Data Mining Project

Saman Arzaghi samanarzaghi@ut.ac.ir.

Report date: 20th August 2022.

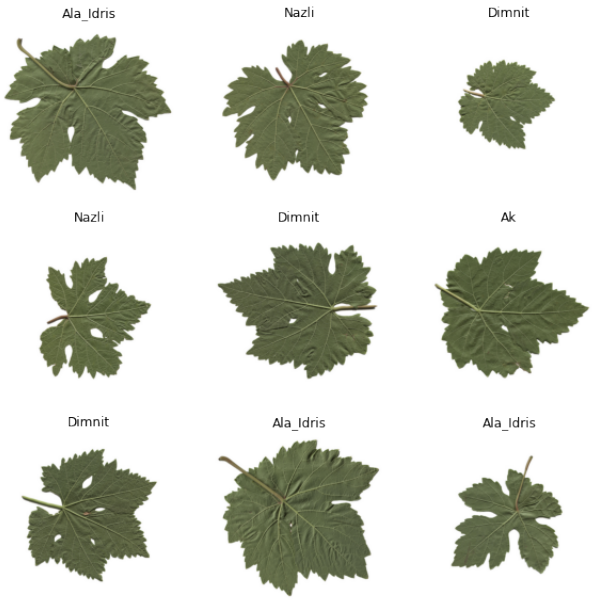
Abstract

There is a data set, five types of tree leaves were given, and we first implemented a simple CNN network and then a pre-trained network(MobileNet2). The main code is implemented in TensorFlow.

we also used the SVM method to increase the accuracy of the model.

1 Load and pre-process the data

Since we use tensorflow library, we use ready methods to load photos from the folder, and first we divide it into two parts, training and testing, with a ratio of eighty to twenty. After loading the photos, we take a first look at the number of photos and their labels, which you can see below.



Then we need to increase the number of training data using augmentation methods. For this purpose, we almost triple the training data using the following changes:

- RandomFlip in horizontal and vertical
- RandomTranslation in the range of [-15 pxl, +15 pxl]
- RandomRotation in the range of [-25 %, +25 %]
- RandomZoom in the range of [0.2 %, 0.2 %]

Now it is necessary to separate twenty percent of this data for the purpose of validation. For the purpose of pre-processing, we resize the data from 500 x 500 pixels to 256 x 256 pixels and divide each pixel by 255 so that each pixel is between 0 and 1 to make the work of our neural network easier.

2 Simple CNN model

First, we need to build a simple CNN model using the data we have and train it. To build a simple CNN model, we use a model with the following architecture:

(type)	Output Shape	Param #
(Conv2D)	(None, 254, 254, 16)	448
(MaxPooling2D)	(None, 127, 127, 16)	0
(Conv2D)	(None, 125, 125, 32)	4640
(MaxPooling2D)	(None, 62, 62, 32)	0
(Conv2D)	(None, 60, 60, 16)	4624
(MaxPooling2D	(None, 30, 30, 16)	0
(Flatten)	(None, 14400)	0
(Flatten)	(None, 14400)	0
(Dense)	(None, 256)	3686656
(Dense)	(None, 5)	1285
Total params: 3,697,653		
Trainable params: 3,697,653		
Non-trainable params: 0		

Table 1: Model summary for Simple CNN.

After running this model 10 times for 5 epochs and with 10 different codes, we obtained the following average accuracy for training, validation and test data:

Train	Validation	Test
100 %	36%	49%

And our confusion matrix we know that we have 5 classes (AK, AlaIdris, Buzgulu, Dimnit and Nazli), so we can create our confusion matrix with the predicted labels as below:

Table 2: Confusion Matrix

Actual Classes \ Predicted Classes	AK	AlaIdris	Buzgulu	Dimnit	Nazli
AK	8	4	3	2	2
AlaIdris	5	6	4	3	4
Buzgulu	0	2	8	5	4
Dimnit	2	0	6	5	13
Nazli	0	1	3	2	8

3 Transfer Learning(preTrained models)

A very effective work that can be done in order to increase the accuracy of the model is the use of pre-trained models (or transfer learning). This makes us directly use a pre-trained model and others don't need to train the model ourselves from the beginning.

In this part, we use the mobile net model and freeze the initial part of the model before the recognition part so that the recognition part can be trained faster. We also use the following architecture for the diagnostic part:

Model: test

(type)	Output Shape	Param #
(InputLayer)	[(None, 256, 256, 3)]	0
MobileNetv2(Functional)	(None, 8, 8, 1280)	2257984
(GlobalAveragePooling2D)	(None, 1280)	0
(Dense)	(None, 512)	655872
(Dense)	(None, 256)	131328
(Dense)	(None, 5)	1285

Total params: 3,046,469
Trainable params: 788,485
Non-trainable params: 2,257,984

Table 3: Model summary for Transfer learning.

The accuracy result is as below(after 10 times training with 10 different seeds):

Train	Validation	Test
95 %	88%	87%

and for confusion matrix we have:

Table 4: Confusion Matrix

Actual Classes \ Predicted Classes	AK	AlaIdris	Buzgulu	Dimnit	Nazli
AK	12	3	0	3	1
AlaIdris	0	21	1	0	0
Buzgulu	0	0	19	0	0
Dimnit	1	1	0	23	1
Nazli	1	0	0	0	13

3.1 Same thing, different libraries...

I have also deployed same transfer learning model and same architecture in PyTorch to hands on different libraries.

The interesting thing is that the result was a little different because the top part of the model(the classifaier) was also pre trained and we did not freez any part of the MobileNetV2.

for the accuracy we achived the followig result(with 10 times training with 10 different seeds):

Train	Validation	Test
97 %	90%	89%

4 Effect of the Auto-encoders

Another part of the project was related to the use of auto encoders. What we did in this part to use auto-encoders was to define an auto-encoder with the following architecture:

Model: test

(type)	Output Shape	Param #
(InputLayer)	[(None, 500, 500, 3)]	0
(Conv2D)	(None, 500, 500, 3)	84
(MaxPooling2g2D)	(None, 250, 250, 3)	0
(UpSampling2d)	(None, 500, 500, 3)	0
(Conv2D)	(None, 500, 500, 3)	84

Total params: 168
Trainable params: 168
Non-trainable params: 0

Table 5: Model summary for AutoEncoder.

after that, we trained this auto encoder with the whole dataset, you can the sample input image, the encoded image and the out put image of the auto encoder(we had train the auto encoder for 30 epochs):

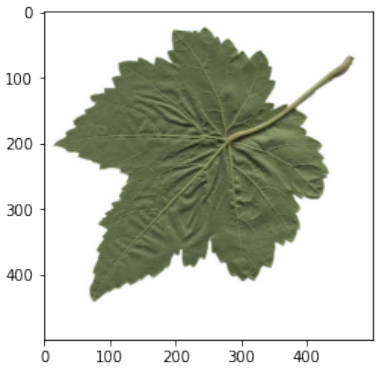


Figure 1: original

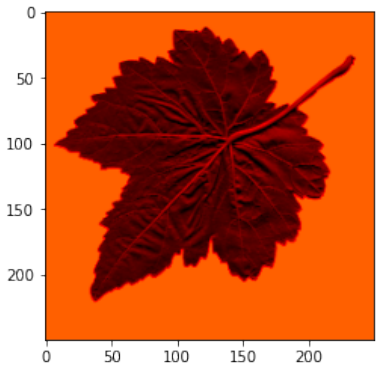


Figure 2: encoded

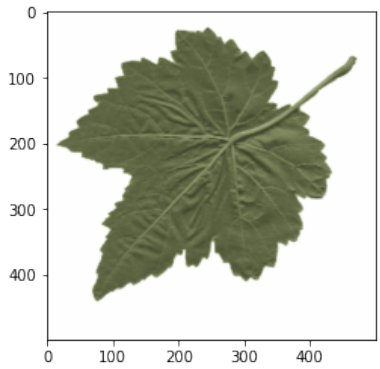


Figure 3: decoded

Now we only have to half the auto encoder and only use the encoder part to encode the images and reduct the dimantion of images(convert (500*500*3) to (250*250*3)), then pass the new images to the same MobileNet that we have created and train it for about 10 epochs ten times with ten different seeds.

the accuracy table is as below:

Train	Validation	Test
65 %	55%	60%

and the confusion matrix is:

Table 6: Confusion Matrix

		Predicted Classes				
		AK	AlaIdris	Buzgulu	Dimnit	Nazli
Actual Classes	AK	10	3	2	3	1
	AlaIdris	1	12	3	4	2
	Buzgulu	1	1	12	2	3
	Dimnit	0	0	2	16	8
	Nazli	2	1	0	1	10

As it is clear from the results, using the auto-encoder to reduce the dimensions in this case does not have such a good result, and resizing only helps the accuracy of the model more. The reason for this is also logical, because in this particular data set that is related to leaves, even the veins of the leaves and small details help us a lot, and the use of auto-encoder to reduce dimensions is not recommended in this particular case, but in other cases Maybe it will help us...

5 Improve the accuracy with SVM

One thing that we can do to improve the accuracy of our model, is that to use SVM after our convolution layers instead of dense network.

So we had create the same model as section 3(the MobileNetV2), freez the first convolution layers and at the end, use SVM as classification layer(not dens network).

To import the pre trained mobile net, we used tensorflow and for the svm part we used sklearn.

the accuracy of the model was imporved! Now we have the following results for test data set:

Test
94 %

The confusion matrix:

Table 7: Confusion Matrix

		Predicted Classes				
		AK	AlaIdris	Buzgulu	Dimnit	Nazli
Actual Classes	AK	17	1	0	1	0
	AlaIdris	1	19	2	0	0
	Buzgulu	1	0	16	2	0
	Dimnit	3	0	0	23	0
	Nazli	1	0	0	0	13

6 10 validation cross fold

The last part of the report is usage of the cross validation...

The whole point of croos validation is that we separate the train data to "k" parts, the train the model "k" times and each time, the k'th part of data is the validation data and the others is the train data.

For the neural network we have used MobileNetV2 for the feature selection part and used same dens network as section 3.

The accuracy result is as below:

	ACC at each k'fold									
	1	2	3	4	5	6	7	8	9	10
train	95%	96%	95%	95%	95%	93%	95%	95%	96%	95%
val	87%	87%	86%	88%	89%	92%	85%	90%	80%	82%
b test	88%	86%	87%	87%	87%	88%	89%	86%	86%	89%

Also we have the confusion matrix, but it's not good looking to create 10 confusion matrix here. If you want to check the confusion matrix they are all in the code(last cell).

7 Conclusion

The best accuracy that we had achived is 94% which was achived by the pre trained MobileNetV2 as feature selection and for the classifier, we have used SVM method.