كليـــــة الحاسبات والذكاء الإصطناعى

كنترول الفرقة الثالثة

العـــام الجامعـــى 2019 / 2020 – دور مايـــو

الغلاف الخارجى للبحث

أولاً: البيانات الخاصة بالطالب				
الفرقة الدراسية	الثالثة	التخصص	علوم حاسب	
اسم القسم	علوم حاسب– عام			
اسم المقرر	التعرف على الانماط			
استاذ المقرر	دكتور : وسام			
ثانياً: البيانات الخاصة بالبحث				
عنوان البحث	Fruits 360 Dataset			
طبيعة المشاركة	بحث فردى [ ]		بحث جماعى [✓]	
ارسال البحث	بواسطة البريد الالكتروني			
اسماء الطلاب المشاركين فى البحث (يكتب الاسم رباعيا)	م	الاسم رباعى	رقم الجلوس	الرقم القومى
	1	احمد عبدالرحمن على عبدالرحمن	3110	29904012102912
	2	أيه إمام محمد إمام	3301	29906180102661
	3	جهاد رضوان احمد عبد الرحمن	3141	29902242103141
	4	هاجر محمد على بيومى	3236	29811281100367
	5			
تاريخ الإرسال	2020 / 6 /			
ثالثاً: البيانات الخاصة بالكنترول				
النتيجة	ناجح [ ]		راسب [ ]	
أعضاء لجنة تقييم البحث		الاسماء		التوقيع
	1			
	2			
	3			

فى حالة عدم قبول البحث يرجى ذكر الأسباب	- ..... - ..... - ..... - .....

CS 342 Pattern Recognition
Research Project

Report Submitted for Fulfillment of the Requirements and ILO's for Pattern Recognition course for Spring 2020

Team No : 25

Candidate graduates

	ID	Name	Level
1.			

Regular Students

1.	20170044	احمد عبدالرحمن على عبدالرحمن	three
2.	20170134	أيه إمام محمد إمام	three
3.	20170166	جهاد رضوان احمد عبد الرحمن	three
4.	20170617	هاجر محمد على بيومى	three

Delivered to:

Dr. Wessam El-Behaidy

Table of Content

1. PROJECT TASKS AND CORRESPONDING MEMBER

Candidate graduates

Team Member	Responsible Tasks	Page No.

Regular Students

Team Member	Responsible Tasks
احمد عبدالرحمن على عبدالرحمن	SVM
أيه إمام محمد إمام	LR
جهاد رضوان احمد عبد الرحمن	LR
هاجر محمد على بيومى	SVM

2. PROJECT INTRODUCTION

2.1 Dataset Used

Fruits 360 Dataset

2.2 Number of classes and their labels

40 classes.

Labels = ['Apple Braeburn', 'Apple Crimson Snow', 'Apple Golden 1',
'Apple Golden 2', 'Apple Golden 3', 'Apple Granny Smith', 'Apple Pink Lady',
'Apple Red 1', 'Apple Red 2', 'Apple Red 3', 'Apple Red Delicious',
'Apple Red Yellow 1','Apple Red Yellow 2', 'Apricot', 'Avocado', 'Avocado ripe',
'Banana', 'Banana Lady Finger', 'Banana Red', 'Beetroot', 'Blueberry', 'Cactus fruit',
'Cantaloupe 1', 'Cantaloupe 2', 'Carambula', 'Cauliflower', 'Cherry 1', 'Cherry 2',
'Cherry Rainier', 'Cherry Wax Black', 'Cherry Wax Red', 'Cherry Wax Yellow',
'Chestnut', 'Clementine', 'Cocos', 'Dates', 'Eggplant', 'Ginger Root', 'Granadilla',
'Grape Blue']

2.3 Dataset Images Numbers and size

Total number of images = 27,262 image
Each image size is 70*70*3

2.4 Training, Validation and Testing

(The number of images used in training, validation and testing.)
Total images divided to two section :
The number of images used in training = 20425 image
The number of images used in testing = 6837 image

3. IMPLEMENTATION DETAILS

3.1. Non-Deep Models

3.1.1. Extracted Features

(How many features are extracted and name them?)

it is the same for LR and SVM

All the images used are extracted .

The images are in RGP and resized to (70*70*3)

//converting each image to rgp in training set and test set

```
image= cv2.imread(path)
b,g,r = cv2.split(image)
image = cv2.merge([r,g,b])
image = cv2.resize(image,(img_size,img_size))
#normalization the image
image = image/255.0
```

//reshape all the images in training set and test set

```
X_train = np.array(X_train).reshape(-1,features)
X_test = np.array(X_test).reshape(-1,features)
```

3.1.2. Cross-validation

-LR ---> NO
-SVM ---> NO

3.1.3. Logistic Regression (LR)

℘ Hyper-parameters

(Specify all the hyper-parameters (optimizer, regularization, …) with their specified value in implementation)
-the optimizer initial learning rate is 0.0035
- no. of epochs = 50
- batch size = 50
- no. of batches=408

3.1.4. Support Vector Machine (SVM)

℘ Hyper-parameters

(Specify all the hyper-parameters (optimizer, regularization, …) with their specified value in implementation)
-the optimizer initial learning rate is 0.0035
- no. of epochs = 50
- batch size = 50
- no. of batches= 408

-regulation Parameter alpha is 0.00000001
-penalty parameter delta is 2.1

3.2. Deep Models

3.2.1. Preprocessing Step

(data augmentation, resize,…. on input images, specify their values)
No augmentation done.
Resize images from 100*100 to 70*70

3.2.2. Convolutional Neural Network (CNN

℘ CNN Number of Layers

7 layers

℘ CNN Architecture

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 70, 70, 16)        448
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 35, 35, 16)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 35, 35, 32)        4640
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 17, 17, 32)        0
_________________________________________________________________
flatten (Flatten)            (None, 9248)              0
_________________________________________________________________
dense (Dense)                (None, 256)               2367744
_________________________________________________________________
dense_1 (Dense)              (None, 40)                10280
=================================================================
Total params: 2,383,112
Trainable params: 2,383,112
Non-trainable params: 0
_________________________________________________________________
```

℘ Hyper-parameters

initial learning rate =0.001 with decay = 0.9 for the 1st moment estimates. And decay = 0.999 for the exponentially weighted infinity norm.
optimizer (Adamax)
no regularization
Batch size = 32
No. of epochs = 10

3.2.3. Pre-Trained Model

℘ Chosen Pre-trained Model

VGG16

℘ Hyper-parameters

Optimizer Adamax
Initial learning rate =0.001 with decay = 0.9 for the 1st moment estimates. And decay = 0.999 for the exponentially weighted infinity norm.
No regularization
Batch size = 32
No. of epochs = 10

℘ Tuning Layers

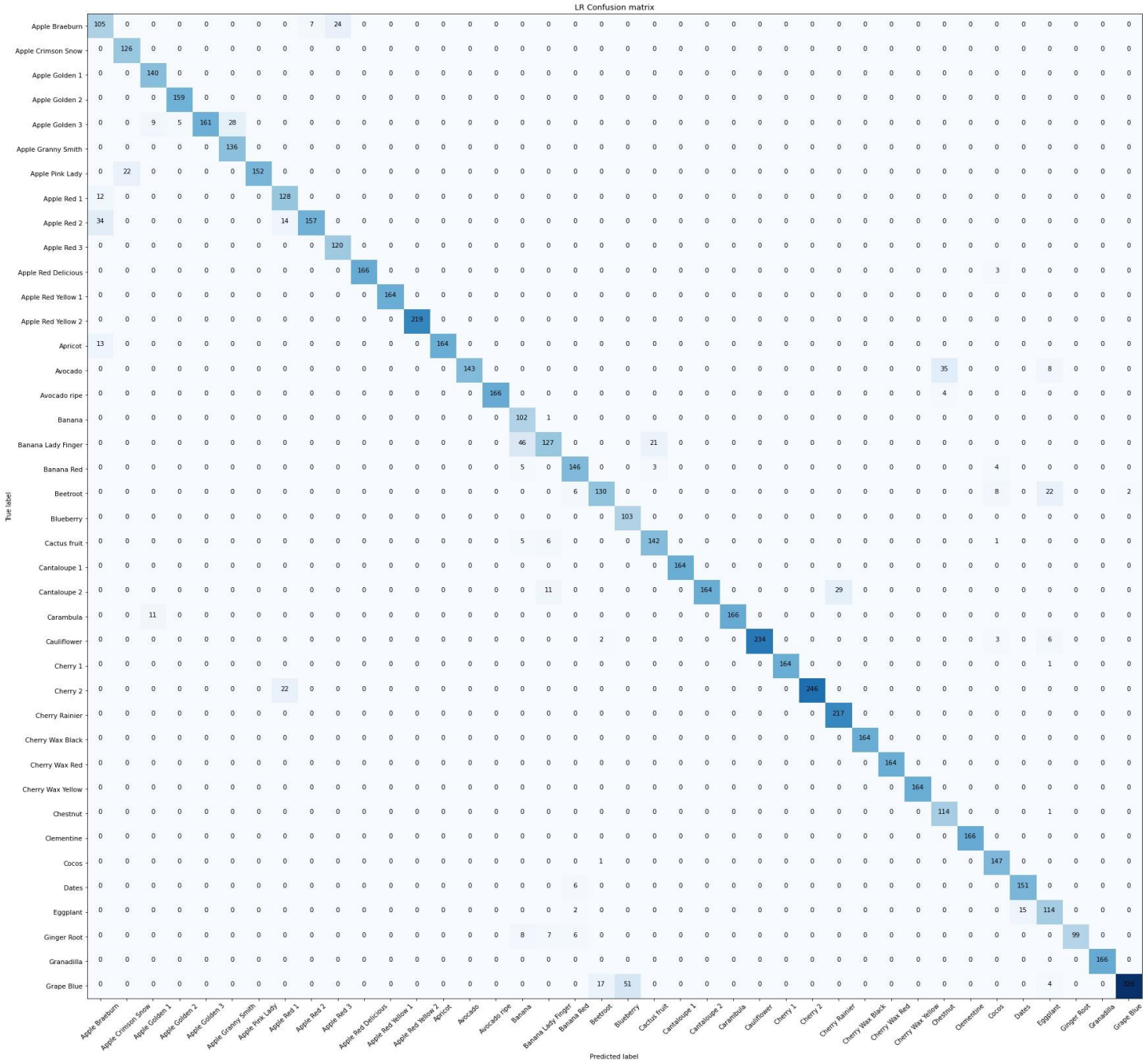
There is no tuning performed.

4. MODELS RESULTS

The results of your model on testing data (accuracy, confusion matrix)

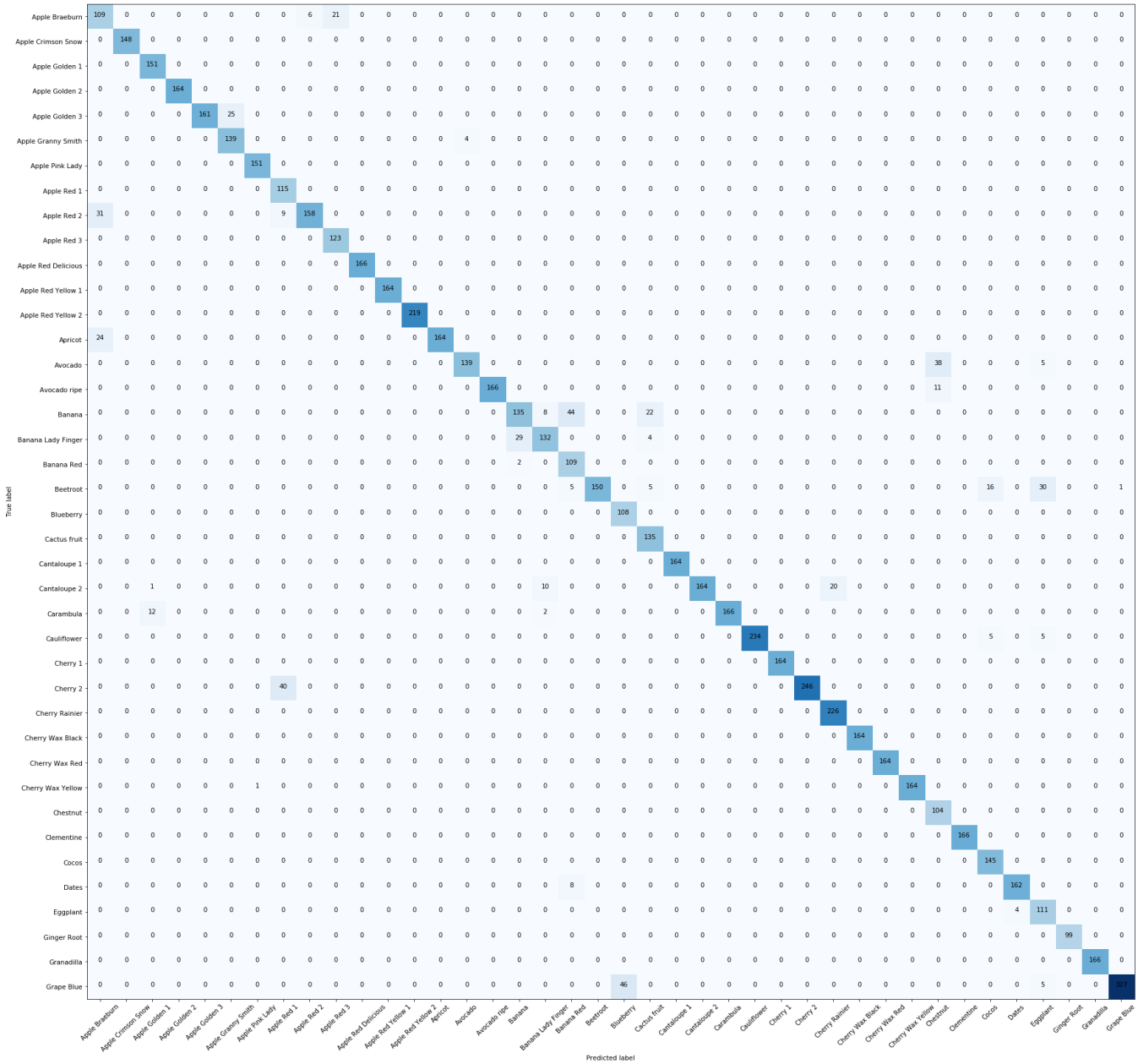
4.1. LR Results

Accuracy = 91.94%



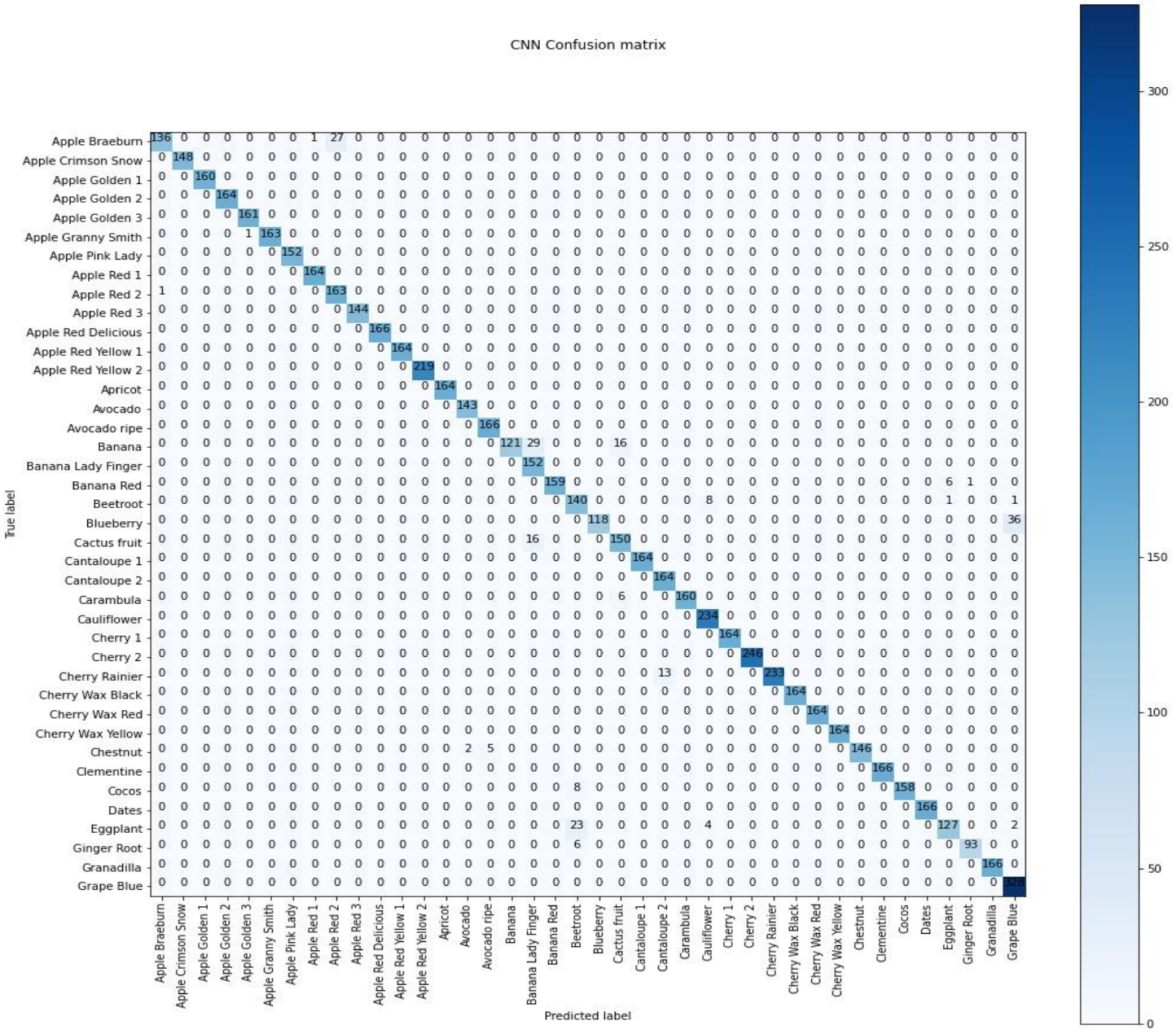
4.2. SVM Results

Accuracy = 92.71%

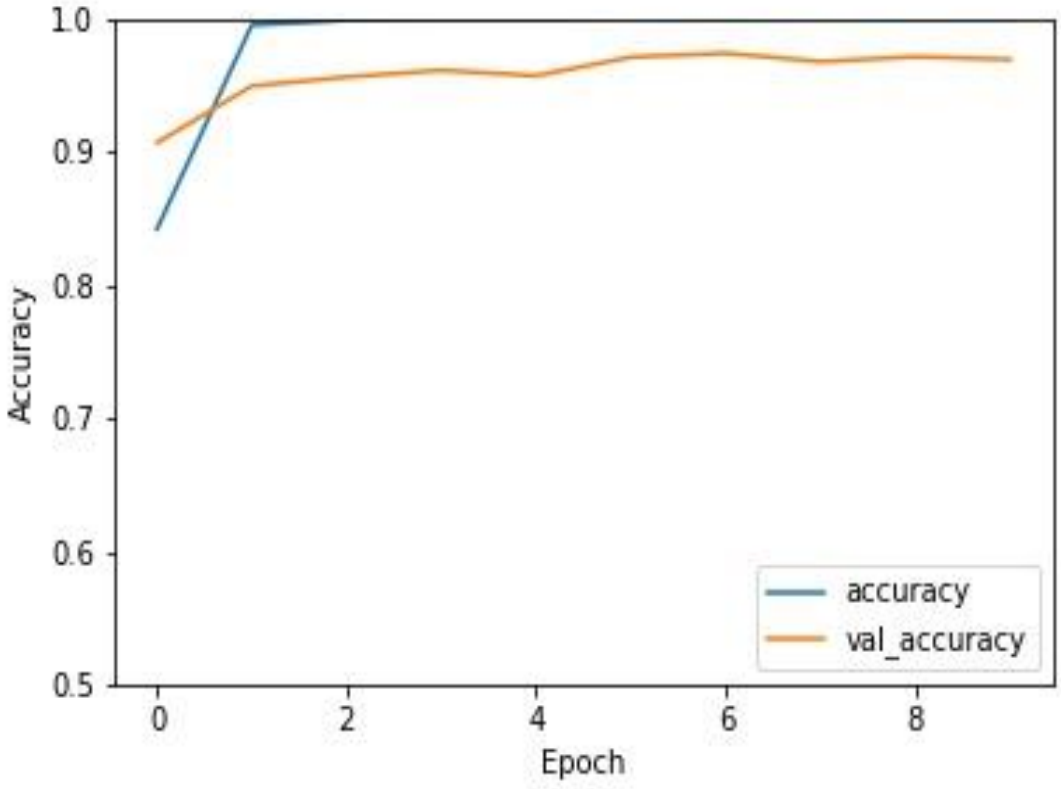


4.3. CNN Results and Loss Screen Shots

Accuracy: 96.88%



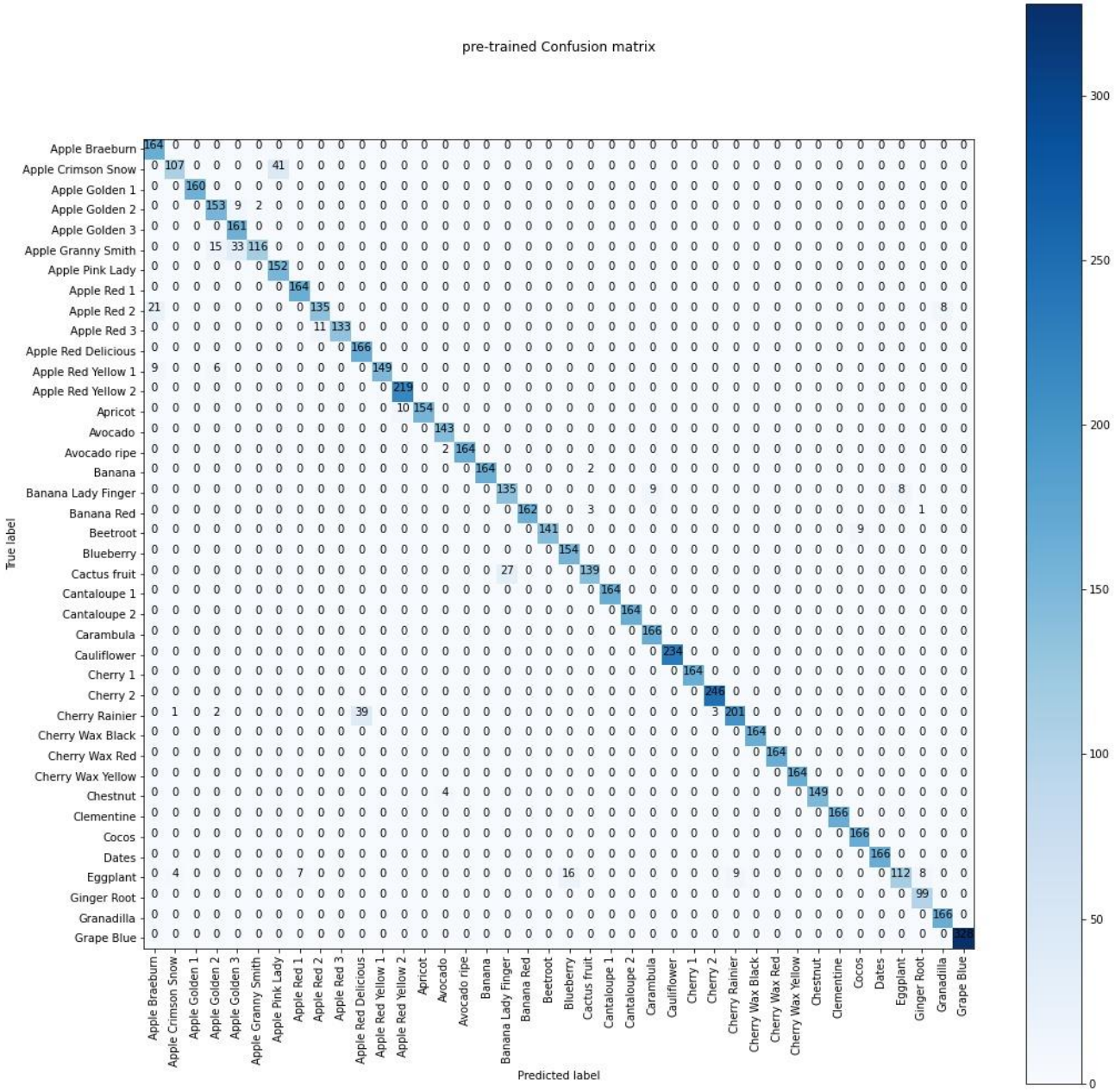
validation loss function



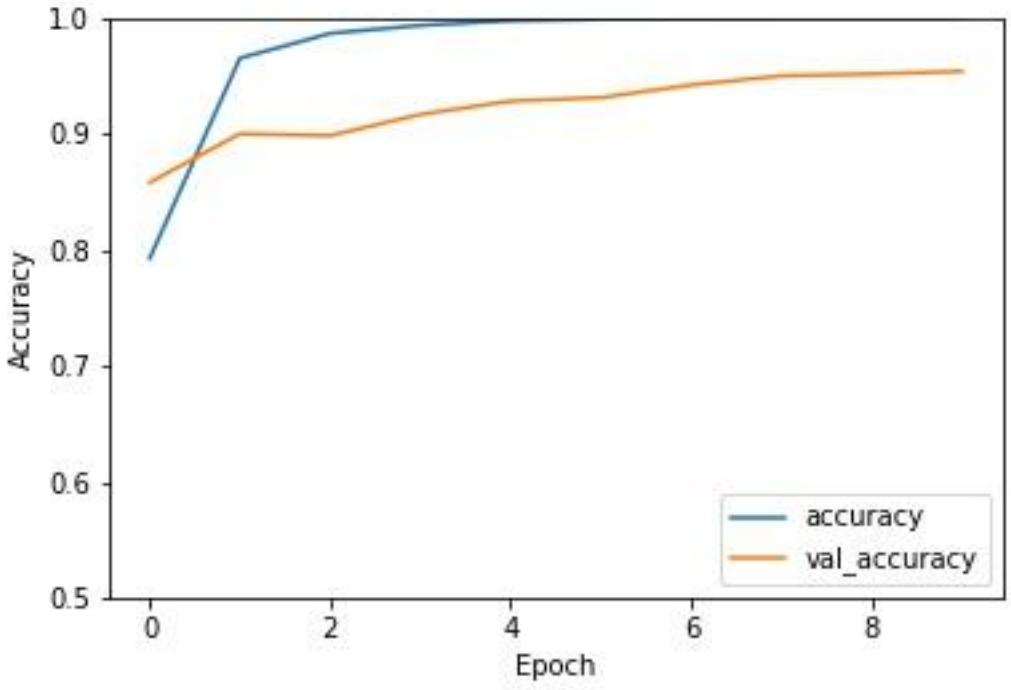
4.4. Pre-trained Results and Loss Screen Shots

+ (Screen Shots of training with validation loss function.)

Accuracy: 95.33%



validation loss function



5. COMPARISON AND CONCLUSION

(Compare the previous results, what do you think about extracted features and classification in the four models?)

5.1. LR and SVM

	SVM	LR
Brief	It used to maximize the margin among class variables	It used to maximize the probability of reaching to a certain label
classification point	-1<C<1	0<C<1
Speed	Faster	slower
theory	geometrical properties	statistical approaches
Accuracy	92.71%	91.94%
Hyper_parameters	learning rate=0.0035 epochs=50 bsize=50 batches= 408 alpha = 0.00000001 delta = 2.1	learning rate=0.0035 epochs=50 bsize=50 batches= 408

5.2. CNN and Pre-trained

	CNN	Pre-trained
No. layers	7	21
No. parameters	2,383,112	14.796.648
No. trainable parameters	2,383,112	81,960
weights	Trainable weights	ImageNet weights(freeze)
Speed	Faster	slower
Accuracy	96.88%	95.33%

5.3. All Models

	SVM	LR	CNN	PR
Hyper-parameters	-learning rate =0.0035 -epochs=50 -bsize=50 -batches=408 -alpha=0.00000001 -delta = 2.1	-learning rate =0.0035 -epochs=50 -bsize=50 -batches=408	-learning rate =0.001 -decay=0.9 (for the 1st Moment estimates) -decay=0.999 (for the exponentially weighted infinity norm) -Optimizer (Adamax) -Batch size=32 -epochs=10	-learning rate =0.001 -decay=0.9 (for the 1st moment estimates) -decay=0.999 (for the exponentially weighted infinity norm) -Optimizer (Adamax) -Batch size=32 -epochs=10
acc	92.71%	91.94%	96.88%	95.33%
Best in order (from 1 to 4)	**3**	**4**	**1**	**2**

6. APPENDIX

(The complete runnable code)

6.1. LR Code

```
#Connect to Dataset
!git clone https://github.com/Horea94/Fruit-Images-Dataset.git

#import ts 1.14
!pip install tensorflow==1.14.0
import tensorflow as tf
#------------------DataSet intialize-------------------------
category_list=['Apple Braeburn', 'Apple Crimson Snow', 'Apple Golden 1', 'Apple Golden 2
','Apple Golden 3', 'Apple Granny Smith','Apple Pink Lady', 'Apple Red 1', 'Apple Red 2'
,
'Apple Red 3', 'Apple Red Delicious', 'Apple Red Yellow 1','Apple Red Yellow 2', 'Aprico
t','Avocado', 'Avocado ripe', 'Banana', 'Banana Lady Finger', 'Banana Red', 'Beetroot',
'Blueberry', 'Cactus fruit', 'Cantaloupe 1', 'Cantaloupe 2', 'Carambula', 'Cauliflower',
'Cherry 1', 'Cherry 2','Cherry Rainier', 'Cherry Wax Black', 'Cherry Wax Red',
'Cherry Wax Yellow', 'Chestnut', 'Clementine', 'Cocos', 'Dates',
'Eggplant', 'Ginger Root', 'Granadilla', 'Grape Blue']
img_size=70
classes=len(category_list)
features=3*img_size*img_size
#------------------Training Preprocessing-------------------------
import numpy as np
import matplotlib.pyplot as plt
import os
import cv2
from random import shuffle
# from tqdm import tqdm
def get_training_data():
    i=0
    train_dir="content/Fruit-Images-Dataset/Training" #PC
#   train_dir="/content/Fruit-Images-Dataset/Training" #Colab
    training_data=[]
    for c in category_list:
        category=os.path.join(train_dir,c)
        for img in os.listdir(category):
            path=os.path.join(category,img)
            print(path)
            image= cv2.imread(path)
            b,g,r = cv2.split(image)
```

```
            image = cv2.merge([r,g,b])
            image = cv2.resize(image,(img_size,img_size))
            image = image/255.0
            image = image.flatten()
            training_data.append([image,i])
        i+=1
    return training_data
#------------------Test Preprocessing-------------------------
def get_testing_data():
    i=0
    test_dir="content/Fruit-Images-Dataset/Test" #PC
#   test_dir="/content/Fruit-Images-Dataset/Test" #Colab
    testing_data=[]
    for c in category_list:
        category=os.path.join(test_dir,c)
        for img in os.listdir(category):
            path=os.path.join(category,img)
            image= cv2.imread(path)
            b,g,r = cv2.split(image)
            image = cv2.merge([r,g,b])
            image = cv2.resize(image,(img_size,img_size))
            image = image/255.0
            image = image.flatten()
            testing_data.append([image,i])
        i+=1
    return testing_data
train_data=get_training_data()
test_data=get_testing_data()
import random
from random import shuffle
random.shuffle(train_data)
random.shuffle(test_data)
X_train = []
y_train = []
X_test = []
y_test = []
for value,index in train_data :
    X_train.append(value)
    y_train.append(index)
train_data.clear()

for value,index in test_data :
    X_test.append(value)
    y_test.append(index)
```

```
test_data.clear()
X_train = np.array(X_train).reshape(-1,features)
X_test = np.array(X_test).reshape(-1,features)
y_train = np.array(y_train)
y_test =np.array(y_test)

with tf.Session() as sesh:
    y_train=sesh.run(tf.one_hot(y_train,classes))
    y_test=sesh.run(tf.one_hot(y_test,classes))

#------------------LR Model-------------------------
learning_rate=0.0035
epochs=50
bsize=50
batches=int(X_train.shape[0] / bsize)
X = tf.placeholder(tf.float32,[None,features])
Y = tf.placeholder(tf.float32,[None,classes])
W=tf.Variable(tf.ones([features,classes],tf.float32))
B=tf.Variable(tf.zeros([classes],tf.float32))

prediction=tf.nn.softmax(tf.add(tf.matmul(X,W),B))
error=tf.reduce_mean(-tf.reduce_sum(Y * tf.log(prediction), axis=1))
optimizer=tf.train.GradientDescentOptimizer(learning_rate).minimize(error)

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for e in range(epochs):
        for i in range(batches):
            x=X_train[i*e:(i*e)+bsize]
            y=y_train[i*e:(i*e)+bsize]
            sess.run(optimizer,feed_dict={X: x, Y:y})
            h=sess.run(error,feed_dict={X:x, Y:y})
        print(f'e:{e:2d} cost={h:.4f}')
    the_prediction=tf.equal(tf.argmax(prediction, 1), tf.argmax(Y, 1))
    accuracy=tf.reduce_mean(tf.cast(the_prediction,tf.float32))
    acc=accuracy.eval({X:X_test,Y:y_test})*100
    print(f'Accuracy:{acc:.2f}%')

    prediction_values=sess.run(prediction,feed_dict={X:X_test})


#------------------Confusion Matrix-------------------------
import itertools
import matplotlib.pyplot as plt
with tf.Session() as sess:
```

```
true_class=np.argmax(y_test,1)
predicted_class=np.argmax(prediction_values,1)
cm=tf.confusion_matrix(predicted_class,true_class)
print(sess.run(cm))
plt.figure(figsize = (30,30))
plt.imshow(sess.run(cm), interpolation='nearest', cmap=plt.cm.Blues)
plt.title('LR Confusion matrix')
plt.colorbar()
tick_marks = np.arange(len(category_list))
plt.xticks(tick_marks, category_list, rotation=45)
plt.yticks(tick_marks, category_list)

for i, j in itertools.product(tick_marks, tick_marks):
    plt.text(j, i, format(sess.run(cm)[i, j]), horizontalalignment="center",
    color="black")
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.tight_layout()
plt.show()
```

6.2. SVM Code

Note:we use the same preprocessing code from LR

```
learning_rate=0.0035
epochs=50
bsize=50
batches=int(X_train.shape[0] / bsize)
X = tf.placeholder(tf.float32,[None,features])
Y = tf.placeholder(tf.float32,[None,classes])
W=tf.Variable(tf.ones([features,classes],tf.float32))
B=tf.Variable(tf.zeros([classes],tf.float32))

model_output = tf.matmul(X, W) + B
alpha = tf.constant([0.00000001]) #0.00000001 1
delta = tf.constant([2.1], tf.float32)
regulation_term = alpha * tf.nn.l2_loss(W) #make balaced bettwwen cost & margen
S_truelabel = tf.reduce_sum(tf.multiply(Y,model_output),axis=1 ,keepdims=True)
loss = tf.reduce_mean(tf.reduce_sum(tf.maximum(0.0, model_output -
 S_truelabel + delta), 1) - delta )
loss += regulation_term
SVM_Optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(loss)

with tf.Session() as sesh:
    sesh.run(tf.global_variables_initializer())
    for e in range(epochs):
        for i in range(batches):
            x=X_train[i*e:(i*e)+bsize]
            y=y_train[i*e:(i*e)+bsize]
            sesh.run(SVM_Optimizer,feed_dict={X: x, Y:y})
            h=sesh.run(loss,feed_dict={X:x, Y:y})
        # print(f'e:{e:2d} cost={h:.4f}')
    the_prediction=tf.equal(tf.argmax(model_output, 1), tf.argmax(Y, 1))
    accuracy=tf.reduce_mean(tf.cast(the_prediction,tf.float32))
    acc=accuracy.eval({X:X_test,Y:y_test})*100
    print(f'Accuracy:{acc:.2f}%')
    prediction_values=sesh.run(model_output,feed_dict={X:X_test})


#confusion_matrix
import itertools
import matplotlib.pyplot as plt
with tf.Session() as sess:
    true_class=np.argmax(y_test,1)
    predicted_class=np.argmax(prediction_values,1)
```

```
cm=tf.confusion_matrix(predicted_class,true_class)
print(sess.run(cm))
plt.figure(figsize = (30,30))
plt.imshow(sess.run(cm), interpolation='nearest', cmap=plt.cm.Blues)
plt.title('SVM Confusion matrix')
plt.colorbar()
tick_marks = np.arange(len(category_list))
plt.xticks(tick_marks, category_list, rotation=45)
plt.yticks(tick_marks, category_list)


for i, j in itertools.product(tick_marks, tick_marks):
    plt.text(j, i, format(sess.run(cm)[i, j]), horizontalalignment="center",
    color="black")

plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.tight_layout()
plt.show()
```

6.2. CNN Code

```
#Connect to Dataset
!git clone https://github.com/Horea94/Fruit-Images-Dataset.git
#Preprocessing Step***********
import os
import cv2
import random
import numpy as np
import matplotlib.pyplot as plt

img_shape=70

category_list= ['Apple Braeburn', 'Apple Crimson Snow', 'Apple Golden 1', 'Apple
Golden 2','Apple Golden 3', 'Apple Granny Smith', 'Apple Pink Lady', 'Apple Red
1', 'Apple Red 2', 'Apple Red 3','Apple Red Delicious', 'Apple Red Yellow1',
'Apple Red Yellow 2', 'Apricot', 'Avocado', 'Avocado ripe', 'Banana',
'Banana Lady Finger', 'Banana Red', 'Beetroot', 'Blueberry', 'Cactus fruit',
'Cantaloupe 1', 'Cantaloupe 2', Carambula', 'Cauliflower', 'Cherry 1', 'Cherry
2', 'Cherry Rainier', 'Cherry Wax Black', 'Cherry Wax Red', 'Cherry Wax Yellow',
'Chestnut', 'Clementine', 'Cocos', 'Dates', 'Eggplant', 'Ginger Root',
'Granadilla', 'Grape Blue']
print(len(category_list))

#get training data
i=0
directory= '/content/fruits-360/Training'
training_data=[]
for c in category_list:

    category= os.path.join(directory,c)
    for img in os.listdir(category):
        path= os.path.join(category,img)
        print(path)
        image= cv2.imread(path,cv2.COLOR_BGR2RGB)    #1 for convert image to rgp
        image= cv2.resize(image,(img_shape,img_shape))# resize image to 50*50 to
 reduse number of bixel
        image = cv2.cvtColor(image,cv2.COLOR_BGR2RGB)
        image = image/255.0     #normalize image as each pixel in grayscale has v
alue in range 0:255
        image = image.flatten()  #convert image 50*50 image matrix  to array 1*1
4700
        training_data.append([image,c,i])
    i +=1

#get test data
```

```
i=0
test_directory= '/content/fruits-360/Test'
test_data=[]
for c in category_list:
    category= os.path.join(test_directory,c)
    for img in os.listdir(category):
        path= os.path.join(category,img)
        print(path)
        image= cv2.imread(path,1)            #1 for convert image to rgp
        image= cv2.resize(image,(img_shape,img_shape)) # resize image to 50*50 t
o reduse number of bixel
        image = cv2.cvtColor(image,cv2.COLOR_BGR2RGB)
        image = image/255.0    #normalize image as each pixel in grayscale has v
alue in range 0:255
        image = image.flatten()  #convert image 50*50 image matrix  to array 1*1
4700
        test_data.append([image,c,i])
    i +=1

#split&shuffle
random.shuffle(training_data)
random.shuffle(test_data)

#split train and test data to values and labels
X_train = []
y_train = []
y_trainLabel =[]

X_test = []
y_test = []
y_testLabel = []
for value,label,index in training_data :
    X_train.append(value)
    y_train.append(index)
    y_trainLabel.append(label)
for value,label,index in test_data :
    X_test.append(value)
    y_test.append(index)
    y_testLabel.append(label)

X_train = np.array(X_train).reshape(-
1,img_shape,img_shape,3) #convert list  to matrix in shape 50*50 for each image
insted os 2500,
X_test = np.array(X_test).reshape(-1,img_shape,img_shape,3)
y_train = np.array(y_train)
```

```
y_test =np.array(y_test)
y_trainLabel =np.array(y_trainLabel)
y_testLabel =np.array(y_testLabel)
training_data.clear() # free some space of ram
test_data.clear()

# show some of test images
plt.figure(figsize=(20,20))
for i in range (25):
    #print(X_train[i])
    plt.subplot(5,5,i+1)  #nrow , ncol ,index
    plt.grid(False)
    plt.imshow(X_test[i])
    plt.xlabel(y_testLabel[i])

#cnn model
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D,MaxPooling2D ,Dense ,Dropout ,Activat
ion ,Flatten
from tensorflow.keras.callbacks import  TensorBoard
from tensorflow.keras.utils import to_categorical
import itertools

#reshape images to fit keras model
X_train = X_train.reshape(20425,img_shape,img_shape,3)
X_test = X_test.reshape(6837,img_shape,img_shape,3)

#cnn model

model= Sequential()

model.add(Conv2D(16,kernel_size=(3,3) , padding = "same",activation='relu',input
_shape=(img_shape,img_shape,3)))

model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Conv2D(32,kernel_size=(3,3), activation='relu'))

model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Flatten())

model.add(Dense(256,activation='relu'))
```

```
model.add(Dense(40,activation='softmax'))

model.summary()

print(len(model.layers))

#compile model
model.compile(tf.keras.optimizers.Adamax( learning_rate=0.001, beta_1=0.9, beta_
2=0.999), loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Load the TensorBoard notebook extension
%load_ext tensorboard

log_file_name = "model_1"
logdir = os.path.join("/content/drive/My Drive/pattern/logs", log_file_name )
tensorboard_callback = tf.keras.callbacks.TensorBoard(logdir)

##train the model
model_train = model.fit(X_train, y_train, validation_data=(X_test, y_test), batc
h_size=32,callbacks=[tensorboard_callback], epochs= 10 )

model.save('/content/drive/My Drive/pattern/model_1.h5')

#Evaluate the model
plt.plot(model_train.history['accuracy'], label='accuracy')
plt.plot(model_train.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')
plt.savefig('/content/drive/My Drive/acc.jpg')
test_loss, test_acc = model.evaluate(X_test,  y_test, verbose=2)

#confusion_matrix
y_pred= model.predict(X_test)
predClass = np.argmax(y_pred , 1)  #return index of largest value ,1 for work on
 second dimension
matrix = tf.math.confusion_matrix(y_test, predClass)

print(matrix)

plt.figure(figsize = (15,15))

plt.imshow(matrix, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('CNN Confusion matrix', y=1.1)
```

```
plt.colorbar()
tick_marks = np.arange(len(category_list))
plt.xticks(tick_marks, category_list, rotation=90)
plt.yticks(tick_marks, category_list)


for i, j in itertools.product(tick_marks, tick_marks):
    plt.text(j, i, format(matrix[i, j]), horizontalalignment="center",color="bla
ck")


plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.tight_layout()
plt.savefig('/content/drive/My Drive/matrix15.jpg')
plt.show()
```

6.3. Pre-trained Code

```
#same preprocessing code in cnn code

from keras import applications
from keras import optimizers
from keras.models import Sequential, Model
from keras.layers import Dense, GlobalAveragePooling2D
from keras.layers import Activation, Dropout, Flatten, Dense
from keras import backend as K

#import vgg16 with pre-
trained weights. do not include fully connected layers (Remove vgg default class
ifier)
vgg = applications.VGG16(input_shape=[70 ,70] + [3],weights='imagenet', include_
top=False)

# don't train existing weights
for layer in vgg.layers:
layer.trainable = False

#add own classifier
x =  Flatten()(vgg.output)
# and a fully connected output/classification layer
```

```
predictions = Dense(40, activation='softmax')(x)
# create the full network so we can train on it
pre_model = Model(inputs=vgg.input, outputs=predictions)
for i, layer in enumerate(pre_model.layers):
print(i, layer.name, layer.trainable)

pre_model.summary()



# compile the model (should be done *after* setting layers to non-trainable)
pre_model.compile(tf.keras.optimizers.Adamax( learning_rate=0.001, beta_1=0.9, b
eta_2=0.999),loss='sparse_categorical_crossentropy', metrics=['accuracy'])

pretrained_model = pre_model.fit(X_train, y_train,epochs=10, batch_size=32,shuff
le = True, verbose = 1, validation_data = (X_test, y_test))

#Evaluate the model
plt.plot(pretrained_model.history['accuracy'], label='accuracy')
plt.plot(pretrained_model.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')

plt.savefig('/content/drive/My Drive/accvgglast.jpg')
test_loss, test_acc = pre_model.evaluate(X_test,  y_test, verbose=2)

#confusion_matrix
y_pred= pre_model.predict(X_test)
predClass = np.argmax(y_pred , 1)  #return index of largest value 1 for work on
second dimantion

import itertools
matrix = tf.math.confusion_matrix(y_test, predClass)

print(matrix)

plt.figure(figsize = (15,15))

plt.imshow(matrix, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('pre-trained Confusion matrix', y=1.1)
plt.colorbar()
tick_marks = np.arange(len(category_list))
plt.xticks(tick_marks, category_list, rotation=90)
```

```
plt.yticks(tick_marks, category_list)


for i, j in itertools.product(tick_marks, tick_marks):
    plt.text(j, i, format(matrix[i, j]), horizontalalignment="center",color="bla
ck")


plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.tight_layout()
plt.savefig('/content/drive/My Drive/matrixvgglast.jpg')
plt.show()
```