Computer Vision Systems Programming VO

Object Category Recognition

Christopher Pramerdorfer

Computer Vision Lab, Vienna University of Technology

Topics

Scene classification using the bag of words model

Fast face detection using boosted Haar features

Convolutional neural networks for large-scale problems

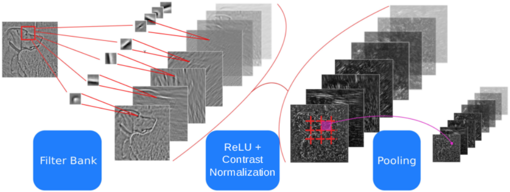


Image adaoted from Kavukcuoglu 2011

We want to distinguish between c scene categories

- So $w \in \{0, \ldots, c-1\}$ (classification problem)

Street Scenes

Sea Scenes

Image adapted from Prince 2012

Scene Classification

Selecting $\mathbf{x}$

We represent an image as a collection of visual words

- Images can be compared based on visual word distribution

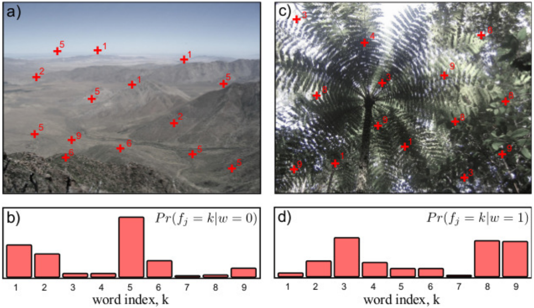
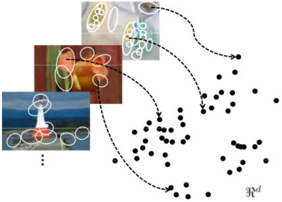


Image from Prince 2012

Scene Classification
Selecting $\mathbf{x}$

Visual words are learned from an image collection

- Compute (SIFT) keypoints and descriptors for all images
- Cluster descriptors into k clusters using k-means
- k cluster means represent visual words



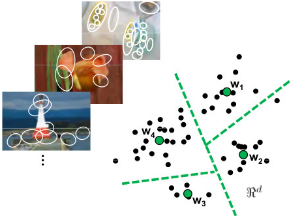


Image from Grauman and Leibe 2011

Visual word distribution $\mathbf{x} \in \mathbb{N}^k$ of image obtained by

- Computing keypoints and descriptors
- Assigning each feature to closest visual word
- Summing up the assignment counts for each visual word

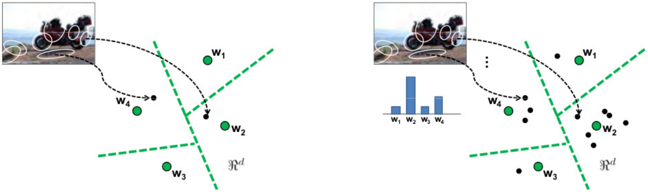


Image from Grauman and Leibe 2011

This image representation is called bag of (visual) words

Now that we have $\mathbf{x}$ we can select and learn a suitable model

- SVMs are often used in the literature
- For a probabilistic alternative see Prince 2012

Many improvements to this model exist

- Better clustering schemes
- Fuzzy assignment to visual words
- Spatial information (constellation model)

Popular and can work well, but no longer state of the art

Scene Classification

Bag of Visual Words Using OpenCV

```
// init SIFT
cv::Ptr<cv::FeatureDetector> kp = cv::FeatureDetector::create("SIFT");
cv::Ptr<cv::DescriptorExtractor> desc = cv::DescriptorExtractor::create("SIFT");

// compute visual words from training data
const int k = 50; // number of visual words
cv::BOWKMeansTrainer trainer(k);

for(const cv::Mat& im : images) { // std::vector of training images
    std::vector<cv::KeyPoint> keypoints; kp->detect(im, keypoints);
    cv::Mat descriptors; desc->compute(im, keypoints, descriptors);
    trainer.add(descriptors);
}

cv::Mat visualWords = trainer.cluster(); // k*128 (SIFT dimension)
```

Scene Classification
Bag of Visual Words Using OpenCV

```
// setup visual word frequency (our x) extractor
cv::Ptr<cv::DescriptorMatcher> fm = cv::makePtr<cv::BFMatcher>(cv::NORM_L2);
cv::BOWImgDescriptorExtractor extractor(desc, fm);
extractor.setVocabulary(visualWords);

// compute x for all training images
cv::Mat xTrain(images.size(), k, CV_32FC1);
for(std::size_t i = 0; i != images.size(); i++) {
    std::vector<cv::KeyPoint> keypoints; kp->detect(images[i], keypoints);
    cv::Mat x; extractor.compute(images[i], keypoints, x);
    xTrain(cv::Rect(0, i, k, 1)) = x;
}

// and corresponding w
cv::Mat wTrain(images.size(), 1, CV_32FC1); // fill me
```

Scene Classification
Bag of Visual Words Using OpenCV

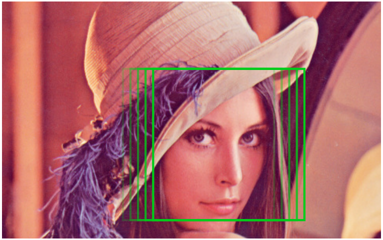
```
// train our model (we use an SVM)
CvSVM svm;
svm.train(xTrain, wTrain);

// now we can predict the class of new images
std::vector<cv::KeyPoint> keypoints; kp->detect(newImage, keypoints);
cv::Mat x; extractor.compute(newImage, keypoints, x);
float w = svm.predict(x); // predicted class label
```

Face Detection

Image from olympus-europa.com

We don't know where the faces are so we

- Slide a fixed-size window over the image
- Compute $\Pr(w|\mathbf{x})$ for each window ($w = 1$ if face, 0 if not)



Face Detection
Selecting $\mathbf{x}$

Which are good features for this task?

Must be fast to compute (many windows)

Must be robust to illumination, so we use gradient information

Different approaches to encoding gradient information

- Compute gradients, pool orientations in blocks (e.g. SIFT)
- Use a collection of Gabor filters

Face Detection
Selecting $\mathbf{x}$

We want something faster

- So we use a "blocky" approximation of Gabor filters
- Difference between rectangular subwindows (Haar features)
- Can be computed in constant time using integral images

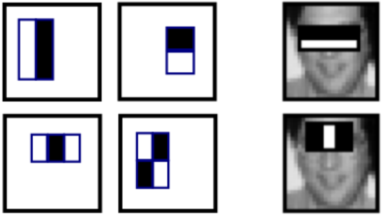


Image adapted from Prince 2012

Face Detection
Selecting $\mathbf{x}$

Computing a Haar feature at a location yields a scalar f_i

We define $\mathbf{x} = (x_1, \ldots, x_I)$ with $x_i = \text{heaviside}(f_i - t_i)$

I is very large

- Different Haar features, subwindow locations, t_i
- We thus learn which features x_i work best

We model $\Pr(w|\mathbf{x})$ as a weighted sum of a feature subset

$$\Pr(w|\mathbf{x}) \propto a = \phi_0 + \sum_k \phi_k x_k \qquad (1 \leq k \leq I)$$

And learn the parameters $\boldsymbol{\theta} = (\phi_0, \phi_k, x_k)$ from training samples

- For each x_k in a large precomputed set
- Find optimal ϕ_0, ϕ_k
- Add best x_k (and ϕ_0, ϕ_k) to sum and repeat

This incremental approach is called boosting

We stop adding features at some point

- If the classification error no longer decreases significantly
- After a specified maximum number of iterations

We end up with K good features, $K \ll I$

- For prediction we compute only these K features
- Stop early if $P(w = 0) > t$ after processing $J \ll K$ features

If we don't care about probabilities, we choose $w = \text{heaviside}(a)$

If we do, we use logistic regression, $\Pr(w|\mathbf{x}) = \text{Bern}_w(\text{sig}(a))$

- We model w as a Bernoulli distribution
- Pass a through a logistic sigmoid to map it to $[0, 1]$
- Called logitboost in this context

Face Detection
Remarks

This method was proposed in Viola and Jones 2001

Very efficient, ideal for e.g. digital cameras

Trades off efficiency for accuracy

- Features capture gradients coarsely, no color information
- More powerful but slower methods exist

Not invariant to scale changes (fixed-size window)

- Repeat detection at different image scales

Detect faces using a pretrained model

OpenCV also supports training

```
# detect faces using a pretrained cascade
image = cv2.imread('faces.jpg', cv2.IMREAD_GRAYSCALE)
cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
faces = cascade.detectMultiScale(image) # should tune parameters
```

Selecting good features $\mathbf{x}$ for object recognition is challenging

- Why we previously learned $\mathbf{x}$

Learned features were low-level

- Based on SIFT descriptors or Haar wavelets

We want task-specific high-level features

- Virtually impossible to design manually
- So we learn them as well

We learn these features hierarchically

- Model consists of layers
- The higher up the layer, the higher-level the feature
- Features in layer n are based on those in layer $n-1$
- Results in a *deep* model, hence deep learning

At the same time we learn to predict $\mathbf{w}$

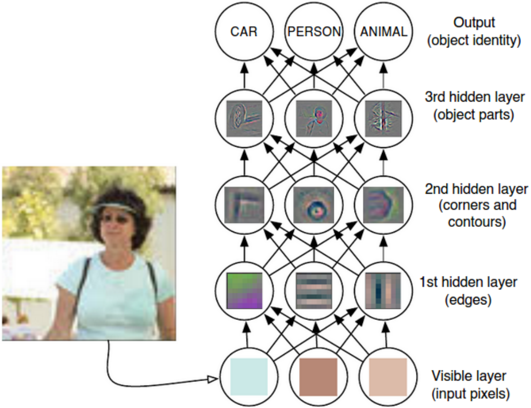


Image from Bengio, Goodfellow, and Courville 2015

Deep model optimized for images

- Exploits correlations between adjacent pixels
- Performs remarkably well in many recognition tasks

"From now on, deep learning has to be considered as the primary candidate in essentially any visual recognition task"

[Razavian et al. 2014]

Multilayer Perceptrons (MLPs) with some twists

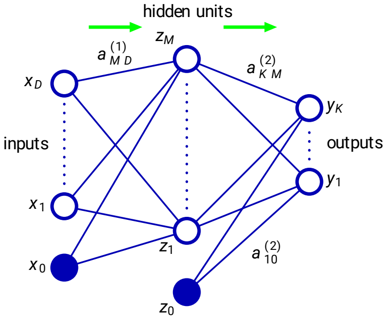


Image adapted from Bishop 2006

Remember the Perceptron?

- Model for binary classification, $w = \text{heaviside}(\mathbf{a}^\top \mathbf{x} + b)$
- Parameters $\boldsymbol{\theta} = (\mathbf{a}, b)$ learned from data

Limitations

- Linear decision boundaries
- Only two classes, not probabilistic
- Learning never converges for non-separable data

To overcome these limitations we

- Replace step function with continuous f (e.g. tanh)
- Add a layer of M such “Perceptrons” (hidden units)
- Add K output units (number of classes)

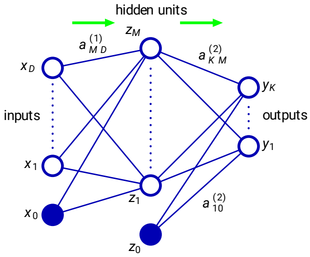


Image adapted from Bishop 2006

Output of mth hidden unit is $z_m(\mathbf{x}) = f(\mathbf{a}_m^\top \mathbf{x})$

- Bias b included in $\mathbf{a}$ and $\mathbf{x}$, $a_0 = b$, $x_0 = 1$

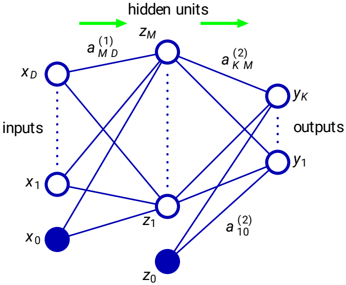


Image from Bishop 2006

Output of kth output unit is $y_k(\mathbf{z}) = g(\mathbf{a}_k^\top \mathbf{z})$

- g depends on problem (regression, classification)

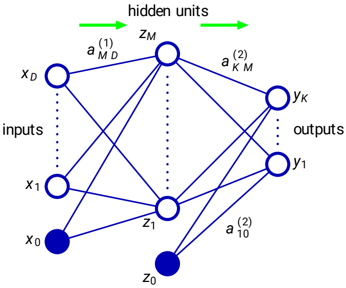


Image from Bishop 2006

Both f and g are differentiable

- Learn parameters using gradient descent
- Gradients evaluated via error backpropagation

Properties

- Powerful, can approximate any decision boundary if M is large
- Does not scale to images due to full connectivity
- Not deep (not possible due to scaling issues)

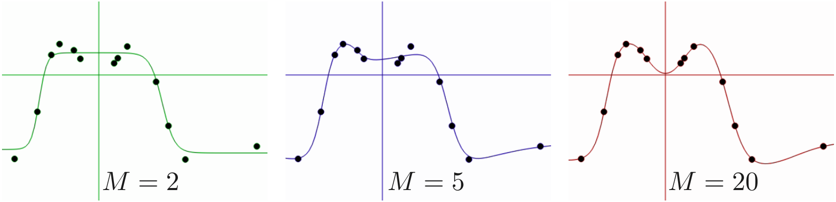
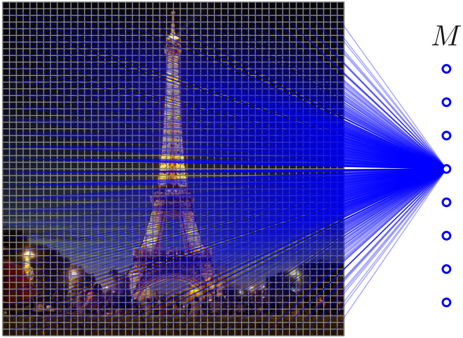


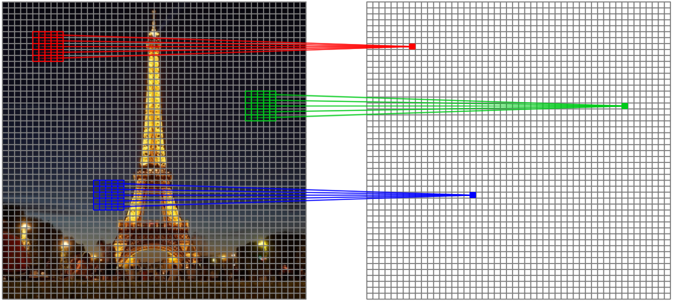
Image adapted from `cs.stanford.edu/people/karpathy/convnetjs/`

QVGA image ($D = 76800$), $M = 500$: ~ 38m parameters



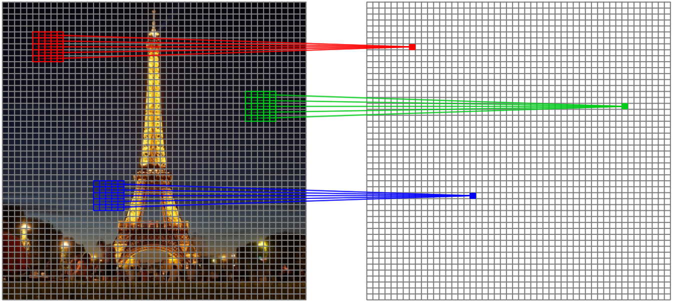
Nearby pixels are closely correlated, the rest is not

- We use D hidden units (one per input pixel)
- Connect each only to nearby pixels (local receptive field)



M much higher but much fewer parameters

- ~ 2m in example with 5×5 receptive field



Learned features should work well everywhere in the image

- We don't know where objects will be

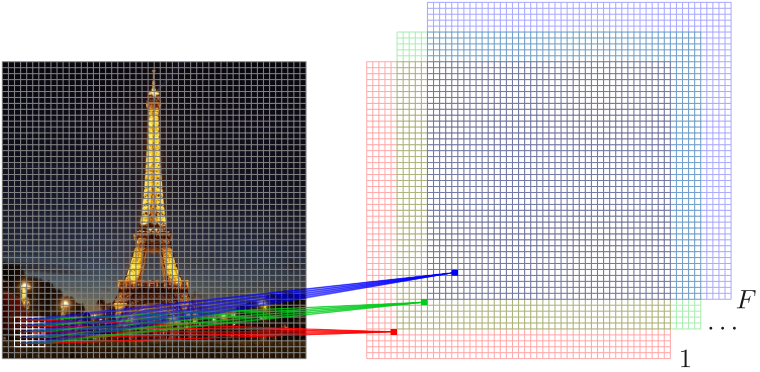
So we define that all hidden units must have *same* weights

- Number of parameters now independent of M
- But we now can learn only a single feature

So we duplicate the hidden layer F times

- Each duplicate (feature map) learns a feature

$F = 100$, 5×5 receptive field : ~ 2600 parameters

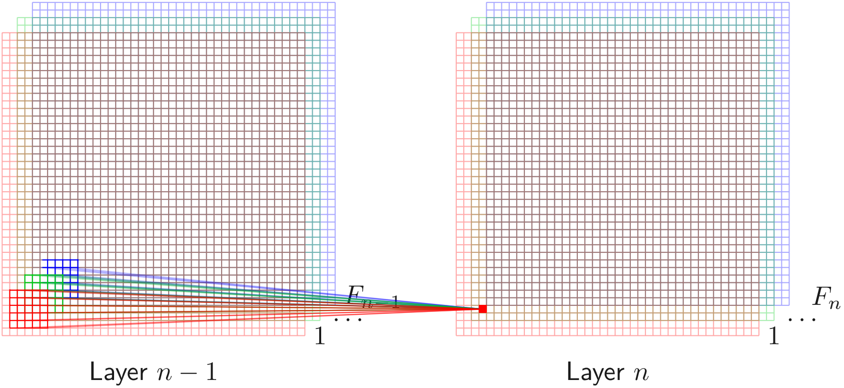


Such layers are called convolutional (conv) layers

- Feature map evaluation equals convolution with kernel $\mathbf{a}$
- Followed by f (often $f(\cdot) = \max(0, \cdot)$ (ReLU))

Few parameters, so we can stack conv layers (deep network)

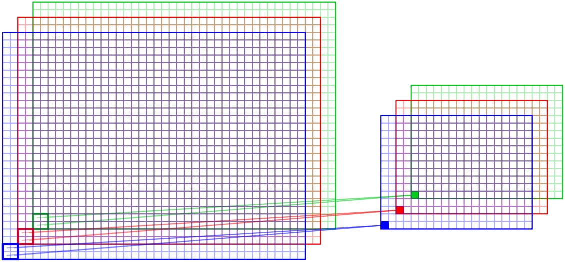
- Conv layer n operates on feature maps in layer $n - 1$
- Learns features by combining those learned in layer $n - 1$



F_{n-1}
1
...
F_n
1
...
Layer $n-1$
Layer n

Most CNNs also contain pooling layers

- Pool (aggregate) information locally (e.g. max, mean)
- Reduces data size, robustness to small object movement



Convolutional Neural Networks (CNNs) consist of

- One or several conv layers (and others)
- Followed by a traditional MLP operating on learned features

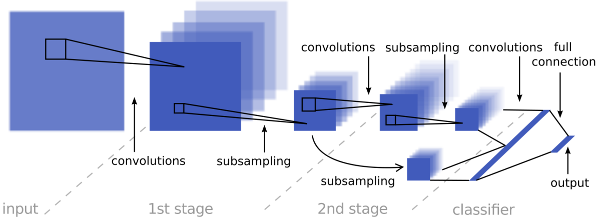


Image adapted from Sermanet et al. 2012

Convolutional Neural Networks Learn High-Level Features

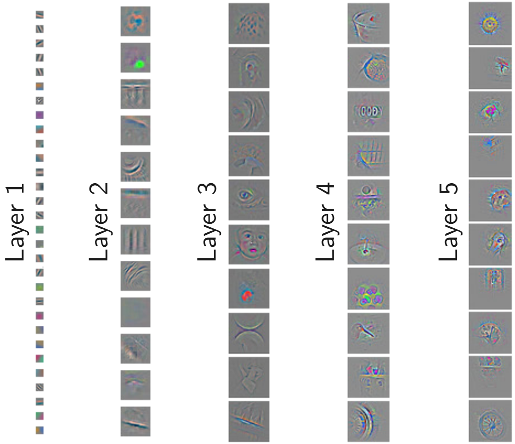


Image adapted from Zeiler and Fergus 2014

CNNs were proposed long ago

- LeCun et al. 1989 : Zip code recognition using a CNN

Large, deep CNNs possible now

- Powerful and flexible GPUs
- Large datasets to avoid overfitting (e.g. ImageNet)

Simple CNN for Zip Code Recognition

- Using ConvNetJS (Java Script library)
- `cs.stanford.edu/people/karpathy/convnetjs/demo/mnist.html`

Image from `convnetjs.com`

Face recognition via 3D face alignment and CNNs

- conv $\Rightarrow$ pooling $\Rightarrow$ conv for low-level features
- Locally connected layers for high-level features
- Human-level face verification performance

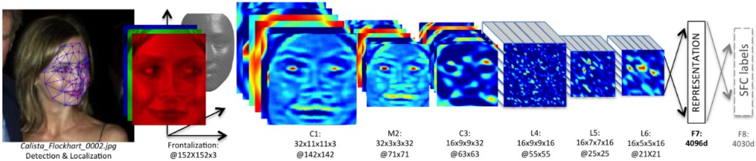


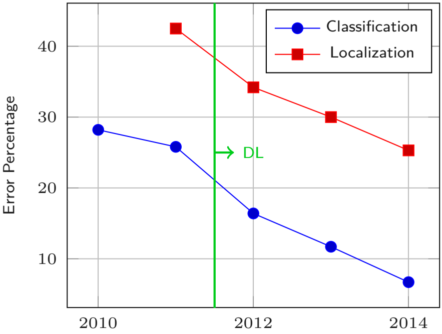
Image from Taigman et al. 2013

Object recognition on the ImageNet database

- ~ 14 million images categorized hierarchically
- Annual object recognition challenges

Image from http://web.eecs.umich.edu/~jiadeng/

CNNs have lead to significant performance gains on ImageNet



Demo of winner of 2013 ImageNet classification challenge

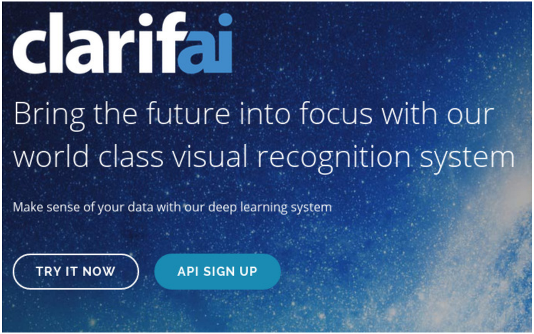
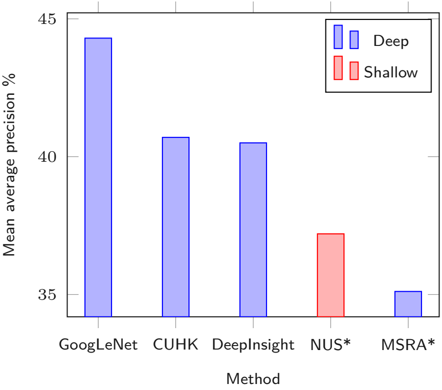


Image from clarifai.com

Results of 2014 ImageNet object detection challenge (excerpt)



Many open-source libraries available

- Caffe (C++, Matlab, Python)
- Keras, Lasagne (Python)
- ConvNetJS (Java Script)
- Torch7 (LUA)

Network definitions and trained networks available online

- E.g. Caffe model zoo

Deep learning and CNNs are powerful concepts

- State of the art in image recognition

Large CNNs have millions of parameters

- Training takes long (up to weeks on GPUs)
- Requires large datasets to avoid overfitting

Bibliography I

Bengio, Yoshua, Ian Goodfellow, and Aaron Courville (2015). *Deep Learning (Draft)*. MIT Press.

Bishop, Christopher (2006). *Pattern recognition and machine learning*. Springer.

Grauman, Kristen and Bastian Leibe (2011). *Visual object recognition*. Morgan & Claypool.

Kavukcuoglu, Koray (2011). *Learning feature hierarchies for object recognition*. PhD thesis.

LeCun, Yann et al. (1989). *Backpropagation applied to handwritten zip code recognition*.

Bibliography II

Prince, S.J.D. (2012). *Computer Vision: Models Learning and Inference*. Cambridge University Press.

Razavian, Ali Sharif et al. (2014). *CNN Features off-the-shelf: an Astounding Baseline for Recognition*.

Sermanet, Pierre et al. (2012). *Pedestrian Detection with Unsupervised Multi-Stage Feature Learning*.

Taigman, Yaniv et al. (2013). *Deepface: Closing the gap to human-level performance in face verification*.

Viola, Paul and Michael Jones (2001). *Rapid object detection using a boosted cascade of simple features*.

Bibliography III

Zeiler, Matthew D and Rob Fergus (2014). *Visualizing and understanding convolutional networks*.