Fruit Recognition From Images

OUR TEAM

Kiwi Yu

Jiahui Chen

Yunxi Zhang

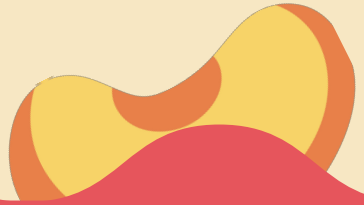
Yuzheng Xie

ABSTRACT

Grocery stores constantly strive to make sure their inventory and sales numbers are accurate. Considering this, our project is designed to reduce the loss associated with mistake while identifying them.

PROJECT WORKFLOW





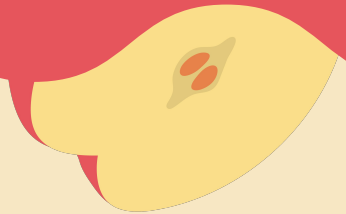
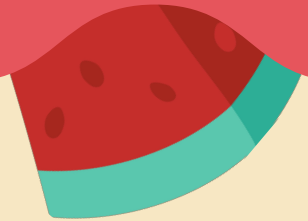
01 Get training and testing data

02 Define the layers

03 Compile the models

04 Train and evaluate the models

05 Get the predictions

ABOUT THE DATA

Dataset: Fruits 360
https://www.kaggle.com/moltean/fruits

The data set includes images of 131 types of fruits and vegetables, and 90,483 images. Different varieties of the same fruit are stored as belonging to different categories.

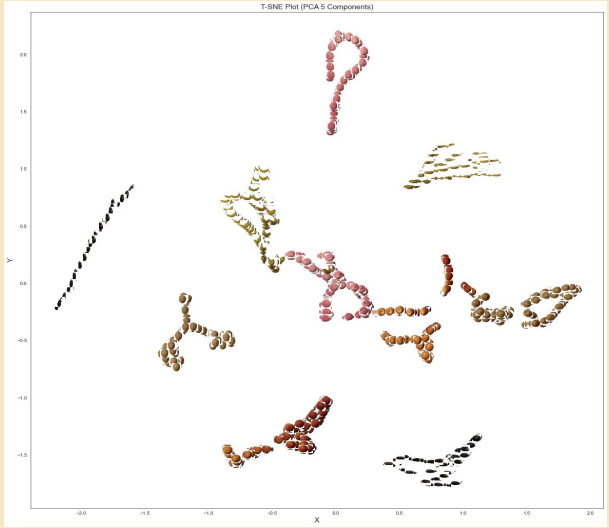
We select 5 different shapes of fruits from the dataset:

Lychee / Peach / Kiwi / Eggplant / Banana

```
The testing set size is :  808
The training set size is :  2406
```

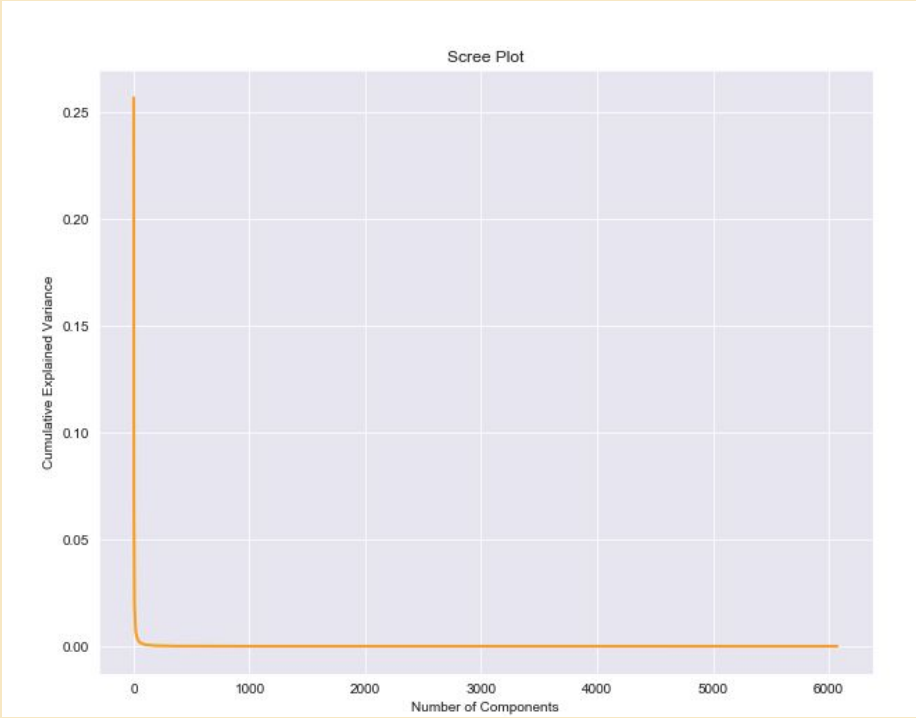
PCA - Principal Component analysis

Linear dimension reduction technique



For our dataset, we used Principal Components Analysis (PCA). This technique allows our dataset to be represented the same while greatly reducing the number of dimensions in our dataset.

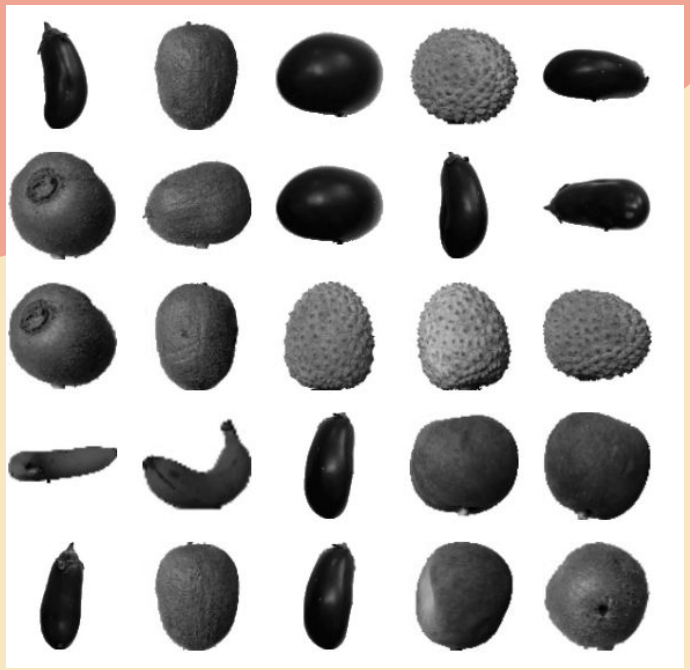
Scree plot



In order to know whether PCA will be useful or not, we need to create a scree plot. A scree plot shows the number of components plotted against explained variance. The plot below shows that using PCA is beneficial, and as a result, we are able to reduce our dataset to 52 components with 99.87% variance retained.

```
components      52.000000
explained_var    0.001361
shifted_var      0.001341
difference       0.000020
Name: 4, dtype: float64
```

BEFORE PCA



AFTER PCA

MODEL - KNN

WHY KNN

KNN is used to classify and group images together, and it can actually predict where new fruits will be grouped

GOAL

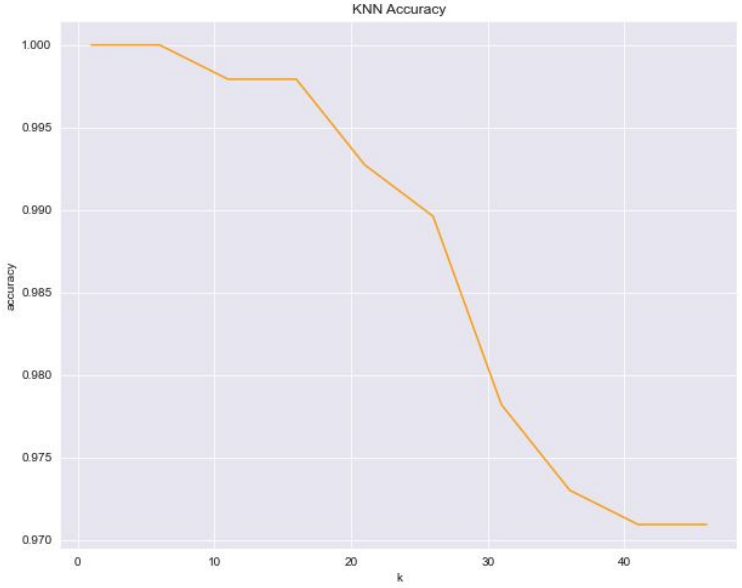
Pick k to have the highest accuracy on our test set

RESULT

As it turns out, this happened to be when k = 1, with accuracy of 97.65%

```
knn = KNeighborsClassifier(n_neighbors = 1)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
precision = (accuracy_score(test_label_ids, y_pred))*100
print("Accuracy with KNN: {0:.2f}%".format(precision))
```

```
Accuracy with KNN: 97.65%
```



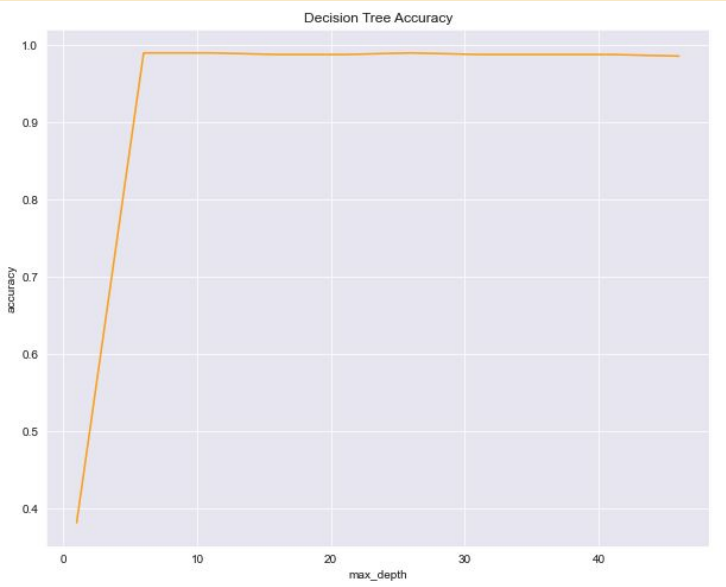
	max_depth	accuracy	accuracy_shift	difference
0	1	0.382139	0.989616	-0.607477
1	6	0.989616	0.989616	0.000000
2	11	0.989616	0.987539	0.002077
3	16	0.987539	0.987539	0.000000
4	21	0.987539	0.989616	-0.002077
5	26	0.989616	0.987539	0.002077
6	31	0.987539	0.987539	0.000000
7	36	0.987539	0.987539	0.000000
8	41	0.987539	0.985462	0.002077
9	46	0.985462	0.000000	0.985462

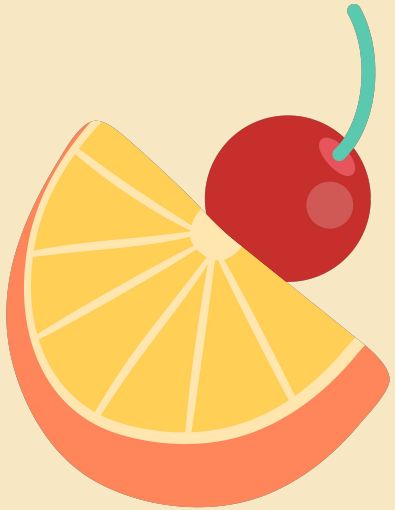
MODEL - DECISION TREE

Decision trees are pretty intuitive in that they split the data based on certain features that differentiate different groups. We fit the data to our training set to see what the best depth of our tree would be. The final accuracy is 88.48%.

```
dtree = DecisionTreeClassifier(max_depth = 36)
dtree.fit(X_train, y_train)
y_pred = dtree.predict(X_test)
precision = accuracy_score(y_pred, y_test)*100
print("Accuracy with Decision Tree: {0:.2f}%".format(precision))
```

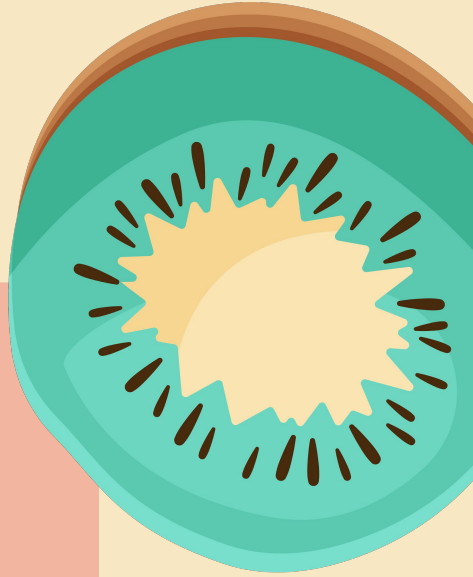
```
Accuracy with Decision Tree: 88.48%
```



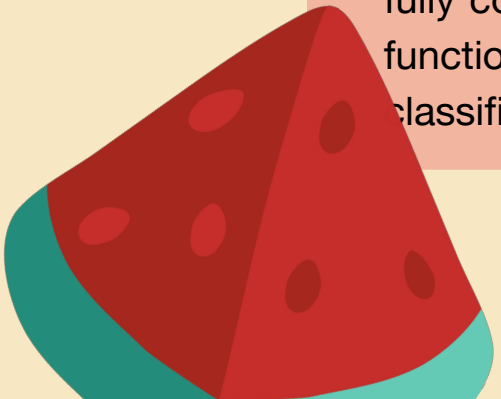


	max_depth	accuracy	accuracy_shift	difference
0	1	0.809969	0.986501	-0.176532
1	6	0.986501	0.991693	-0.005192
2	11	0.991693	0.991693	0.000000
3	16	0.991693	0.991693	0.000000
4	21	0.991693	0.991693	0.000000
5	26	0.991693	0.991693	0.000000
6	31	0.991693	0.991693	0.000000
7	36	0.991693	0.991693	0.000000
8	41	0.991693	0.991693	0.000000
9	46	0.991693	0.000000	0.991693

MODEL - CNN



Given CNN's ability to classify images well, we decided to apply this model to our dataset. Our CNN had four convolutional layers. For each layer, we added the ReLU activation function and used the max operation for pooling to reduce the spatial dimensions. For the fully connected layers, we used the softmax activation function, which is specifically used for multilabel classification.



MODEL - CNN RESULT

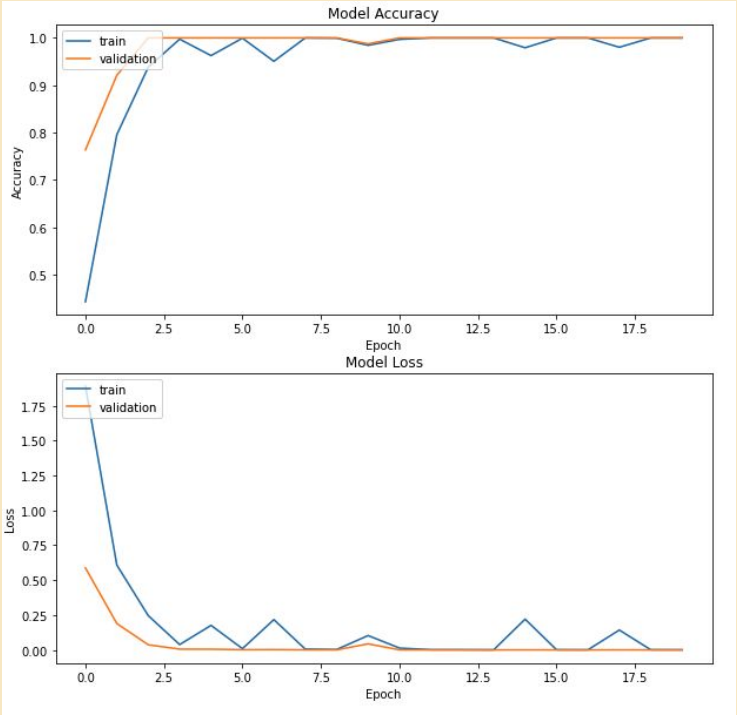
20 EPOCHS

ACCURACY

99.99%

SOFTMAX ACTIVATION

120 BATCH SIZE



MODEL - VGG16

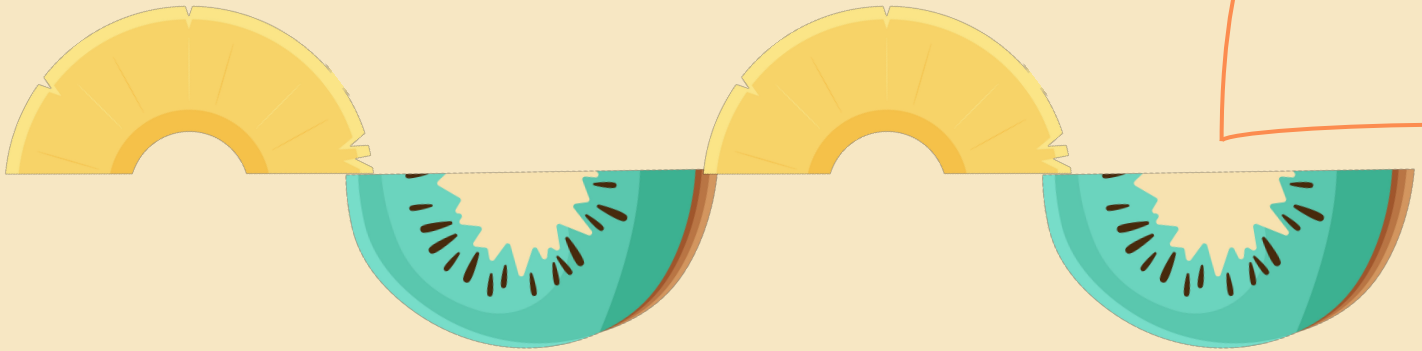
Transfer learning is a machine learning method in which a model that has already been developed is reused as the starting point model for another task, and we wanted to see if the results could justify the resources needed to run it

DRAWBACK

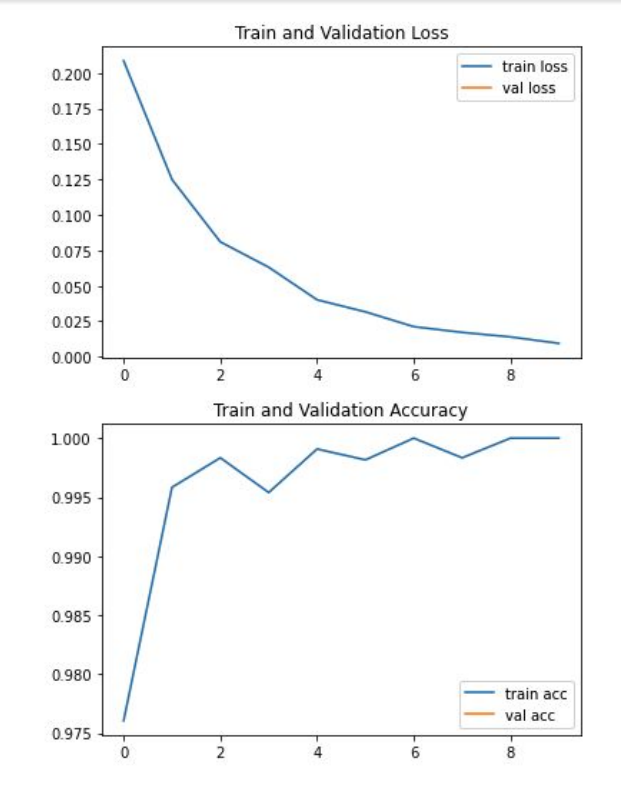
VERY SLOW

BUT

always good result

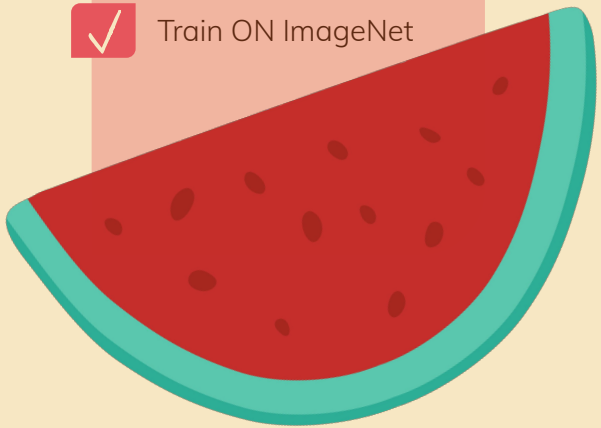


MODEL - VGG16 RESULT



ACCURACY

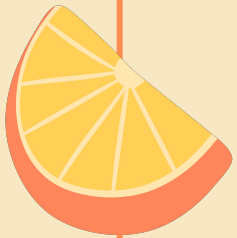
- TAKES HOURS
- 99.17% ACCURACY
- Train ON ImageNet



Result comparison

KNN

97.65%

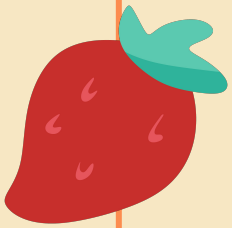


01

02

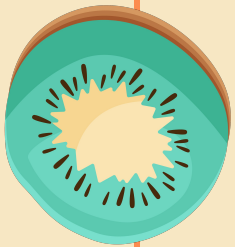
Decision tree

88.48%



CNN

99.99%

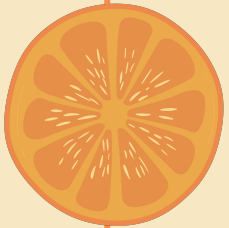


03

04

VGG16

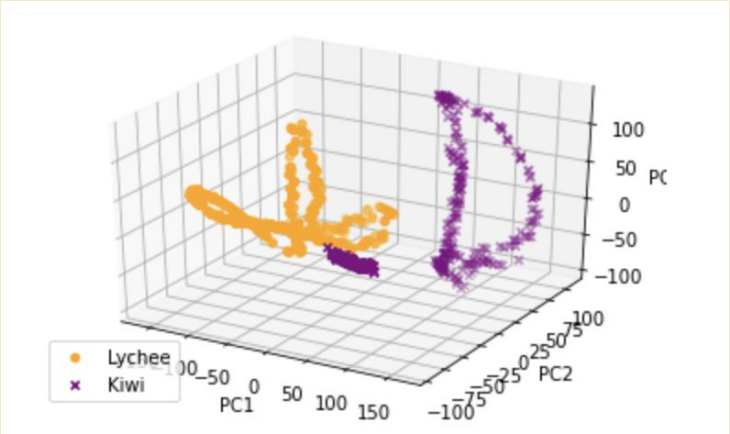
99.17%

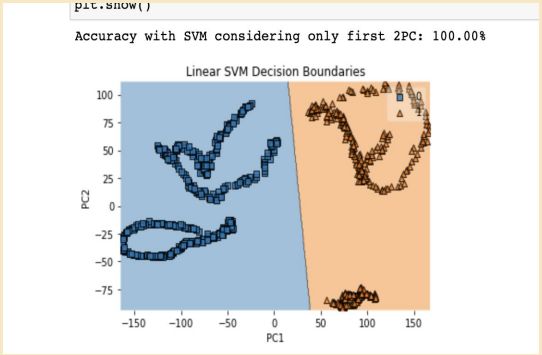


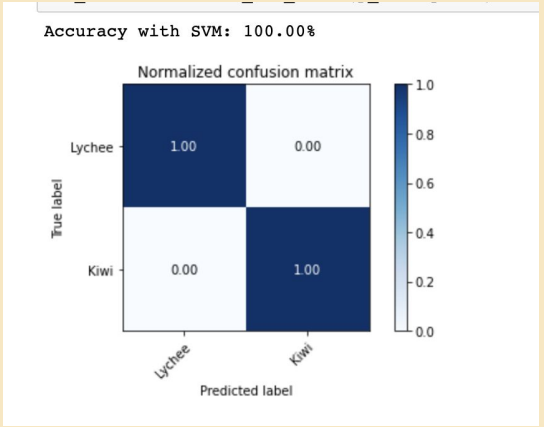
CONCLUSION

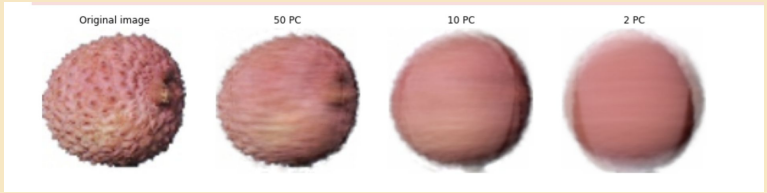
While our highest performing model is currently a CNN, the VGG16 model might be the best for long-term classification as more images are trained in the larger ImageNet database. Additionally, it is valuable to note that although KNN has a slightly lower accuracy than CNN, it took much less time to run. On the other hand, VGG16 took the longest to run and was only the third highest accuracy.

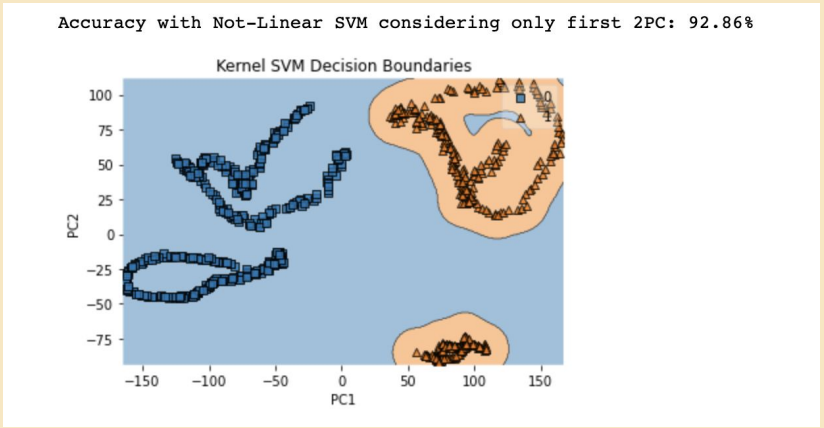
Improvement_SVM












THANKS