EE4146 Report

Group member: KO Tsz Shan (55696782)
IP Wai Lam (55679712)
CHEN Jing (55682083)
CHEUNG Yu Hin (55693466)
NG Cheuk Lung (55116872)

Abstract
Computer vision is commonly used in the field of automation technology, and it is made possible by utilizing machine learning to train a machine to be able to distinguish between received images. From this project, we aim to build a computer vision model that performs object classification on images.

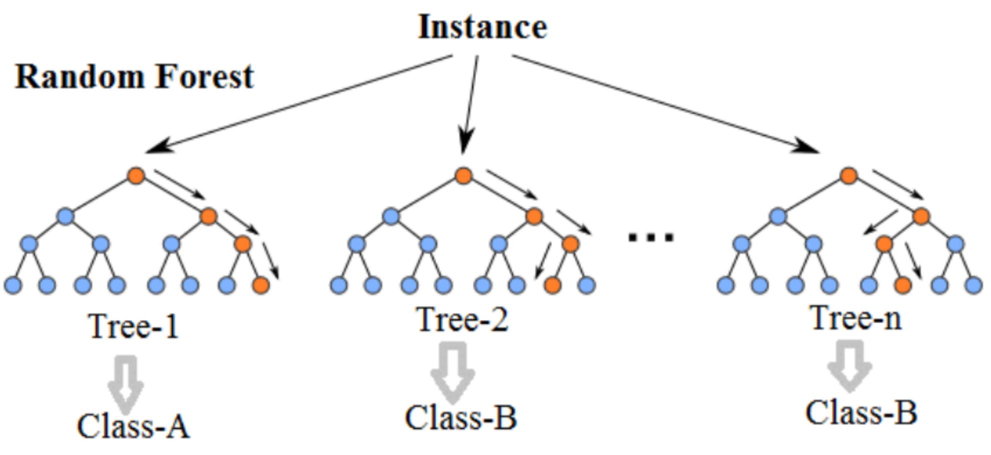
I. Introduction
The objective of this project is to classify a set of images into 6 classes including buildings, forest, glacier, mountain, sea and street. In this project, we need to find the machine learning problems and try to solve them by using different methods such as classification and feature extractor. Also, we are expected to learn how to read official documents and how to implement the machine learning algorithms. Finally, we can optimize the model with offline validation. The procedure is we first have a set of testing images and a set of training images. Those sets of images will pass through the process of feature extractor which make them become feature vectors. Then they go through the classification algorithm and the final result will be generated as a csv sheet.

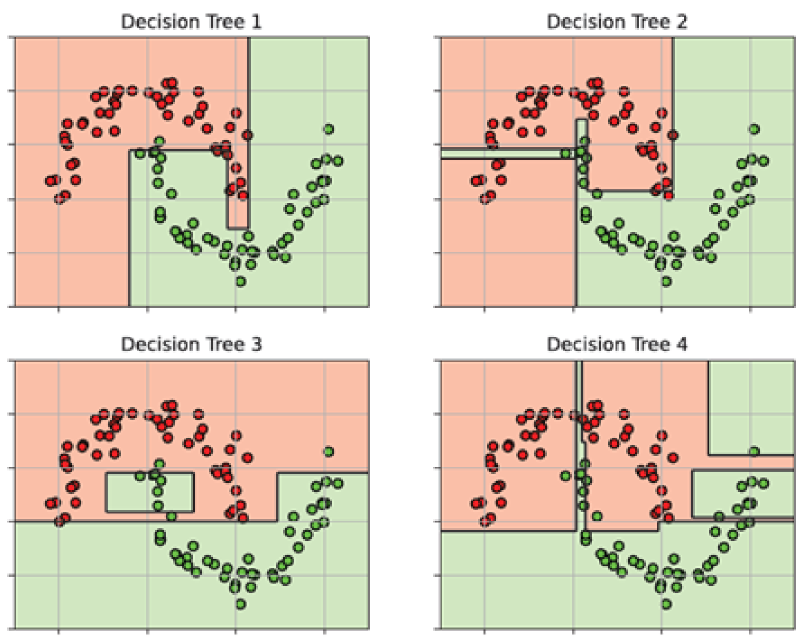
II. Method
A. Classification
First, we need to investigate different classifiers. Then, we find out which classifier gives the highest validation accuracy. We use the classifier which has the highest validation accuracy to generate the '.csv' file. In this project, we use 6 classifiers to investigate the highest validation accuracy including AdaBoostClassifier, sigmoid svm, random forest classifier, linear svm, rbf kernel svm and polynomial svm.

i. Ada boosting classifier
Ada boosting is one of the ensemble boosting classifiers. It will combine multiple weak classifiers to increase the accuracy of classifiers. It is to set the classifier weight of weak learners according to

classification error. Then, it is added to the ensemble according to classifier weight： Classifier = A1 classifier + A2 classifier +… Finally, it updates data weight for each sample. Furthermore, when it assigns the weight to the trained classifier during each iteration, the more accurate will get the highest weight. The advantage of the ada boosting classifier is easy to achieve and it can boost the accuracy through combining multiple weak learners and correcting the mistakes of them. But this classifier is sensitive to noise data and affected by outliers.

ii. Random forest classifier
The basic principle of random forest classifier is to combine multiple decision trees which use the GINI algorithm and add randomly allocated training data in order to improve the final results. In another word, it is to combine multiple weak learners to construct stronger learners. In this project, we use the bagging method to generate different data sets to generate multiple different decision trees. Here is an example of its working principle.



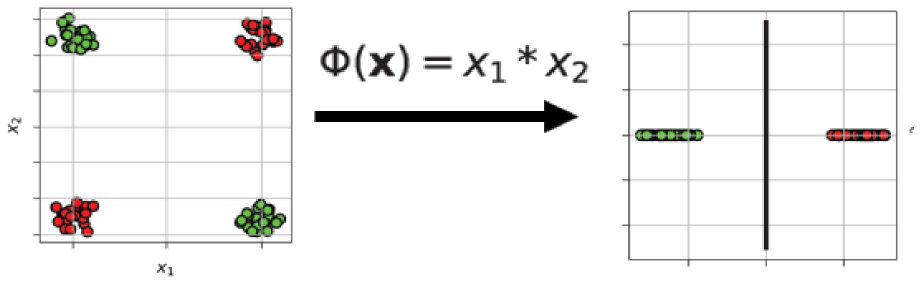


In order to train each decision tree, it will create a new subset of data by randomly sampling from all training data. Then, it aggregates all predictions which are given by voting for each test sample.
It is good to use this classifier because it is also a non-linear decision boundary. It has good generalization and it costs less time for training. It is very fast. But it can be sensitive to outliers and as a tree-based classifier, it is hard to represent a "diagonal" decision boundary.

iii. SVM
Support Vector Machine(SVM) is a well-known binary classifier and its algorithm is based on statistical learning theory. In another word, SVM is a supervised learning algorithm which attempts to construct a hyperplane from the data and divide the data into two classes. Finally, it will predict and classify the data. It will define the margin points and maximum the margin.

$$d_i = \frac{|f(\mathbf{x}_i)|}{||\mathbf{w}||} \qquad \gamma = \min_i \frac{|f(\mathbf{x}_i)|}{||\mathbf{w}||}$$

However, SVM can get better performance in classification problems when it is matched with the kernel functions. The kernel-based classifier is similar to kernel PCA which transform the inputs with non-linear kernel function. The principle of kernel-based classifiers is to transform the input features with kernel functions and then it will train a linear classifier in the transformed space. Here is a toy example.



There are some common kernel functions which are used in the project.
-Sigmoid kernel
It is one of the commonly used kernel functions of linear inseparable SVM.
The equation of sigmoid kernel is

$$K(x_i, x_j) = tanh(\gamma x_i^T x_j + r), \gamma > 0, r < 0$$

-Linear kernel

It is the ordinary inner product. The equation of linear kernel is

$$K(x_i, x_j) = x_i^T x_j$$

-RBF kernel

It is called the Gaussian kernel. The equation of RBF kernel is

$$k(\mathbf{x_i}, \mathbf{x_j}) = \exp(-\gamma\|\mathbf{x_i} - \mathbf{x_j}\|^2)$$

-Polynomial kernel

It is another commonly used kernel function of linear inseparable SVM. The equation of polynomial kernel is

$$K(x_i, x_j) = (\gamma x_i^T x_j + r)^d, d > 1$$

The advantage of using kernel based classifier is that the kernel function can be used in various data formats instead of vector-data and it is a non-linear decision boundary. However, this classifier is also sensitive to the used kernel function and it is expensive on large kernel matrices computationally.

B. Feature

When vectors with better features are provided to the classifier, the classification result will be more accurate.

i. Dimensionality Reduction

Dimensionality Reduction is a process of projecting high-dim data on a low-dim linear surface. This is done because the raw images are difficult to analyze during object classification, as the images are in their original high dimensions, the images are computationally intractable. We have considered 3 common methods of Dimensionality Reduction: Kernel Principal component analysis (KernelPCA), Fast Independent Component Analysis (ICA), and Non-Negative Matrix Factorization (NMF).

KernelPCA is a method of non-linear dimensionality reduction through the use of kernels. By applying non-linear transformation on data and then project data on low-dim linear surface, we can obtain matrices of high-dim data on low-dim non-flat surfaces.

The formula for KernelPCA is an alternative version of the linear PCA’s formula:

$$C = \frac{1}{N} \sum_{i=1}^{N} \mathbf{x}_i \mathbf{x}_i^\top$$

That introduces Kernel to the original formula to form a new formula:

$$\mathbf{V}^{k^T}\Phi(\mathbf{x}) = \left(\sum_{i=1}^{N} \mathbf{a_i}^k \Phi(\mathbf{x_i})\right)^T \Phi(\mathbf{x})$$

Where $\Phi : \mathbb{R}^d \to \mathbb{R}^N$

FastICA is an improved version of the original Independent Component Analysis(ICA) method. In ICA, the data matrix (i.e the raw images) is considered to be a linear combination of non-Gaussian (independent) components, and the algorithm attempts to separate the components by estimating an un-mixing matrix.
In FastICA, non-gaussianity is measured using approximations to neg-entropy (J), which makes the algorithm more robust to noise than other variations of ICA and faster to compute.
The algorithm of FastICA is as shown below:

Input: C Number of desired components
Input: $\mathbf{X} \in \mathbb{R}^{N\times M}$ Prewhitened matrix, where each column represents an N-dimensional sample, where $C <= N$
Output: $\mathbf{W} \in \mathbb{R}^{N\times C}$ Un-mixing matrix where each column projects $\mathbf{X}$ onto independent component.
Output: $\mathbf{S} \in \mathbb{R}^{C\times M}$ Independent components matrix, with M columns representing a sample with C dimensions.

```
for p in 1 to C:
    $\mathbf{w_p} \leftarrow$ Random vector of length N
    while $\mathbf{w_p}$ changes
        $\mathbf{w_p} \leftarrow \frac{1}{M}\mathbf{X}g(\mathbf{w_p}^T\mathbf{X})^T - \frac{1}{M}g'(\mathbf{w_p}^T\mathbf{X})\mathbf{1_M}\mathbf{w_p}$
        $\mathbf{w_p} \leftarrow \mathbf{w_p} - \sum_{j=1}^{p-1}(\mathbf{w_p}^T\mathbf{w_j})\mathbf{w_j}$
        $\mathbf{w_p} \leftarrow \frac{\mathbf{w_p}}{\|\mathbf{w_p}\|}$

output $\mathbf{W} \leftarrow [\mathbf{w_1}, \dots, \mathbf{w_C}]$

output $\mathbf{S} \leftarrow \mathbf{W^T X}$
```

NMF is a method that finds two non-negative matrices (W, H) which are the factorization result of the original non-negative matrix X, where all three matrices have no negative elements. This non-negativity makes the resulting matrices easier to classify. The factorization result is then used as an example for dimensionality reduction.
The algorithm of NMF is shown below:
Repeat these 2 equations

$$\mathbf{H}^{n+1}_{[i,j]} \leftarrow \mathbf{H}^{n}_{[i,j]} \frac{((\mathbf{W}^n)^T\mathbf{V})_{[i,j]}}{((\mathbf{W}^n)^T\mathbf{W}^n\mathbf{H}^n)_{[i,j]}}$$

$$\mathbf{W}^{n+1}_{[i,j]} \leftarrow \mathbf{W}^{n}_{[i,j]} \frac{(\mathbf{V}(\mathbf{H}^{n+1})^T)_{[i,j]}}{(\mathbf{W}^n\mathbf{H}^{n+1}(\mathbf{H}^{n+1})^T)_{[i,j]}}$$

where V=WH, n is the index of the iteration, until W and H are stable.

ii. Feature Extractors
Different models of feature extractors can improve the accuracy of prediction.

-Vggnet
It is the model proposed in the Very Deep Convolutional Networks for Large-Scale Image Recognition. It investigates the effect of the convolutional network depth on its accuracy in the large-scale image recognition setting.

-Efficientnet
It uses neural architecture search to design a new baseline network and scale it up to obtain a family of models.

-Densenet
It is the model proposed in the Densely Connected Convolutional Networks which connects each layer to every other layer in a feed-forward fashion.

-Resnet
It explicitly reformulates the layers as learning residual functions with reference to the layer inputs, instead of learning unreferenced functions.

-Googlenet
It is the model based on the Inception model. The architectural decisions were based on the Hebbian principle and the intuition of multi-scale processing

III. <u>Results</u>

A. Classification

i. Ada boosting classifier
The AdaBoost Classifier validation accuracy = 0.7249523809523809

```
svmclf = AdaBoostClassifier(n_estimators=75, learning_rate=1)
svmclf.fit(trainX, trainY)
predY_svm = svmclf.predict(valX)
acc_svm = metrics.accuracy_score(valY, predY_svm)
print("AdaBoostClassifier validation accuracy =", acc_svm)
```

```
AdaBoostClassifier validation accuracy = 0.724952380952380
9
```

ii. Sigmoid svm
The sigmoid svm validation accuracy = 0.8853333333333333

```
svmclf = svm.SVC(kernel='sigmoid', degree=3)
svmclf.fit(trainX, trainY)
predY_svm = svmclf.predict(valX)
acc_svm = metrics.accuracy_score(valY, predY_svm)
print("sigmoid svm validation accuracy =", acc_svm)
```

```
sigmoid svm validation accuracy = 0.8853333333333333
```

iii. Random forest classifier
The Random Forest Classifier validation accuracy = 0.9139047619047619

```
svmclf = RandomForestClassifier(n_estimators=100)
svmclf.fit(trainX, trainY)
predY_svm = svmclf.predict(valX)
acc_svm = metrics.accuracy_score(valY, predY_svm)
print("RandomForestClassifier validation accuracy =", acc_sv
m)
```

```
RandomForestClassifier validation accuracy = 0.91390476190
47619
```

iv. Linear svm
The linear svm validation accuracy = 0.9112380952380953

```
# paramgrid = {'C': logspace(-1,5,20), 'gamma': logspace(-4,3,2
0) }
# svmclf = model_selection.GridSearchCV(svm.SVC(kernel='linea
r'), paramgrid, cv=5, n_jobs=-1, verbose=True)
svmclf = svm.SVC(kernel='linear')
svmclf.fit(trainX, trainY)
predY_svm = svmclf.predict(valX)
acc_svm = metrics.accuracy_score(valY, predY_svm)
print("linear svm validation accuracy =", acc_svm)
```

```
linear svm validation accuracy = 0.9112380952380953
```

v. RBF kernel svm
The rbf svm validation accuracy = 0.9325714285714286

```
svmclf = svm.SVC(kernel='rbf')
svmclf.fit(trainX, trainY)
predY_svm = svmclf.predict(valX)
acc_svm = metrics.accuracy_score(valY, predY_svm)
print("rbf svm validation accuracy =", acc_svm)
```

```
rbf svm validation accuracy = 0.9325714285714286
```

vi. Polynomial svm
The polynomial svm validation accuracy = 0.9337142857142857

```
svmclf = svm.SVC(kernel='poly', degree=2)
svmclf.fit(trainX, trainY)
predY_svm = svmclf.predict(valX)
acc_svm = metrics.accuracy_score(valY, predY_svm)
print("polynomial svm validation accuracy =", acc_svm)
```

```
polynomial svm validation accuracy = 0.9337142857142857
```

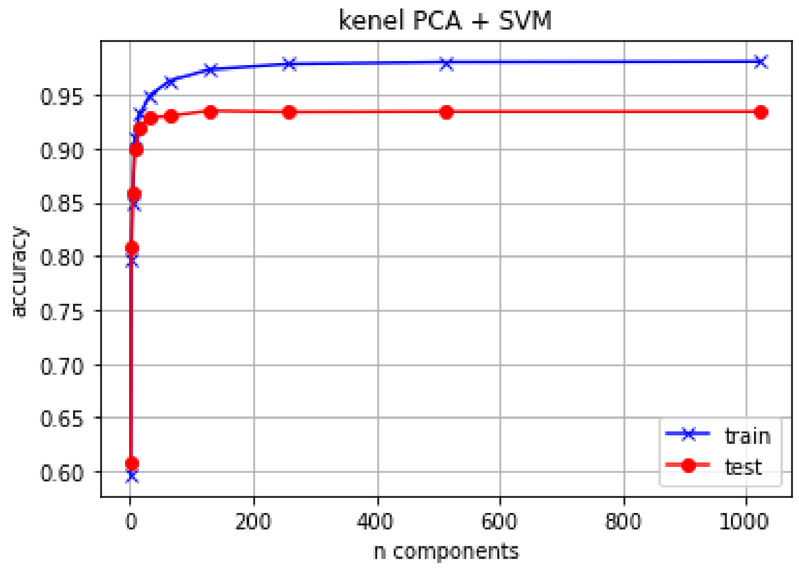
We choose polynomial svm for the classifier and generate the .csv file. The Score is 0.80181.

B. Feature

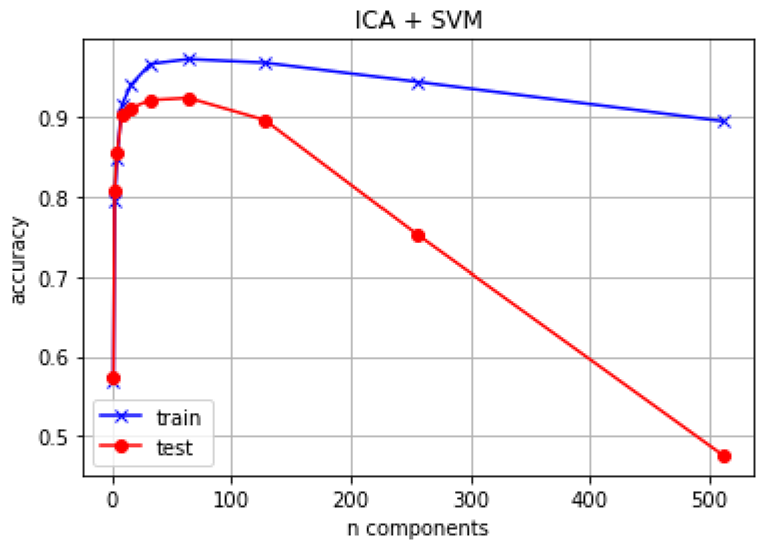
i. Dimensionality Reduction

The classification accuracy of a dimensionality reduction method is based on how many images from the training set are fitted. We have tested the accuracy of the 3 methods using different numbers of fitted components (denoted as n-components) including 1,2,4,8,16,32,64,128,256,512,1024. All the tests are performed with the same SVM setting (polynomial svm).

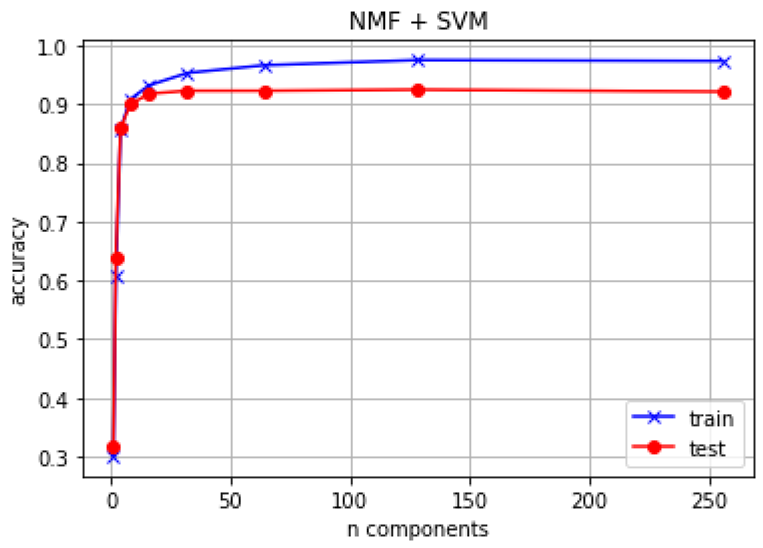
When Kernel PCA is used, the accuracy increases as n-component increase. Maximum accuracy 0.93410 is reached when n-component = 512. This method has the shortest execution time of all 3 methods.



When FastICA is used, the accuracy increases as n-component increase, but decreases after n-component = 64 as n-component increase. In the test, max iteration is set to 20000 and since the execution time is extremely long when n > 1000, the accuracy of n-component = 1024 is not tested. The maximum accuracy achieved is 0.92457 when n-component = 64. The execution time of this method is longer than Kernel PCA but shorter than NMF



When NMF is used, the accuracy increases as n-component increase, but decreases when n-component increases from 128 to 256. In the test, max iteration is set to 20000 and since the execution time increases exponentially as n-component increases, the accuracy of n-component = 512 and 1024 are not tested. The maximum accuracy achieved is 0.92457 when n-component = 128. This method has a far longer execution time than the other two methods.



Kernel PCA provides the highest classification accuracy out of all 3 methods, so we choose Kernel PCA to be the dimensionality reduction method for this project.

ii. Feature Extractors

- Efficientnet

```
# Define pretrained deep feature extractor
model = torchvision.models.efficientnet_b6(pretrained=True).cuda()
```

We choose Efficientnet_b6 for the feature extractor model and generate the train feat and test feat.
The accuracy is 0.72363

- Densenet

```
# Define pretrained deep feature extractor
model = torchvision.models.densenet161(pretrained=True).cuda()
```

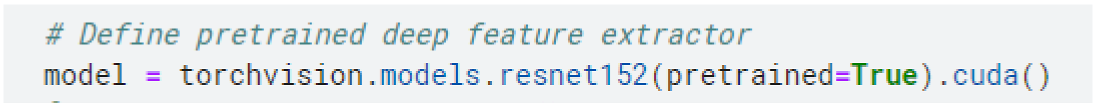
We choose Densenet161 for the feature extractor model and generate the train feat and test feat.
The accuracy is 0.78454

- Vggnet

```
# Define pretrained deep feature extractor
model = torchvision.models.vgg19_bn(pretrained=True).cuda()
```

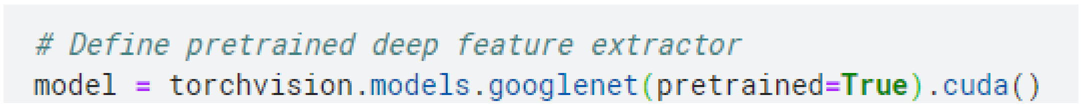
We choose Vgg19_bn for the feature extractor model and generate the train feat and test feat.
The accuracy is 0.74454

- Resnet



```
# Define pretrained deep feature extractor
model = torchvision.models.resnet152(pretrained=True).cuda()
```

We choose Resnet152 for the feature extractor model and generate the train feat and test feat.
The accuracy is 0.80363
- Googlenet



```
# Define pretrained deep feature extractor
model = torchvision.models.googlenet(pretrained=True).cuda()
```

We choose Googlenet for the feature extractor model and generate the train feat and test feat.
The accuracy is 0.81454

IV. <u>Conclusion</u>

In general, we have tried different classification and feature extractor models. The score for the classification method using polynomial svm is the highest. The score is 0.80 and the validation accuracy of it is 0.934. Then, to improve our extracted features' quality, we used Googlenet for the feature extractor model to gain the highest accuracy for the output .csv file. Wel also used Kernel PCA to perform dimensionality reduction, which has the accuracy of 0.92. The accuracy is 0.81.