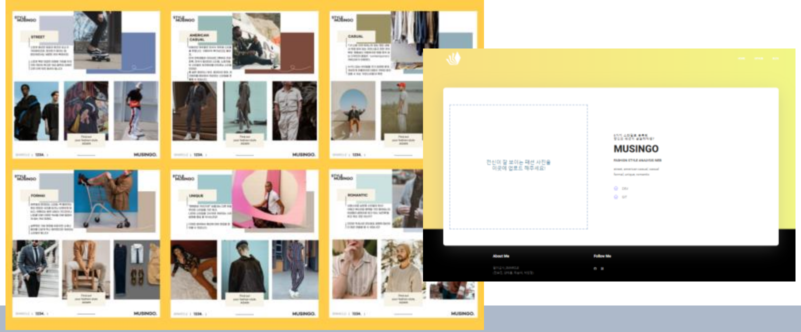
Papaya Disease Detector
using python tensorflow & SVM

Presentation by Yujin Jeon

My introduction

- B.S. majoring ‘Unmanned Vehicle Engineering’ of ‘AI robot’ department in Sejong univ.

- Experience
  - Mini autonomous car (Xy-car) driving competition 6th place overall, 2021
  - Matlab AI challenge competition Bronze award, 2023
  - Studying in lab, IVPG, 2023~
  - Creation of AI fashion recommendation webpage (in charge of backend and CV), 2023



Contents

Introduction

Journal of King Saud University – Computer and Information Sciences 32 (2020) 300–309

Contents lists available at ScienceDirect

Journal of King Saud University – Computer and Information Sciences

journal homepage: www.sciencedirect.com

Machine vision based papaya disease recognition

Md. Tarek Habib [a,b,*], Anup Majumder [b], A.Z.M. Jakaria [b], Morium Akter [a], Mohammad Shorif Uddin [a], Farruk Ahmed [c]

[a] Department of Computer Science and Engineering, Jahangirnagar University, Dhaka, Bangladesh
[b] Department of Computer Science and Engineering, Daffodil International University, Dhaka, Bangladesh
[c] Department of Computer Science and Engineering, Independent University, Bangladesh, Dhaka, Bangladesh

ARTICLE INFO

Article history:
Received 22 March 2018
Revised 7 June 2018
Accepted 13 June 2018
Available online 18 June 2018

Keywords:
Papaya disease
Agro-medical expert system
Machine vision
k-means clustering
Support vector machine

ABSTRACT

Over the years little research has been performed for vision-based papaya disease recognition system in order to help distant farmers, most of whom require proper support for cultivation. Due to advancement of vision-based technology we find a good solution to this problem. Papaya disease recognition mainly involves two challenging problems: one is disease detection and another is disease classification. Considering this scenario, here we present an online machine vision-based agro-medical expert system that processes an image captured through mobile or handheld device and determines the diseases in order to help distant farmers to address the problem. Some experiments are performed to show the utility of the proposed expert system. First, we propose a set of features from the view point of distinguishing attributes. K-means clustering algorithm is used in order to segment out the disease-attacked region from the captured image and then required features are extracted to classify the diseases with the help of support vector machine. More than 90% classification accuracy has been achieved, which appears to be good as well as promising by comparing performances obtained with recently reported relevant works.

1. Introduction

Bangladesh is a developing as well as densely populated country, where the prominent portion of the population depends on agriculture. It is agriculture that has managed employment for

quality of the agricultural product must be maintained with proper care. In Bangladesh, the rate of literacy is only 64.6% (List of countries by literacy rate, 2018) and most of the farmers are illiterate. They have hardly any training on farming with advanced technology. Again, there are few problems that come into play. One of

- <Machine vision based papaya disease recognition>, Md. Tarek Habib, Anup Majumder, A.Z.M. Jakaria, Morium Akter, Mohammad Shorif Uddin, Farruk Ahmed, 22 March 2018
  - This paper is written **to help farmers in need of support in agriculture**, and deals with machine vision technology for papaya disease recognition.

Introduction

- According to this paper, some countries are especially dependent on agriculture.
- In Bangladesh, for an example, 47% of the total population of the country is employed in agriculture.
- However, 39.9% of post-harvest losses were occurred in papaya agriculture, which is immensely popular product in Bangladesh. Moreover, there were two problems in dealing with this loss problems.

1) Transportation problem: they cannot reach the agriculture centers easily.
2) ignorance of the farmers towards the advanced technology

- So, If they are provided with **proper support** for producing better and disease-free crops, then it will be highly beneficial for them.

Introduction

First, the size of the picture is changed to a fixed size using 'Bicubic interpolation'. And then, the pictures are applied histogram equalization to improve contrast. Then, 'K-means clustering' technology is used to distinguish between defective and non-defective parts. There are the five most frequent papaya diseases, which are called black spot, powdery mildew, brown spot, phytophthora light, and anthracnose.

And then, we extract features from the 'disease part' which is distinguished via clustering step. There are two main features for distinguishing papaya diseases: one is a statistical feature, and the other is a co-occurrence feature. "Statistical features" have mean, standard deviation, variance, kurtosis, and skewness. "co-occurrence features" have contrast, correlation, energy, entropy, and homogeneity. (Each feature has a formula that can be calculated which is referred from previous published papers. Especially, these features were selected by referring to what was concluded to be meaningful in other papers.) After clustering, the vector of these features is extracted from the defective part.

Then, finally, you can solve the multi-class problem by using the feature vector in the SVM, which is one of the way to teach a machine about features.

Performance evaluation was conducted using images that were manually labeled with defects (ground truth images). As a result, the accuracy reached 95.2%. K-means clustering and SVM technologies were selected due to the excellent performance found in other existing papers, and in the case of SVM, the performance of other classification technologies, decision tree and naive bayes, was compared for comparison, but the SVM was finally selected with overwhelming accuracy.

- As this paper figure out the best way to make papaya disease classifier, I decided to **make the papaya disease classifier for myself.**

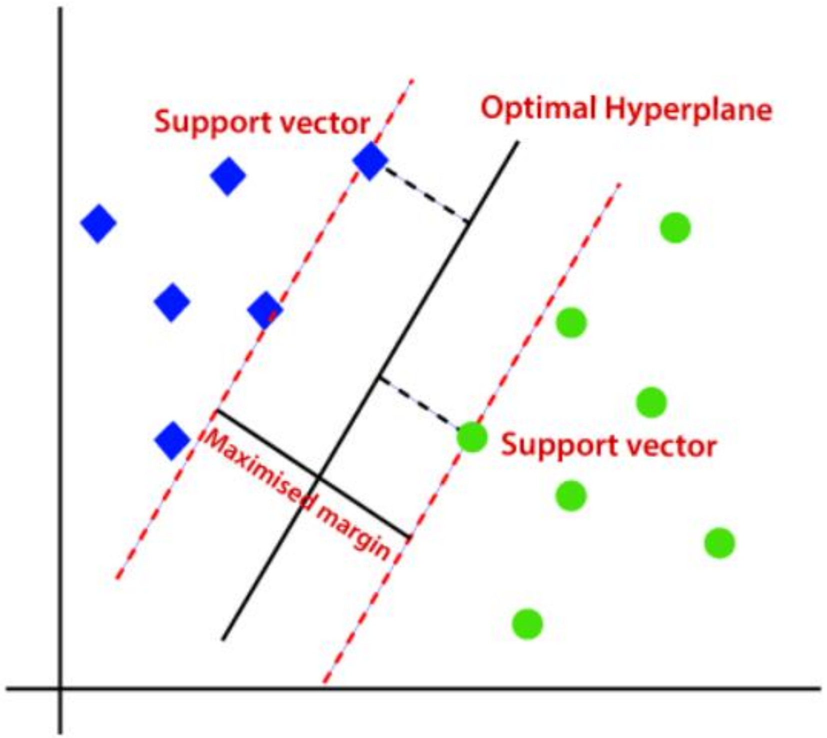
- Papaya disease recognition has two challenges:

1. **Detecting papaya disease**
2. **Classifying the disease**

Explanation

1. SVM (support vector machine)

- It is a supervised learning model that classifies and regresses based on a decision boundary, that is, a line similar to a linear model.



Explanation

2. Image_dataset_from_directory / Dataset

```
tf.keras.utils.image_dataset_from_directory(
    directory,
    labels='inferred',
    label_mode='int',
    class_names=None,
    color_mode='rgb',
    batch_size=32,
    image_size=(256, 256),
    shuffle=True,
    seed=None,
    validation_split=None,
    subset=None,
    interpolation='bilinear',
    follow_links=False,
    crop_to_aspect_ratio=False,
    **kwargs
)
```

`labels`	Either `"inferred"` (labels are generated from the directory structure), `None` (no labels), or a list/tuple of integer labels of the same size as the number of image files found in the directory. Labels should be sorted according to the alphanumeric order of the image file paths (obtained via `os.walk(directory)` in Python).
`validation_split`	Optional float between 0 and 1, fraction of data to reserve for validation.
`subset`	Subset of the data to return. One of `"training"`, `"validation"`, or `"both"`. Only used if `validation_split` is set. When `subset="both"`, the utility returns a tuple of two datasets (the training and validation datasets respectively).

Returns

A `tf.data.Dataset` object.

Explanation

2. Image_dataset_from_directory / Dataset

```
tf.data.Dataset(
    variant_tensor
)
```

- Consist of tensor

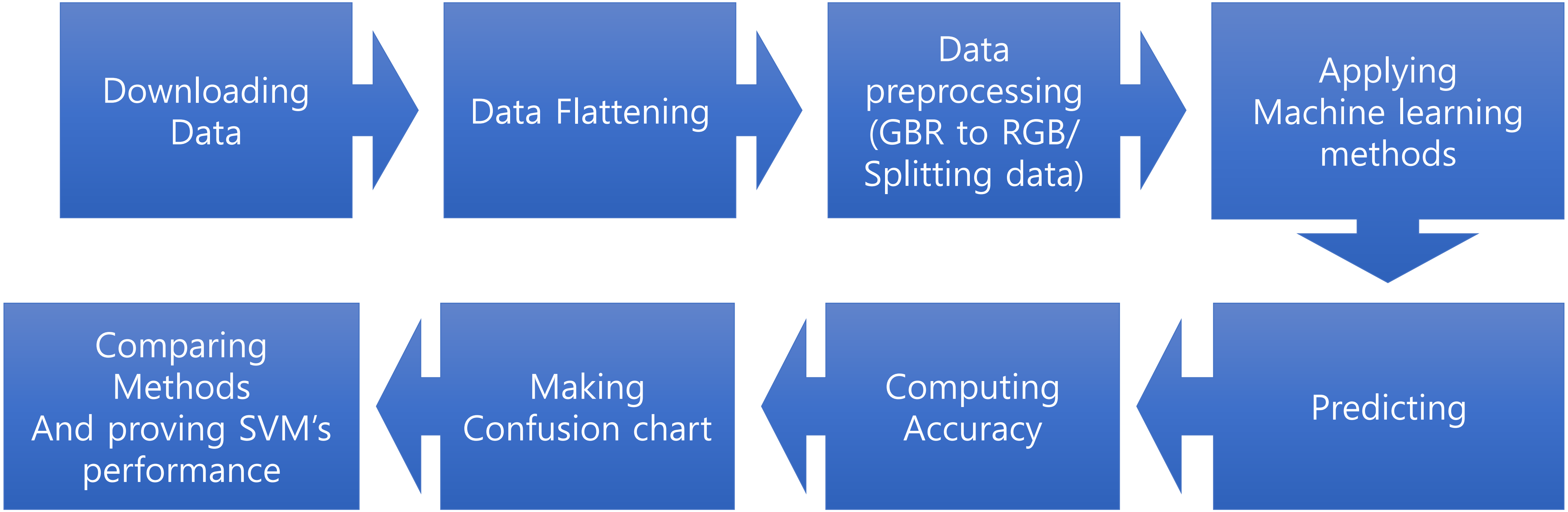
```
take(
    count, name=None
)
```

Creates a `Dataset` with at most `count` elements from this dataset.

```
>>> dataset = tf.data.Dataset.range(10)
>>> dataset = dataset.take(3)
>>> list(dataset.as_numpy_iterator())
[0, 1, 2]
```

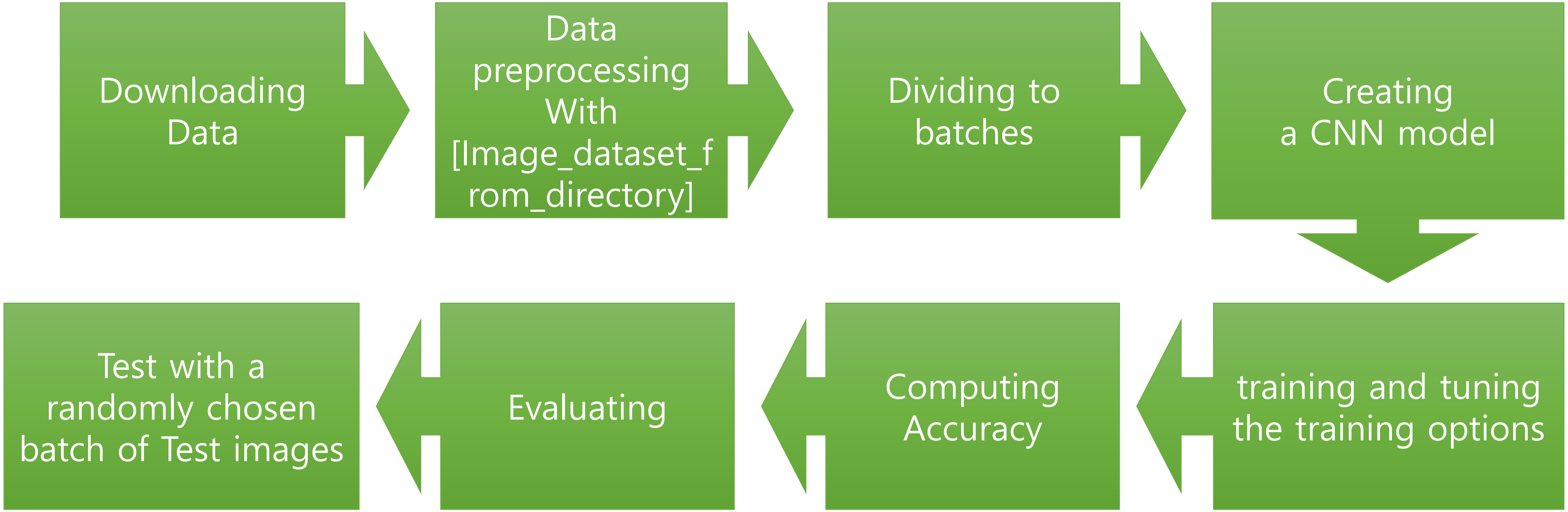
Flow Chart

- **Part 1) SVM (Machine Learning Part)**



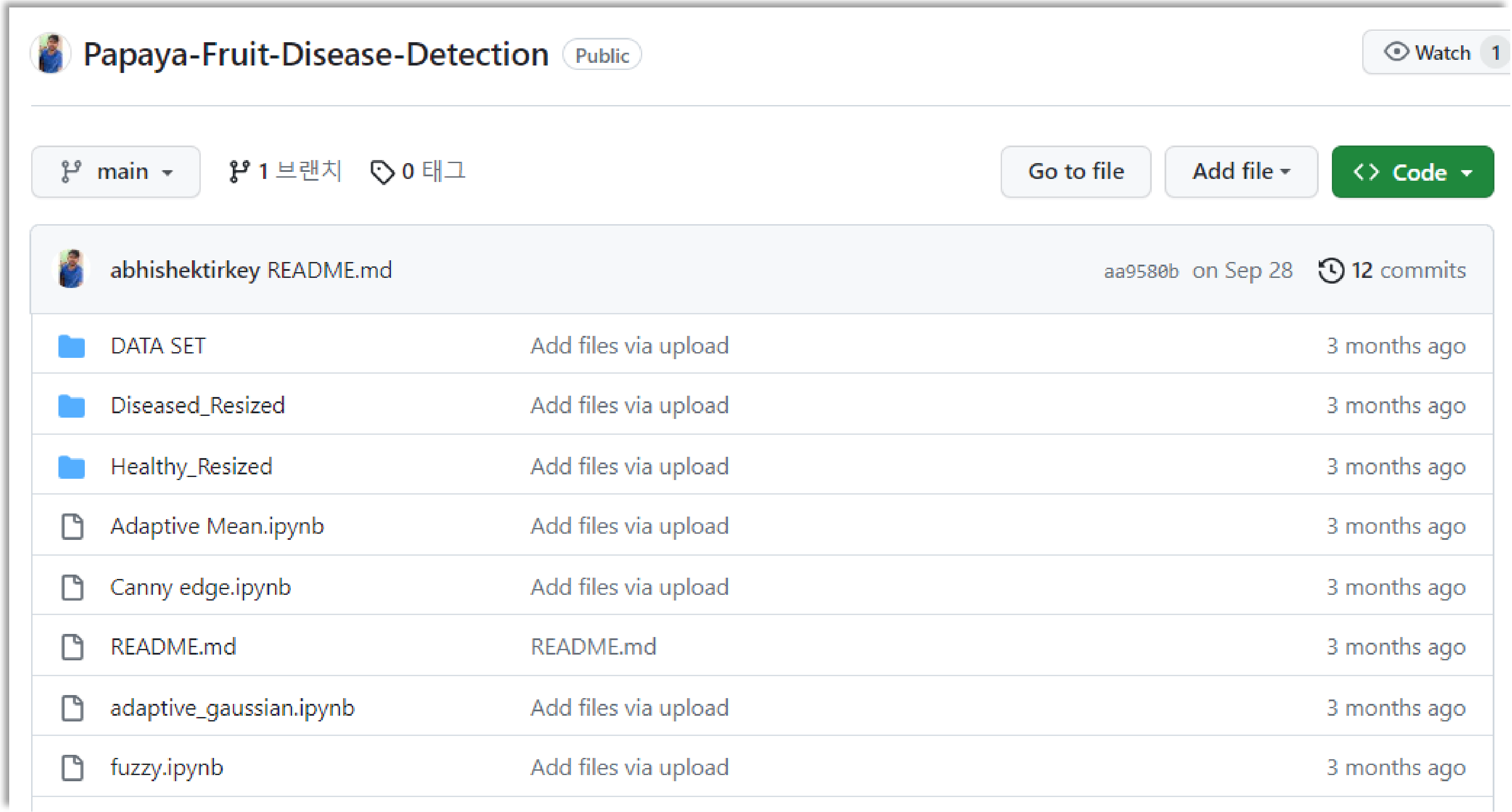
Flow Chart

- **Part 2) CNN (Deep Learning Part)**



Screenshots of Code

1. Downloading Data



A papaya data set in github

Screenshots of Code 2. Applying Machine Learning to make a classifier

1) Import Data
2) Data Flattening
3) Data split (train/test)

```
class_names=['Diseased','Healthy']
flat_data_arr=[] #input array
target_arr=[] #output array
data_dir='/content/drive/MyDrive/Colab Notebooks/machine vision system project'

#path which contains all the class_names of images
for i in class_names:

    print(f'loading... category : {i}')
    path=os.path.join(data_dir,i)
    print(path)
    for img in os.listdir(path):
        img_array=cv2.imread(os.path.join(path,img))
        flat_data_arr.append(img_array.flatten())
        target_arr.append(class_names.index(i))
    print(f'loaded category:{i} successfully')

flat_data=np.array(flat_data_arr)
target=np.array(target_arr)
df=pd.DataFrame(flat_data) #dataframe
df['Target']=target

x=df.iloc[:,:-1] #input data
y=df.iloc[:,-1] #output data

print("x:",x.shape)
print("y:",y.shape)

x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=.20,random_state=77,stratify=y)

print('Splitted Successfully')
```

```
loading... category : Diseased
/content/drive/MyDrive/Colab Notebooks/machine vision system project/Diseased
loaded category:Diseased successfully
loading... category : Healthy
/content/drive/MyDrive/Colab Notebooks/machine vision system project/Healthy
loaded category:Healthy successfully
x: (421, 30000)
y: (421,)
Splitted Successfully
```

```
x_train.head()
```

	0	1	2	3	4	5	6	7	8	9	...	29990	29991	29992	29993	29994	29995	29996	29997	29998	29999
223	248	222	206	249	223	207	253	221	208	253	...	139	177	148	139	177	148	139	177	148	139
266	253	253	253	253	253	253	253	253	253	253	...	253	253	253	253	253	253	253	253	253	253
352	183	160	152	183	160	152	183	160	152	183	...	192	233	197	191	234	198	192	237	201	195
325	172	138	132	172	138	132	172	138	132	172	...	176	224	185	176	223	184	175	223	184	175
386	188	158	153	188	158	153	190	158	153	190	...	187	239	196	187	239	196	187	239	196	187

5 rows × 30000 columns

```
y_train.head()
```

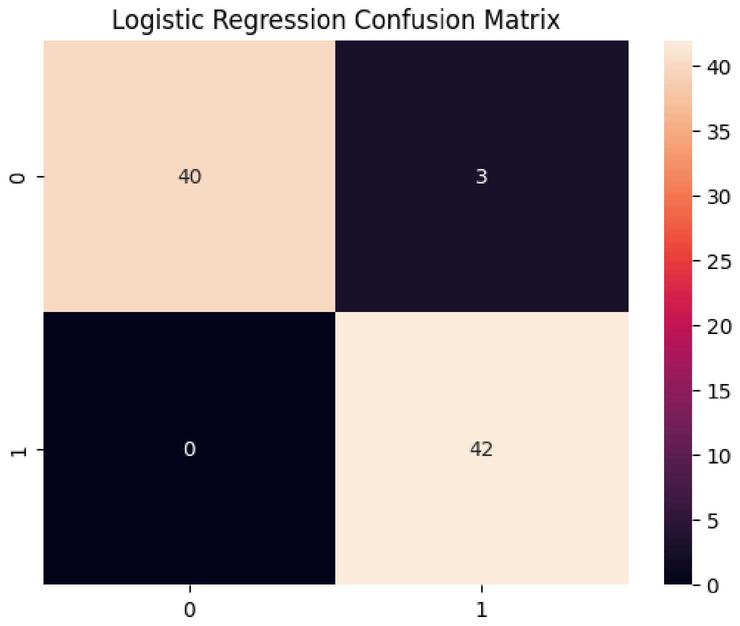
```
223    1
266    1
352    1
325    1
386    1
Name: Target, dtype: int64
```

Screenshots of Code 2. Applying Machine Learning to make a classifier

1) Logistic Regression

```
lr = LogisticRegression(penalty='l1', solver = 'liblinear', random_state=42)
lr.fit(x_train, y_train)

y_pred = lr.predict(x_test)
lr_train_acc = round(accuracy_score(y_train, lr.predict(x_train)) *100, 2)   # 소숫점 2자리로 반올림
lr_test_acc = round(accuracy_score(y_test, y_pred)*100, 2)
print('₩nAccuracy = ', lr_test_acc, ' %₩n')

cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True)
plt.title('Logistic Regression Confusion Matrix')
```

```
Accuracy =  96.47  %
```

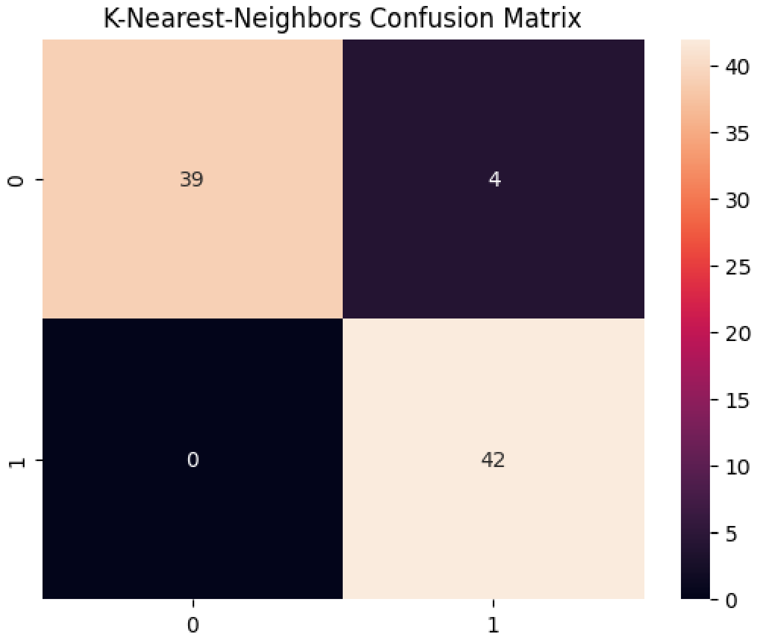


Screenshots of Code 2. Applying Machine Learning to make a classifier

2) KNN

```
] knn = KNeighborsClassifier(n_neighbors=5)
  knn.fit(x_train, y_train)

  y_pred = knn.predict(x_test)
  knn_train_acc = round(accuracy_score(y_train, lr.predict(x_train)) *100, 2)   # 소숫점 2자리로 반올림
  knn_test_acc = round(accuracy_score(y_test, y_pred)*100, 2)
  print('₩nAccuracy = ', knn_test_acc, ' %₩n')

  cm = confusion_matrix(y_test, y_pred)
  sns.heatmap(cm, annot=True)
  plt.title('K-Nearest-Neighbors Confusion Matrix')
```

```
Accuracy =  95.29  %
```

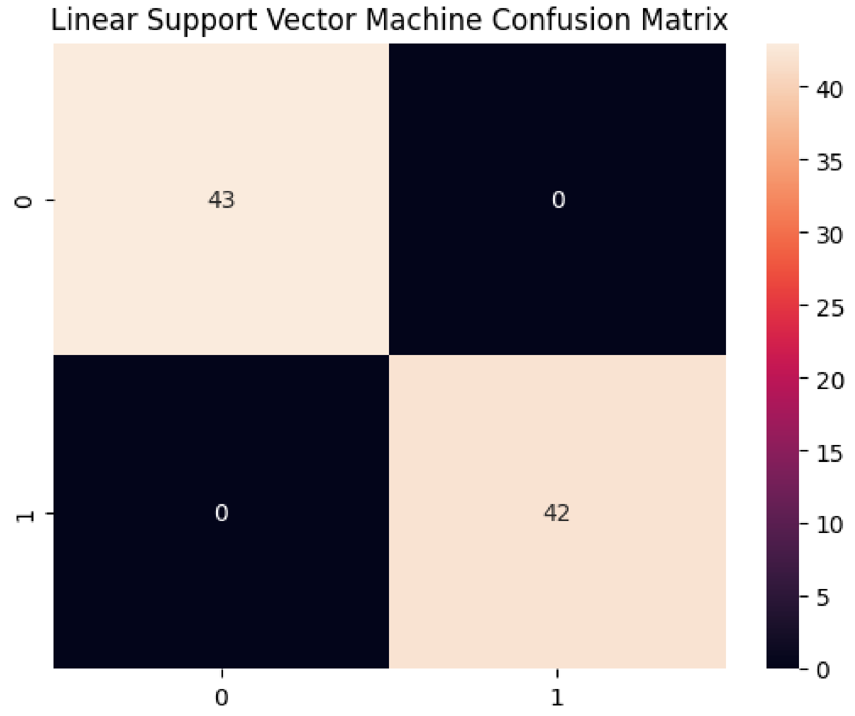


Screenshots of Code 2. Applying Machine Learning to make a classifier

3) SVM

```
] ln_svm = SVC(kernel='linear')
  ln_svm.fit(x_train, y_train)
  y_pred = ln_svm.predict(x_test)
  ln_svm_train_acc = round(accuracy_score(y_train, ln_svm.predict(x_train)) *100, 2)
  ln_svm_test_acc = round(accuracy_score(y_test, y_pred)*100, 2)
  print('₩nAccuracy = ', ln_svm_test_acc, ' %₩n')

  cm = confusion_matrix(y_test, y_pred)
  sns.heatmap(cm, annot=True)
  plt.title('Linear Support Vector Machine Confusion Matrix')
```

```
Accuracy =  100.0  %
```

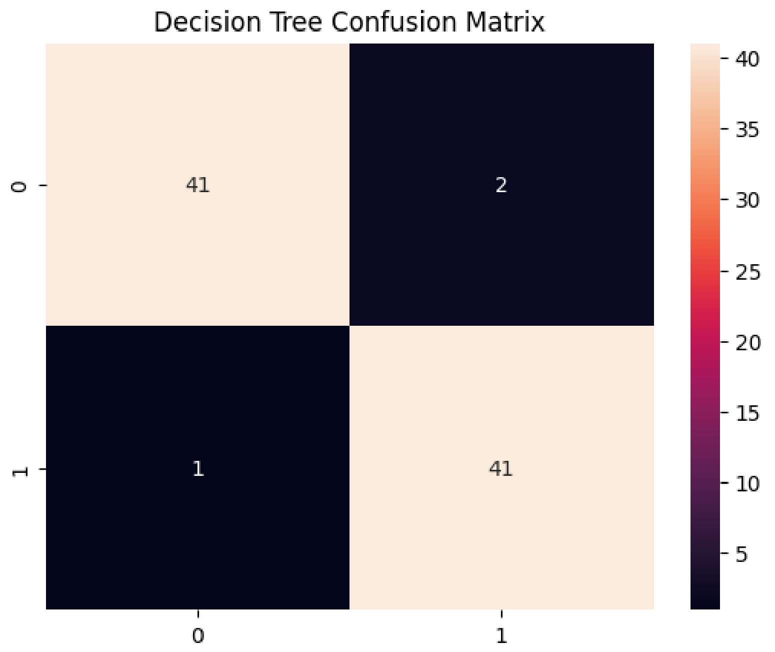


Screenshots of Code 2. Applying Machine Learning to make a classifier

4) Decision Tree

```
] tree = DecisionTreeClassifier(max_depth=5)
  tree.fit(x_train, y_train)

  y_pred = tree.predict(x_test)
  tree_train_acc = round(accuracy_score(y_train, tree.predict(x_train)) *100, 2)
  tree_test_acc = round(accuracy_score(y_test, y_pred)*100, 2)
  print('₩nAccuracy = ', tree_test_acc, ' %₩n')

  cm = confusion_matrix(y_test, y_pred)
  sns.heatmap(cm, annot=True)
  plt.title('Decision Tree Confusion Matrix')
```

```
Accuracy =  96.47  %
```

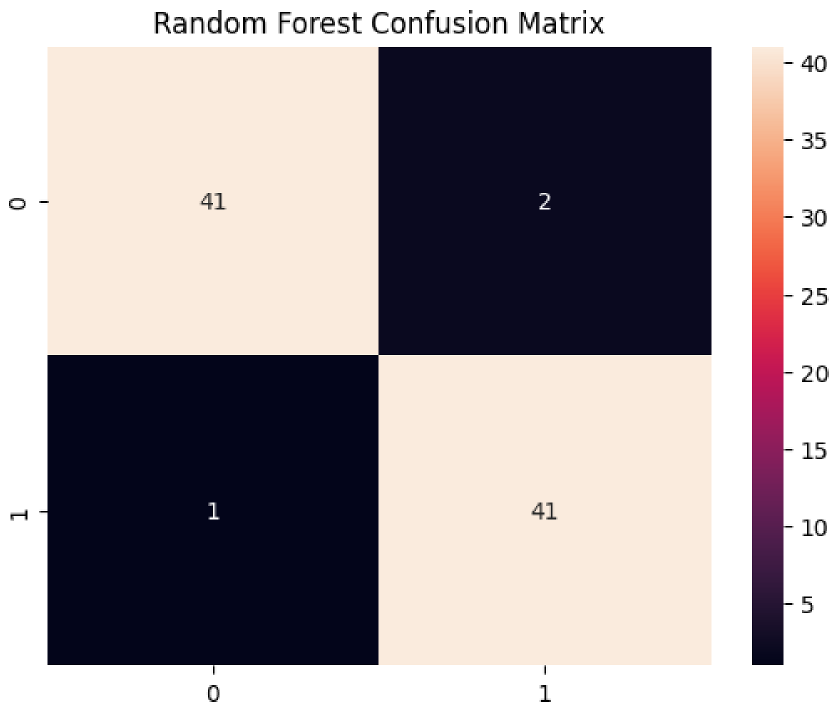


Screenshots of Code 2. Applying Machine Learning to make a classifier

5) Random Forest

```
rdm_frst = RandomForestClassifier(n_estimators=5, random_state=10)
rdm_frst.fit(x_train, y_train)

y_pred = rdm_frst.predict(x_test)
rdm_train_acc = round(accuracy_score(y_train, rdm_frst.predict(x_train)) *100, 2)
rdm_test_acc = round(accuracy_score(y_test, y_pred)*100, 2)
print('₩nAccuracy = ', rdm_test_acc, ' %₩n')

cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True)
plt.title('Random Forest Confusion Matrix')
```

```
Accuracy =  96.47  %
```

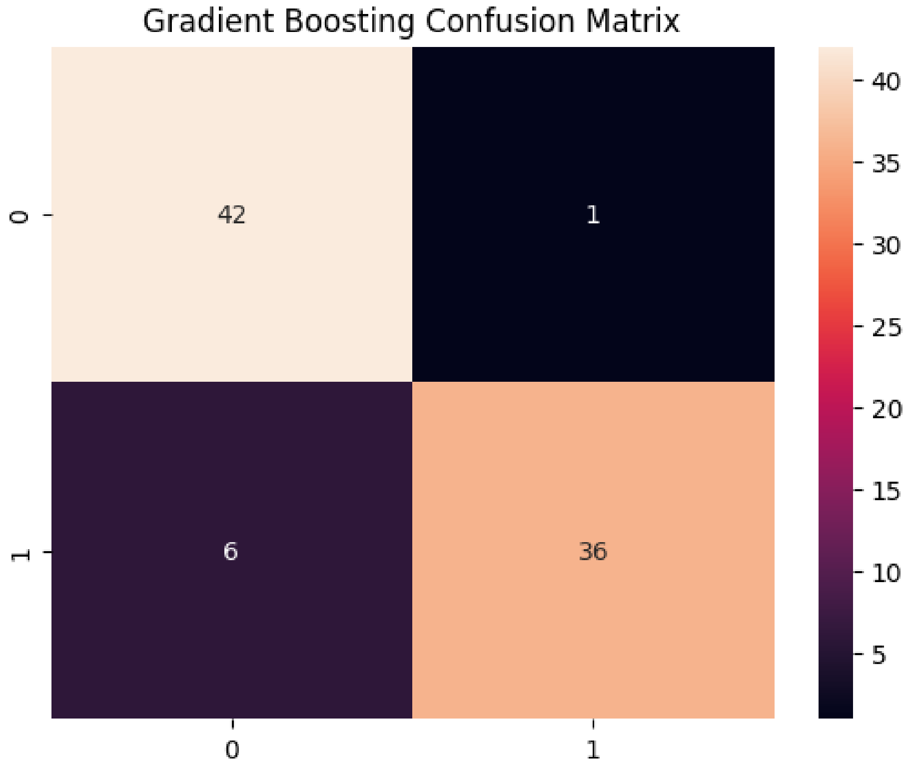


Screenshots of Code 2. Applying Machine Learning to make a classifier

6) Gradient Boosting

```
] gb = GradientBoostingClassifier(n_estimators=5, random_state=10)
  gb.fit(x_train, y_train)

  y_pred = gb.predict(x_test)
  gb_train_acc = round(accuracy_score(y_train, gb.predict(x_train)) *100, 2)
  gb_test_acc = round(accuracy_score(y_test, y_pred)*100, 2)
  print('₩nAccuracy = ', gb_test_acc, ' %₩n')

  cm = confusion_matrix(y_test, y_pred)
  sns.heatmap(cm, annot=True)
  plt.title('Gradient Boosting Confusion Matrix')
```

```
Accuracy =  91.76  %
```



Screenshots of Code 2. Applying Machine Learning to make a classifier

Comparing Classification Models

```
models = pd.DataFrame({
    'Model': [
        'Logistic Regression','K Nearest Neighbors','Linear Support Vector Machines',
        'Decision Tree','Random Forest', 'Gradient Boosting'
    ],
    'Model Accuracy Score': [
       lr_test_acc,knn_test_acc,ln_svm_test_acc,tree_test_acc,rdm_test_acc,gb_test_acc
    ]
})
models.sort_values(by='Model Accuracy Score', ascending=False)
```

	Model	Model Accuracy Score
2	Linear Support Vector Machines	100.00
0	Logistic Regression	96.47
3	Decision Tree	96.47
4	Random Forest	96.47
1	K Nearest Neighbors	95.29
5	Gradient Boosting	91.76

Then, finally, you can solve the multi-class problem by using the feature vector in the SVM, which is one of the way to teach a machine about features.

Performance evaluation was conducted using images that were manually labeled with defects (ground truth images). As a result, the accuracy reached 95.2%. K-means clustering and SVM technologies were selected due to the excellent performance found in other existing papers, and in the case of SVM, the performance of other classification technologies, decision tree and naive bayes, was compared for comparison, but the SVM was finally selected with overwhelming accuracy.

As stated in the paper, SVM shows the highest performance.

Screenshots of Code 2. Applying Machine Learning to make a classifier

- With SVM

```
img=cv2.imread("/content/drive/MyDrive/Colab Notebooks/machine vision system project/Diseased/a0_a49.jpg")
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB) #brg->rgb
plt.imshow(img)
plt.axis('off')
plt.show()

sample=[img.flatten()]
probability_svm=ln_svm.predict(sample)
print("The predicted image is : "+class_names[probability_svm[0]])
```

The predicted image is : Diseased

Screenshots of Code 2. Applying Machine Learning to make a classifier

- With SVM

```
img=cv2.imread("/content/drive/MyDrive/Colab Notebooks/machine vision system project/Healthy/r3_1 (25).jpg")
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB) #brg->rgb
plt.imshow(img)
plt.axis('off')
plt.show()

sample=[img.flatten()]
probability_svm=ln_svm.predict(sample)
print("The predicted image is : "+class_names[probability_svm[0]])
```

The predicted image is : Healthy

Screenshots of Code 3. Applying CNN deep Learning to make a classifier

Importing Data

```
PATH = '/content/drive/MyDrive/Colab Notebooks/machine vision system project'
```

```
#diseased_dir = os.path.join(PATH, 'Diseased')
#healthy_dir = os.path.join(PATH, 'Healthy')
```

```
data_dir = PATH
```

```
batch_size = 3
sz=(100,100)
```

데이터 split

```
train_dataset = tf.keras.utils.image_dataset_from_directory(data_dir,
                                                            shuffle=True,
                                                            batch_size=batch_size,
                                                            validation_split=0.3,
                                                            subset="training",
                                                            interpolation='bicubic',
                                                            image_size=sz,
                                                            seed=1204)

test_and_val_dataset = tf.keras.utils.image_dataset_from_directory(data_dir,
                                                            shuffle=True,
                                                            batch_size=batch_size,
                                                            validation_split=0.3,
                                                            subset="validation",
                                                            image_size=sz,
                                                            interpolation='bicubic',
                                                                 seed=1204)
```

```
Found 421 files belonging to 2 classes.
Using 295 files for training.
Found 421 files belonging to 2 classes.
Using 126 files for validation.
```

Screenshots of Code 3. Applying CNN deep Learning to make a classifier

split validation set

```
test_batches_num = tf.data.experimental.cardinality(test_and_val_dataset)
val_dataset = test_and_val_dataset.take(test_batches_num // 5)
test_dataset = test_and_val_dataset.skip(test_batches_num // 5)
```

```
print('Number of available test and validation batches: %d' %test_batches_num)
```

```
Number of available test and validation batches: 42
```

```
print('Number of validation batches: %d' % tf.data.experimental.cardinality(val_dataset))
print('Number of test batches: %d' % tf.data.experimental.cardinality(test_dataset))
```

```
Number of validation batches: 8
Number of test batches: 34
```

```
print(train_dataset.class_names)
print(test_and_val_dataset.class_names)
class_names = train_dataset.class_names
```

```
['Diseased', 'Healthy']
['Diseased', 'Healthy']
```

Screenshots of Code 3. Applying CNN deep Learning to make a classifier

- Showing dataset

```
plt.figure(figsize=(10, 10))
for images, labels in train_dataset.take(1):
  for i in range(3):
    ax = plt.subplot(3, 3, i + 1)
    #plt.imshow(images[i].numpy().astype("uint8"))
    plt.imshow(images[i]/255)
    plt.title(class_names[labels[i]])
    plt.axis("off")
```

Screenshots of Code 3. Applying CNN deep Learning to make a classifier

- Dividing to batches

```python
test_dataset_batch1 = list(test_dataset.take(1))[0]
test_dataset_batch1
```

```
(<tf.Tensor: shape=(3, 100, 100, 3), dtype=float32, numpy=
 array([[[[190., 201., 233.],
          [191., 202., 234.],
          [192., 200., 236.],
          ...,
          [144., 145., 176.],
          [144., 145., 176.],
          [141., 142., 173.]],

         [[192., 200., 236.],
          [192., 200., 236.],
          [192., 200., 236.],
          ...,
          [145., 146., 176.],
          [144., 145., 175.],
          [141., 142., 172.]],

         [[195., 200., 242.],
```

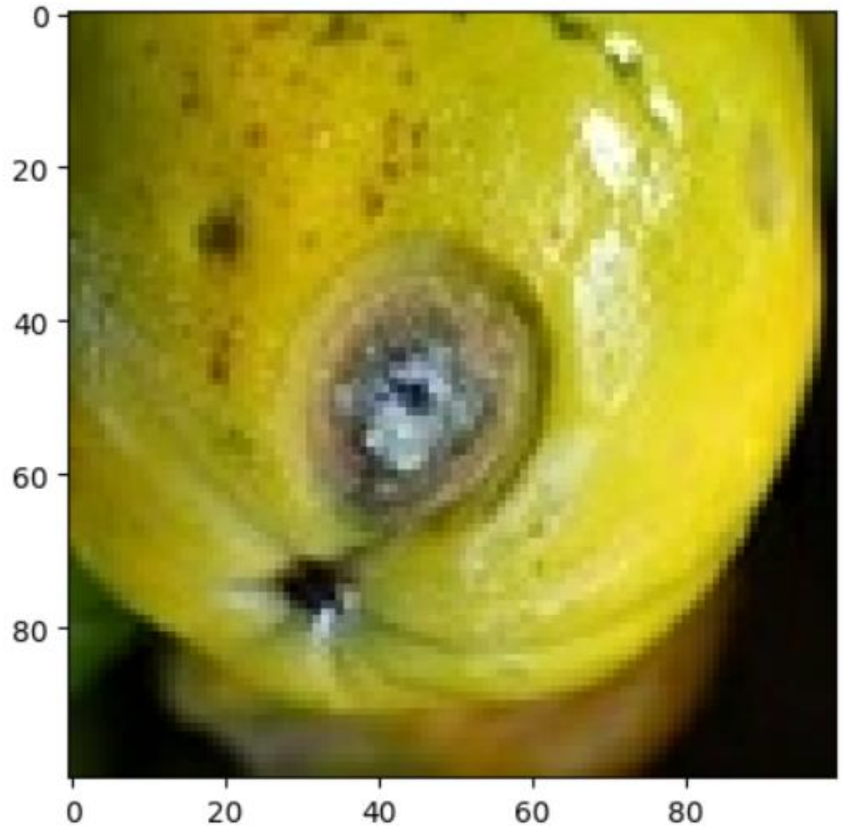
```
          [144., 148., 177.],
          [143., 147., 176.],
          ...,
          [152., 156., 191.],
          [152., 156., 191.],
          [152., 156., 191.]],

         [[123., 127., 156.],
          [123., 127., 156.],
          [123., 127., 156.],
          ...,
          [151., 155., 190.],
          [151., 155., 190.],
          [151., 155., 190.]]]], dtype=float32)>,
 <tf.Tensor: shape=(3,), dtype=int32, numpy=array([1, 0, 1], dtype=int32)>)
```

Screenshots of Code 3. Applying CNN deep Learning to make a classifier

- Showing one of the image in a Batch

```
plt.imshow(test_dataset_batch1[0][0]/255)
```

```
<matplotlib.image.AxesImage at 0x7c2a2ac5ef20>
```



Screenshots of Code 3. Applying CNN deep Learning to make a classifier

- Creating a CNN model

```
num_classes = 2

model = tf.keras.Sequential([
    tf.keras.layers.Rescaling(1./255),  # lightness to range in [0,1]
    tf.keras.layers.Conv2D(32, 3, activation='relu',padding='same',input_shape=sz),
    tf.keras.layers.MaxPooling2D(strides=(2,2)),
    tf.keras.layers.Conv2D(32, 3, activation='relu',padding='same'),
    tf.keras.layers.MaxPooling2D(strides=(2,2)),
    tf.keras.layers.Conv2D(32, 3, activation='relu',padding='same'),
    tf.keras.layers.MaxPooling2D(strides=(2,2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(num_classes),
    tf.keras.layers.Softmax()
    ])
```

Screenshots of Code 3. Applying CNN deep Learning to make a classifier

- Compiling and fitting

```
model.compile(
  optimizer='adam',
  loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False), # loss=tf.keras.losses.BinaryCrossentropy(from_logits=True)
  metrics=['accuracy'])
```

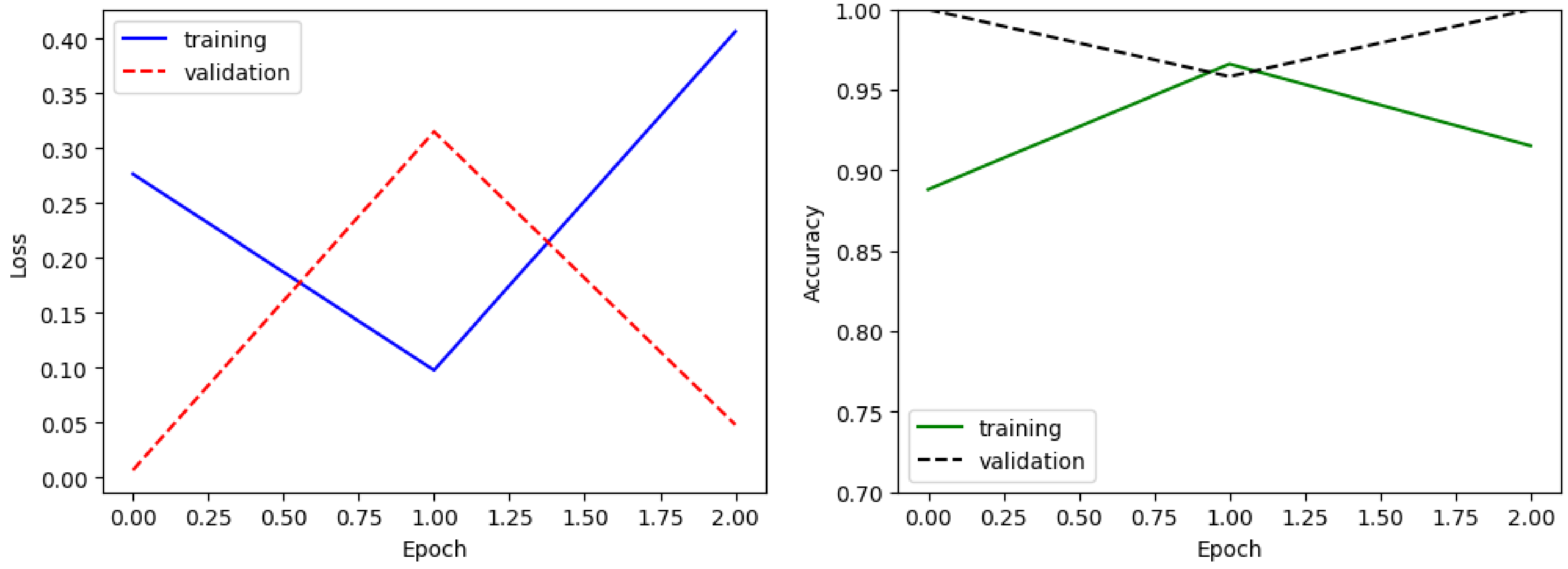
```
history = model.fit(
    train_dataset,
    validation_data=val_dataset,
    epochs=3
)
```

```
Epoch 1/3
99/99 [==============================] - 14s 99ms/step - loss: 0.2768 - accuracy: 0.8881 - val_loss: 0.0067 - val_accuracy: 1.0000
Epoch 2/3
99/99 [==============================] - 9s 90ms/step - loss: 0.0978 - accuracy: 0.9661 - val_loss: 0.3154 - val_accuracy: 0.9583
Epoch 3/3
99/99 [==============================] - 10s 100ms/step - loss: 0.4067 - accuracy: 0.9153 - val_loss: 0.0481 - val_accuracy: 1.0000
```

Screenshots of Code 3. Applying CNN deep Learning to make a classifier

```
plt.figure(figsize=(12, 4)), plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], 'b-', label='training')
plt.plot(history.history['val_loss'], 'r--', label='validation')
plt.xlabel('Epoch'), plt.ylabel('Loss'), plt.legend()
plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], 'g-', label='training')
plt.plot(history.history['val_accuracy'], 'k--', label='validation')
plt.xlabel('Epoch'), plt.ylabel('Accuracy'), plt.ylim(0.7, 1), plt.legend()
plt.show()
```

- Checking progress with plots



Screenshots of Code 3. Applying CNN deep Learning to make a classifier

- Evaluating

```
history.history
```

```
{'loss': [0.27677813172340393, 0.09782513976097107, 0.4067358970642 09],
 'accuracy': [0.8881356120109558, 0.9661017060279846, 0.9152542352676392],
 'val_loss': [0.006747635547071695, 0.3153701722621918, 0.04811384156346321],
 'val_accuracy': [1.0, 0.9583333134651184, 1.0]}
```

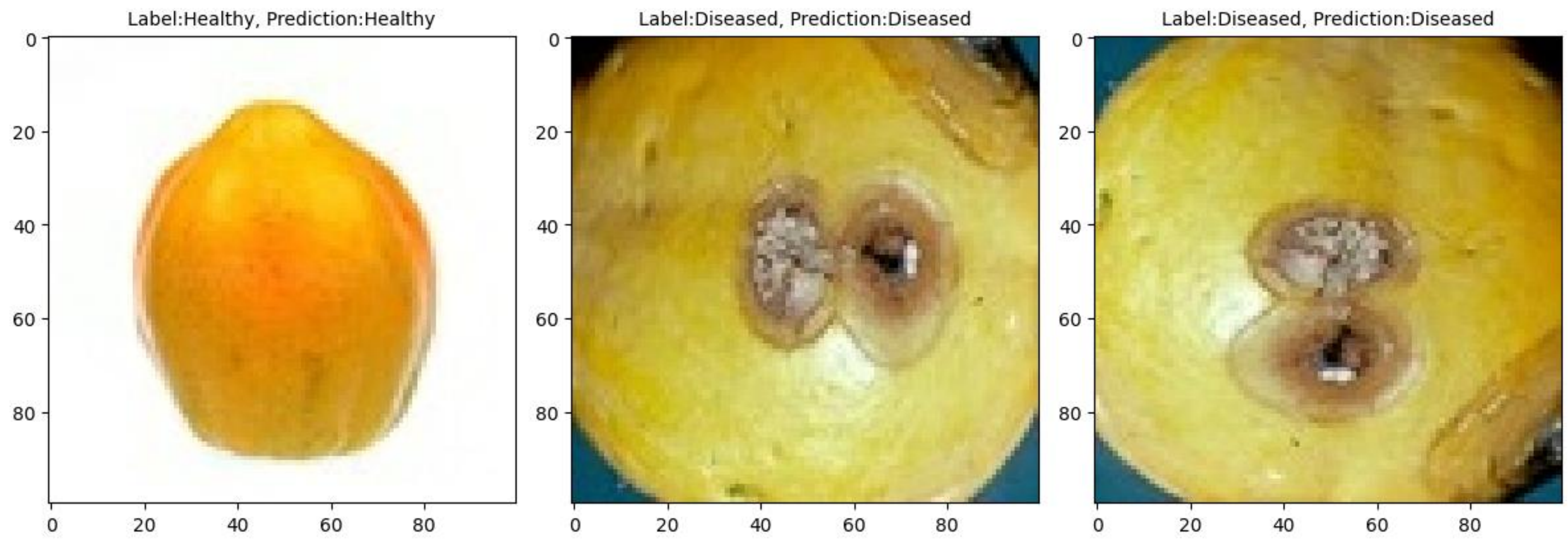
```
score = model.evaluate(test_dataset)
```

```
34/34 [==============================] - 1s 27ms/step - loss: 0.0537 - accuracy: 0.9902
```

Screenshots of Code 3. Applying CNN deep Learning to make a classifier

- Test with a randomly chosen batch of Test images

```
test_dataset_batch1 = list(test_dataset.take(1))[0]
#image_batch, label_batch = next(iter(test_dataset))

prediction = model.predict(test_dataset_batch1[0])
predicted_labels = np.argmax(prediction, axis=1)

#test_labels = np.argmax(test_dataset, axis=1)
test_labels = test_dataset_batch1[1]

count = 0

plt.figure(figsize=(12,8))
for n in range(3):
    count += 1
    plt.subplot(1, 3, count)
    plt.imshow(test_dataset_batch1[0][n]/255)
    #plt.imshow(test_dataset_batch1[0][n].reshape(28, 28, 3)/255, interpolation='bicubic')
    tmp = "Label:" + class_names[test_labels[n]] + ", Prediction:" + class_names[predicted_labels[n]]
    plt.title(tmp,fontdict = {'fontsize' : 10})

plt.tight_layout()
plt.show()
```




```
prediction
```

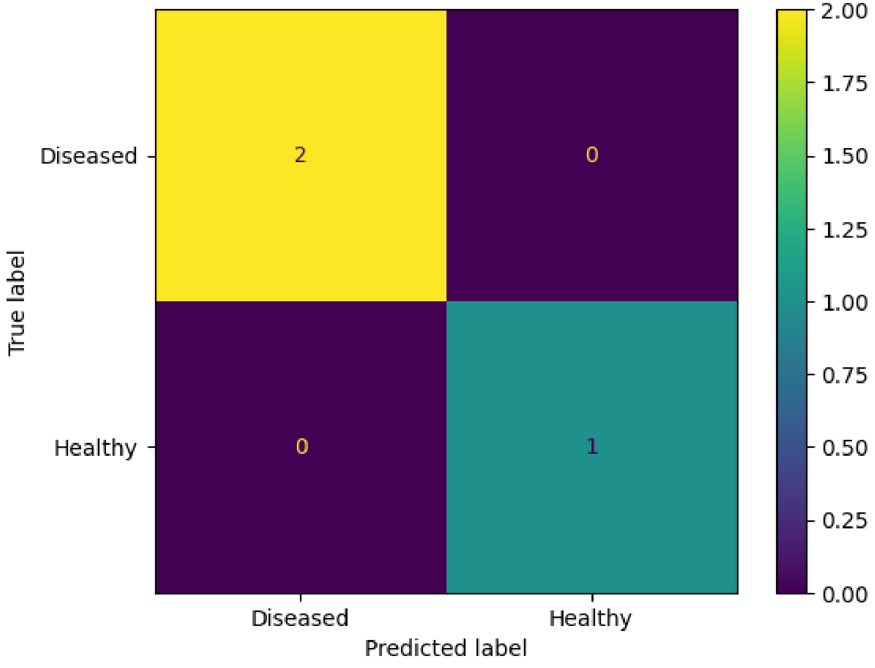
```
array([[0.03853271, 0.9614673 ],
       [0.04772344, 0.9522765 ],
       [0.02524399, 0.974756  ]], dtype=float32)
```

▲score

Screenshots of Code   3. Applying CNN deep Learning to make a classifier

- Making confusion chart

```
cm = confusion_matrix(test_labels.numpy(), predicted_labels)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=class_names)
disp.plot()
plt.show()
```



Screenshots of Code

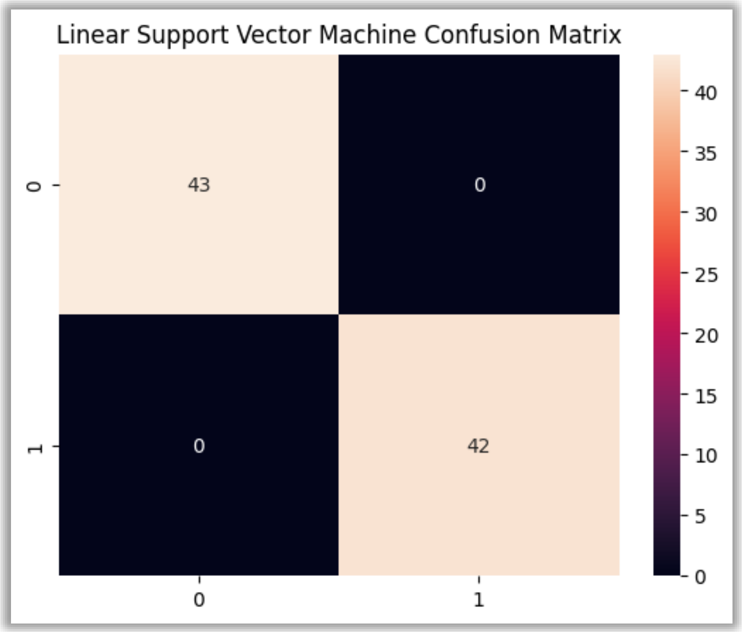
4. Evaluating

- As this paper,
- "K-means clustering and SVM technologies were selected due to the excellent performance found in other existing papers." and "SVM was finally selected with overwhelming accuracy."
- It was confirmed that the contents of the paper were true with this try.

Comparing Classification Models

```
models = pd.DataFrame({
    'Model': [
        'Logistic Regression','K Nearest Neighbors','Linear Support Vector Machines',
        'Decision Tree','Random Forest', 'Gradient Boosting'
    ],
    'Model Accuracy Score': [
        lr_test_acc,knn_test_acc,ln_svm_test_acc,tree_test_acc,rdm_test_acc,gb_test_acc
    ]
})
models.sort_values(by='Model Accuracy Score', ascending=False)
```

	Model	Model Accuracy Score
2	Linear Support Vector Machines	100.00
0	Logistic Regression	96.47
3	Decision Tree	96.47
4	Random Forest	96.47
1	K Nearest Neighbors	95.29
5	Gradient Boosting	91.76

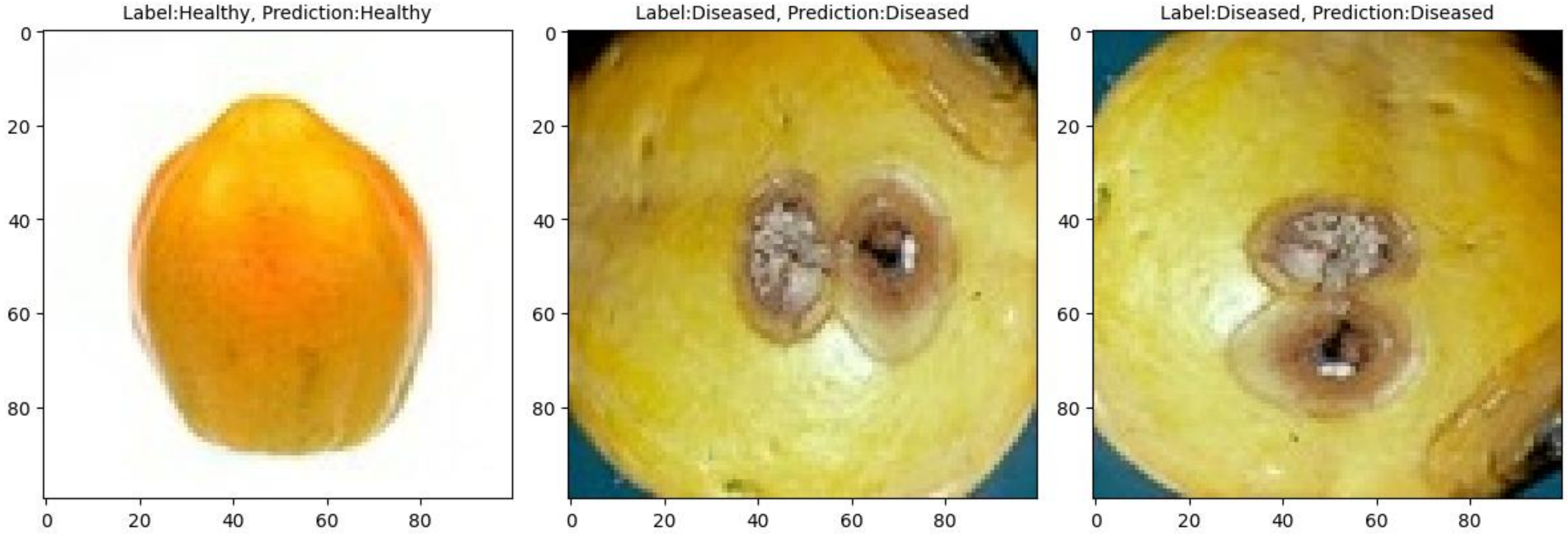


Screenshots of Code

4. Evaluating

- But Because machine learning involves the inconvenience of having to manually extract features,

I think using deep learning, which has sufficiently good performance, will be more helpful in the application development process.




```
prediction
```

```
array([[0.03853271, 0.9614673 ],
       [0.04772344, 0.9522765 ],
       [0.02524399, 0.974756  ]], dtype=float32)
```

Result (Demonstration)

https://colab.research.google.com/drive/1I8VxTbtETkzZR2y7DphHx8UhnLWFpLu5?usp=sharing

https://colab.research.google.com/drive/1dyBReLY0U1dDGTd17wsH_58cIlUJBR3w?usp=sharing