

intelligent sensing in robotics

Assignment 2

Dr. Sameer Kishore

Course code – PDE 4433

Table of Contents

[01Introduction 2](#_Toc133540588)

[02Approach 2](#_Toc133540589)

[03Gathering Information & Execution 4](#_Toc133540590)

[05Limitations/ Evaluation and Conclusion 5](#_Toc133540591)

[06Appendix 5](#_Toc133540592)

# *01*Introduction

Our objective for PDE4434 Assignment 2 is to demonstrate conceptual and practical understanding of intelligent sensing and decision making by implementing an uno card detection system that can determine the shape and color of the card thrown on deck and according to that make a game play decision.

The system needs to be able to understand the rules of the game and play different cards at hand depending on the cards that are present on the floor and depending on how these cards tweak the game in different directions. It should be able to abide by the game rules and complete the game depending on the situation or the move of the previous players. For this coursework we tried to implement a version of the system that can learn and detect the card.

To begin with, the uno game has a total of 108 cards on its deck and the rules of the game are that each player gets 7 cards at the beginning of the game and there’s a deck where players can take cards. The objective of the game is for players to finish all their cards. The challenges are that some cards in the game have instructions that make other players slow down. There are cards that reverse the play and there are cards that change the color of the cards needed on the floor. The game starts with each player having 7 cards in hand and each player must play once during their turn. If the player doesn’t have the same color, number as the ones on the floor, the player is forced to take another card from the deck which increases the number of cards in hand and slows them down. The cards are a collection of different colors (red, green, yellow, blue) and then there are wildcards in black that can be played anytime. These cards if played against the robot will either force the robot to skip his turn, take up to 4 extra cards from the deck or play another color.

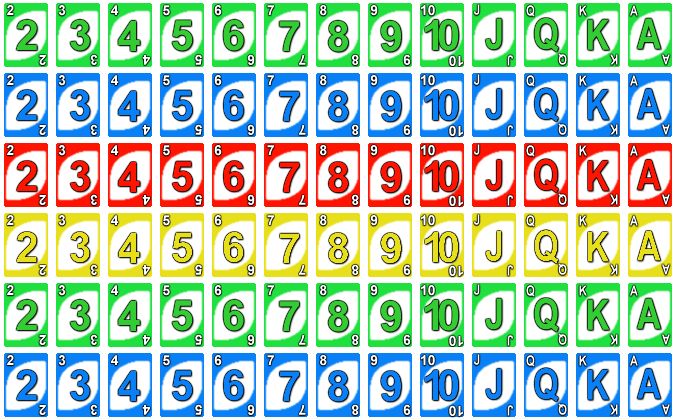
# *02*Approach

The first step of creating this is to create datasets for the robot. We need the robot to understand all the inputs of the game so that it can be ready to release an output depending on the input (card shown on the floor) that are played by the opponent.

This is where our datasets come in. We will create different datasets for the robot to be able to understand all the different cards of the game with their different colors.

For each number a dataset will be created so that the robot understands the type of card released by the opponent and the options they have in their hand.

The next step would be wildcards. The wildcards will also be fed separately in the same manner and this way the system now knows all the cards of the game. The wildcards that have colors (red, green, blue, yellow) will be trained in the same respective dataset created for each number color. Then a separate dataset for the black wild cards will be created and those will be fed in due to their ability to tweak the rules of the game with respect to numbers and colors. For this dataset we will use neural networks because of its efficiency with detecting pictures based on color and shape. Figure 1.1 shows how the uno cards are designed. For this demonstration, we have only focused on detecting the cards with one color.

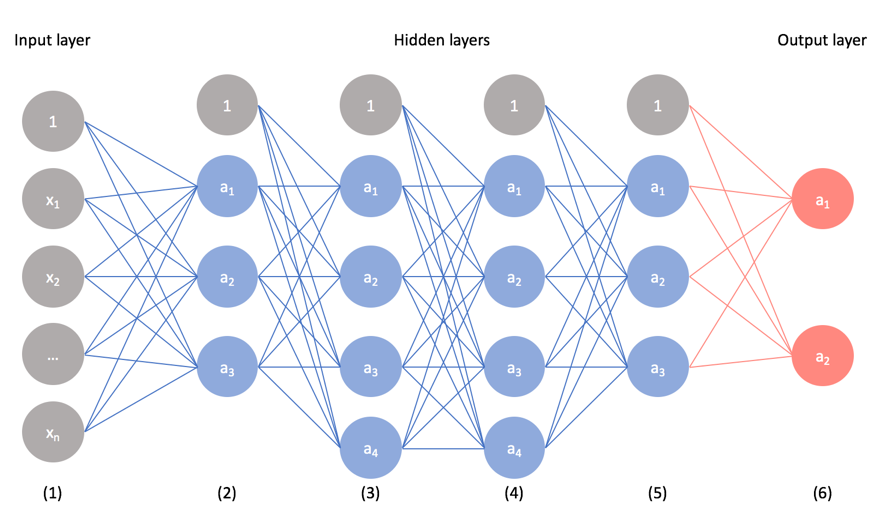
A picture containing shape

Description automatically generated

Research showed Convolutional Neural Networks to be very efficient with Image Detection. The idea of convolutional networks is that it uses the same way the brain uses for the learning process which makes it one of the most effective ways of machine learning when referring to image detection.

Due to some limitations with hardware (Mac m1 silicon) we were unable to carry out our tests using CNN as some of the well-known libraries for image detection (TensorFlow) are not compatible. For testing we used K-neighbor Classifier to learn images from the dataset and use it to predict other images. If the testing using K-neighbor classifier was successful, it is safe to say that the model could work theoretically.

According to research Convolutional Neural Networks has shown the most promising results when dealing with image recognition. Theoretically it uses an input layer to take the input and then uses the hidden layers to compare images together and refine its results in the output layer. One of the libraries most used for image detection is TensorFlow which is based on convolutional neural networks.

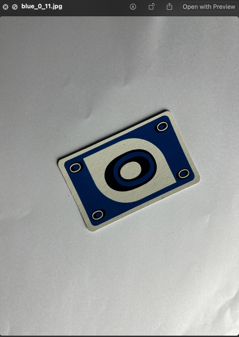
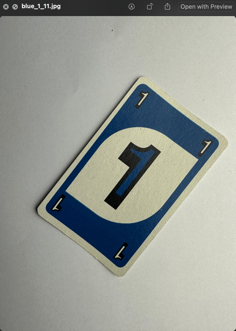


For conceptual testing, we used K-neighbor classifier to detect the color of the image. There was a hardware limitation with the system during implementation, since TensorFlow doesn’t work on mac m1 silicon chips.

# *03*Gathering Information & Execution

The first step for creating a machine learning system is to create a dataset that can be used by the system to learn and then be used to predict live traffic lights. The wider the dataset the more the learning accuracy would be. The dataset was captured using an iPhone with a white background to try to eliminate any noise from the image. We also used a light stand above the image to ensure that we take the image in all form of lighting to increase the learning accuracy of the system.

The images were in the form of labelled images of each uno card from different sides and different lighting conditions. Below is an example of some of the images captured for the dataset.



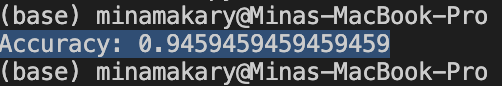
The next step would be determining an appropriate machine learning technique that we can train and test so that we can evaluate our result. In this case, k-neighbor classifier was used to evaluate learning accuracy and result.

We used K-neighbor classifier to train our dataset initially. We trained our models using the dataset by splitting into training and testing sets. We applied some image thresholding techniques like blurring, grayscale along with canny edge detection technique. These image thresholding techniques will later determine the shape of the card as well so the outcome would be to predict the card ex: “five red”.

In this project we used Python with Visual studio code and used libraries from OpenCV to open the camera and use it for reading the images from the camera (using an external camera as well for trial).

First, we set up our deployment environment and installed necessary KNN libraries and open cv for displaying and analyzing image.

When the model was trained, the testing result on the K-neighbor Classifier was nearly 0.95 with detecting cards. Theoretically, the model was able to identify red, green, and yellow under different circumstances.



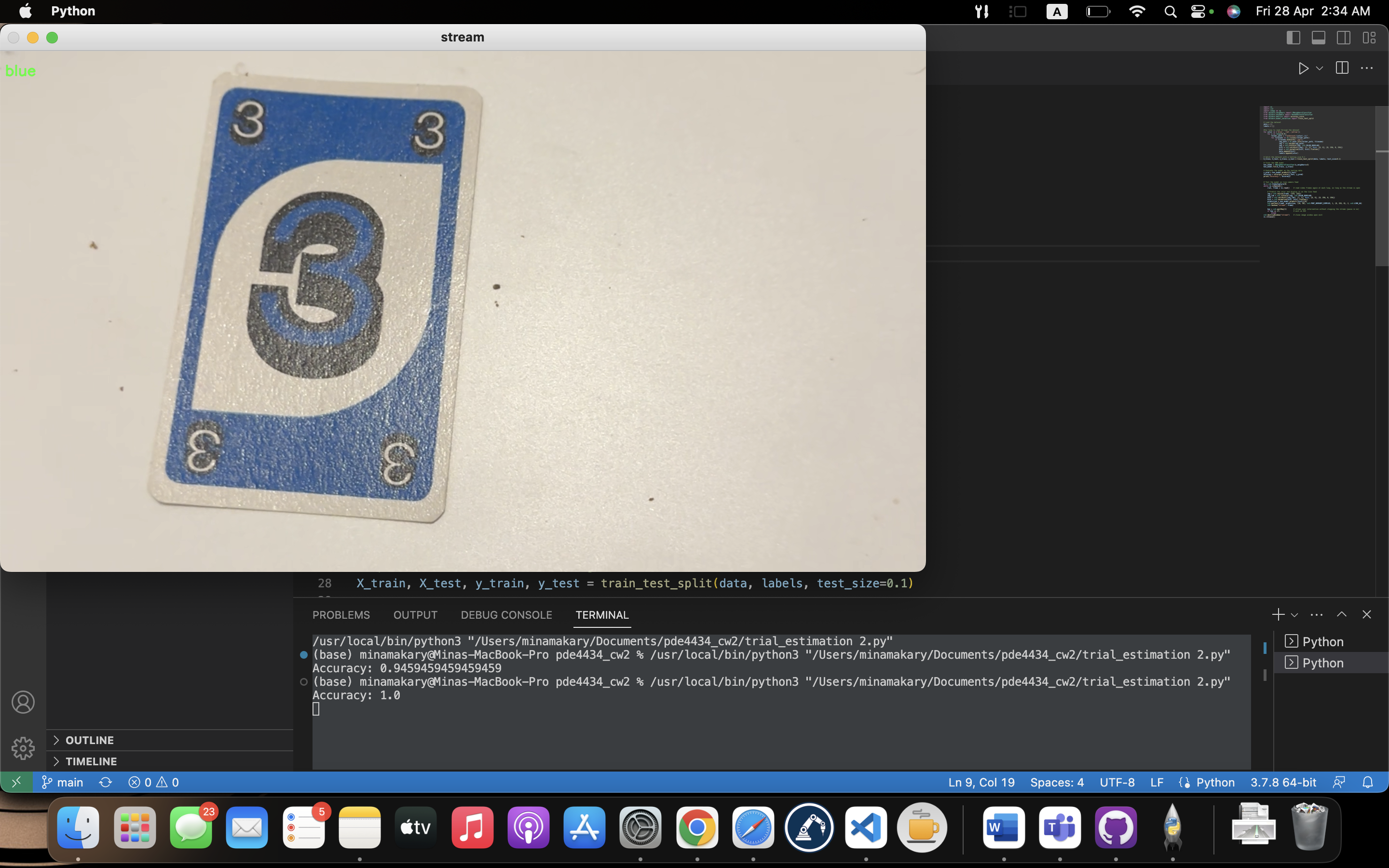
The script was later modified to take a live feed from the camera and detect images as soon as they are displayed. The script runs a continuous live feed from the camera (iPhone camera for accuracy) and once a card is displayed the system predicts the color.

# *05*Limitations/ Evaluation and Conclusion

In conclusion, the implementation of the detection system was a success. The KNN model was mostly successful in accurately defining the color of an uno card when the camera is working and when a card is displayed. The K-neighbor classifier technique did show us a bit of error rate which can be worked on by increasing and refining the dataset. When we created the dataset, we made sure that the images are all in the same format JPG, as one of the limitations faced during execution was having jpeg and PNG files not read correctly by visual studio code.

Deployment would involve trying the system on real world simulation application.

A limitation met is that CNN requires a lot of computational power as it has the highest capabilities so far with regards to image detection.



# 

# *06*Appendix

*Github link*

<https://github.com/minaamakary/pde4434_cw2.git>

*Code (colordetect.py)*

import os

import cv2

import numpy as np

from sklearn.neighbors import KNeighborsClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score

from sklearn.model\_selection import train\_test\_split

# Load the dataset

data = []

labels = []

for color in ['blue', 'red','yellow']:

for i in range(0,9):

folder\_path = f"myDataset/{color}\_{i}"

for filename in os.listdir(folder\_path):

if filename.endswith(".jpg"):

img\_path = os.path.join(folder\_path, filename)

img = cv2.imread(img\_path)

img = cv2.cvtColor(img, cv2.COLOR\_BGR2LAB)

hist = cv2.calcHist([img], [1, 2], None, [8, 8], [0, 256, 0, 256])

hist = cv2.normalize(hist, hist).flatten()

data.append(hist)

labels.append(color)

# Split the dataset

X\_train, X\_test, y\_train, y\_test = train\_test\_split(data, labels, test\_size=0.1)

# Train the KNN model

knn\_model = KNeighborsClassifier(n\_neighbors=3)

knn\_model.fit(X\_train, y\_train)

# Evaluate the model on the testing data

y\_pred = knn\_model.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

print("Accuracy:", accuracy)

# Test the model on live camera feed

vc = cv2.VideoCapture(0)

while vc.isOpened():

rval, frame = vc.read() # read video frames again at each loop, as long as the stream is open

# Predict the color and display it on the live feed

img = cv2.resize(frame, (224, 224))

img\_lab = cv2.cvtColor(img, cv2.COLOR\_BGR2LAB)

hist = cv2.calcHist([img\_lab], [1, 2], None, [8, 8], [0, 256, 0, 256])

hist = cv2.normalize(hist, hist).flatten()

prediction = knn\_model.predict([hist])[0]

cv2.putText(frame, prediction, (10, 50), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (0, 255, 0), 2, cv2.LINE\_AA)

cv2.imshow("stream", frame)

key = cv2.waitKey(1) # allows user intervention without stopping the stream (pause in ms)

if key == 27: # exit on ESC

break

cv2.destroyWindow("stream") # close image window upon exit

vc.release()