



Grades Auto-Filler

TEAM #9

Name		Work division
Adham Ali Abdelaal	Sec:1 BN:12	<ol style="list-style-type: none"> 1. Detect check mark 2. Bubble sheet
Mohamed Waleed Fathy	Sec:2 BN:20	<ol style="list-style-type: none"> 1. Image Orientation Fixing 2. Detect question mark 3. Classifier (Handwritten Digits)
Menna Allah Hossam Eldin	Sec:2 BN:27	<ol style="list-style-type: none"> 1. Fix orientation on Bubble sheet 2. Classifier (IDs for bubble sheet and Grade Auto filler)
Eslam Ashraf Ibrahim	Sec:1 BN:13	<ol style="list-style-type: none"> 1. Cutting cells from table 2. detect square, vertical lines, horizontal lines, empty cell and (-) 3. detect digits and id by OCR

Used Algorithms:

- **Image Orientation Fixing:**

Applied Canny Edge Detector to get the edges in the pictures

Then, applied dilation to connect some disconnected edges.

Used OpenCV's Find contours to get the contours, sorted them, then returned the second largest contour in Area.

Calculated Max Height and Width of that contour; to calculate new corners coordinates to use it in perspective transform.

Fed both the old and new coordinates to OpenCV's warpPerspective, to get the image with vertical orientation.



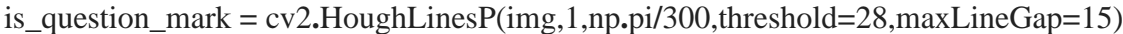
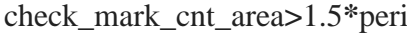
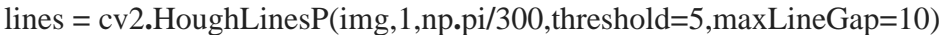
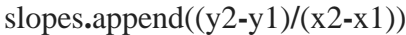


- **Getting cells from table**

To get cells I get horizontal lines in image and vertical lines and make cv2.bitwiseand to get intersection between them and I loop on intersections and get cells after sorting intersections (top to bottom left to right) and cv2.boundingRect to get x,y,w,h and cut cell Using cv2.adaptiveTheshold to make good threshold

- **Detect Symbols**

In detecting we convert image to gray scale and make thresholding to detect symbols using Thorold

- To detect square shape, I get horizontal lines and vertical lines by cv2.HoughLinesP make bitwise if there are 4 intersections, I said true otherwise false after erosion and dilation to make noise less
- To detect vertical lines, I search if there are any horizontal lines by cv2.HoughLinesP in image if not, I get contours and number of contours is number of vertical lines after erosion and dilation to make noise less
- To detect horizontal lines, I search if there are any vertical lines in image if not I get contours and number of contours is number of horizontal lines by cv2.HoughLinesP if one line I get width if less than image/2 it will be (-) otherwise horizontal line after erosion and dilation to make noise less

- To detect empty cell check if there are no contours in image after erosion and dilation to make noise less
- Used two features, together, they are distinctive for Question Marks: Including Circles of neither too small nor too large radii lengths. Perimeter length of the Contour. Used Hough Transform -> OpenCV's HoughCircles to get the results. And arcLength to get the Perimeter.
- **Check mark**
 1. Clear borders of the image because some borders forms false contours.

`img=img&border_mask`
 2. Filter check mark from horizontal and vertical lines by count the number of the contours in the image (check mark will have at most 2 contours but horizontal and vertical lines will have more than 2)

`cnts,_=cv2.findContours(img, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)`
`if(len(cnts)>2):`
`return False`
 3. Filter check mark from question mark by check that if there is exists at least one line with length 28.

`is_question_mark = cv2.HoughLinesP(img,1,np.pi/300,threshold=28,maxLineGap=15)`
 4. Filter check mark from squares by checking if there is at least one closed contours in the image so the symbol will not be check mark

`check_mark_cnt_area>1.5*peri`
 5. Get all lines in the image with minimum length 5

`lines = cv2.HoughLinesP(img,1,np.pi/300,threshold=5,maxLineGap=10)`
 6. Calculate all lines slopes and store it into list

`slopes.append((y2-y1)/(x2-x1))`
 7. If there are negatives and positives slopes
 - a. Calculate the median of each sign to ignore outliers
 - b. Add the medians and apply threshold to validate the check mark

`diff_ang>30 and diff_ang<100`
 8. If the slopes have the same sign
 - a. Calculate the max and the min of slopes
 - b. Subtract the values and apply threshold to validate the check mark

`diff_ang>80 and diff_ang<150`

- **Classification**

Used 4 classifiers:

1. Support Vector Classification (SVC)
2. K-Nearest Neighbor (KNN)
3. Random Forest (RF)
4. Logistic Regression (LR)

The choice of the result depended on a weighted voting system, where a weight was given to each digit according to how accurately the 4 classifiers predicted it. Weights were tuned; to finally choose a value that best matches the actual test data

- **Bubble sheet**

1. Extract the paper from the image.

2. Convert the RGB image to gray scale image using

a. `cv2.cvtColor(paper,cv2.COLOR_BGR2GRAY)`

3. Convert the image to binary image using local thresholding using

a. `cv2.adaptiveThreshold(gray_scale_paper,255,cv2.ADAPTIVE_THRESH_MEAN_C,cv2.THRESH_BINARY_INV, 73, 5)`

4. Erode the image to get rid of false chosen bubbles using

a. `cv2.erode(thresholded,np.ones((3,3)),iterations=1)`

5. Detect the external contours using

a. `cv2.findContours(thresholded, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)`

6. Calculate the bounding rectangle of the contours and get its coordinates, width and height and calculate aspect ratio

a. `(x,y,w,h)=cv2.boundingRect(cnt)`

7. Calculate the contour perimeter with

a. `cv2.arcLength(cnt, True)`

8. Approximate each contour to polygon using

a. `cv2.approxPolyDP(cnt, ratio*peri, True)`

9. Calculate each contour area using

a. `cv2.contourArea(cnt)`

10. Filter the contours so if the aspect ratio $\in [0.8, 1.2]$, the number of contour vertices is greater than 4, the contour area is greater than 30 (threshold to ignore small circles or contours) and the contour is a closed one then

a. It will be considered as a bubble

b. `if(aspect_ratio>=0.8 and aspect_ratio<=1.2 and len(approx)>=4 and curr_cnt_area>30 and curr_cnt_area>1.5*peri)`

11. Collect all contours areas in array and calculate the average area (the mean of areas should be very close to the bubbles area because the most frequent contours are the bubbles contours).

a. `areas.append(curr_cnt_area)`

b. `pre_question_cnts.append(cnt)`

12. Another filter to the output contours that resulted from the first filter by check if the area of each contour is within 30% error with the average contour area

a. `abs(area-bubble_area)<=bubble_area*.3`

13. Sort the contours from top to bottom

a. `question_cnts, _ = imcnts.sort_contours(pre_question_cnts, method='top-to-bottom')`

14. Calculate the number of choices and the number of question in each row by using x and y coordinates of each bubble.

15. Iterate over each row and sort the contours from left to right.

a. `curr_row_cnts, _ = imcnts.sort_contours(question_cnts[i:i+number_of_conts])`

16. Iterate over each question and calculate the number of pixels in each bubble that equal to one.

a. `mask = np.zeros(thresholded.shape, dtype="uint8")`

b. `cv2.drawContours(mask, [c], -1, 255, -1)`

```
c. mask= cv2.bitwise_and(eroded, mask)
```

```
d. cv2.countNonZero(mask)
```

17. Check if the student select more than one choice or no choice the answer will be X but if the student select only one choice so the answer will be the character of the choice (A, B, C, etc.).

```
a. if(total>0.7):
```

```
    i. count_chosen=((total-bubbles)<=0.3).sum()
```

```
b. else:
```

```
    i. count_chosen=((total-bubbles)<=0.15).sum()
```

```
c. final_ans=(ord('X')-ord('A')) if(count_chosen!=1) else bubbled
```

18. Calculate the number of rows and map the result array to the real data.

```
a. final_answers[j*number_of_rows+i]=answers[curr_cnt]
```

Experiment results and analysis:

Cutting cells:

In cutting cells first, I used cv2.findcontours to get cells but it doesn't have a good accuracy that detect wrong contours, so I must cut the cells by hand not by function cv2.findcontours to make sure that I cutting right cells and first I sort them left to right top to bottom I change sort and make it top to bottom left to right so I detect id then digit then symbol it doesn't work correctly if lines of table isn't clear enough

Detect Symbols:

1. Question Marks

Used two features, together, they are distinctive for Question Marks:

Including Circles of neither too small nor too large radii lengths.

Perimeter length of the Contour.

Including Circles Excluded Lines, Checkmarks, Empty cells, as well as most Squares.

Some Squares can have a circular corner when hand-written. To further exclude them, we set a restriction that the contour area should be more than one and a half of its perimeters.

Disconnected Squares persisted since their area was estimated to be zero.

So, we set a restriction on perimeters of the contours, square perimeters were found to be around twice these of the question marks.

Thus, both features hence yielded good results.

2. Square shape

I get horizontal lines and vertical lines make bitwise if there are 4 intersections, I said true otherwise false after erosion and dilation to make noise less but it doesn't work perfect if square has small area so I can't detect it or square sides have obtuse angle so I can't detect shape

3. Horizontal lines && vertical lines && (-)

First, I get contours and make erosion and dilation to delete noise and make image gray and make threshold to detect contours and in detecting horizontal lines we check there aren't vertical lines and in detecting vertical lines we check there aren't horizontal lines. If one horizontal line I get width if less than image/2 it will be (-) otherwise horizontal line

4. Empty Cell

First, I get contours and make erosion and dilation to delete noise and make image gray and make threshold to detect contours and if there aren't any contours it will be detected Empty cell

5. Check mark

Classification:

A Dataset of 42,000 labeled pictures of handwritten digits [size: 28*28] was obtained.

Hog Features were extracted from the images, initially using 2*2 cells, but later changed to 4*4 cells for more immunity to noise.

Our initial approach was to train one model (SVC) and test it. After being trained on 80% of the data, it yielded 99% accuracy when tested with the remaining 20%.

The model was tested on the actual test images, yielded 65% accuracy at best, and a bit less than 47.2% accuracy on more extreme digit shapes.

So, we trained another model (KNN) — with accuracy 98.12% against the dataset— to be compared to SVC. Yielded better results overall, but SVC was still more accurate at one or two digits, 7 for instance.

That's when we thought of training more models to be compared together. So, Random Forest (RF) and Logistic Regression (LR) Classifiers were used for that purpose.

Results were better overall but not coming from one of them, yet collaborative prediction was better.

So, we thought of making use of the outcome of the 4 classifiers for overall better results.

First, tried to formulate a classifier-specific weight for each digit, but analyzing the value counts from the initial testing of each classifier yielded nothing fruitful.

Then, we thought of assigning absolute weights for digits.

We analyzed the results from several pictures to develop a weighted voting system. Initialized weights and tested on the rest of the test data.

That approach yielded the most accurate results from the 4 classifiers together.

Weaknesses:

- Handwritten Digits:
When the 4 classifiers all yield a wrong output when the digit orientation is close to other digits, our voting system won't be of use.
- ID Detection:
We were able to handle connected digits to work in most of the cases, but a few cases still fail.
Digit 6 in ID Detection [Printed Digits, not handwritten] fails to be detected by the 4 classifiers.
- Detect square
If square has small area may be not detect as square or if sides aren't straight enough
- OCR doesn't work great in handwritten digits like printed digit

- if the bubbles connected to each other the testcase will fail
- if the bubbles aspect ratio is not close to 1 the test case will fail

Strengths:

- Digits and ID Detection:

The voting system applied in both modules allowed us to achieve good analytical results unreachable by any single classifier out of the 4 we used.

Connected printed digits in ID Detection were perfectly handled in many cases but failed with a few though.

- Signs Detection:

Question Marks were handled with minimal conflicts with other signs using the two aforementioned features.

Results:

Grades sheet has good results in image 1,5,6,7,8,9,10,11,12,13,14,15

And not good at 2,3,4

Bubble sheet has good results in folder TestCases

And not good in folder Failed Test Cases

Repo in gitHub

[Grades-autofiller](#)

Libraries

Libraries	functions
OpenCV	<ol style="list-style-type: none">1. findContours2. ContourArea3. ArcLength4. approxPolyDP5. cvtColor6. dilate7. Canny8. GaussianBlur9. bilateralFilter10. getPerspectiveTransform11. warpPerspective12. HOGDescriptor13. Resize14. boundingRect15. adaptiveThreshold
CSV	<ol style="list-style-type: none">1. reader
Pickle	<ol style="list-style-type: none">1. laod2. punp
Imutils	<ol style="list-style-type: none">1. contours
sklearn	<ol style="list-style-type: none">1. svm.SVC2. neighbors.KNeighborsClassifier3. ensemble.RandomForestClassifier4. model_selection.train_test_split5. linear_model.LogisticRegression
statistics	<ol style="list-style-type: none">1. median
skimage	<ol style="list-style-type: none">1. thin2. skeletonsize
OS	<ol style="list-style-type: none">1. isdir
shutils	<ol style="list-style-type: none">1. rmtree

Accuracy:

SVM: 99.0%

KNN: 98.24%

RF: 98.1%

LR: 98.4%

References:

Histogram of Gradients:

<https://learnopencv.com/histogram-of-oriented-gradients/>

SVC:

<https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html>

KNN:

<https://scikit-learn.org/stable/modules/neighbors.html#nearest-neighbors-classification>

Random Forest:

<https://scikit-learn.org/stable/modules/ensemble.html#random-forest-parameters>

<https://scikit-learn.org/stable/modules/ensemble.html#random-forests>

getting cells:

<https://medium.com/coinmonks/a-box-detection-algorithm-for-any-image-containing-boxes-756c15d7ed26>