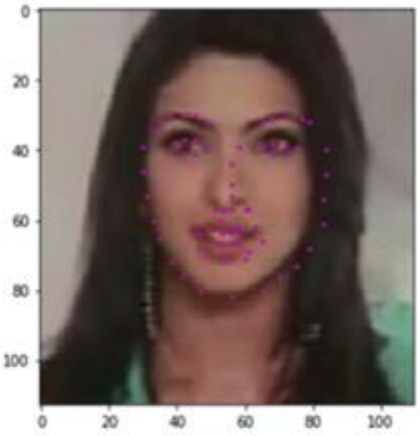
Introduction to Deep Learning (I2DL)

Tutorial 9: Facial Keypoint Detection

Overview

- Exercise 08: Case Study
- Fully Connected & Convolutional Layers
  - Recap
  - Changes to Dropout & BatchNorm
- Exercise 09: Facial Keypoint Detection

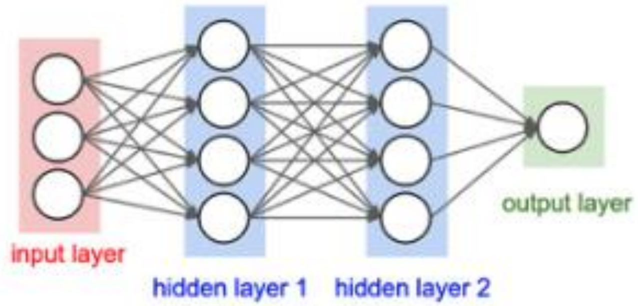


Fully Connected
vs
Convolutional Layers

Recap: Fully Connected Layers

- **Fully Connected (FC) networks / Multi-Layer Perceptron (MLP):** Receive an input vector and transform it through a series of hidden layers (weights & activation functions).

- **Fully Connected layers:** Each layer is made up of a set of neurons, where each single neuron is connected to all neurons in the previous layer

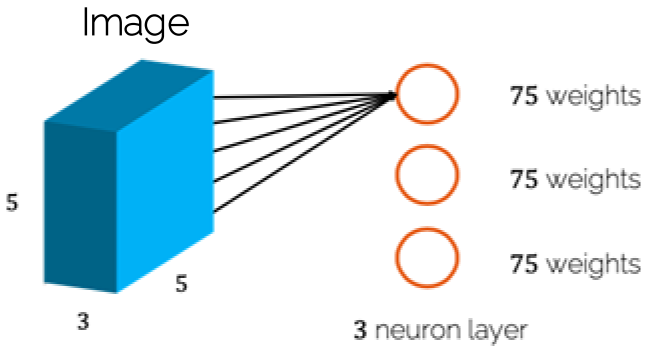


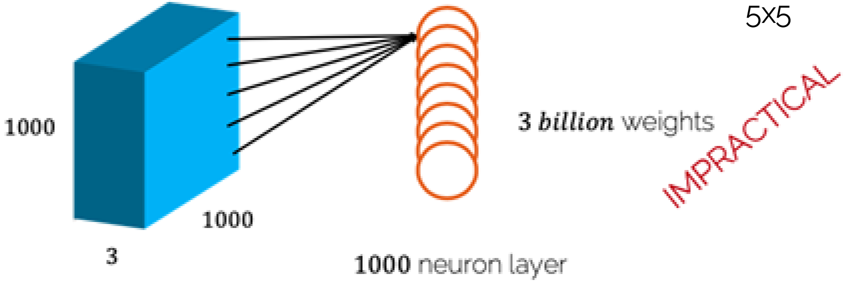
Computer Vision – MLP

- **Assumption:** Input to the network are images
- **Disadvantage:** Images need to have a certain resolution to contain enough information

238x238

5X5

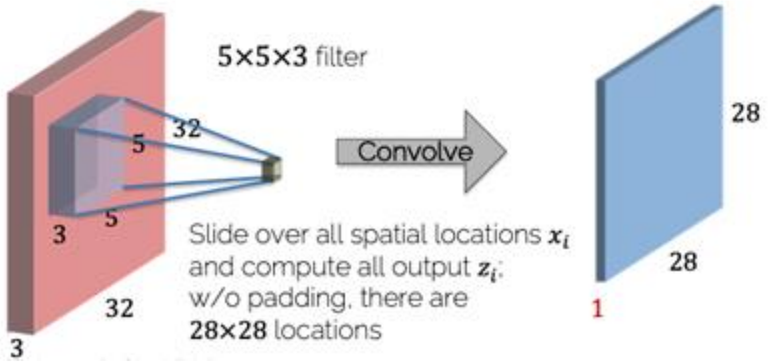




Can we reduce the number of weights in our architecture?

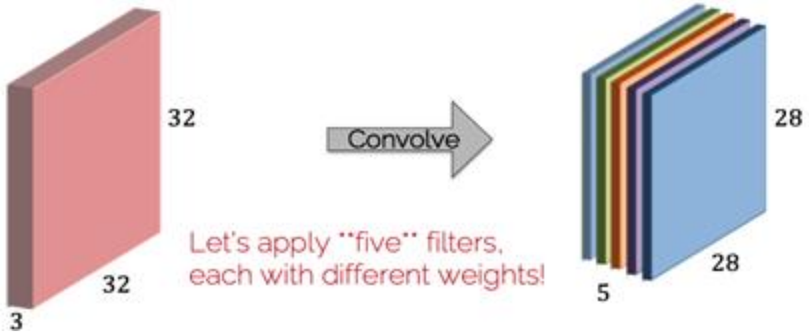
Computer Vision - CNN

- **Assumption:** Input to the network are images
- **Idea:** Sliding filter over the input image (convolution) instead of passing the entire image through all neurons individually



Computer Vision - CNN

- **Assumption:** Input to the network are images
- **Filters:** Sliding window with the same filter parameters to extract image features
- **Advantage:** Learn translation-invariant "concepts" and weight sharing



Convolution: Hard-coded

3x3 kernel

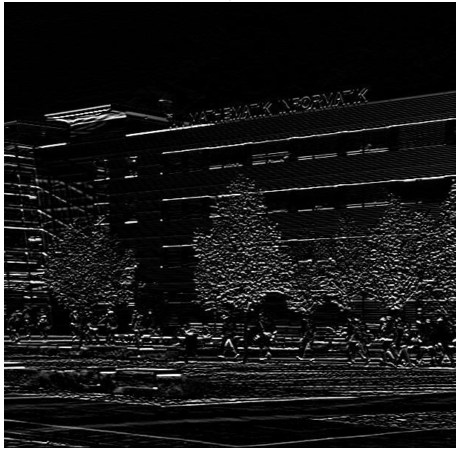
```
[-1, -1, -1]
[ 0,  0,  0]
[ 1,  1,  1]
```

*

*

3x3 kernel

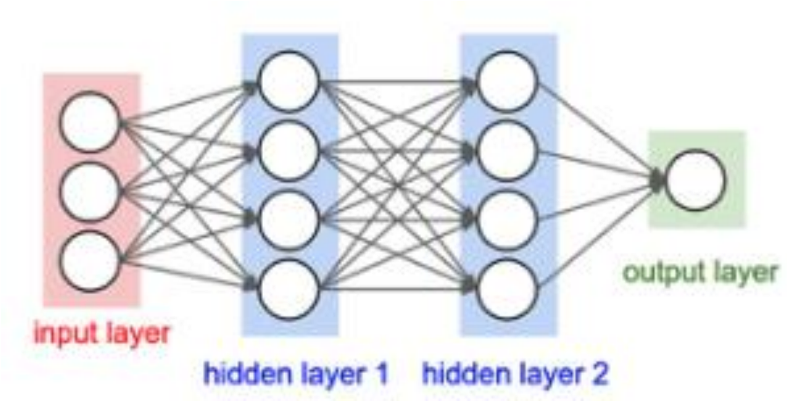
```
[-1, 0, 1]
[-1, 0, 1]
[-1, 0, 1]
```

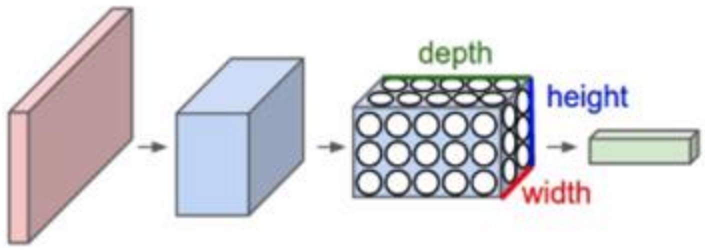


Convolutional Layers: BatchNorm and Dropout

Fully Connected vs Convolution

- Output Fully-Connected layer: One layer of neurons, independent
- Output Convolutional Layer: Neurons arranged in 3 dimensions





http://graphics.stanford.edu/courses/cs468-17-spring/LectureSlides/L10%20-%20intro_to_deep_learning.pdf

Recap: Batch Normalization

- Batch norm for **FC** neural networks
  - Input size (N, D)
  - Compute minibatch mean and variance across N (i.e. we compute mean/var for each feature dimension)

Input: $x : N \times D$

Learnable params:
$\gamma, \beta : D$

Intermediates: $\mu, \sigma : D$
$\hat{x} : N \times D$

Output: $y : N \times D$

$$\mu_j = \frac{1}{N} \sum_{i=1}^{N} x_{i,j}$$

$$\sigma_j^2 = \frac{1}{N} \sum_{i=1}^{N} (x_{i,j} - \mu_j)^2$$

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$

http://cs231n.stanford.edu/slides/2018/cs231n_2018_lecture07.pdf

Recap: Batch Normalization

- Batch norm for **FC** neural networks
  - Input size (N, D)
  - Compute minibatch mean and variance across N (i.e. we compute mean/var for each feature dimension)

Batch Normalization for **fully-connected** networks

$\mathbf{x}: \mathrm{N} \times \mathrm{D}$

Normalize

$\boldsymbol{\mu}, \boldsymbol{\sigma}: 1 \times \mathrm{D}$

$\gamma, \beta: 1 \times \mathrm{D}$

$\mathbf{y} = \gamma(\mathbf{x} - \boldsymbol{\mu}) / \boldsymbol{\sigma} + \beta$

http://cs231n.stanford.edu/slides/2018/cs231n_2018_lecture07.pdf

Spatial Batch Normalization

- Batchnorm for **convolutional** NN = spatial batchnorm
  - Input size (N, C W, H)
  - Compute minibatch mean and variance across N, W, H (i.e. we compute mean/var for each channel C)

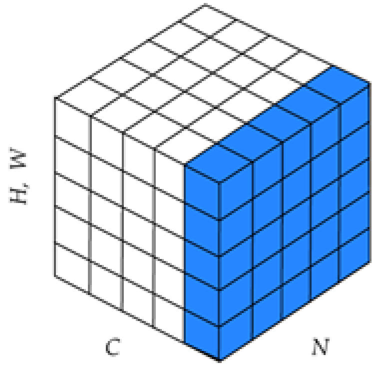
$\mathbf{x}$: N×C×H×W

Normalize (N, H, W)

μ, σ: 1×C×1×1

γ, β: 1×C×1×1

$y = \gamma(x-\mu)/\sigma+\beta$



http://cs231n.stanford.edu/slides/2018/cs231n_2018_lecture07.pdf

Spatial Batch Normalization

Fully Connected

– Input size (N, D)
– Compute minibatch mean and variance across N (i.e. we compute mean/var for each feature dimension)

$\mathbf{x}: N \times D$

Normalize $\downarrow$ (N)

$\boldsymbol{\mu}, \boldsymbol{\sigma}: 1 \times D$

$\gamma, \beta: 1 \times D$

$y = \gamma(x - \mu)/\sigma + \beta$

Convolutional = spatial BN

– Input size (N, C, W, H)
– Compute minibatch mean and variance across N, W, H (i.e. we compute mean/var for each channel C)

$\mathbf{x}: N \times C \times H \times W$

Normalize $\downarrow$ (N, H, W)

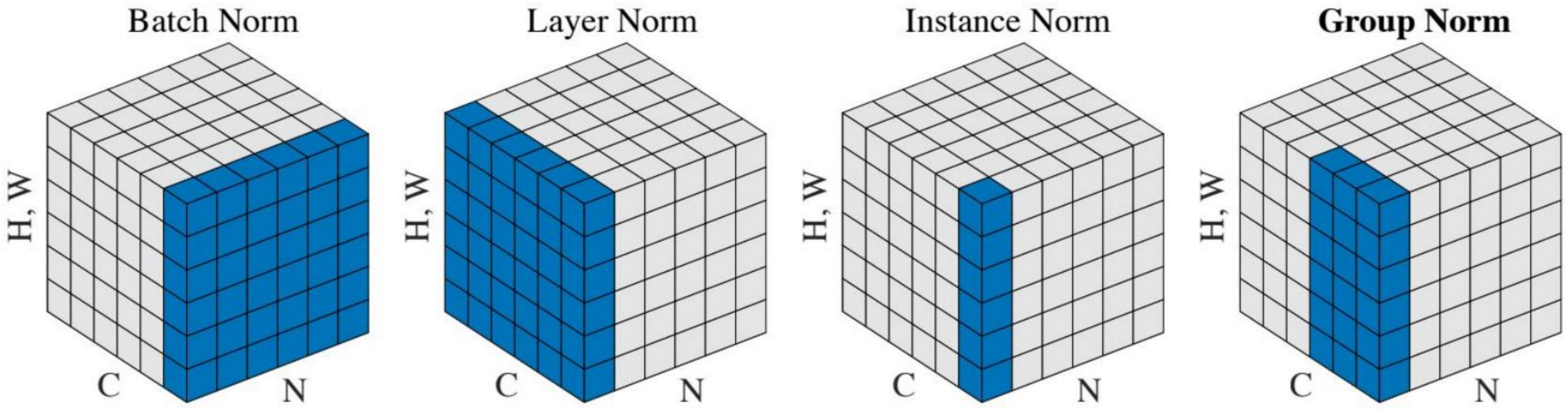
$\boldsymbol{\mu}, \boldsymbol{\sigma}: 1 \times C \times 1 \times 1$

$\gamma, \beta: 1 \times C \times 1 \times 1$

$y = \gamma(x - \mu)/\sigma + \beta$

http://cs231n.stanford.edu/slides/2018/cs231n_2018_lecture07.pdf

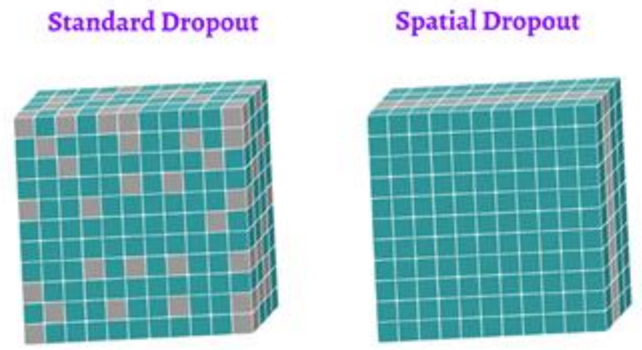
Other normalizations



http://cs231n.stanford.edu/slides/2018/cs231n_2018_lecture07.pdf

Dropout for convolutional layers

- **Regular Dropout:** Deactivating specific neurons in the networks (one neuron "looks" at whole image)
- **Dropout Convolutional Layers:** Standard neuron-level dropout (i.e. randomly dropping a unit with a certain probability) does not improve performance in convolutional NN
- **Spatial Dropout** randomly sets entire feature maps to zero

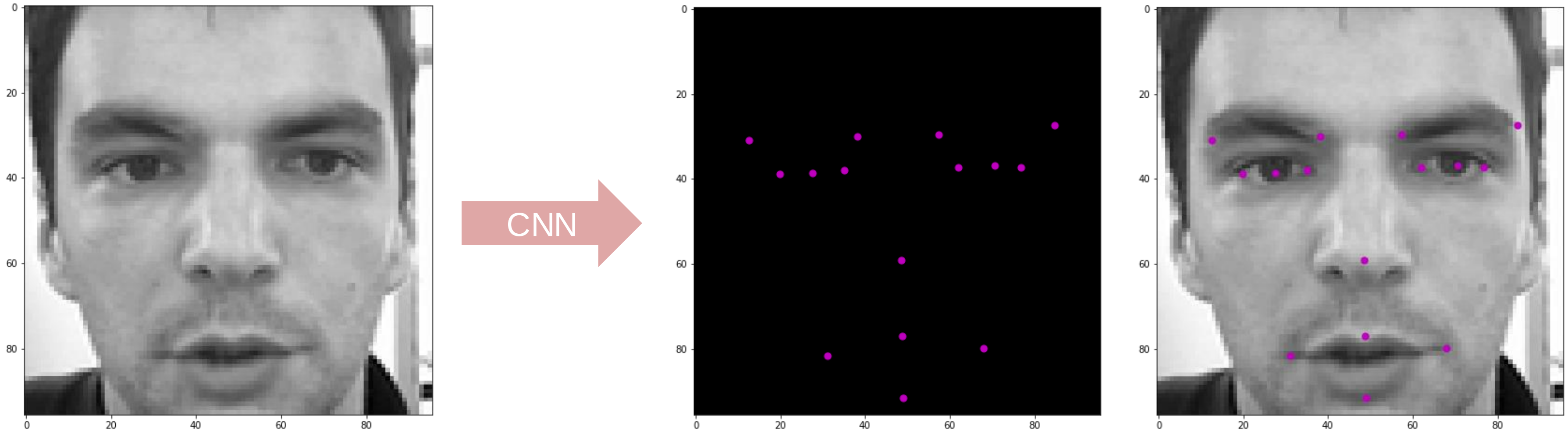


Exercise 9:
Facial Keypoints Detection

Submission: Facial Keypoints

Input:
(1, 96, 96) grayscale image

Output:
(2, 15) keypoint coordinates



Dataset:
- train: 1546 images
- validation: 298 images

Submission: Metric

Accuracy (Classification) → Score (Regression)

```
def evaluate_model(model, dataset):
    model.eval()
    criterion = torch.nn.MSELoss()
    dataloader = DataLoader(dataset, batch_size=1, shuffle=False)
    loss = 0
    for batch in dataloader:
        image, keypoints = batch["image"], batch["keypoints"]
        predicted_keypoints = model(image).view(-1,15,2)
        loss += criterion(
            torch.squeeze(keypoints),
            torch.squeeze(predicted_keypoints)
        ).item()
    return 1.0 / (2 * (loss/len(dataloader)))

print("Score:", evaluate_model(dummy_model, val_dataset))
```

Submission Requirement: Score >= 100

Good luck &

see you next week

☺