3), activation='relu'))
  model.add(Conv2D(128, (3, 3), activation='relu'))
  model.add(MaxPooling2D(pool_size=(2, 2)))
  model.add(Dropout(.2))

  # Neural network for classification with one hidden layer
  # flatten to transform the 2D maps into a 1D vector for NN:
  # => input layer of size 32x32x3 = 3072
  model.add(Flatten())
  # hidden layer with 128 nodes
  model.add(Dense(256, activation='relu'))
  # output layer with 8 nodes because of the 8 classes
  model.add(Dense(8, activation='softmax'))

  # MODEL COMPILING: choosing loss function, optimizer and
  # metrics for perf evaluation
  model.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy'],

###############################################################################

x_train,x_test,y_train,y_test = load_data(X_CNN,Y_CNN)

# create data generator
datagen = ImageDataGenerator(width_shift_range=0.1, height_shift_range=0.1,
    horizontal_flip=True)
# prepare iterator
it_train = datagen.flow(x_train,y_train,batch_size = 32)

# EARLY STOPPING
# simple early stopping
callbacks = [EarlyStopping(monitor='val_loss', patience=15)]


```

```python
steps = int(x_train.shape[0] / 32)
model_log = model.fit(
    it_train,
    steps_per_epoch=steps,
    epochs=100,
    validation_data=(x_test,y_test),
    verbose=1,
    callbacks=callbacks
)
```

Listing 8: CNN with regularization technics to reduce overfitting

The results that are obtained are summarized in Fig 5. It is already good enough for the expected model, but we wanted to push the CNN model further with transfer learning.

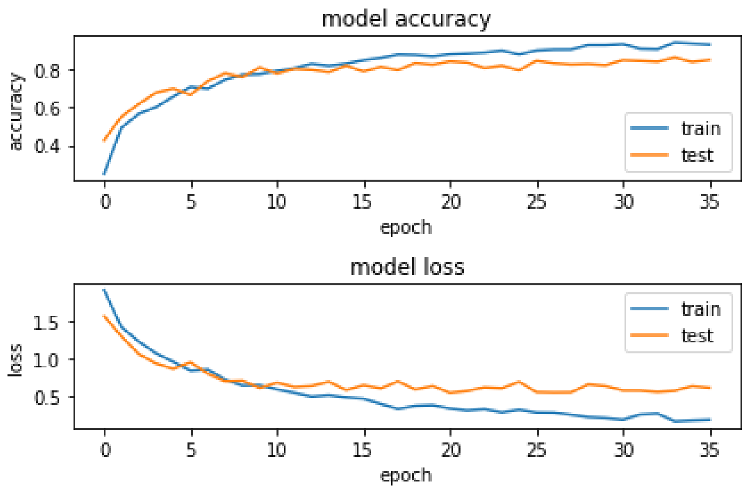

Figure 5: Accuracy and Loss functions with the epochs

3.5 Transfer learning

The final method that we will use in order to increase our accuracy again will be to do transfer learning. Transfer learning is a machine learning technique that involves using a pre-trained model on a new task, rather than training a model from scratch. We will use VGG16 model to implement our transfer learning, the script can be found in Listing 9 .

```python
# VGG16 Model
model = VGG16(include_top=False,weights='imagenet',input_shape=(64,64,3))
# Freezer les couches du VGG16
for layer in model.layers:
    layer.trainable = False

conv1 = Conv2D(64, (3, 3), activation='relu')(model.output)
pool1 = MaxPooling2D(pool_size=(2, 2))(conv1)
```

```
flat1 = Flatten()(pool1)
class1 = Dense(512,activation='relu')(flat1)
class2 = Dense(1024,activation='relu')(class1)
class3 = Dense(2056,activation='relu')(class2)
prediction = Dense(8, activation='softmax')(class3)

model=Model(inputs=model.input,outputs=prediction)

# loss='kullback_leibler_divergence'
model.compile(optimizer='adam', loss='kullback_leibler_divergence', metrics=['
    acc'])
```

Listing 9: Transfer Learning with the VGG16 Model

The results that are obtained are summarized in Fig 6. Accuracy and Loss are what we wished, however divergence can be observed, it is not smooth curves. Work can be done with transfer learning by defreezing and modifying some upper layers, add regularization technics. It can be a prospect to increase our model ability.

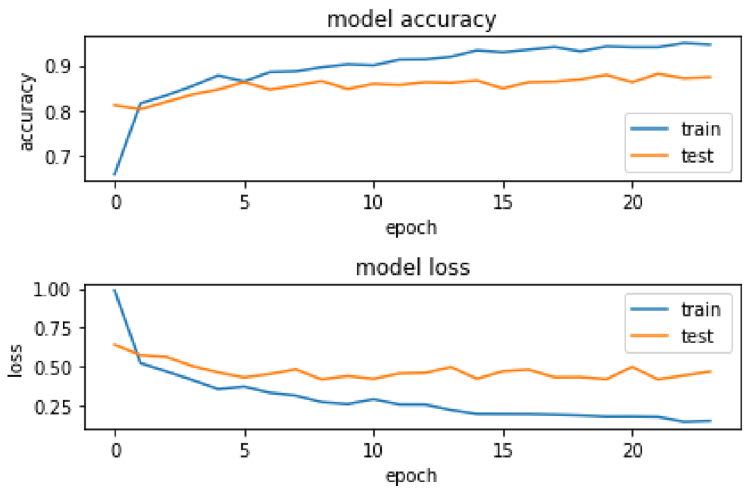

Figure 6: Accuracy and Loss functions with the epochs

3.6 CNN Summary

Here is a sum-up of the CNN accuracy, loss and complexity.

We can therefore conclude that CNN was a good method to solve the 8-class classification problem even if we are aware that there are many parameters to tune (especially concerning the regularization part) to obtain a nice accuracy and not a too big complexity. Transfer learning is a nice solution to have increased accuracy, however it is a more black box, and it takes a bit more time to understand how it works. We can consider that we were able to solve the 8-class classification problem.

Model	Test loss (%)	Testing accuracy (%)	Time cost (s)
CNN 1	0.69	0.83	173
CNN 2	0.48	0.85	131
CNN + VGG16	0.47	0.87	98

Table 5: CNN with Images accuracy and time cost

4 Image prediction

Now, that we have a nice model with a good accuracy, we wanted to try whether the model was able to predict the label of an image from our phone that we had taken. We wrote the script from Listing 10.

```python
# load and prepare the image
def load_image(filename):
  # load the image
    img = tf.keras.utils.load_img(filename, target_size=(64, 64))
  # convert to array
    img = tf.keras.utils.img_to_array(img)
  # reshape into a single sample with 3 channels
    img = img.reshape(1,64, 64, 3)
  # prepare pixel data
    img = img.astype('float64')
    img = img / 255.0
    return img



# load the image
dirpath = 'gdrive/MyDrive/Colab Notebooks/forettest.jpg'
# img = load_image('sample_image.png')
img = load_image(dirpath)
# load model
model = load_model('final_model_v2')
# predict the class
result = model.predict(img)
print(max(result[0]))
lab = np.array(['SEE SHORE','FOREST','HIGHWAY','CITY','MOUNTAIN','OPEN COUNTRY'
    ,'STREET','BIG BUILDING'])
for i in range (8):
  if result[0,i] == max(result[0]):
    print (i+1)
    print("This Image is a : ",lab[i])


img = skio.imread(dirpath)
plt.figure()
skio.imshow(img)
```

Listing 10: Prediction from an image from our own

The Image from Fig 7 was well detected with the label FOREST. We did a rapid preprocessing to adjust the dimension in a 64 $\times$ 64 image so it can fit to the input data. Similar experiments were made with mountains and see shores. The model was loaded from a h5-format so it was possible to use the predict answers on new data.

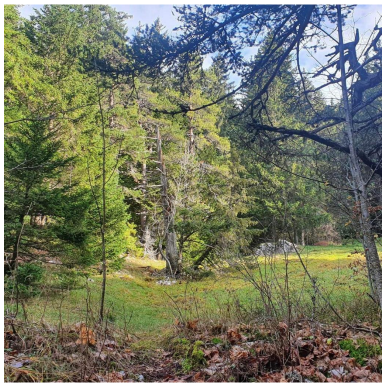

Figure 7: A picture representing a forest

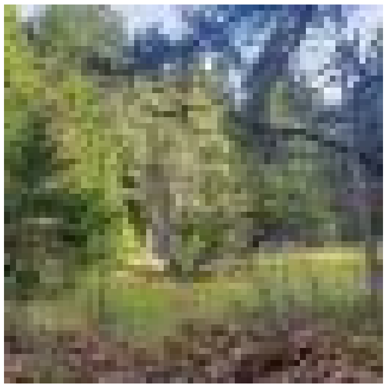

Figure 8: A picture representing a forest after the preprocessing 64 x 64

5 Skills / Competencies' approach

We were able to solve the given problem by first thoroughly understanding the problem statement and breaking it down into smaller, more manageable pieces. We then identified the specific skills and knowledge that were needed to address each aspect of the problem, and researched and learned about those skills as needed. We also consulted our lectures and sought feedback from peers to ensure that our approaches were on track. Once we had a clear understanding of the problem and the necessary skills, we developed and implemented a plan to address the problem, paying careful attention to each step and making adjustments as needed. We also monitored my progress and made any necessary adjustments to stay on track. Overall, the process of solving the problem required a combination of research, critical thinking, problem-solving skills, and persistence.

6 Summary and Conclusions

To put it in a nutshell, this project was really helpful and well organized as it gave us the opportunity to review all the methods learned in the lectures and apply to them a real world example. Therefore, we get to understand these techniques with more accuracy.

To go in further details, we can conclude that for the 2-class classification problem , simple classification methods are enough and give good accuracy with a low complexity. However, for the 8-class classification problem, it is more complex and more lengthy protocols have to be put in place. In that case, a CNN has to be implemented, and we have to take care of the parameters to be tuned. Many models are available and the range of improvement is still very high with for example transfer learning that would have needed a bit more time to master.

We conducted the project smoothly going from the most simple solution to most complex ones. We solved the problem with efficiency and in the due time. Maybe a point that was problematic was with Google Colab as even if it is "Colab", it was complicated to work together on the same file without the risk of overwritting the partner work. An other platform should be looked for a future project such as GitHub for example.