Fashion Classification and Detection Using Convolutional Neural Networks

Prateek Srivastava & Archit Arora

November 10, 2016

Fashion Classification Problem

- Given an image, identify the apparel a particular person is wearing





Applications

- Targeted ads
- Product recommendations
- Identification of suspects through CCTV footage
- Brand adoption trend for clothing companies

Challenges

- Intra-class variation
- Viewpoint and locational variation
- Scale variation
- Background clutter

Challenges

- Intra-class variation
- Viewpoint and locational variation
- Scale variation
- Background clutter

Challenges

- Intra-class variation
- Viewpoint and locational variation
- Scale variation
- Background clutter

Challenges

- Intra-class variation
- Viewpoint and locational variation
- Scale variation
- Background clutter

Overview of fashion classification process

- Data collection
- Data augmentation
- Learning from data
- Image classification & object detection

Data collection

- Used **Scrapy tool** to search and crawl Macy's website
- Generated a labelled data set of around 3500 images belonging to these categories

14 Fashion Categories

Jacket	Long-dress	Polo shirts	Suit	Short-dress
T-shirt	Sweater	Trousers	Shoes	Robe
Shirt	Jeans	Swimsuit	Shorts	

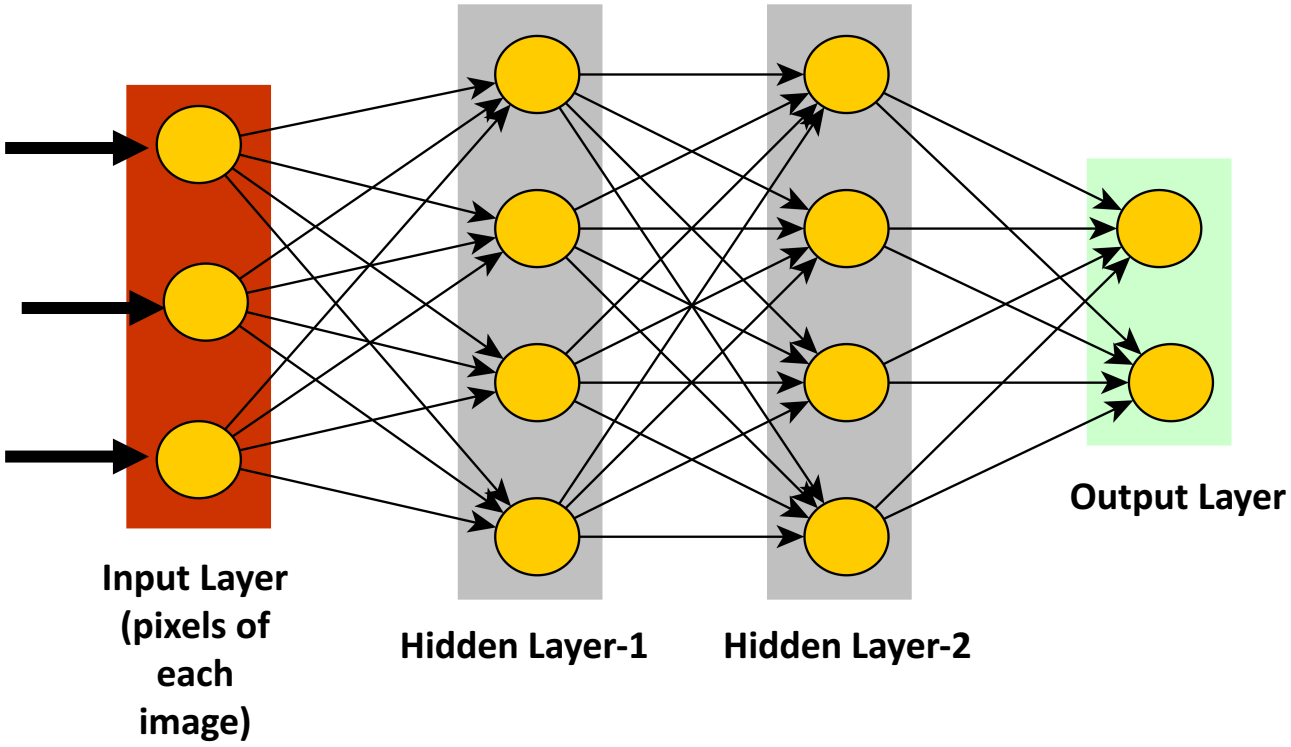




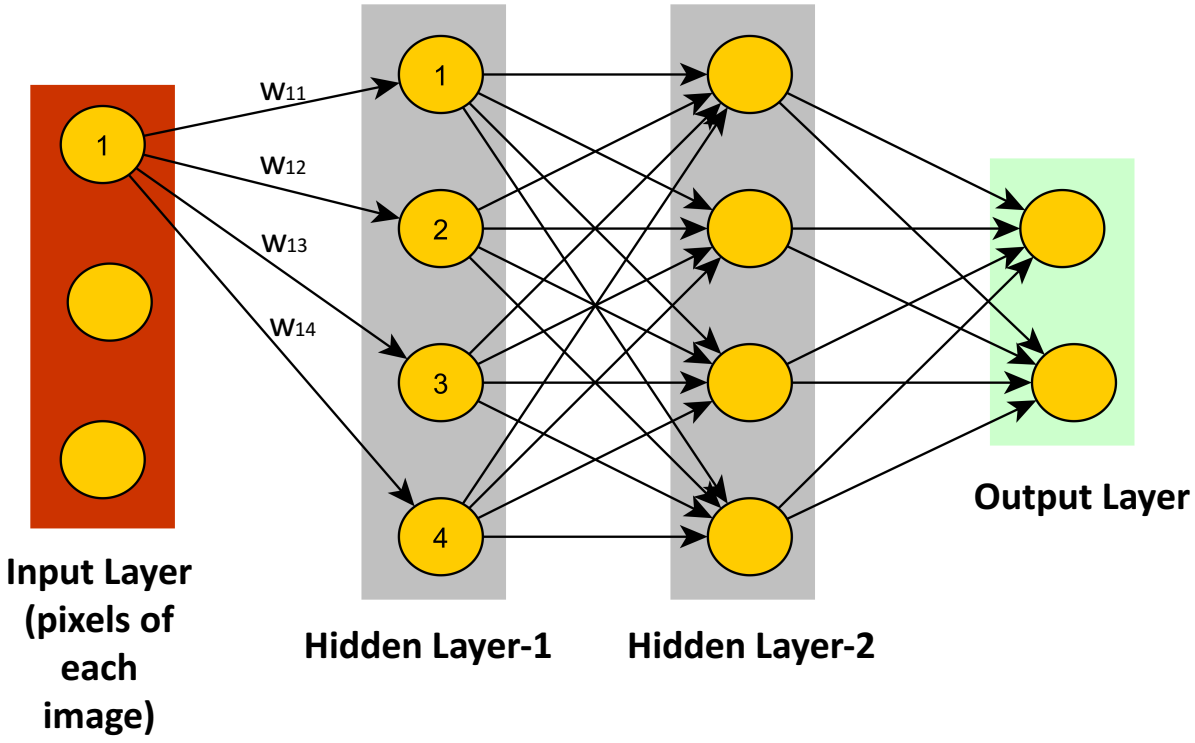

Data augmentation/preprocessing

- Implemented to improve robustness of the classifier
- Effects introduced to account for deviations from training data set:
    1. Background noise (white background converted to noisy background)
    2. Random jitter (Random jitter contrast: Apply PCA to all pixels in training set, sample a ”color offset” along principal component and then add offset to all pixels)
    3. Rotation
    4. Contrast change
    5. Crop to tackle scale variances
    6. Resizing to tackle resolution variances
    7. Image normalization (subtracting the mean from respective RGB channels)
- **scipy, scikit-image** were used to add the above effects

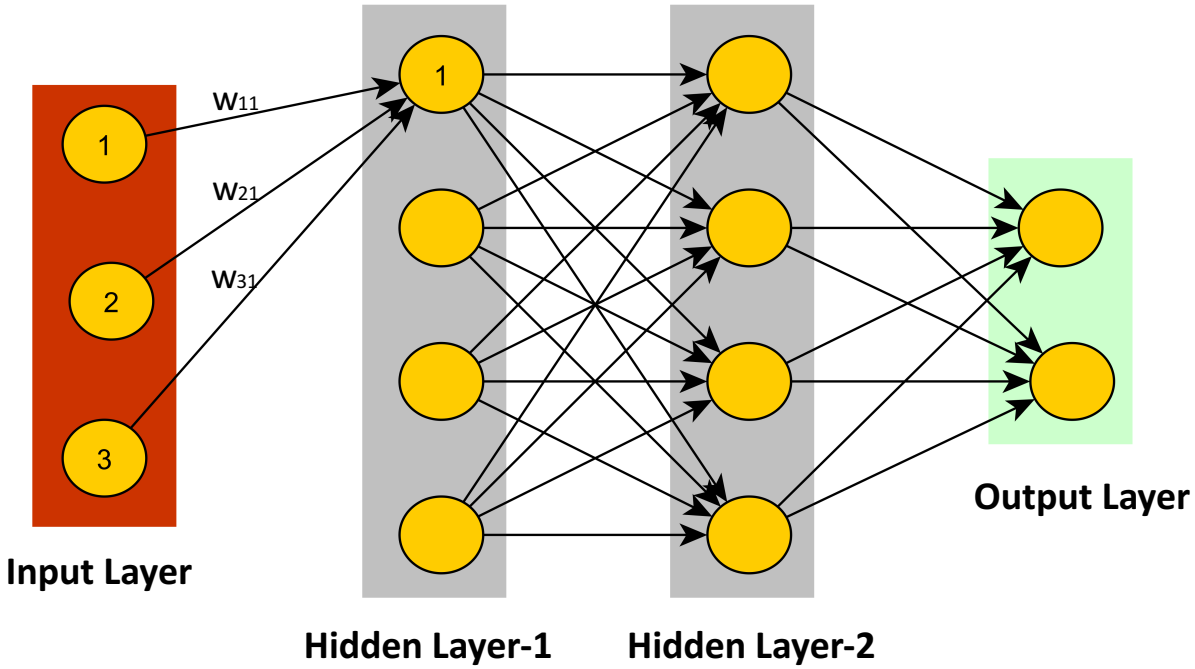
Learning from data: Multi-layer perceptron



Learning from data: Multi-layer perceptron

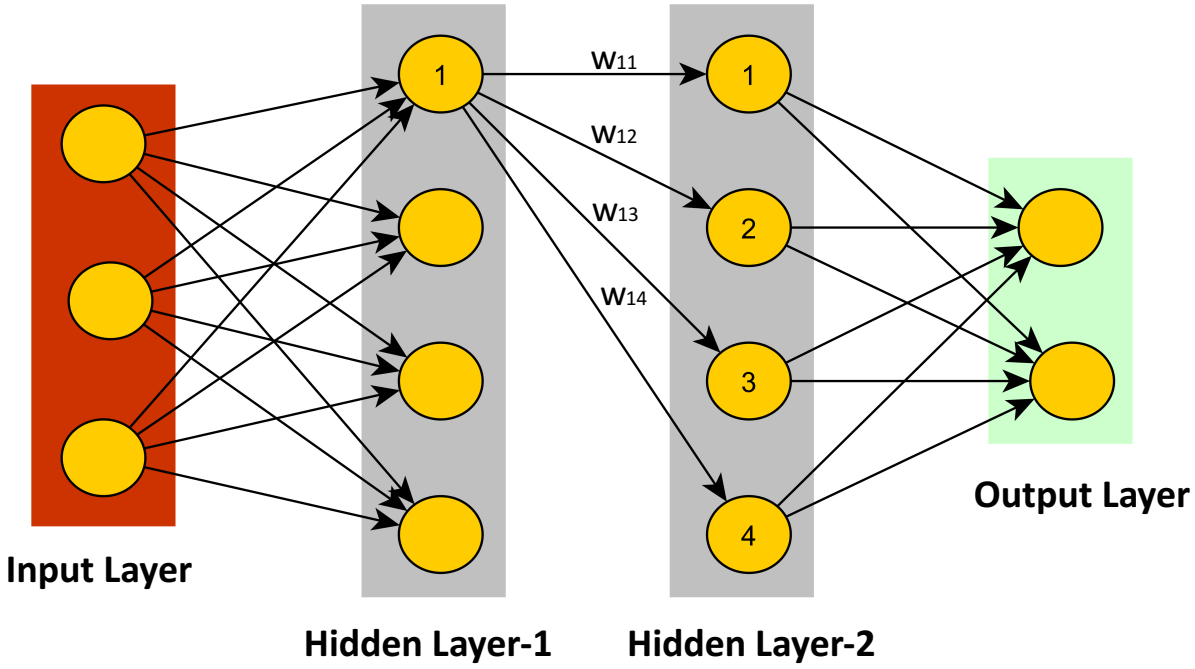


Learning from data: Multi-layer perceptron

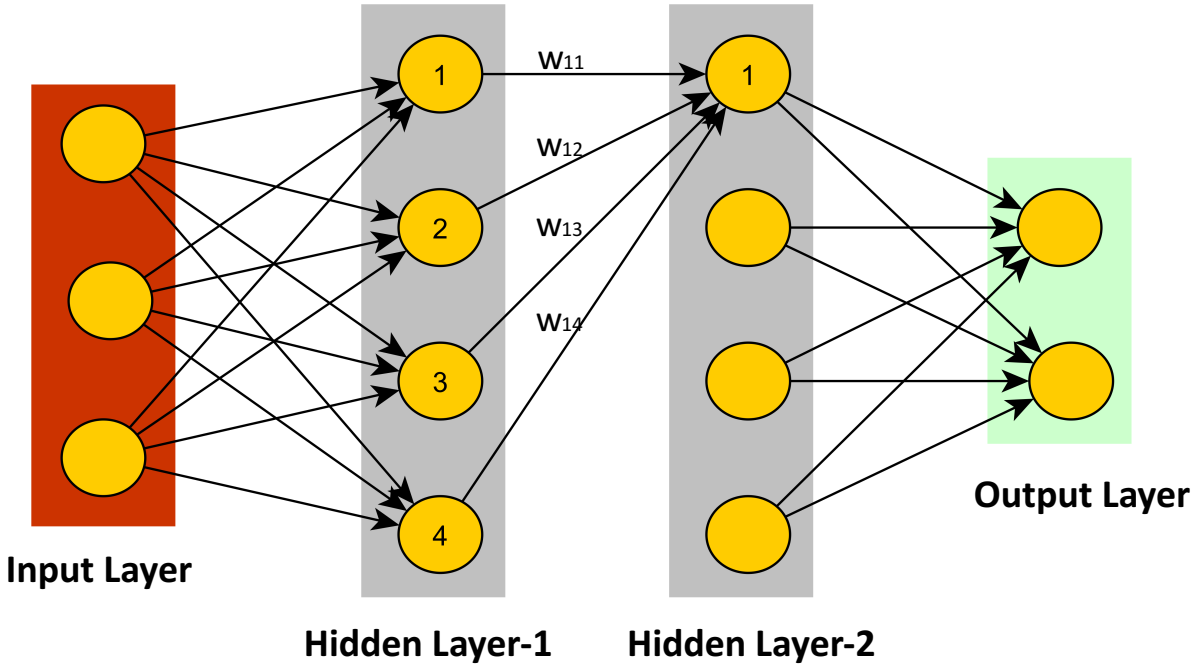


Non-linear activation function to identify complex data patterns.
Eg: f(x) = max(0,x) (ReLu)

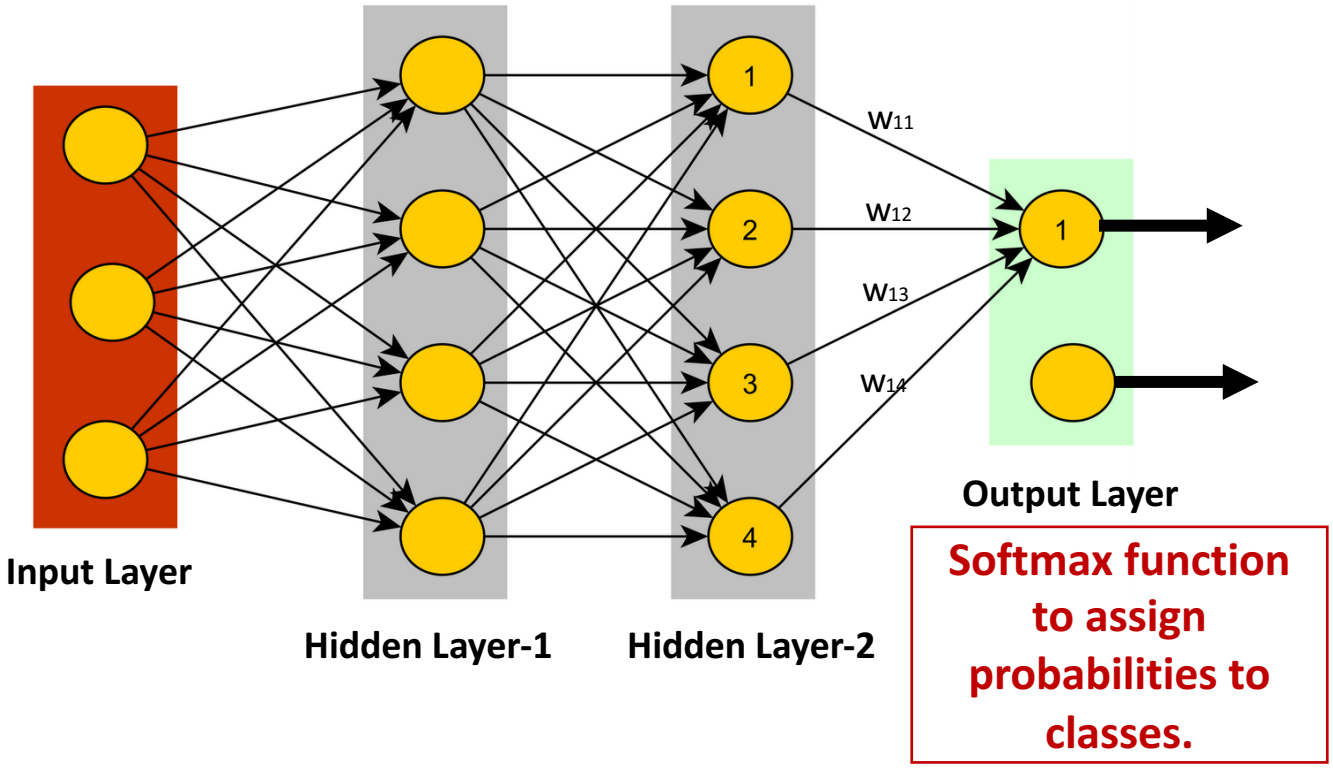
Learning from data: Multi-layer perceptron



Learning from data: Multi-layer perceptron



Learning from data: Multi-layer perceptron



Learning from data: Convolutional Neural Networks (CNN)

32x32x3 image

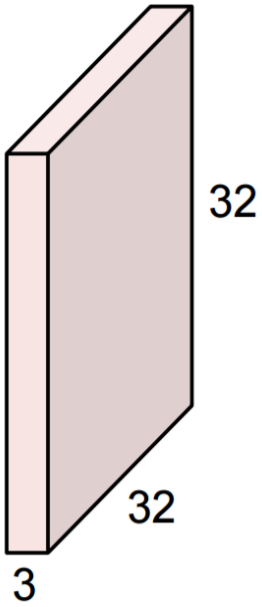


Image input as **volume**

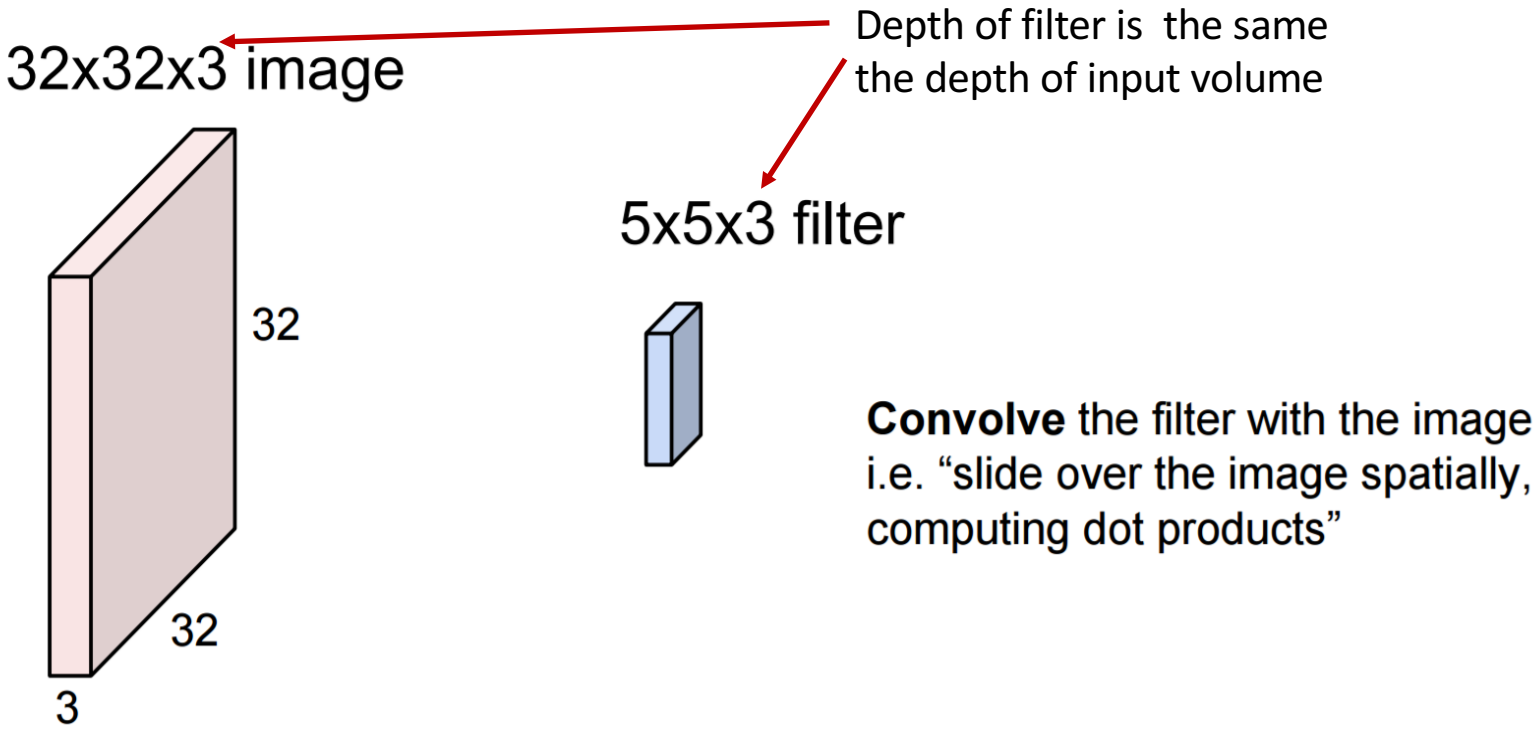
Main Issue: Multi-Layer Perceptron does not scale well to image data. Use CNNs!

5x5x3 filter

Convolve the filter with the image
i.e. "slide over the image spatially, computing dot products"

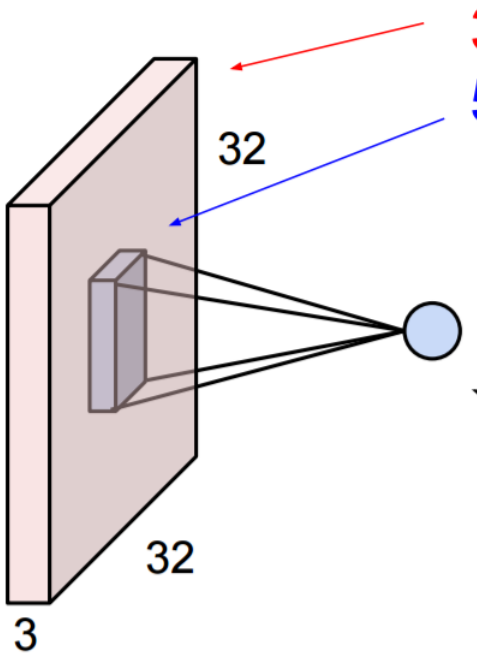
[Slides taken from Lec-7 CS231N Convolutional Neural Networks for Computer Vision]

Learning from data: Convolutional Neural Networks (CNN)



[Slides taken from Lec-7 CS231N Convolutional Neural Networks for Computer Vision]

Learning from data: Convolutional Neural Networks (CNN)



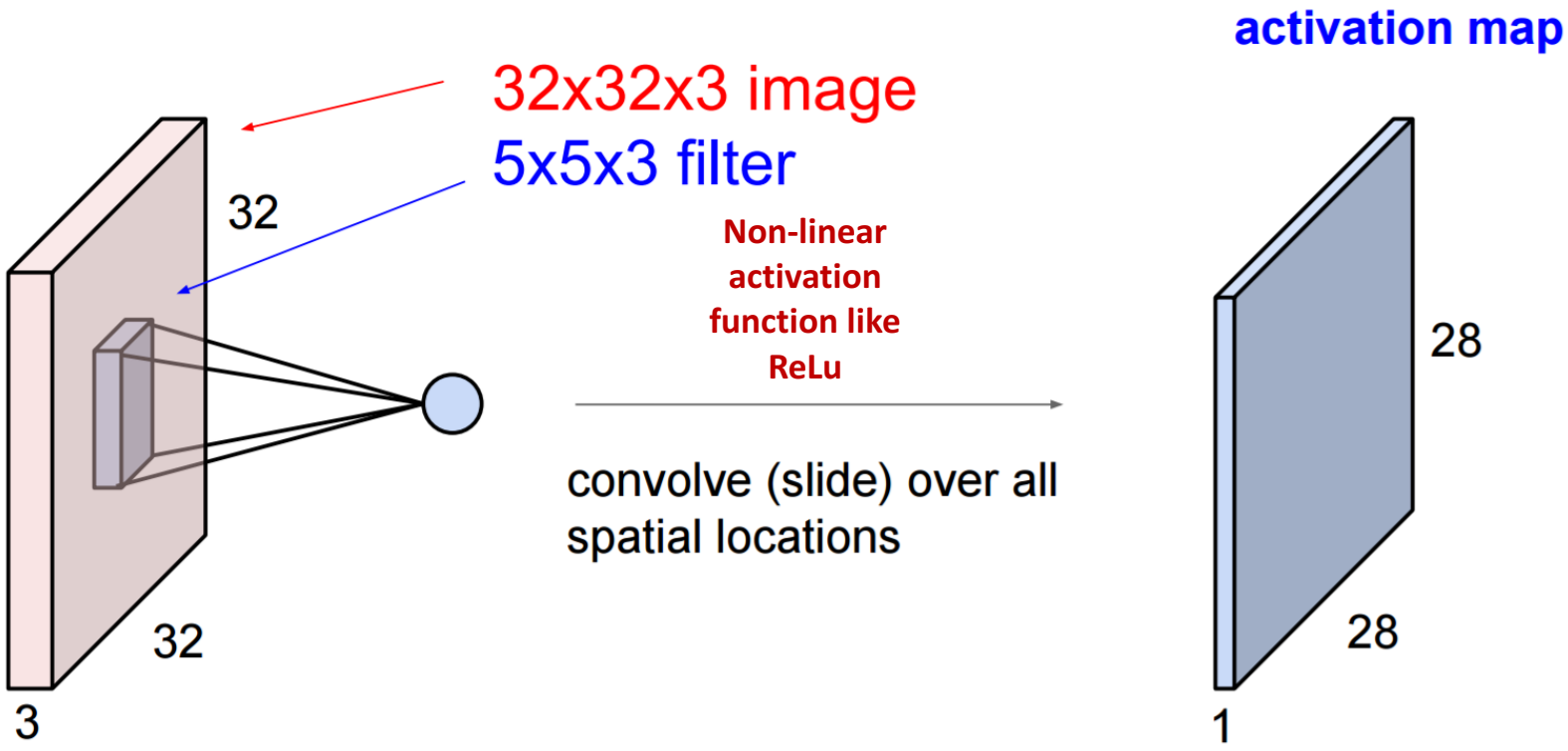
1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

Evaluates image similarity with filter template

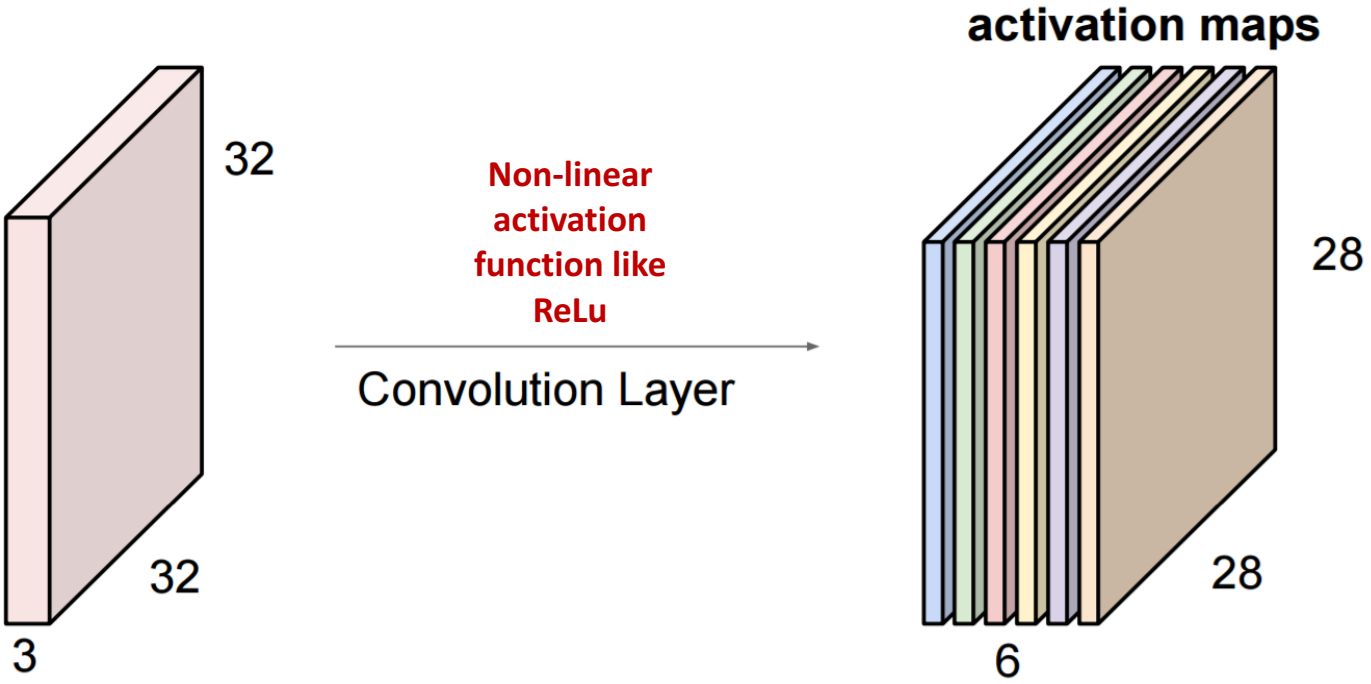
[Slides taken from Lec-7 CS231N Convolutional Neural Networks for Computer Vision]

Learning from data: Convolutional Neural Networks (CNN)



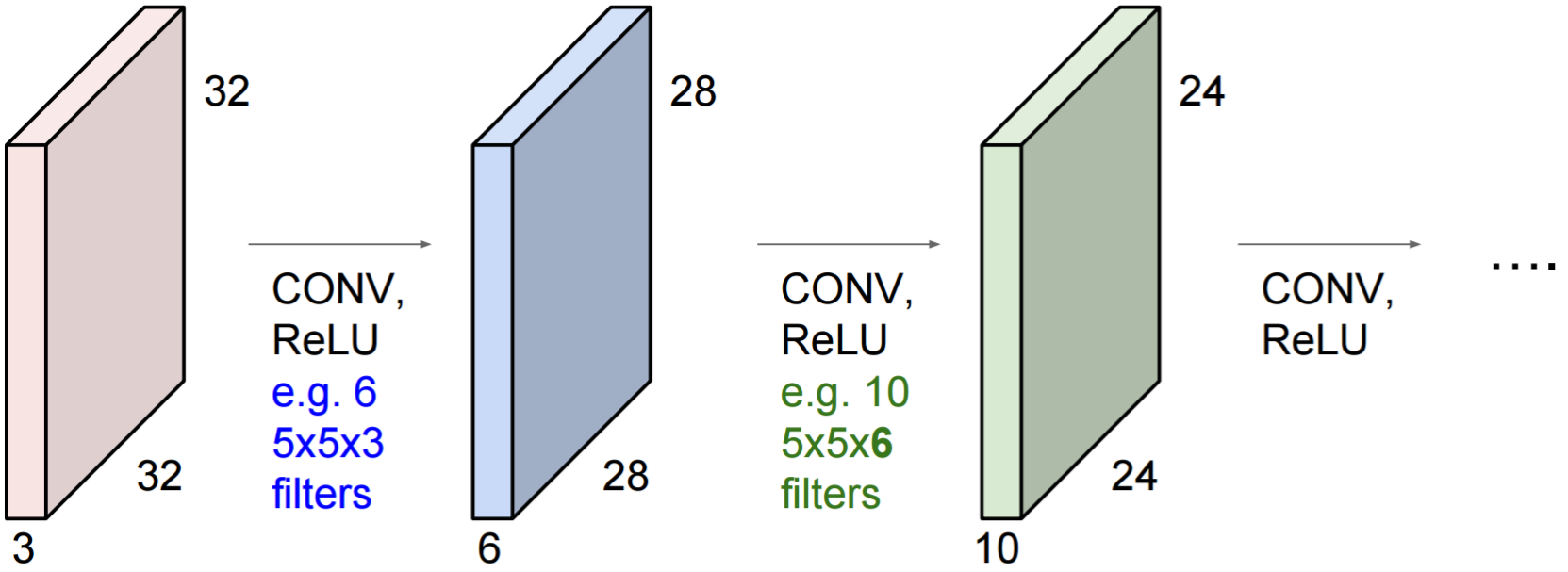
[Slides taken from Lec-7 CS231N Convolutional Neural Networks for Computer Vision]

Learning from data: Convolutional Neural Networks (CNN)



[Slides taken from Lec-7 CS231N Convolutional Neural Networks for Computer Vision]

Learning from data: Convolutional Neural Networks (CNN)



[Slides taken from Lec-7 CS231N Convolutional Neural Networks for Computer Vision]

Steps involved in Training Data

- Define Loss function for minimization: Cross-entropy or Negative Likelihood used

$$L(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{W}) = \frac{1}{n} \sum_{i=1}^{n} \sum_{c \in C} I_{\{y_i = c\}} \log P(Y_i = c | \, x_i) + \boldsymbol{\lambda} ||\boldsymbol{W}||_{\boldsymbol{2}}$$

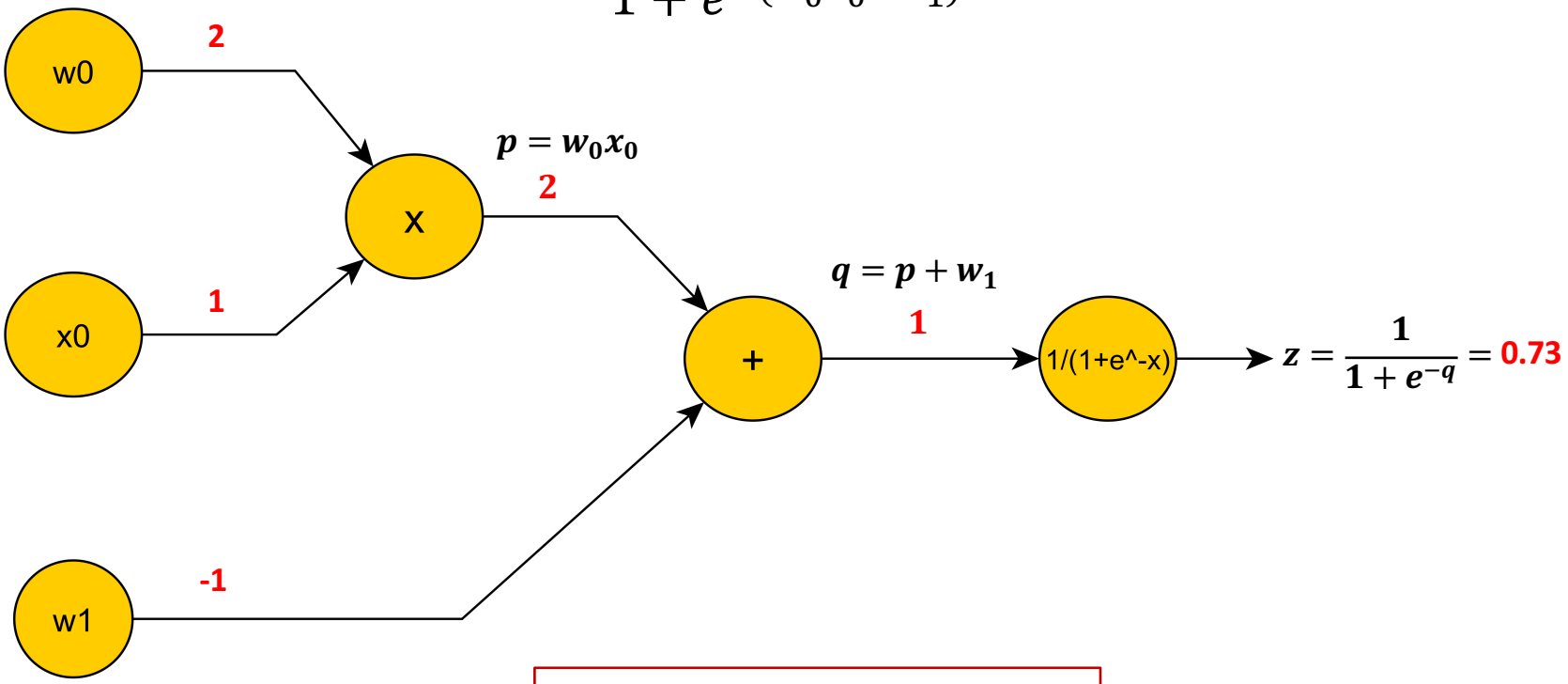
- Use mini-batch SGD to minimize loss
- Initialize weights $\boldsymbol{W_0}$ for all the layers
- Update rule:

$$\boldsymbol{W^{(k+1)}} = \boldsymbol{W^{(k)}} - \boldsymbol{s_k}(\boldsymbol{\nabla_W L(W)})$$

Gradient (Difficult to compute directly)

Gradient Computations based on Computational Graphs

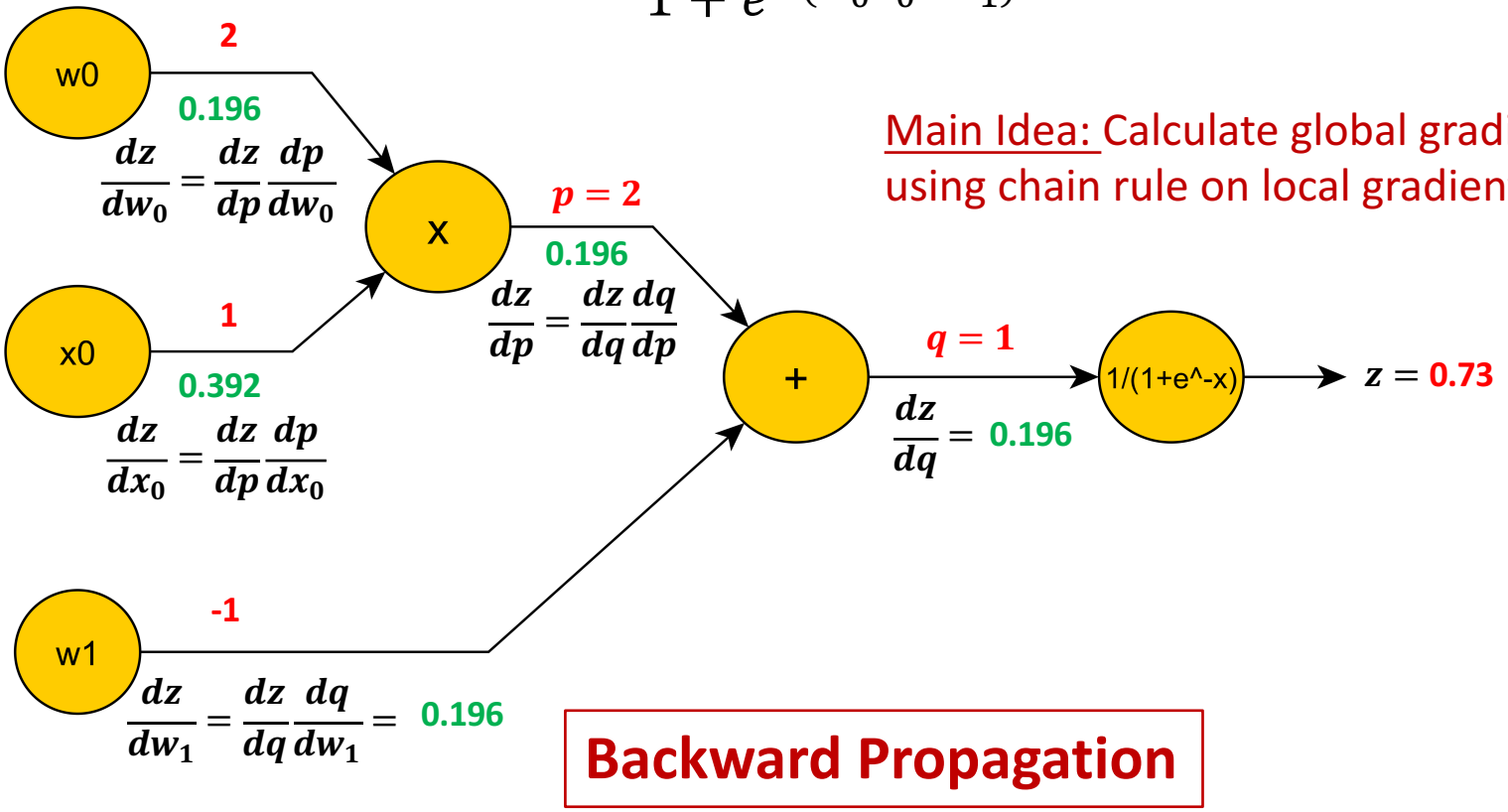
Gradient of $f(x_o, w_o, w_1) = \frac{1}{1 + e^{-(w_0 x_0 + w_1)}}$ at (2,1,1)



Forward Propagation

Gradient Computations based on Computational Graphs

Gradient of $f(x_o, w_o, w_1) = \frac{1}{1 + e^{-(w_0 x_0 + w_1)}}$ at (2,1,1)



Main Idea: Calculate global gradients using chain rule on local gradients

Backward Propagation

Using Pre-trained models

- In practice, training CNNs from scratch is hardly used due to computational considerations
- Instead, pre-trained models trained on other dataset are used
- Main idea: Low level features are common to all image classification problems
- For our project we used VGG-16 model trained for ImageNet classification challenge on 1000 categories

VGG-16 Architecture

- Input: **224x224 size images with 3 channels**

1. Convolutional (224x224x64)
2. Convolutional (224x224x64)

Max Pool (112x112x64)

3. Convolutional (112x112x128)
4. Convolutional (112x112x128)

Max Pool (56x56x128)

5. Convolutional (56x56x256)
6. Convolutional (56x56x256)
7. Convolutional (56x56x256)

Max Pool (28x28x256)

8. Convolutional (28x28x512)
9. Convolutional (28x28x512)
10. Convolutional (28x28x512)

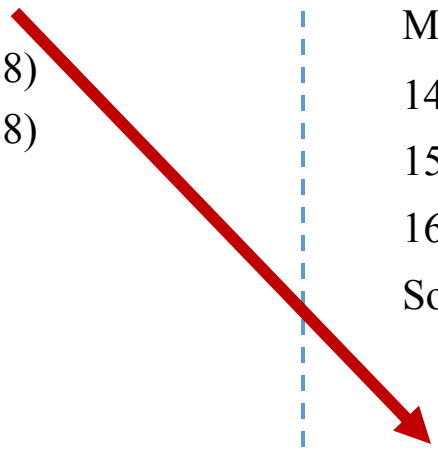
Max Pool (14x14x512)

11. Convolutional (14x14x512)
12. Convolutional (14x14x512)
13. Convolutional (14x14x512)

Max Pool (7x7x512)

14. Fully Connected (1x1x4096)
15. Fully Connected (1x1x4096)
16. Fully Connected (1x1x1000)

Soft-max (1x1x1000)



Reduces spatial dimensions

Implementation: Modified VGG-16

- Input: 224x224 size images with 3 channels

1. Convolutional (224x224x64)
2. Convolutional (224x224x64)

Max Pool (112x112x64)

3. Convolutional (112x112x128)
4. Convolutional (112x112x128)

Max Pool (56x56x128)

5. Convolutional (56x56x256)
6. Convolutional (56x56x256)
7. Convolutional (56x56x256)

Max Pool (28x28x256)

8. Convolutional (28x28x512)
9. Convolutional (28x28x512)
10. Convolutional (28x28x512)

Max Pool (14x14x512)

11. Convolutional (14x14x512)
12. Convolutional (14x14x512)
13. Convolutional (14x14x512)

Max Pool (7x7x512)

14. Fully Connected (1x1x4096)
15. Fully Connected (1x1x4096)
16. Fully Connected (1x1x1000)

Soft-max (1x1x1000)

Treat the previous layers (weights) as fixed and tune the parameters on the last 3 fully connected layers

Implementation: Modified VGG-16

- Input: 224x224 size images with 3 channels

1. Convolutional (224x224x64)
2. Convolutional (224x224x64)

Max Pool (112x112x64)

3. Convolutional (112x112x128)
4. Convolutional (112x112x128)

Max Pool (56x56x128)

5. Convolutional (56x56x256)
6. Convolutional (56x56x256)
7. Convolutional (56x56x256)

Max Pool (28x28x256)

8. Convolutional (28x28x512)
9. Convolutional (28x28x512)
10. Convolutional (28x28x512)

Max Pool (14x14x512)

11. Convolutional (14x14x512)
12. Convolutional (14x14x512)
13. Convolutional (14x14x512)

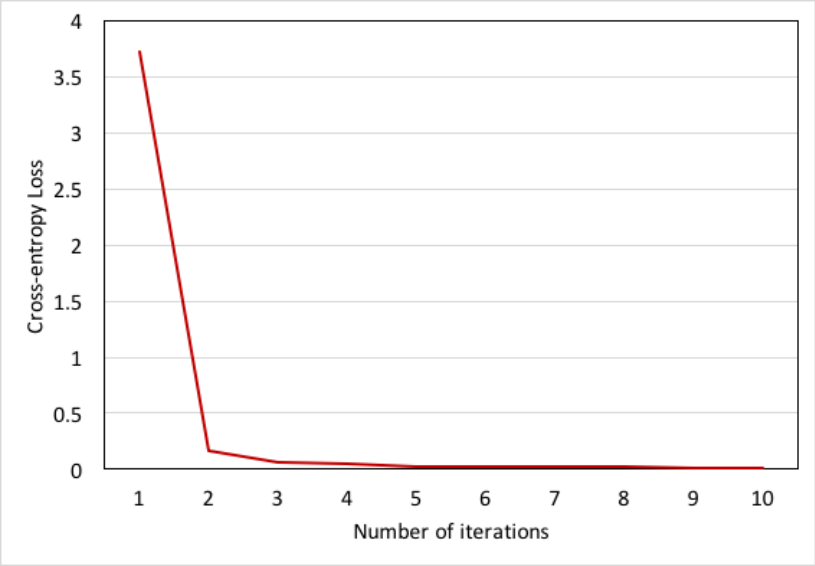
Max Pool (7x7x512)

14. Fully Connected (1x1x4096)
15. Fully Connected (1x1x4096)
16. Fully Connected (1x1x1000)

Soft-max (1x1x1000)

Implemented CNN architecture using Theano (library) and Lasagne (wrapper) with CUDA GPU on AWS EC2 server

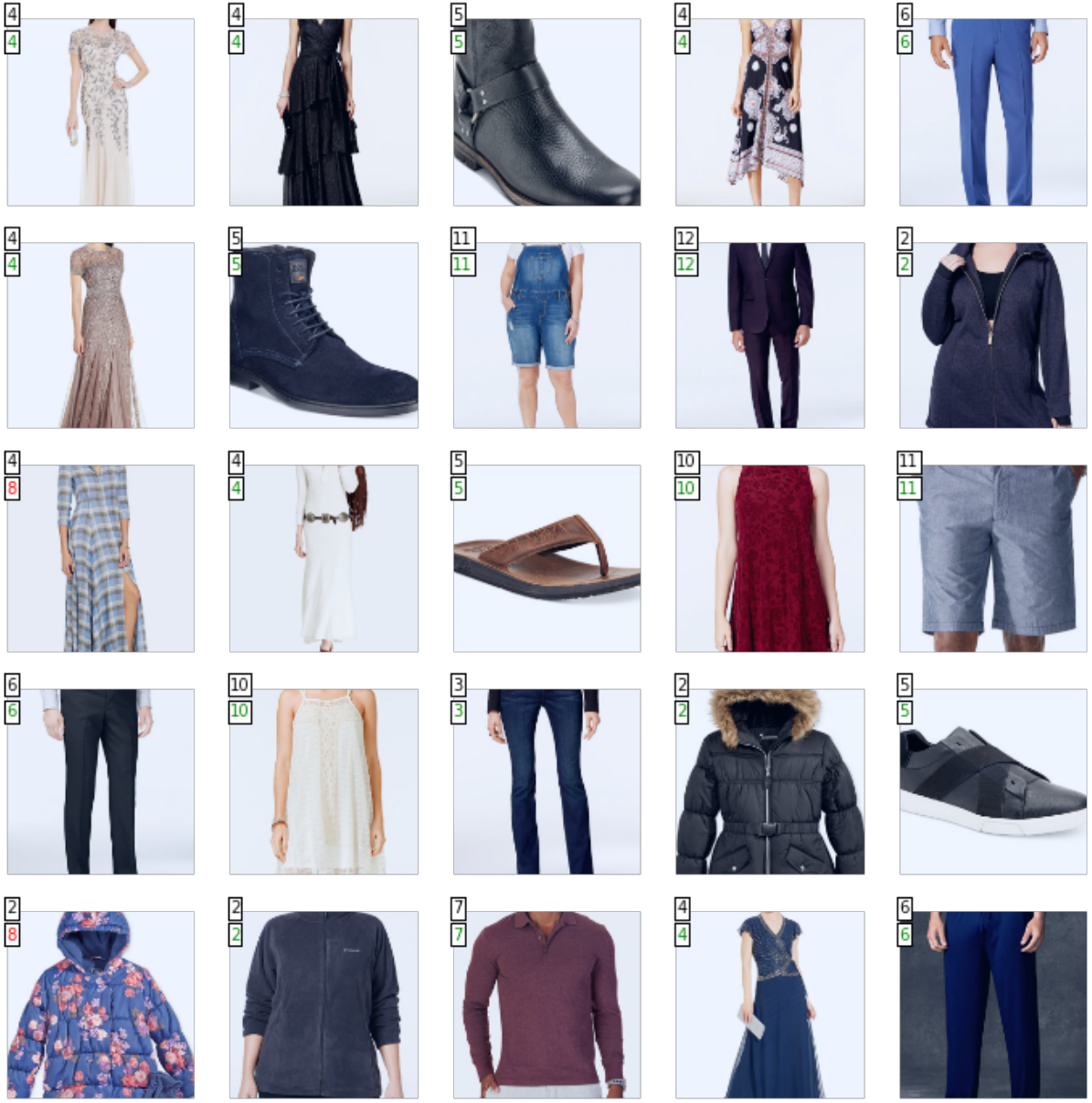
Results using Macy's data



Cross-entropy loss versus number of iterations using Nesterov momentum update rule

- Initially, results were observed using the train-test split within Macy's data (without data augmentation)
- ~92% accuracy was achieved

Results using Macy's data



Results using Macy's data


Training Data


Test Data

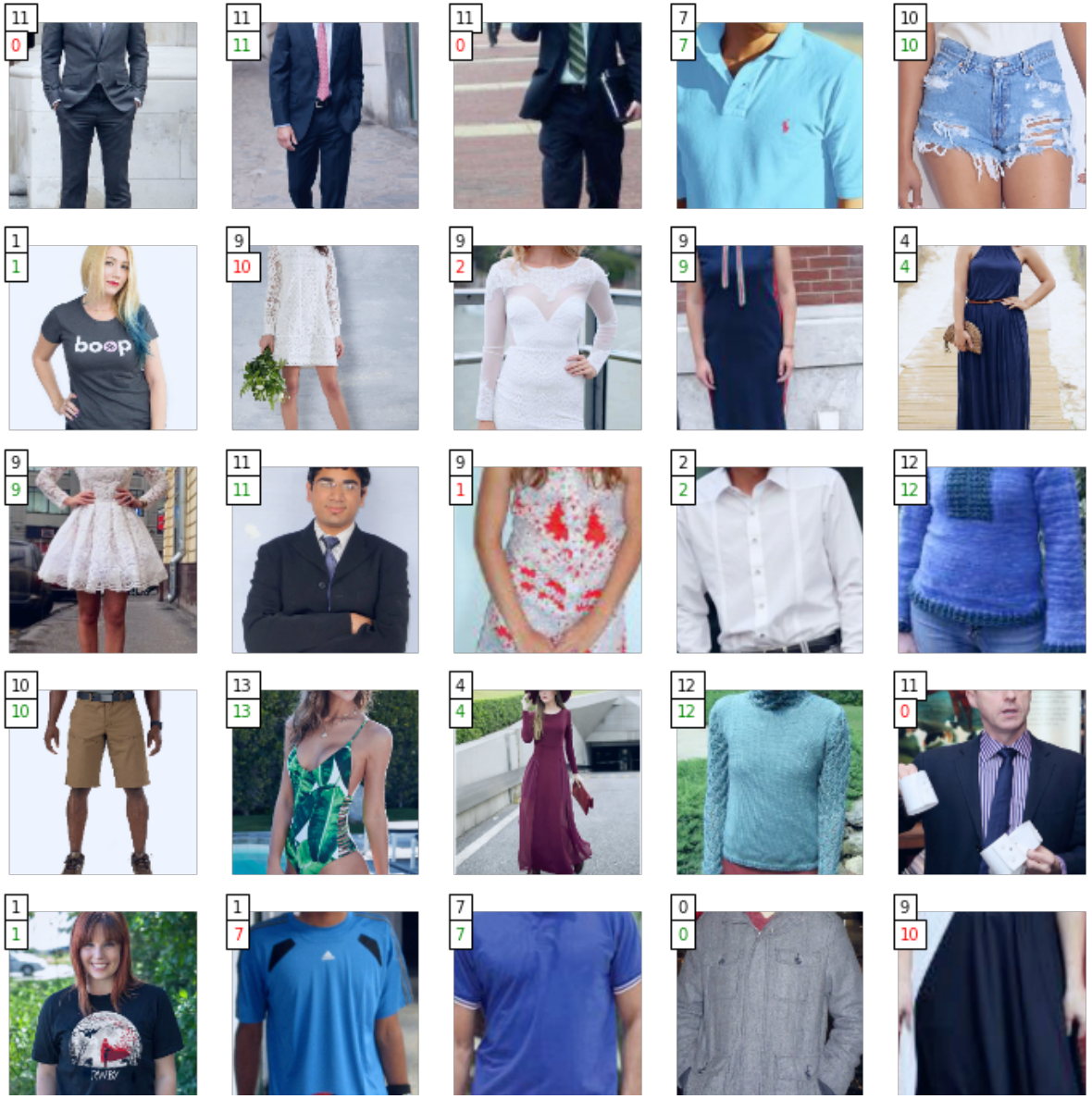
- Results were exceptionally bad (~20% accuracy) when using the same training data to predict a test data of real life examples from Google Images
- Hence, there was a need for data augmentation

Results for test data after data augmentation

Cross-entropy loss versus number of iterations using Nesterov momentum update rule

- The test data consisted of around 60 images from Google Images comprising of real world photos
- ~71% accuracy was achieved

Results for test data after data augmentation



Next Steps…

- Extend analysis to multiple clothing object detection
- Approach: Use a sliding window approach with fast linear classifiers to identify regions of interest and apply CNNs
- Determine specific object location using regression head with a L2 loss function instead of classification head with cross-entropy loss

THANK YOU

Results using Macy's data

Label	Category
0	Jacket
1	T-Shirt
2	Shirt
3	Jeans
4	Long-dress
5	Shoes
6	Trousers
7	Polo-shirts
8	Robe
9	Short-dress
10	Shorts
11	Suit
12	Sweater
13	Swimsuit