AI programming assignment#1

1. Public non-image dataset

Mobile Price datasets

This dataset contains training dataset and testing dataset.

```
trian_shape: (2000, 21)
test_shape (1000, 21)
```

Algorithms

- Decision tree
- Random forest
- SVM

Analysis

- **Results when using different amounts of training data**
  Here I choose decision tree as the classifier and compare the result of different size (1000, 1500, 1800) of training data.

Training dataset size	Precision (macro)	Recall (macro)	F1 (macro)	Accuracy	Train Score	Test Score
1000	0.793362	0.792049	0.792523	0.794000	1.000000	0.794000
1500	0.820926	0.820619	0.824613	0.824000	1.000000	0.824000
1800	0.882495	0.881940	0.880847	0.880000	1.000000	0.880000

From the above chart we can find that as we increase the size of training dataset, the F1-score and accuracy also get better. It might be underfitting that impact the performance of models trained with fewer data. But it's a shame that the original dataset isn't big enough to do more comparison, because I also want to observe if there will be negative impact as we keep increasing the size of training data.

- **Results when using different classifiers**

	DTs	RF	SVC
Precision	0.825125	0.853572	0.955357
Recall	0.825181	0.855520	0.954335
F1	0.824565	0.854394	0.954309
Accuracy	0.827500	0.857500	0.955000
Train_Score	1.000000	1.000000	0.954375
Test_Score	0.827500	0.857500	0.955000

In this part I choose three classifiers, which are decision tree, random forest and svm. From the chart we can notice that svm has the best performance compare with the other classifiers, it might because of the advantages of support vector machine, such as its robustness to noise and outliers, because in this experiment I didn't preprocess the data, so there might exist data that are noisy, and also svm could perform well on unseen data. We can also find that although decision tree and random forest both achieve 100% accuracy on training set, their test score is >10% less than svm, it might because that they are overfit with the training data, so their test score are not that ideal. It's also reasonable that random forest performs better than decision tree since random forest is constituted by several decision tree using ensemble learning.

- **Results when using different hyper-parameters**

In this part, I use GridSearchCV to search over a specified hyperparameter grid and cross-validation using StratifiedKFold to evaluate the performance of different hyperparameter combinations of SVM, and get the best parameter of SVM. At first, I compare the results using different kernels of SVM, we can see that using 'linear' as kernel returns the best test score.

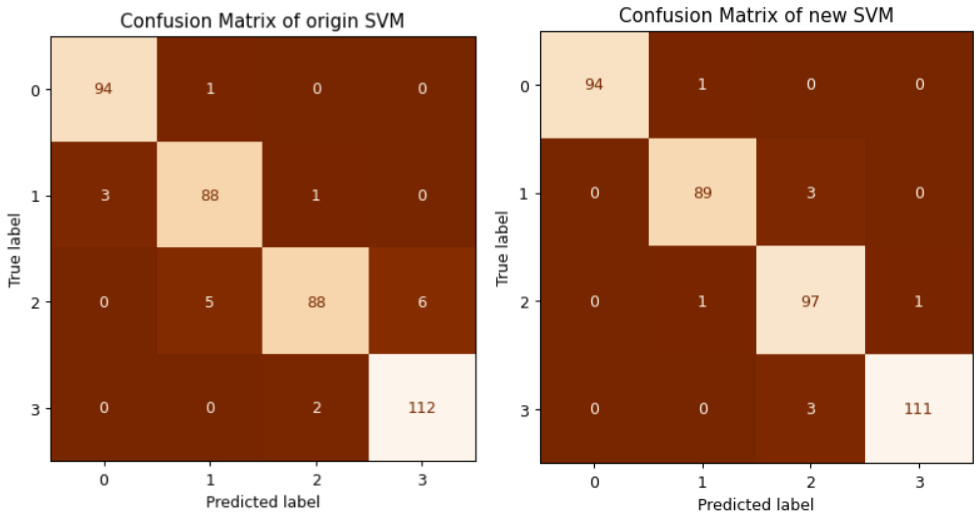
	0	1	2	3
mean_fit_time	0.13617	0.252448	1.11609	41.811477
std_fit_time	0.001209	0.006825	0.029246	16.898383
mean_score_time	0.011955	0.028134	0.049089	0.003374
std_score_time	0.000264	0.002559	0.001849	0.000509
param_kernel	poly	rbf	sigmoid	linear
params	{'kernel': 'poly'}	{'kernel': 'rbf'}	{'kernel': 'sigmoid'}	{'kernel': 'linear'}
split0_test_score	0.959375	0.946875	0.16875	0.9625
split1_test_score	0.940625	0.934375	0.1625	0.953125
split2_test_score	0.959375	0.959375	0.2	0.96875
split3_test_score	0.953125	0.9375	0.209375	0.9625
split4_test_score	0.95625	0.953125	0.18125	0.99375
mean_test_score	0.95375	0.94625	0.184375	0.968125
std_test_score	0.00696	0.009354	0.017897	0.01375
rank_test_score	2	3	4	1

The next step, I choose linear as kernel and use different C value(from 1-50) and degrees(from 2-10) to see which combination performance the best. Since the comparison result is too big to put it in the report, so I only print out the best combination and its scores.

```
{'C' : 7, 'degree' : 4}
0.975
```

Then compare the results of the svm after tuning hyper-parameter and the original svm using default hyper-parameter.

	SVC	new SVC
Precision	0.955357	0.977710
Recall	0.954335	0.977587
F1	0.954309	0.977521
Accuracy	0.955000	0.977500
Train_Score	0.954375	0.990000
Test_Score	0.955000	0.977500



We can see that the performance of svm whose parameter has been reset beats the svm using default parameter in every aspect, and from the confusion matrix we can find out that the biggest change happens in class 2, from 88/99 grows to 97/99, showing that changing parameters of model correctly can cause huge impact which is positive.

Discussion

- Yes, the results are what I expected. SVM performs better than decision tree and random forest, and random forest performs better than decision tree. Performances when training data increases also get better, and the tuning of

hyper-parameter is successful, too.

- There are many factors that will affect the performance. From the experiments, we can observe that by simply changing the model, the performance could increase drastically, and tuning the hyper parameter of models also improve the performance. The balance of dataset could also affect performance, the dataset I choose this time is a completely balanced dataset, which means the amount of each label is equal, if we choose imbalanced dataset, algorithms such as decision tree and random forest could be misled and classify most of cases into specific label.
- I would like to preprocess the dataset and see if there's any outlier or observe the characteristic of each feature, I think both of them could help me improve the performance of model.
- I think in this experiment I got to learn more about SVM and the new method GridSearch which help me tuning hyper-parameter easily than before.

2. Public image data

Flower recognition dataset

This dataset contains 4242 images of flowers, divided into five classes: chamomile, tulip, rose, sunflower, dandelion. For each class there are about 800 photos. Photos are not high resolution, about 320x240 pixels.

Algorithms

- SVM
- InceptionNetV3

Analysis

- **Results when using different amounts of training data**

  In this part, I use non-linear SVM (kernel = 'rbf') to see the impact of different train data sizes, using train data size : test data size = 7:3, 8:2 and 9:1.

	precision	recall	f1-score
daisy	0.53	0.42	0.47
dandelion	0.47	0.69	0.56
rose	0.60	0.28	0.38
sunflower	0.60	0.62	0.61
tulip	0.47	0.50	0.49
accuracy			0.51
macro avg	0.53	0.50	0.50
weighted avg	0.53	0.51	0.50

	precision	recall	f1-score
daisy	0.57	0.50	0.53
dandelion	0.52	0.71	0.60
rose	0.61	0.38	0.47
sunflower	0.65	0.67	0.66
tulip	0.57	0.56	0.57
accuracy			0.57
macro avg	0.58	0.56	0.57
weighted avg	0.58	0.57	0.57

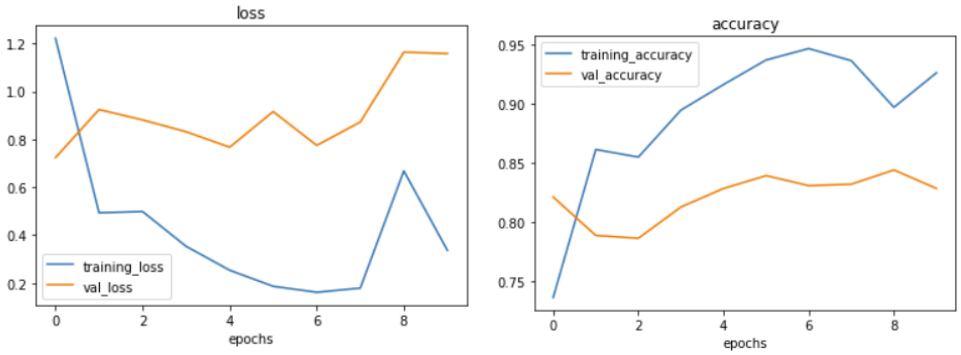
	precision	recall	f1-score
daisy	0.61	0.53	0.56
dandelion	0.55	0.78	0.65
rose	0.62	0.39	0.48
sunflower	0.68	0.70	0.69
tulip	0.61	0.57	0.59
accuracy			0.60
macro avg	0.61	0.59	0.59
weighted avg	0.61	0.60	0.59

As the figure shows, we can see that increasing training data size positively impact the performance of model.

- **Results when using different classifiers**

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input-layer (InputLayer)     [(None, 300, 300, 3)]     0
_________________________________________________________________
inception_v3 (Functional)    (None, None, None, 2048)  21802784
_________________________________________________________________
global_average_pooling_layer (None, 2048)              0
_________________________________________________________________
output-layer (Dense)         (None, 500)               1024500
=================================================================
Total params: 22,827,284
Trainable params: 1,024,500
Non-trainable params: 21,802,784
```

I choose InceptionNetV3 as my CNN-based model to compare with SVM, the model structure is as shown above. As for the parameter, I set batch size = 32, learning rate = 0.01 and epoch = 10, and below are the results.

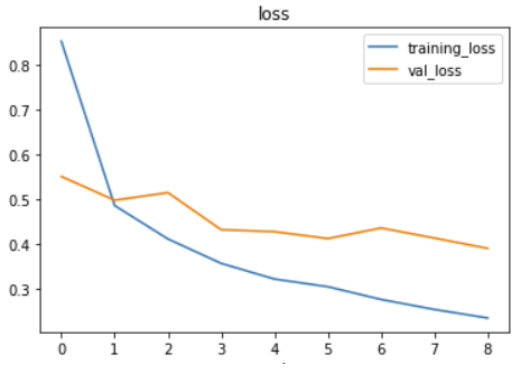


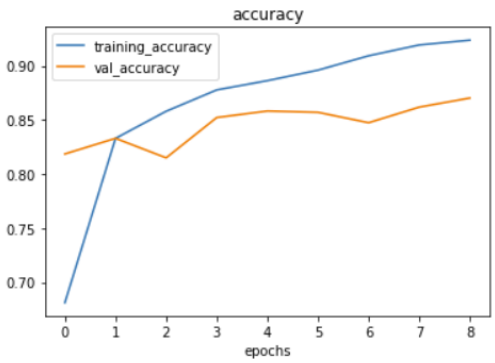
We can see that the accuracy of InceptionNet achieves 0.84, which is much higher compares to accuracy of SVM, but it's apparent that both curves of training loss and validation loss is not very ideal, the loss increases during training instead of decrease. I think it's because of the value of learning rate is

too large, so it may has caused overfitting.

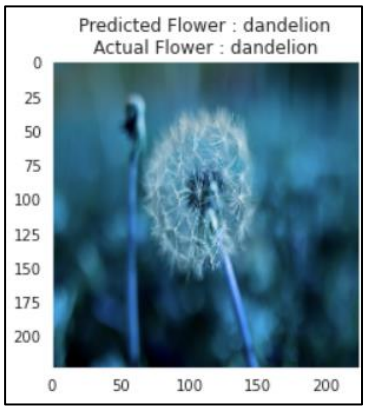
- **Results when using different hyper-parameters**

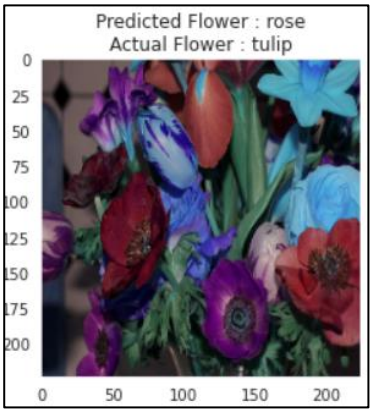
  Because of the result of last experiment, in this part I decide to change learning rate down to 0.001, epoch to 9 and increase the batch size to 64 to see if the performance can be better.





From the above figures, we can find that both curves of training loss and validation loss are much closer to ideal situation, and also the accuracy of validation data achieve 0.8702 in the last epoch, which is almost 3% higher than the last experiment.





Examples of correctly classified and misclassified data.

Discussion

- Yes, the results are what I expected. The performance using pretrained InceptionNet model is much better than simply using SVM.
- The tuning of hyper-parameter, the method to preprocess data and the model we choose will all affect the performance. From the experiments we can find that different parameter, including learning rate, batch size and epoch can affect model performance
- If I have more time, I think I will use different cnn-based model, such as ResNet, vgg16 and compare their performance. I also see a man using transfer learning +

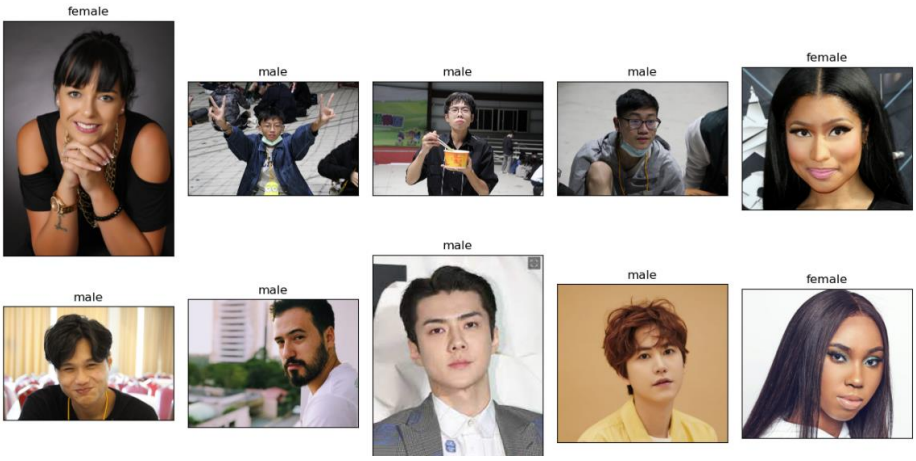
ensemble learning to achieve 99% accuracy, I would like to try his method if I have more time, too.

- From this experiment I've learned more about CNN model and how to use them correctly because I 've never used them to do experiments before and some helpful function like ImageDataGenerator to prepare data of image form.

Self-made dataset

Male vs Female dataset

This dataset consists of train/test data and two classes: male and female. For train/test dataset, there are 50/10 pictures in each class which I collected on internet and some of them are my friends. The size of photo isn't specified, so it should be resized by user if needed in later experiment.



Algorithm

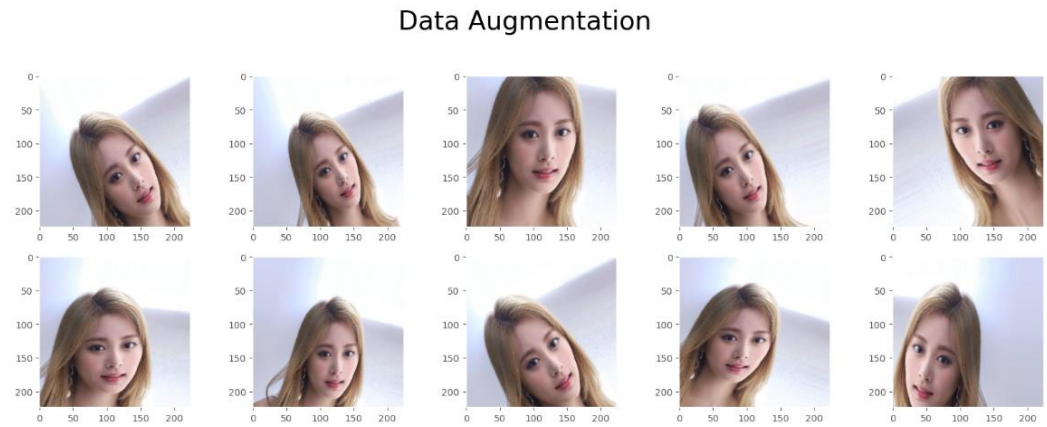
- SVM
- InceptionNetV3

Analysis

- **Compare results using data augmentation and different amount of training data**

  Data Augmentation allows to generate images with modifications to the original ones. The model will learn from these variations (changing angle, size and position),

being able to predict better never seen images that could have the same variations in position, size and position. I think it's also a way to increase training data size. I use ImageDataGenerator to help me done data augmentation, the example data after transform is shown below.




```
score0 = model_1.evaluate(test_data)
```

```
1/1 [==============================] - 3s 3s/step - loss: 0.4198 - accuracy: 0.8000
```

```
model_0.evaluate(test_data)
```

```
1/1 [==============================] - 3s 3s/step - loss: 0.3539 - accuracy: 0.9000
```

Model_1 is the model trained without augmentation and model_0 is the model trained with augmentation model, other setting such as model structure and hyper-parameter is same. We can see that accuracy of model_0 is 10% higher than model_1, meaning that it answers two more question right than model_1. It proofs that data augmentation does works.

- **Results when using different classifiers**

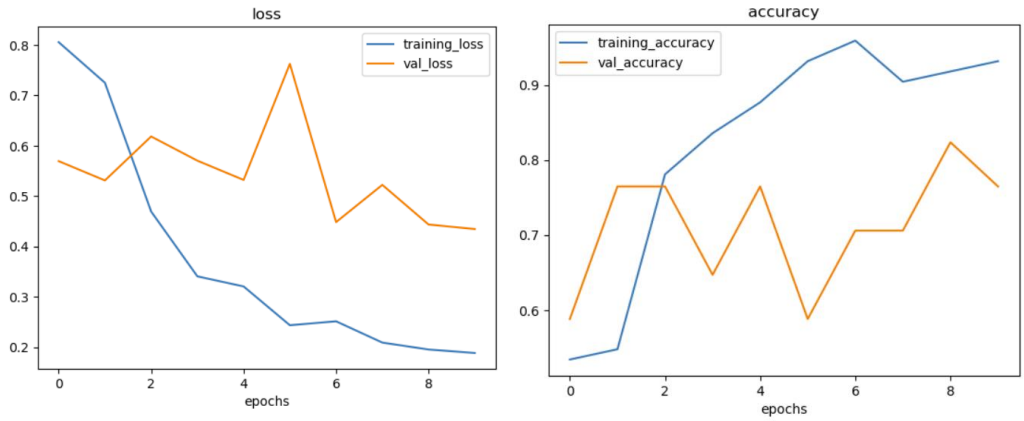
	precision	recall	f1-score	support
male	0.67	0.80	0.73	10
female	0.75	0.60	0.67	10
accuracy			0.70	20
macro avg	0.71	0.70	0.70	20
weighted avg	0.71	0.70	0.70	20

Above is the classification report of SVM, we can find that in this simple dataset, SVM has achieve accuracy of 70%, which is 10% fewer than model_1 and 20% fewer than model_0. The gap is much less than the gap of SVM and InceptionNet in public image dataset. I think the complexity of dataset also affects models' performance.

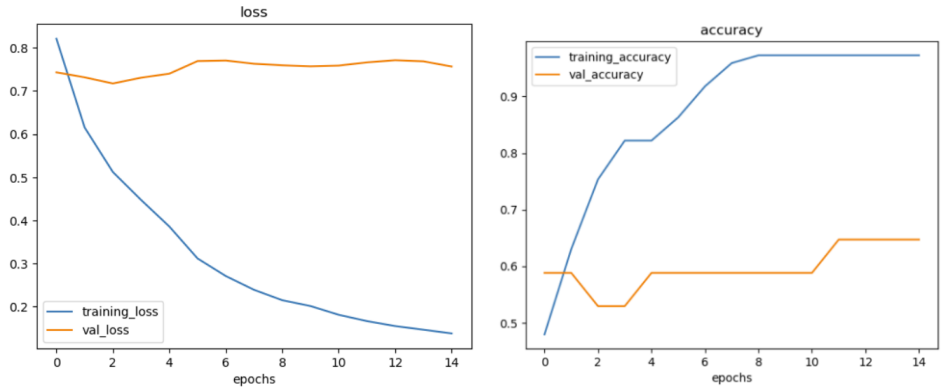
- **Results when using different hyper-parameters**

These two figures of loss and accuracy is the situation when I set learning rate to 0.001, epoch = 10, it's obvious that both curve is not regular and don't meet the

ideal curve.



Then I turn the learning rate down to 0.0005 and change epoch to 15. Below are the results of changing parameters, we can see that both validation loss and validation accuracy are almost the same as beginning, might mean that this combination of parameter is still not working. But when I evaluate this model on test dataset, it surprisingly achieve 95% accuracy, so I'm not very sure if the model is fitting well or not and what's the real reason of improvement of performance.



Discussion

- I think the results of experiments on this dataset is not what I expect, because from the figure of the last experiment I thought it's performance would be poor, but it has the best performance instead. I think if the data size is larger, it could have more specific results.
- I think that data augmentation is a helpful method to improve performance from the experiments this time. This is also a completely balance dataset, so there's no need to preprocess data and transform data, it might also make the performance better. I think the test dataset I choose may affect the model

performance too, because I choose pictures to be put in the test data randomly, so there might be same person or same background in both train data and tests data that will affect model.

- If I have more time, I would like to try more tuning of hyper parameter and find the combination of parameter that could have best performance, because in the previous experiment, both results are not too satisfying, I also want to try other CNN-based model such as ResNet in this experiment, I think it's probable that it achieves better accuracy.
- From this experiment I learned how to collect dataset from scratch and learn method to augment data. I think both of them are going to be helpful to my future ai programming.

Appendix

```
# Generate image generator for data augmentation
datagen =  ImageDataGenerator(
  #preprocessing_function=preprocess_input,
  rotation_range=30,
  width_shift_range=0.2,
  height_shift_range=0.2,
  shear_range=0.2,
  zoom_range=0.2,
  horizontal_flip=True
)

# load one image and reshape
img = load_img(EXAMPLE_PIC)
img = cv2.imread(EXAMPLE_PIC)
img = cv2.resize(img,(224,224))
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
x = img_to_array(img)/255.
x = x.reshape((1,) + x.shape)
# img = cv2.imread(EXAMPLE_PIC)
# x = cv2.resize(img,(224,224))

# # plot 10 augmented images of the loaded iamge
plt.figure(figsize=(20,10))
plt.suptitle('Data Augmentation', fontsize=28)
```

```
datagen = ImageDataGenerator(preprocessing_function = tf.keras.applications.inception_v3.preprocess_input,
test_datagen = ImageDataGenerator(preprocessing_function = tf.keras.applications.inception_v3.preprocess_in

train_data = datagen.flow_from_directory(directory = "/kaggle/input/male-vs-female/male vs female/train",
                                         batch_size= 32,
                                         target_size= (224,224),
                                         class_mode = "categorical",
                                        shuffle=True,
                                       seed=42,
                                       subset='training')

val_data = datagen.flow_from_directory(directory = "/kaggle/input/male-vs-female/male vs female/train",
                                         target_size = (224,224),
                                         class_mode = "categorical",
                                    shuffle=True,
                                     seed=42,
                                           subset='validation')

test_data = test_datagen.flow_from_directory(directory = "/kaggle/input/male-vs-female/male vs female/test"
                                         target_size = (224,224),
                                         class_mode = "categorical")
```

```python
base_model = tf.keras.applications.InceptionV3(include_top= False, weights = "imagenet")

# 2. Freeze the base model
base_model.trainable = False

#3. Create inputs into models
inputs = tf.keras.layers.Input(shape =(300,300,3), name = "input-layer")

#4. Rescaling
#x = tf.keras.layers.experimental.preprocessing.Rescaling(1/255.)(inputs)

#5. Pass the inputs
x = base_model(inputs)
print(f"Shape after passing inputs through base model: {x.shape}")

# 6. Average pool the outputs of the base model
x = tf.keras.layers.GlobalAveragePooling2D(name = "global_average_pooling_layer")(x)
print(f"Shape after GlobalAveragePooling2D: {x.shape}")

#7. Create the output activation layer
outputs = tf.keras.layers.Dense(2, activation = "softmax", name = "output-layer")(x)

# 8. Combine the inputs with outputs into a model
model_1 = tf.keras.Model(inputs, outputs)

# 9. Compile the model
model_1.compile(loss = "categorical_crossentropy",
                optimizer = tf.keras.optimizers.Adam(learning_rate = 0.001),
```

```python
def getYourFruits(fruits, data_type, print_n=False, k_fold=False):
    images = []
    labels = []
    dim = 100
    val = ['Training', 'Test']
    if not k_fold:
        path = "../input/flowers-recognition/flowers/"
        for i,f in enumerate(fruits):
            p = path + f
            j=0
            for image_path in glob.glob(os.path.join(p, "*.jpg")):
                image = cv2.imread(image_path, cv2.IMREAD_COLOR)
                image = cv2.resize(image, (dim, dim))
                image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
                images.append(image)
                labels.append(i)
                j+=1
            if(print_n):
                print("There are " , j , " " , data_type.upper(), " images of " , fruits[i].upper())
        images = np.array(images)
        labels = np.array(labels)
        return images, labels
```

```python
# flower type
fruits = ["daisy", "dandelion", "rose", "sunflower", "tulip"]

#Get Images and Labels
X, y =  getYourFruits(fruits, 'Training')
# X_test, y_test = getYourFruits(fruits, 'Test')

#Scale Data Images
scaler = StandardScaler()
X_train = scaler.fit_transform([i.flatten() for i in X])
# X_test = scaler.fit_transform([i.flatten() for i in X_test])
```

```python
svm_nlin = svm.SVC(C=1.0, kernel="rbf")
svm_nlin.fit(train_X, train_Y)
y_pred = svm_nlin.predict(test_X)
print(classification_report(test_Y, y_pred,
                            target_names=classes))
```