Project presentation

Image analysis and pattern recognition

Catinca Mujdei, Erick Maraz (Group 17)

May 29, 2020

1. Extracting the frames from the video

1. Extracting the frames from the video

1. create list of all extracted frames called `frames`

1. Extracting the frames from the video

1. create list of all extracted frames called `frames`
2. binarize the first frame: `first_frame_binarized`

1. Extracting the frames from the video

1. create list of all extracted frames called `frames`
2. binarize the first frame: `first_frame_binarized`
3. track the arrow:

1. Extracting the frames from the video

1. create list of all extracted frames called `frames`
2. binarize the first frame: `first_frame_binarized`
3. track the arrow:
    1. normalize intensity of all frames

1. Extracting the frames from the video

1. create list of all extracted frames called `frames`
2. binarize the first frame: `first_frame_binarized`
3. track the arrow:
   1. normalize intensity of all frames
   2. apply red mask on each normalized frame

1. Extracting the frames from the video

1. create list of all extracted frames called `frames`
2. binarize the first frame: `first_frame_binarized`
3. track the arrow:
   1. normalize intensity of all frames
   2. apply red mask on each normalized frame
   3. compute location of the arrow in each frame, defined as the center of the red mask

1. Extracting the frames from the video

1. create list of all extracted frames called `frames`
2. binarize the first frame: `first_frame_binarized`
3. track the arrow:
    i. normalize intensity of all frames
    ii. apply red mask on each normalized frame
    iii. compute location of the arrow in each frame, defined as the center of the red mask
    iv. store consecutive locations in list `arrow_locations`:

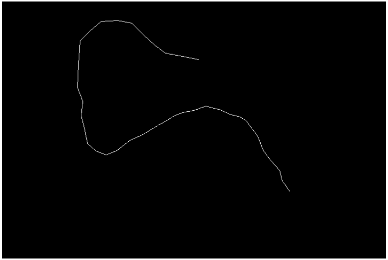


Figure: Arrow path

2. Window sliding

2. Window sliding

1. main concept: slide over `first_frame_binarized` with square window, step size 2, and reject windows that...

2. Window sliding

1. main concept: slide over `first_frame_binarized` with square window, step size 2, and reject windows that...
   1. ...are not fully contained within image

2. Window sliding

1. main concept: slide over `first_frame_binarized` with square window, step size 2, and reject windows that...
   1. ...are not fully contained within image
   2. ...contain too few binary values of one kind

2. Window sliding

1. main concept: slide over `first_frame_binarized` with square window, step size 2, and reject windows that...
   1. ...are not fully contained within image
   2. ...contain too few binary values of one kind
   3. ...contain white border pixels

2. Window sliding

1. main concept: slide over `first_frame_binarized` with square window, step size 2, and reject windows that...
   1. ...are not fully contained within image
   2. ...contain too few binary values of one kind
   3. ...contain white border pixels
2. problem tackling: we do not a priori know the size of the characters

2. Window sliding

1. main concept: slide over `first_frame_binarized` with square window, step size 2, and reject windows that...
   1. ...are not fully contained within image
   2. ...contain too few binary values of one kind
   3. ...contain white border pixels
2. problem tackling: we do not a priori know the size of the characters
   1. start the above concept with large window size (e.g. 80)

2. Window sliding

1. main concept: slide over `first_frame_binarized` with square window, step size 2, and reject windows that...
   1. ...are not fully contained within image
   2. ...contain too few binary values of one kind
   3. ...contain white border pixels
2. problem tackling: we do not a priori know the size of the characters
   1. start the above concept with large window size (e.g. 80)
   2. iteratively decrease window side length (by e.g. 8 pixels), until the window sliding no longer yields any valid windows, and store the minimum window side length that yields valid windows

2. Window sliding

1. main concept: slide over `first_frame_binarized` with square window, step size 2, and reject windows that...
   1. ...are not fully contained within image
   2. ...contain too few binary values of one kind
   3. ...contain white border pixels
2. problem tackling: we do not a priori know the size of the characters
   1. start the above concept with large window size (e.g. 80)
   2. iteratively decrease window side length (by e.g. 8 pixels), until the window sliding no longer yields any valid windows, and store the minimum window side length that yields valid windows
   3. from this minimum window side length, repeat the iterative approach in the previous point in reverse: in each round, remove windows that overlap with previously added windows

2. Window sliding

1. main concept: slide over `first_frame_binarized` with square window, step size 2, and reject windows that...
   1. ...are not fully contained within image
   2. ...contain too few binary values of one kind
   3. ...contain white border pixels
2. problem tackling: we do not a priori know the size of the characters
   1. start the above concept with large window size (e.g. 80)
   2. iteratively decrease window side length (by e.g. 8 pixels), until the window sliding no longer yields any valid windows, and store the minimum window side length that yields valid windows
   3. from this minimum window side length, repeat the iterative approach in the previous point in reverse: in each round, remove windows that overlap with previously added windows
   4. note: to avoid arrow pieces being selected, consider the same window in the first and last frame and check that they are very similar

2. Window sliding

1. main concept: slide over `first_frame_binarized` with square window, step size 2, and reject windows that...
   1. ...are not fully contained within image
   2. ...contain too few binary values of one kind
   3. ...contain white border pixels
2. problem tackling: we do not a priori know the size of the characters
   1. start the above concept with large window size (e.g. 80)
   2. iteratively decrease window side length (by e.g. 8 pixels), until the window sliding no longer yields any valid windows, and store the minimum window side length that yields valid windows
   3. from this minimum window side length, repeat the iterative approach in the previous point in reverse: in each round, remove windows that overlap with previously added windows
   4. note: to avoid arrow pieces being selected, consider the same window in the first and last frame and check that they are very similar
3. with the obtained windows, create list of dictionaries that have two keys: `image_box` and `center` (mean of white pixels)

3. Character classification

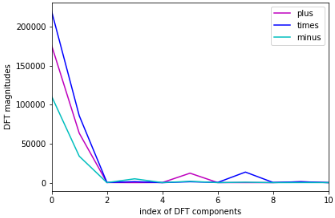
3. Character classification

two different functions that each take as input a selected window

3. Character classification

two different functions that each take as input a selected window

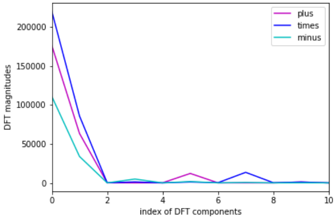
- `classify_operator`: based on number of contours and amplitude of Fourier descriptors



3. Character classification

two different functions that each take as input a selected window

- `classify_operator`: based on number of contours and amplitude of Fourier descriptors

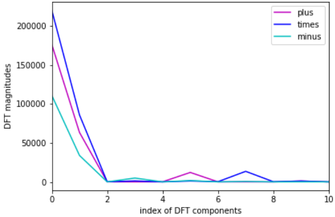


- `classify_number`: CNN trained on augmented MNIST

3. Character classification

two different functions that each take as input a selected window

- `classify_operator`: based on number of contours and amplitude of Fourier descriptors



- `classify_number`: CNN trained on augmented MNIST
  - problems to tackle: digits are hand-written, have varying contours, can be rotated, scaled, have a non-centered sliding window

3. Character classification: CNNs

3. Character classification: CNNs

- advantages

3. Character classification: CNNs

- advantages
  - shift invariance

3. Character classification: CNNs

- advantages
  - shift invariance
  - scaling invariance: max-pooling

3. Character classification: CNNs

- advantages
  - shift invariance
  - scaling invariance: max-pooling
- problems

3. Character classification: CNNs

- advantages
  - shift invariance
  - scaling invariance: max-pooling
- problems
  - no invariance to rotation

3. Character classification: CNNs

- advantages
  - shift invariance
  - scaling invariance: max-pooling
- problems
  - no invariance to rotation
  - tuning of hyperparameters

3. Character classification: CNNs

- advantages
  - shift invariance
  - scaling invariance: max-pooling
- problems
  - no invariance to rotation
  - tuning of hyperparameters
- solutions

3. Character classification: CNNs

- advantages
  - shift invariance
  - scaling invariance: max-pooling
- problems
  - no invariance to rotation
  - tuning of hyperparameters
- solutions
  - make it deep

3. Character classification: CNNs

- advantages
  - shift invariance
  - scaling invariance: max-pooling
- problems
  - no invariance to rotation
  - tuning of hyperparameters
- solutions
  - make it deep
  - data augmentation: rotation - resizing - translation

3. Character classification: CNNs

- advantages
  - shift invariance
  - scaling invariance: max-pooling
- problems
  - no invariance to rotation
  - tuning of hyperparameters
- solutions
  - make it deep
  - data augmentation: rotation - resizing - translation
  - add a little bit of self-written data

3. Character classification: CNNs

- advantages
  - shift invariance
  - scaling invariance: max-pooling
- problems
  - no invariance to rotation
  - tuning of hyperparameters
- solutions
  - make it deep
  - data augmentation: rotation - resizing - translation
  - add a little bit of self-written data

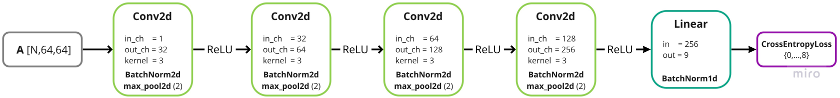


Figure: Architecture of the CNN

4. Tracing the formula and re-making the video

4. Tracing the formula and re-making the video

1. tracing the formula: for loop iterating over original video frames

4. Tracing the formula and re-making the video

1. tracing the formula: for loop iterating over original video frames
   1. at each iteration, determine character that is closest to arrow position by comparing centers of respective objects

4. Tracing the formula and re-making the video

1. tracing the formula: for loop iterating over original video frames
   1. at each iteration, determine character that is closest to arrow position by comparing centers of respective objects
   2. reject if closest character is more than 50 pixels away from arrow, or if it is the same as previous character

4. Tracing the formula and re-making the video

1. tracing the formula: for loop iterating over original video frames
   - i. at each iteration, determine character that is closest to arrow position by comparing centers of respective objects
   - ii. reject if closest character is more than 50 pixels away from arrow, or if it is the same as previous character
   - iii. based on location in formula string (i.e. even or odd index), decide whether new character is operator or digit and apply suitable classification function

4. Tracing the formula and re-making the video

1. tracing the formula: for loop iterating over original video frames
   1. at each iteration, determine character that is closest to arrow position by comparing centers of respective objects
   2. reject if closest character is more than 50 pixels away from arrow, or if it is the same as previous character
   3. based on location in formula string (i.e. even or odd index), decide whether new character is operator or digit and apply suitable classification function
2. evaluate formula

4. Tracing the formula and re-making the video

1. tracing the formula: for loop iterating over original video frames
   1. at each iteration, determine character that is closest to arrow position by comparing centers of respective objects
   2. reject if closest character is more than 50 pixels away from arrow, or if it is the same as previous character
   3. based on location in formula string (i.e. even or odd index), decide whether new character is operator or digit and apply suitable classification function
2. evaluate formula
3. re-making the video: for loop iterating over original video frames

4. Tracing the formula and re-making the video

1. tracing the formula: for loop iterating over original video frames
   1. at each iteration, determine character that is closest to arrow position by comparing centers of respective objects
   2. reject if closest character is more than 50 pixels away from arrow, or if it is the same as previous character
   3. based on location in formula string (i.e. even or odd index), decide whether new character is operator or digit and apply suitable classification function
2. evaluate formula
3. re-making the video: for loop iterating over original video frames
   1. at each iteration, draw dots on all past locations of arrow and draw a line between those points

4. Tracing the formula and re-making the video

1. tracing the formula: for loop iterating over original video frames
   1. at each iteration, determine character that is closest to arrow position by comparing centers of respective objects
   2. reject if closest character is more than 50 pixels away from arrow, or if it is the same as previous character
   3. based on location in formula string (i.e. even or odd index), decide whether new character is operator or digit and apply suitable classification function
2. evaluate formula
3. re-making the video: for loop iterating over original video frames
   1. at each iteration, draw dots on all past locations of arrow and draw a line between those points
   2. at each iteration, write current state of formula