# Machine Learning and Face Recognition

Mads Bøttger Hansen

Department of Chemistry
Aarhus University

Machine Learning Study Group, November 13th 2017

AARHUS UNIVERSITY

# Outline



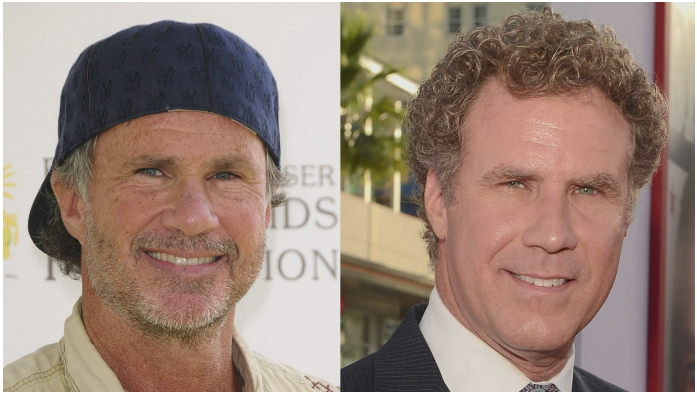
AARHUS UNIVERSITY

# Outline



AARHUS UNIVERSITY

# Telling Chad Smith and Will Ferrell apart



Based on Adam Geitgey's Machine Learning blog (see *Further Reading*).

AARHUS UNIVERSITY

# Courtesy of our Evil Overlords

AARHUS UNIVERSITY

# Courtesy of our Evil Overlords

AARHUS UNIVERSITY

# Courtesy of our Evil Overlords

FaceNet (Schroff et al., 2015)

DeepFace (Taigman et al., 2014)

AARHUS UNIVERSITY

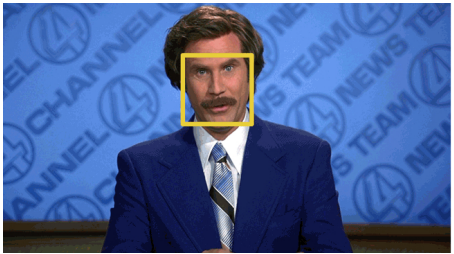
# Face Recognition Is a Complex Process

1. Face detection: e.g. histogram of oriented gradients (HOG), convolutional neural networks (CNN).
2. Input preparation: e.g. linear transformation or 3D-warping.
3. *'Embedding'* from (conventional/fully-connected) neural network (NN).
4. Recognition using simple classification algorithm: e.g. support vector machines (SVM).


AARHUS UNIVERSITY

# Face Recognition Is a Complex Process

1. Face detection: e.g. histogram of oriented gradients (HOG), convolutional neural networks (CNN).
2. Input preparation: e.g. linear transformation or 3D-warping.
3. *'Embedding'* from (conventional/fully-connected) neural network (NN).
4. Recognition using simple classification algorithm: e.g. support vector machines (SVM).

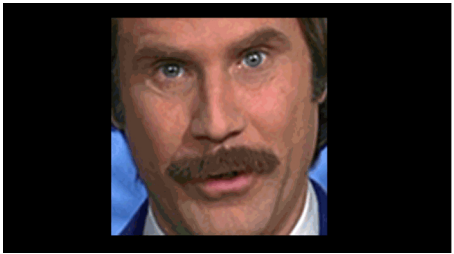


1. Find face in image

AARHUS UNIVERSITY

# Face Recognition Is a Complex Process

1. Face detection: e.g. histogram of oriented gradients (HOG), convolutional neural networks (CNN).
2. Input preparation: e.g. linear transformation or 3D-warping.
3. *'Embedding'* from (conventional/fully-connected) neural network (NN).
4. Recognition using simple classification algorithm: e.g. support vector machines (SVM).



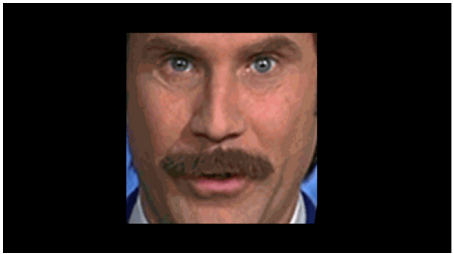
1. Find face in image

2. Analyze facial features

AARHUS UNIVERSITY

# Face Recognition Is a Complex Process

1. Face detection: e.g. histogram of oriented gradients (HOG), convolutional neural networks (CNN).
2. Input preparation: e.g. linear transformation or 3D-warping.
3. *'Embedding'* from (conventional/fully-connected) neural network (NN).
4. Recognition using simple classification algorithm: e.g. support vector machines (SVM).



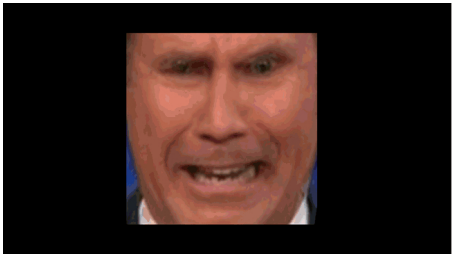
1. Find face in image

2. Analyze facial features

AARHUS UNIVERSITY

# Face Recognition Is a Complex Process

1. Face detection: e.g. histogram of oriented gradients (HOG), convolutional neural networks (CNN).
2. Input preparation: e.g. linear transformation or 3D-warping.
3. *'Embedding'* from (conventional/fully-connected) neural network (NN).
4. Recognition using simple classification algorithm: e.g. support vector machines (SVM).

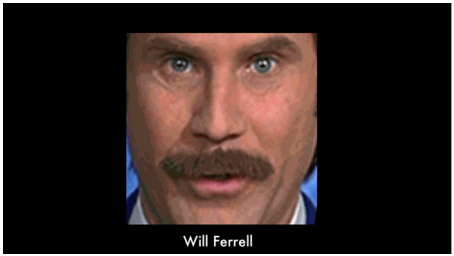


1. Find face in image
2. Analyze facial features
3. Compare against known faces

AARHUS UNIVERSITY

# Face Recognition Is a Complex Process

1. Face detection: e.g. histogram of oriented gradients (HOG), convolutional neural networks (CNN).
2. Input preparation: e.g. linear transformation or 3D-warping.
3. *'Embedding'* from (conventional/fully-connected) neural network (NN).
4. Recognition using simple classification algorithm: e.g. support vector machines (SVM).

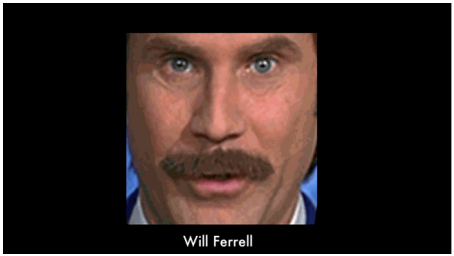



1. Find face in image
2. Analyze facial features
3. Compare against known faces
4. Make a prediction

AARHUS UNIVERSITY

# Face Recognition Is a Complex Process

1. Face detection: e.g. histogram of oriented gradients (HOG), convolutional neural networks (CNN).
2. Input preparation: e.g. linear transformation or 3D-warping.
3. *'Embedding'* from (conventional/fully-connected) neural network (NN).
4. Recognition using simple classification algorithm: e.g. support vector machines (SVM).

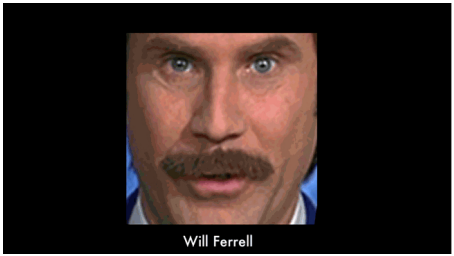



1. Find face in image
2. Analyze facial features
3. Compare against known faces
4. Make a prediction

AARHUS UNIVERSITY

# Face Recognition Is a Complex Process

1. Face detection: e.g. histogram of oriented gradients (HOG), convolutional neural networks (CNN).
2. Input preparation: e.g. linear transformation or 3D-warping.
3. *'Embedding'* from (conventional/fully-connected) neural network (NN).
4. Recognition using simple classification algorithm: e.g. support vector machines (SVM).

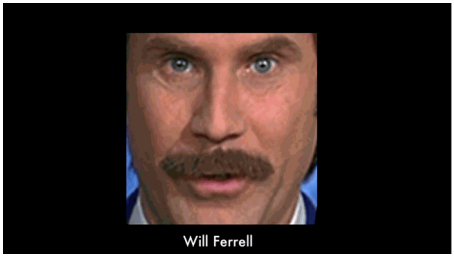



1. Find face in image
2. Analyze facial features
3. Compare against known faces
4. Make a prediction

AARHUS UNIVERSITY

# Face Recognition Is a Complex Process

1. Face detection: e.g. histogram of oriented gradients (HOG), convolutional neural networks (CNN).
2. Input preparation: e.g. linear transformation or 3D-warping.
3. *'Embedding'* from (conventional/fully-connected) neural network (NN).
4. Recognition using simple classification algorithm: e.g. support vector machines (SVM).

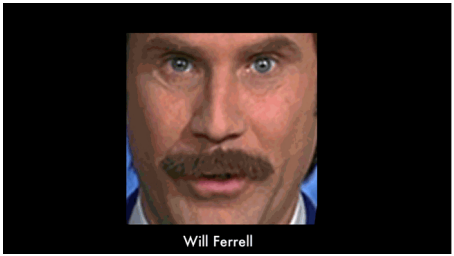



1. Find face in image
2. Analyze facial features
3. Compare against known faces
4. Make a prediction

AARHUS UNIVERSITY

# Face Recognition Is a Complex Process

1. Face detection: e.g. histogram of oriented gradients (HOG), convolutional neural networks (CNN).
2. Input preparation: e.g. linear transformation or 3D-warping.
3. *'Embedding'* from (conventional/fully-connected) neural network (NN).
4. Recognition using simple classification algorithm: e.g. support vector machines (SVM).




AARHUS UNIVERSITY

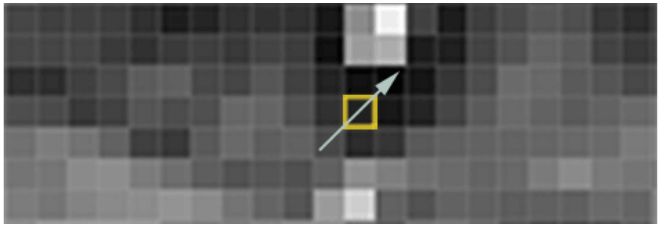
# Step 1, Face Detection: Histogram of Oriented Gradients (HOG)

- B/W: faces depend on shape/contours, not colors.
- Dealing with brightness differences: pixel-wise gradients (HOG). (Now superseded by CNN?)
- Group pixel-wise gradients, only register max. value (pooling).
- Compare HOG version to generic face pattern.

AARHUS UNIVERSITY

# Step 1, Face Detection: Histogram of Oriented Gradients (HOG)

- B/W: faces depend on shape/contours, not colors.
- Dealing with brightness differences: pixel-wise gradients (HOG). (Now superseded by CNN?)
- Group pixel-wise gradients, only register max. value (pooling).
- Compare HOG version to generic face pattern.



AARHUS UNIVERSITY

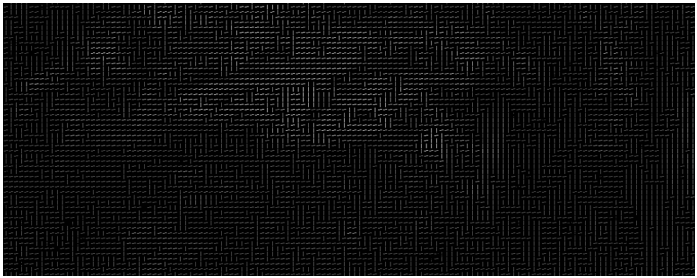
# Step 1, Face Detection: Histogram of Oriented Gradients (HOG)

- B/W: faces depend on shape/contours, not colors.
- Dealing with brightness differences: pixel-wise gradients (HOG). (Now superseded by CNN?)
- Group pixel-wise gradients, only register max. value (pooling).
- Compare HOG version to generic face pattern.

AARHUS UNIVERSITY

# Step 1, Face Detection: Histogram of Oriented Gradients (HOG)

- B/W: faces depend on shape/contours, not colors.
- Dealing with brightness differences: pixel-wise gradients (HOG). (Now superseded by CNN?)
- Group pixel-wise gradients, only register max. value (pooling).
- Compare HOG version to generic face pattern.



AARHUS UNIVERSITY

# Step 1, Face Detection: Histogram of Oriented Gradients (HOG)

- B/W: faces depend on shape/contours, not colors.
- Dealing with brightness differences: pixel-wise gradients (HOG). (Now superseded by CNN?)
- Group pixel-wise gradients, only register max. value (pooling).
- Compare HOG version to generic face pattern.

AARHUS UNIVERSITY

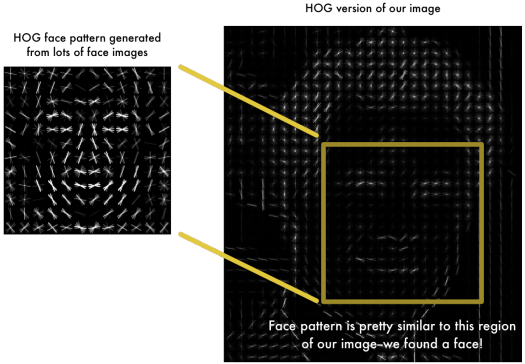
# Step 1, Face Detection: Histogram of Oriented Gradients (HOG)

- B/W: faces depend on shape/contours, not colors.
- Dealing with brightness differences: pixel-wise gradients (HOG). (Now superseded by CNN?)
- Group pixel-wise gradients, only register max. value (pooling).
- Compare HOG version to generic face pattern.

AARHUS UNIVERSITY

# Step 1, Face Detection: Histogram of Oriented Gradients (HOG)

- B/W: faces depend on shape/contours, not colors.
- Dealing with brightness differences: pixel-wise gradients (HOG). (Now superseded by CNN?)
- Group pixel-wise gradients, only register max. value (pooling).
- Compare HOG version to generic face pattern.

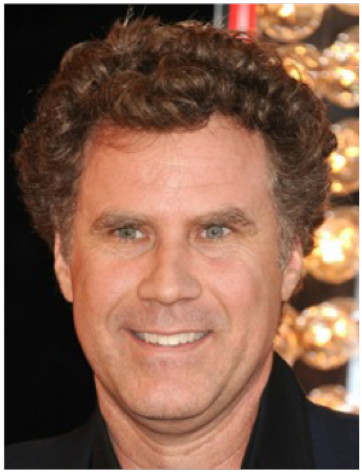


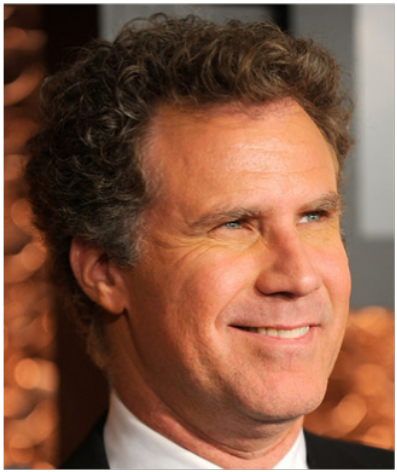

AARHUS UNIVERSITY

# Step 2, Input Preparation: Facial Landmarks

Kazemi, Sullivan (2014)

# Step 2, Input Preparation: Facial Landmarks

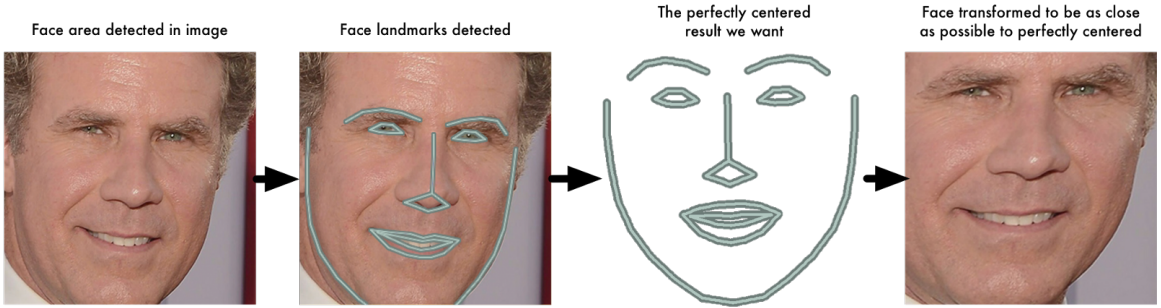




Dealing with different orientations?

AARHUS UNIVERSITY

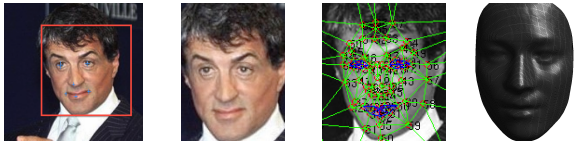
# Step 2, Input Preparation: Facial Landmarks




Linear transformation (preserves parallel lines) (Google's FaceNet?)

AARHUS UNIVERSITY

# Step 2, Input Preparation: Facial Landmarks



(a) (b) (c) (d)

(e) (f) (g) (h)

Facebook's DeepFace: 3D-warp

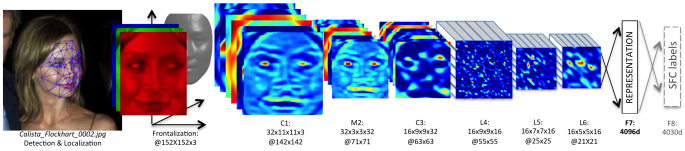
# Step 2, Input Preparation: Facial Landmarks

- Importance: feed suitable input to NN.
- Trade-off: simpler input vs. NN training time.
- Perspective: Behler paper, interatomic distances, symmetrization.

AARHUS UNIVERSITY

# Step 3, Embedding from Neural Network: Google's Triplet Loss Function

- NN: input (pixels, HOG) $\rightarrow$ output (what?)
- Output: 'embedding' = parameter vector ($\in \mathcal{R}^{128}$ for Google's FaceNet)
- If humanly engineered: e.g. eye separation, nose width, etc.
- But machine learned: no concrete physical significance.
- Only important thing: clustering.
- Google: triplet loss function.
- Intensive training this NN! But once trained it can be reused.



# Step 3, Embedding from Neural Network: Google's Triplet Loss Function

- NN: input (pixels, HOG) $\rightarrow$ output (what?)
- Output: 'embedding' = parameter vector ($\in \mathcal{R}^{128}$ for Google's FaceNet)
- If humanly engineered: e.g. eye separation, nose width, etc.
- But machine learned: no concrete physical significance.
- Only important thing: clustering.
- Google: triplet loss function.
- Intensive training this NN! But once trained it can be reused.

Input Image

128 Measurements Generated from Image

| | | | |
|---|---|---|---|
| 0.097496084868908 | 0.045223236083984 | -0.1281466782093 | 0.032084941864014 |
| 0.12529824674129 | 0.060309179127216 | 0.17521631717682 | 0.020976085215807 |
| 0.030809439718723 | -0.019814777253139 | 0.10801389068365 | -0.000521637278451189 |
| 0.036050599068403 | 0.065554238855839 | 0.073130600015444 | -0.13189511100111 |
| -0.097486883401871 | 0.1226262897253 | -0.029626874253154 | -0.005955751053989 |
| -0.0066401711665094 | 0.036753039169292 | -0.15958009660244 | 0.043374512344599 |
| -0.14131525188882 | 0.14114324748516 | -0.031351584941149 | -0.053436127700701 |
| -0.048540540039539 | -0.061901587992907 | -0.15042643249035 | 0.078198105096817 |
| -0.12567175924778 | -0.10568545013666 | -0.12728653848171 | -0.076289616525173 |
| -0.061418771743774 | -0.074287034571171 | -0.065365232572256 | 0.12369407318058 |
| 0.046741496771574 | 0.006176188122481 | 0.14746543765068 | 0.056418422609568 |
| -0.12113650143147 | -0.21055991947651 | 0.004109122790396 | 0.089727647602558 |
| 0.061606746166945 | 0.11345765739679 | 0.021352224051952 | -0.008584329858223 |
| 0.061988940702915 | 0.19372203946114 | -0.086726233363152 | -0.022388197481632 |
| 0.10904195904732 | 0.084853030741215 | 0.094635948353878 | 0.020696049556136 |
| -0.019414527341723 | 0.006481129676106 | 0.21180312335491 | -0.050584398210049 |
| 0.15245945751667 | -0.16582328081131 | -0.035577941885915 | -0.072376452386379 |
| -0.12216668576002 | -0.0072777755558491 | -0.036901291459799 | -0.034365277737379 |
| 0.083934605121613 | -0.059730969369411 | -0.070026844739914 | -0.045013956725597 |
| 0.087945111065905 | 0.11478432267904 | -0.089621491730213 | -0.013955107890069 |
| -0.021407851949334 | 0.14841195940971 | 0.078333757817745 | -0.17898085113387 |
| -0.018298890441656 | 0.049525424838066 | 0.13227833807468 | -0.072600327432156 |
| -0.011014151386917 | -0.051016297191381 | -0.14132921397686 | 0.005051190827528 |
| 0.009367933496832 | -0.062812767922878 | -0.13407498598099 | -0.014829395338893 |
| 0.058139257133007 | 0.004863874055452 | -0.039491076022387 | -0.043765489012003 |
| -0.024210374802351 | -0.11443792283535 | 0.071997955441475 | -0.012062266460602 |
| -0.057223934680223 | 0.014983869667351 | 0.052281547337777 | 0.012774495407939 |
| 0.023535015061498 | -0.081752359867096 | -0.031709920614958 | 0.069833360612392 |
| -0.009803973383324 | 0.037022035568953 | 0.11009479314089 | 0.11638788878918 |
| 0.020220354199409 | 0.12788131833076 | 0.18632389605045 | -0.015367929216059 |
| 0.004033768083902 | -0.094398014247417 | -0.11768248677254 | 0.10281457751989 |
| 0.051597066223621 | -0.10034311562777 | -0.040977258235216 | -0.082041338086128 |

# Step 3, Embedding from Neural Network: Google's Triplet Loss Function

- NN: input (pixels, HOG) $\rightarrow$ output (what?)
- Output: 'embedding' = parameter vector ($\in \mathcal{R}^{128}$ for Google's FaceNet)
- If humanly engineered: e.g. eye separation, nose width, etc.
- But machine learned: no concrete physical significance.
- Only important thing: clustering.
- Google: triplet loss function.
- Intensive training this NN! But once trained it can be reused.

Input Image

128 Measurements Generated from Image

| | | | |
|---|---|---|---|
| 0.097496084868908 | 0.045223236083984 | -0.1281466782093 | 0.032084941864014 |
| 0.12529824674129 | 0.060309179127216 | 0.17521631717682 | 0.020976085215807 |
| 0.030809439718723 | -0.019814777253139 | 0.10801389068365 | -0.00052163278451189 |
| 0.036505099068403 | 0.065554238855839 | 0.073130600015444 | -0.13189511100111 |
| -0.097486883401871 | 0.1226262897253 | -0.029626874253154 | -0.0059557510539889 |
| -0.0066401711665094 | 0.036753039169292 | -0.15958009660244 | 0.043374512344599 |
| -0.14131525188882 | 0.14114324748516 | -0.031351584941149 | -0.053436127700701 |
| -0.048540540039539 | -0.061901587992907 | -0.15042643249035 | 0.078198105096817 |
| -0.12567175924778 | -0.10568545013666 | -0.12728653848171 | -0.076289616525173 |
| -0.061418771743774 | -0.074287034571171 | -0.065365232572256 | 0.12369407318058 |
| 0.046741496771574 | 0.006176188122481 | 0.14746543765068 | 0.056418422609568 |
| -0.12113650143147 | -0.21055991947651 | 0.004109122790396 | 0.089727647602558 |
| 0.061606746166945 | 0.11345765739679 | 0.021522224051952 | -0.0085843298584223 |
| 0.061988940702915 | 0.19372203946114 | -0.086726233363152 | -0.022388197481632 |
| 0.10904195904732 | 0.084853030741215 | 0.094635948353878 | 0.020696049556136 |
| -0.019414527341723 | 0.006481129676106 | 0.21180312335491 | -0.050584398210049 |
| 0.15245945751667 | -0.16582328081131 | -0.035577941685915 | -0.072376452386379 |
| -0.12216668576002 | -0.0072777755558491 | -0.036901291459799 | -0.034365277737379 |
| 0.083934605121613 | -0.059730969369411 | -0.070026844739914 | -0.045013956725597 |
| 0.087945111065905 | 0.11478432267904 | -0.089621491730213 | -0.013955107890069 |
| -0.021407851949334 | 0.14841195940971 | 0.078333757817745 | -0.17898085113387 |
| -0.018298890441656 | 0.049525424838066 | 0.13227833807468 | -0.072600327432156 |
| -0.011014151386917 | -0.051016297191381 | -0.14132921397686 | 0.065051190827528 |
| 0.009367933496828 | -0.062812767922878 | -0.13407498598099 | -0.014829395338893 |
| 0.058139257133007 | 0.004863874055452 | -0.039491076022387 | -0.043765489012003 |
| -0.024210374802351 | -0.11443792283535 | 0.071997955441475 | -0.012062266460602 |
| -0.057223934680223 | 0.014983869667351 | 0.052281547377 | 0.012774495407939 |
| 0.023535015061498 | -0.081752359867096 | -0.031709920614958 | 0.069833360612392 |
| -0.009803973383324 | 0.037022035568953 | 0.11009479314089 | 0.11638788878918 |
| 0.020220354199409 | 0.12788131833076 | 0.18632389605045 | -0.015367929216059 |
| 0.004033768083902 | -0.094398014247417 | -0.11768248677254 | 0.10281457751989 |
| 0.051597066223621 | -0.10034311562777 | -0.040977258235216 | -0.082041338086128 |

# Step 3, Embedding from Neural Network: Google's Triplet Loss Function

- NN: input (pixels, HOG) $\rightarrow$ output (what?)
- Output: 'embedding' = parameter vector ($\in \mathcal{R}^{128}$ for Google's FaceNet)
- If humanly engineered: e.g. eye separation, nose width, etc.
- But machine learned: no concrete physical significance.
- Only important thing: clustering.
- Google: triplet loss function.
- Intensive training this NN! But once trained it can be reused.

Input Image

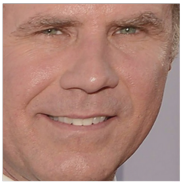
128 Measurements Generated from Image

| | | | |
|---|---|---|---|
| 0.097496084868908 | 0.045223236083984 | -0.1281466782093 | 0.032084941864014 |
| 0.12529824674129 | 0.060309179127216 | 0.17521631717682 | 0.020976085215807 |
| 0.030809439718723 | -0.019814777253139 | 0.10801389068365 | -0.00052163278451189 |
| 0.036505099068403 | 0.065554238855839 | 0.073130600015444 | -0.13189511100111 |
| -0.097486883401871 | 0.1226262897253 | -0.029626874253154 | -0.0059557510539889 |
| -0.0066401711665094 | 0.036753039169292 | -0.15958009660244 | 0.043374512344599 |
| -0.14131525188882 | 0.14114324748516 | -0.031351584941149 | -0.053436127700701 |
| -0.048540540039539 | -0.061901587992907 | -0.15042643249035 | 0.078198105096817 |
| -0.12567175924778 | -0.10568545013666 | -0.12728653848171 | -0.076289616525173 |
| -0.061418771743774 | -0.074287034571171 | -0.065365232572256 | 0.12369407318058 |
| 0.046741496771574 | 0.006176188122481 | 0.14746543765068 | 0.056418422609568 |
| -0.12113650143147 | -0.21055991947651 | 0.0041091227903962 | 0.089727647602558 |
| 0.061606746166945 | 0.11345765739679 | 0.021552224051952 | -0.0085843298584223 |
| 0.061988940702915 | 0.19372203946114 | -0.086726233363152 | -0.022388197481632 |
| 0.10904195904732 | 0.084853030741215 | 0.09463594853878 | 0.020696049556136 |
| -0.019414527341723 | 0.006481129676106 | 0.21180312335491 | -0.050584398210049 |
| 0.15245945751667 | -0.16582328081131 | -0.035577941685915 | -0.072376452386379 |
| -0.12216668576002 | -0.0072777755558491 | -0.036901291459799 | -0.034365277737379 |
| 0.083934605121613 | -0.059730969369411 | -0.070026844739914 | -0.045013956725597 |
| 0.087945111065905 | 0.11478432267904 | -0.089621491730213 | -0.013955107890069 |
| -0.021407851949334 | 0.14841195940971 | 0.078333757817745 | -0.17898085113387 |
| -0.018298890441656 | 0.049525424838066 | 0.13227833807468 | -0.072600327432156 |
| -0.011014151386917 | -0.051016297191381 | -0.14132921397686 | 0.005051190827528 |
| 0.009367933496832 | -0.062812767922878 | -0.13407498598099 | -0.014829395338893 |
| 0.058139257133007 | 0.0048638740554452 | -0.039491076022387 | -0.043765489012003 |
| -0.024210374802351 | -0.11443792283535 | 0.071997955441475 | -0.012062266460602 |
| -0.057223934680223 | 0.014983869667351 | 0.052281547377 | 0.012774495407939 |
| 0.023535015061498 | -0.081752359867096 | -0.031709920614958 | 0.069833360612392 |
| -0.0098039731383324 | 0.037022035568953 | 0.11009479314089 | 0.11638788878918 |
| 0.020220354199409 | 0.12788131833076 | 0.18632389605045 | -0.015367929216059 |
| 0.0040337680839002 | -0.094398014247417 | -0.11768248677254 | 0.10281457751989 |
| 0.051597066223621 | -0.10034311562777 | -0.040977258235216 | -0.082041338086128 |

# Step 3, Embedding from Neural Network: Google's Triplet Loss Function

- NN: input (pixels, HOG) $\rightarrow$ output (what?)
- Output: 'embedding' = parameter vector ($\in \mathcal{R}^{128}$ for Google's FaceNet)
- If humanly engineered: e.g. eye separation, nose width, etc.
- But machine learned: no concrete physical significance.
- Only important thing: clustering.
- Google: triplet loss function.
- Intensive training this NN! But once trained it can be reused.

Input Image



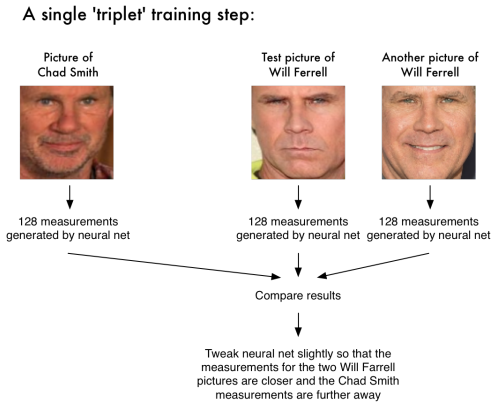
128 Measurements Generated from Image

| | | | |
|---|---|---|---|
| 0.097496084868908 | 0.045223236083984 | -0.1281466782093 | 0.032084941864014 |
| 0.12529824674129 | 0.060309179127216 | 0.17521631717682 | 0.020976085215807 |
| 0.030809439718723 | -0.019814777253139 | 0.10801389068365 | -0.000521637278451189 |
| 0.036050599068403 | 0.065554238855839 | 0.073130600015444 | -0.13189511100111 |
| -0.097486883401871 | 0.122626289723 | -0.029626874253154 | -0.005955751039889 |
| -0.006640171665094 | 0.036753039169292 | -0.15958009660244 | 0.043374512344599 |
| -0.14131525188882 | -0.14114324748516 | -0.031351584941149 | -0.053436127700701 |
| -0.048540540039539 | -0.061901587992907 | -0.15042643249035 | 0.078198105096817 |
| -0.12567175924778 | -0.10568545013666 | -0.12728653848171 | -0.076289616525173 |
| -0.061418771743774 | -0.074287034571171 | -0.065365232572256 | 0.12369407318058 |
| 0.046741496771574 | 0.006176188122481 | 0.14746543765068 | 0.056418422609568 |
| -0.12113650143147 | -0.21055991947651 | 0.004109122790396 | 0.089727647602558 |
| 0.061606746166945 | 0.11345765739679 | 0.021522224051952 | -0.008584329858423 |
| 0.061988940702915 | 0.19372203946114 | -0.086726233363152 | -0.022388197481632 |
| 0.10904195904732 | 0.084853030741215 | 0.094635948353878 | 0.020696049556136 |
| -0.019414527341723 | 0.006481129676036 | 0.21180312335491 | -0.050584398210049 |
| 0.15245945751667 | -0.16582328081131 | -0.035577941685915 | -0.072376452386379 |
| -0.12216668576002 | -0.007277775558491 | -0.036901291459799 | -0.034365277737379 |
| 0.083934605121613 | -0.059730969369411 | -0.070026844739914 | -0.045013956725597 |
| 0.087945111065905 | 0.11478432267904 | -0.089621491730213 | -0.013955107890069 |
| -0.021407851949334 | 0.14841195940971 | 0.078333757817745 | -0.17898085713387 |
| -0.018298890441656 | 0.049525424838066 | 0.13227833807468 | -0.072600327432156 |
| -0.011014151386917 | -0.051016297191381 | -0.14132921397686 | 0.005051190827528 |
| 0.009367933496832 | 0.062812767922878 | -0.13407498598099 | -0.014829395338893 |
| 0.058139257133007 | 0.004863874055452 | -0.039491076022387 | -0.043765489012003 |
| -0.024210374802351 | -0.11443792283535 | 0.071997955441475 | -0.012062266460602 |
| -0.057223934680223 | 0.014983869667351 | 0.052281547377 | 0.012774495407939 |
| 0.023535015061498 | -0.081752359867096 | -0.031709920614958 | 0.069833360612392 |
| -0.009803973383324 | 0.037022035568953 | 0.11009479314089 | 0.11638788878918 |
| 0.020220354199409 | 0.12788131833076 | 0.18632389605045 | -0.015367929216059 |
| 0.004033768083902 | -0.094398014247417 | -0.11768248677254 | 0.10281457751989 |
| 0.051597066223621 | -0.10034311562777 | -0.040977258235216 | -0.082041338086128 |

AARHUS UNIVERSITY

# Step 3, Embedding from Neural Network: Google's Triplet Loss Function

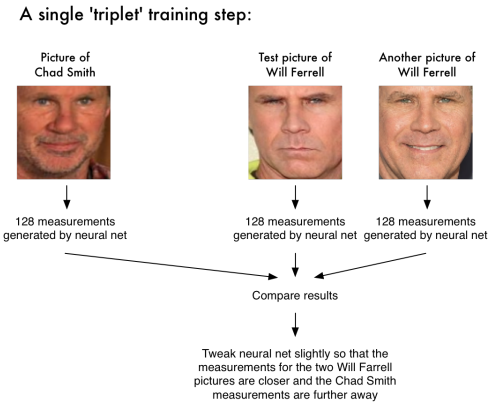
- NN: input (pixels, HOG) $\rightarrow$ output (what?)
- Output: 'embedding' = parameter vector ($\in \mathcal{R}^{128}$ for Google's FaceNet)
- If humanly engineered: e.g. eye separation, nose width, etc.
- But machine learned: no concrete physical significance.
- Only important thing: clustering.
- Google: triplet loss function.
- Intensive training this NN! But once trained it can be reused.


AARHUS UNIVERSITY

# Step 3, Embedding from Neural Network: Google's Triplet Loss Function

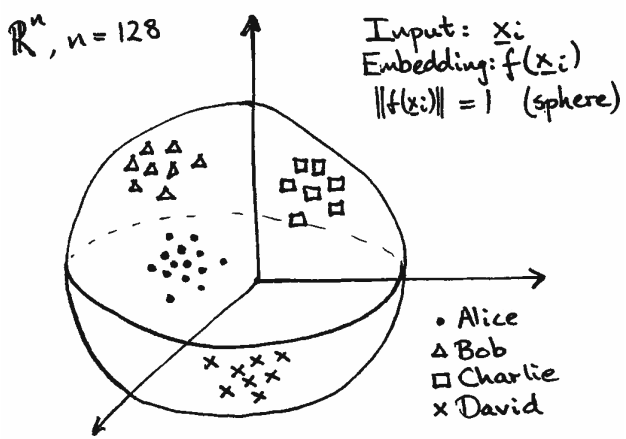
- NN: input (pixels, HOG) $\rightarrow$ output (what?)
- Output: 'embedding' = parameter vector ($\in \mathcal{R}^{128}$ for Google's FaceNet)
- If humanly engineered: e.g. eye separation, nose width, etc.
- But machine learned: no concrete physical significance.
- Only important thing: clustering.
- Google: triplet loss function.
- Intensive training this NN! But once trained it can be reused.


AARHUS UNIVERSITY

# Step 4, Recognition: Classification algorithm

- The NN embeddings:
  - alike for identical persons.
  - distant for different persons.
- Train a ML classification algorithm (such as SVM) with known faces.
- Feed image, generate embedding, classify as known person.




AARHUS UNIVERSITY

# Step 4, Recognition: Classification algorithm

- The NN embeddings:
  - alike for identical persons.
  - distant for different persons.
- Train a ML classification algorithm (such as SVM) with known faces.
- Feed image, generate embedding, classify as known person.


AARHUS UNIVERSITY

# Step 4, Recognition: Classification algorithm

- The NN embeddings:
  - alike for identical persons.
  - distant for different persons.
- Train a ML classification algorithm (such as SVM) with known faces.
- Feed image, generate embedding, classify as known person.

AARHUS UNIVERSITY

# Outline



AARHUS UNIVERSITY

# Convolution Essentials

- In math: $s(t) = \int x(a)w(t-a)da$
  where $w$: weight, $x$: signal
- Discrete: $s(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a)$
- 2D:

$$S(i,j) = \sum_{m,n} I(i+m, j+n)K(m,n)$$

where $I$: image, $K$: kernel [1]

[1] Actually, this formula is called *cross-correlation* while *convolution* has terms like $I(i-m, j-n)K(m,n)$, i.e. the kernel is *flipped* relative to the image. *Convolution* is handier for math proofs, *cross-correlation* is (maybe?) more intuitive to implement and interpret, and in the end the NN doesn't care; it just optimizes the weights in either case. The terms are sometimes used interchangeably by the community. See Goodfellow, ch. 9.

# Convolution Essentials

- In math: $s(t) = \int x(a)w(t-a)da$
  where $w$: weight, $x$: signal
- Discrete: $s(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a)$
- 2D:

$$S(i,j) = \sum_{m,n} I(i+m, j+n)K(m,n)$$

where $I$: image, $K$: kernel [1]

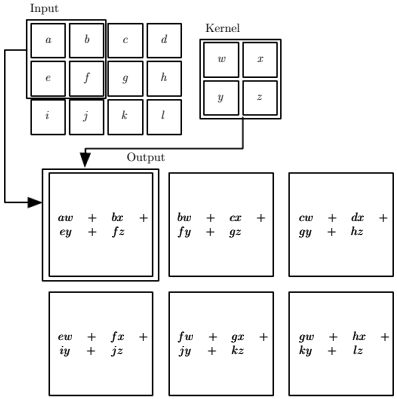
[1] Actually, this formula is called *cross-correlation* while *convolution* has terms like $I(i-m, j-n)K(m,n)$, i.e. the kernel is *flipped* relative to the image. *Convolution* is handier for math proofs, *cross-correlation* is (maybe?) more intuitive to implement and interpret, and in the end the NN doesn't care; it just optimizes the weights in either case. The terms are sometimes used interchangeably by the community. See Goodfellow, ch. 9.

# Convolution Essentials

- In math: $s(t) = \int x(a)w(t-a)da$
  where $w$: weight, $x$: signal
- Discrete: $s(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a)$
- 2D:

$$S(i,j) = \sum_{m,n} I(i+m, j+n)K(m,n)$$

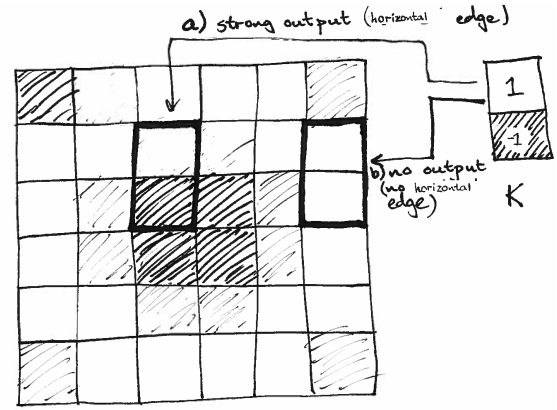
where $I$: image, $K$: kernel [1]




[1] Actually, this formula is called *cross-correlation* while *convolution* has terms like $I(i-m, j-n)K(m,n)$, i.e. the kernel is *flipped* relative to the image. *Convolution* is handier for math proofs, *cross-correlation* is (maybe?) more intuitive to implement and interpret, and in the end the NN doesn't care; it just optimizes the weights in either case. The terms are sometimes used interchangeably by the community. See Goodfellow, ch. 9.

# Convolution as "filter"/feature detector

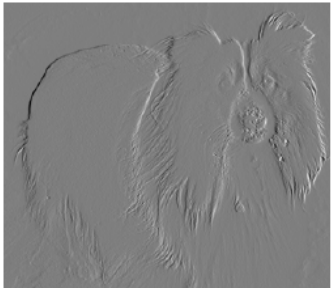
- Convolution extracts features located *anywhere*.
- Edge detection (compare with HOG).
- Humanly engineered: we might look for eye contours, etc.
- ML: just train a CNN to figure out the best kernel(s), i.e. best features to look for.




AARHUS UNIVERSITY

# Convolution as "filter"/feature detector

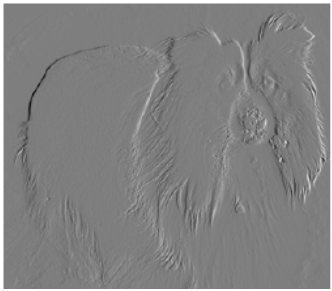
- Convolution extracts features located *anywhere*.
- Edge detection (compare with HOG).
- Humanly engineered: we might look for eye contours, etc.
- ML: just train a CNN to figure out the best kernel(s), i.e. best features to look for.



AARHUS UNIVERSITY

# Convolution as "filter"/feature detector

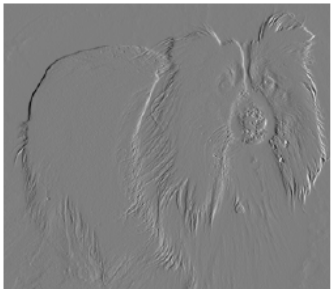
- Convolution extracts features located *anywhere*.
- Edge detection (compare with HOG).
- Humanly engineered: we might look for eye contours, etc.
- ML: just train a CNN to figure out the best kernel(s), i.e. best features to look for.



AARHUS UNIVERSITY

# Convolution as "filter"/feature detector

- Convolution extracts features located *anywhere*.
- Edge detection (compare with HOG).
- Humanly engineered: we might look for eye contours, etc.
- ML: just train a CNN to figure out the best kernel(s), i.e. best features to look for.



AARHUS UNIVERSITY

# Pooling

- A "summary" of statistics of the NN layer output (often used with CNN).
- E.g.: max pooling; just select the max. element in area.
- Robustness towards small, insignificant variations.
- Data reduction (using strides).

AARHUS UNIVERSITY

# Pooling

- A "summary" of statistics of the NN layer output (often used with CNN).
- E.g.: max pooling; just select the max. element in area.
- Robustness towards small, insignificant variations.
- Data reduction (using strides).

AARHUS UNIVERSITY

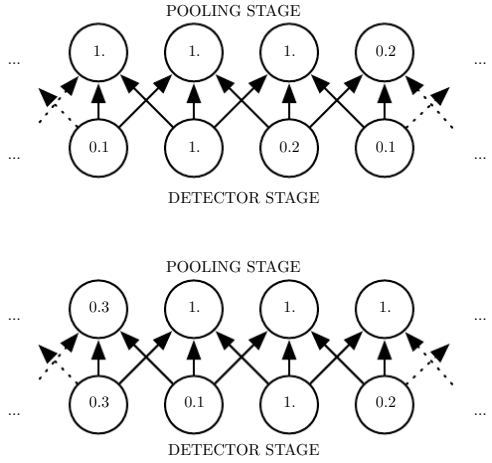
# Pooling

- A "summary" of statistics of the NN layer output (often used with CNN).
- E.g.: max pooling; just select the max. element in area.
- Robustness towards small, insignificant variations.
- Data reduction (using strides).

AARHUS UNIVERSITY

# Pooling

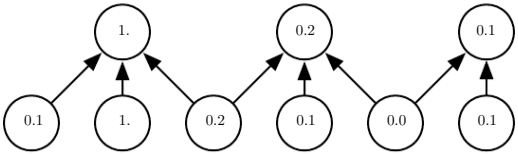
- A "summary" of statistics of the NN layer output (often used with CNN).
- E.g.: max pooling; just select the max. element in area.
- Robustness towards small, insignificant variations.
- Data reduction (using strides).




AARHUS UNIVERSITY

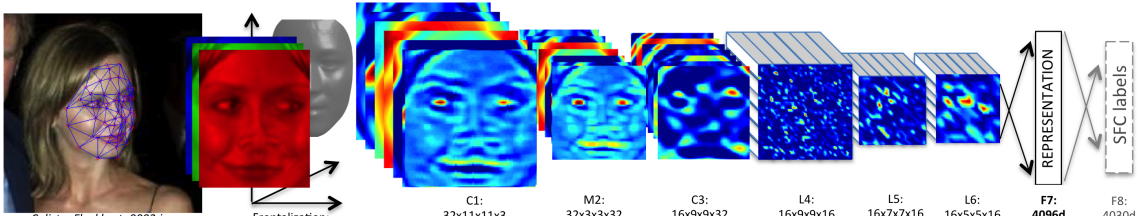
# Pooling

- A "summary" of statistics of the NN layer output (often used with CNN).
- E.g.: max pooling; just select the max. element in area.
- Robustness towards small, insignificant variations.
- Data reduction (using strides).




AARHUS UNIVERSITY

# CNN and pooling: Facebook's DeepFace revisited




[2]"A 3D-aligned 3-channels (RGB) face image of size 152 by 152 pixels is given to a convolutional layer (C1) with 32 filters of size 11x11x3 (we denote this by 32x11x11x3@152x152). The resulting 32 feature maps are then fed to a max-pooling layer (M2) which takes the max over 3x3 spatial neighborhoods with a stride of 2, separately for each channel. This is followed by another convolutional layer (C3) that has 16 filters of size 9x9x16. The purpose of these three layers is to extract low-level features, like simple edges and texture. [...] The subsequent layers (L4, L5 and L6) are instead locally connected, like a convolutional layer they apply a filter bank, but every location in the feature map learns a different set of filters." (MBH: i.e. they extract features specific to that region of the face.) "[...] Finally, the top two layers (F7 and F8) are fully connected: each output unit is connected to all inputs. These layers are able to capture correlations between features captured in distant parts of the face images, e.g., position and shape of eyes and position and shape of mouth."

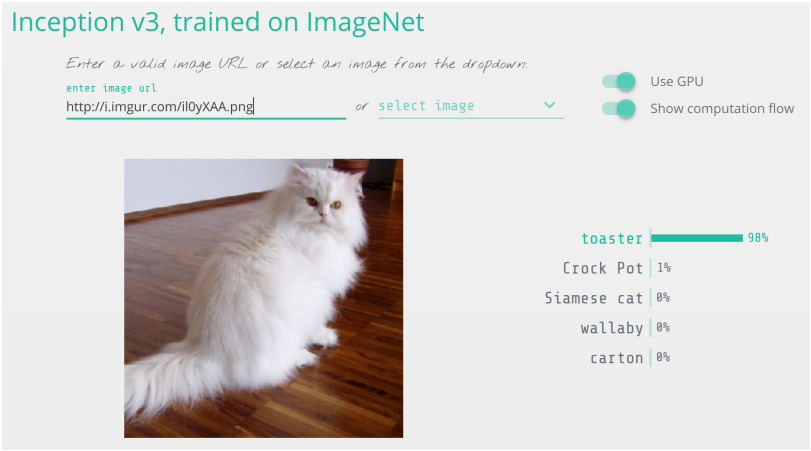
# Outline



AARHUS UNIVERSITY

# How to Hack a Neural Network

Adam Geitgey's block, part 8: Optimize the image, not the NN weights.




AARHUS UNIVERSITY

# Summary/perspectives

- Importance of NN input.
- Convolutional NN/pooling for feature detection.
- High quality, well documented, open source C++ library (dlib) for ML.

AARHUS UNIVERSITY

# Further Reading and Resources I

- Adam Geitgey. *Machine Learning is Fun! Part 4: Modern Face Recognition with Deep Learning* (2016). (Comment: blog that much of this presentation was based upon – the other parts contain good, digestable introductions to many ML aspects as well.)
- Florian Schroff, Dmitry Kalenichenko, James Philbin. *FaceNet: A Unified Embedding for Face Recognition and Clustering* (2015). (Comment: Google's face recognition technique.)
- Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, Lior Wolf. *DeepFace: Closing the Gap to Human-Level Performance in Face Verification* (2014). (Comment: Facebook's face recognition technique.)
- Navneet Dalal, Bill Triggs. *Histograms of Oriented Gradients for Human Detection* (2005).
- Vahid Kazemi, Josephine Sullivan. *One Millisecond Face Alignment with an Ensemble of Regression Trees (2014).* (Comment: About aligning faces using 'facial landmarks'.)

AARHUS UNIVERSITY

# Further Reading and Resources II

- Ian Goodfellow, Yoshua Bengio, Aaron Courville. *Deep Learning (2016).* (Comment: All the essentials, by three of the authorities (from what I understood) in the field. HTML version on site, PDF-version here.)
- *dlib C++ library.* (Comment: (in their own words) high quality, well documented, open source machine learning toolkit. I managed to install it and run a face detection script using it on my Mac more or less out of the box.)

AARHUS UNIVERSITY