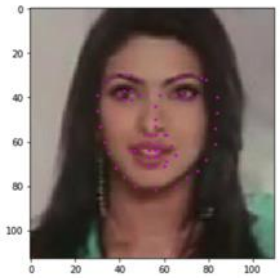
Introduction to Deep Learning (I2DL)

Tutorial 9: Facial Keypoint Detection

Overview

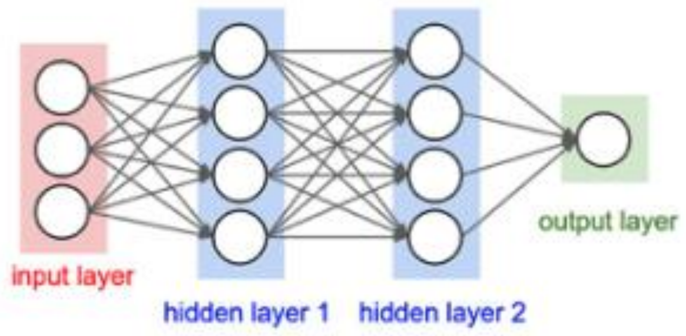
- Convolutional Layers
  - Recap
  - Changes to Dropout & Batchnorm

- Exercise 09: Facial Keypoint Detection
  - Deadline: 18.01.2023 15.59



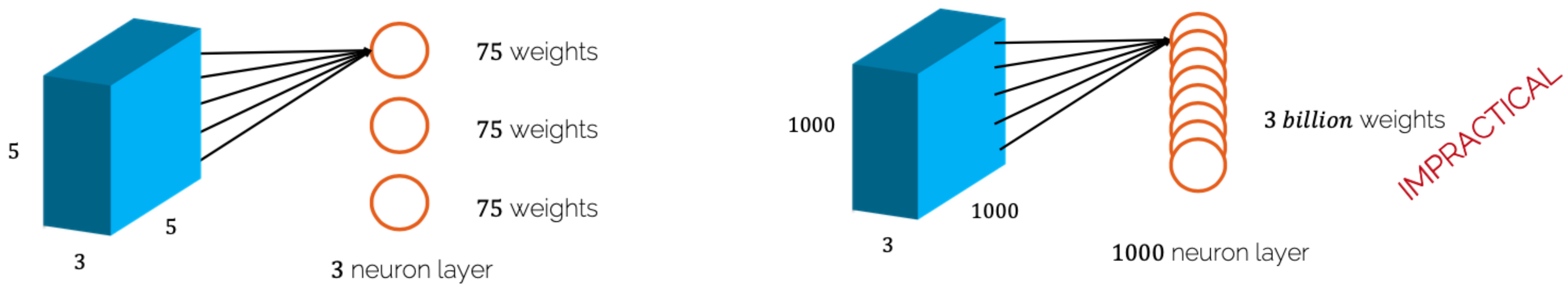
Recap: Fully-Connected Layers

- **Regular Neural Networks:** Receive an input vector and transform it through a series of hidden layers.
- **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



Convolutional Layers

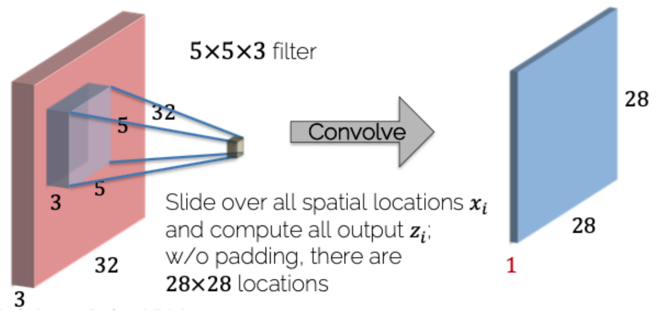
- **Assumption:** Input to our Network are images
- **Disadvantage:** Normal sized images are more likely to produce the right situation



Can we reduce the number of weights in our architecture?

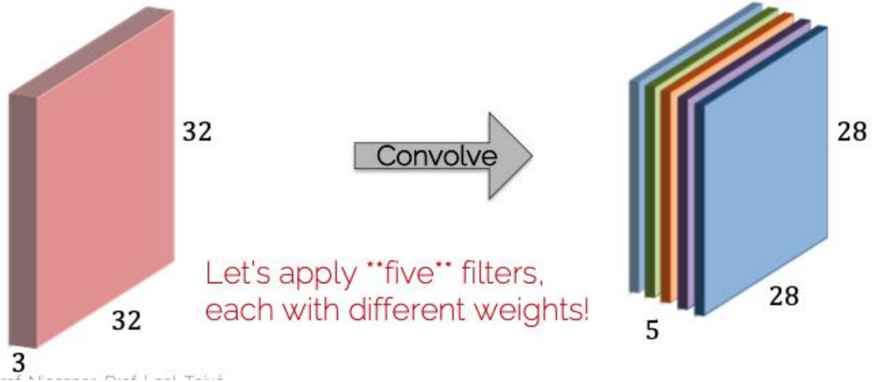
Convolutional Layers

- **Assumption:** Input to our Network are images
- **Advantage:** We can analyze the image by looking at different region instead of looking at the whole image
- **Idea:** Sliding filter over the input image (convolution) instead of matrix multiplication

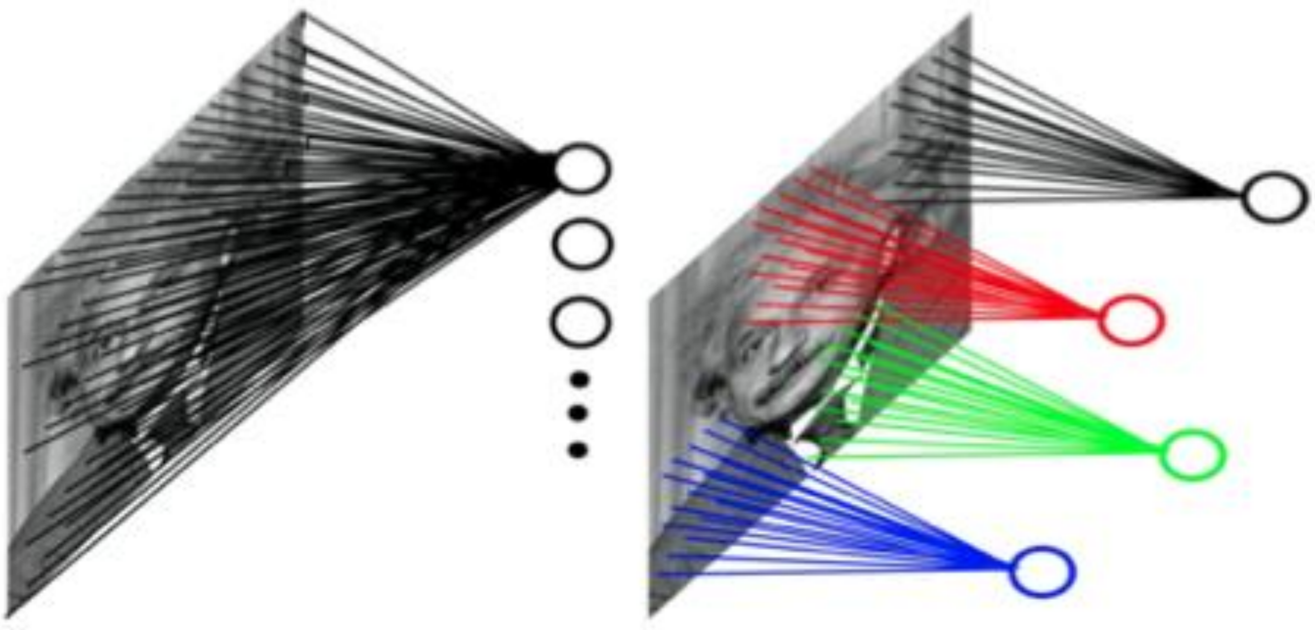


Convolutional Layers

- **Assumption:** Input to our Network are images
- **Filters:** Sliding window with the same filter parameters to extract image features
  - Concept of weight sharing
  - Extract features independent of location



Fully Connected vs Convolution

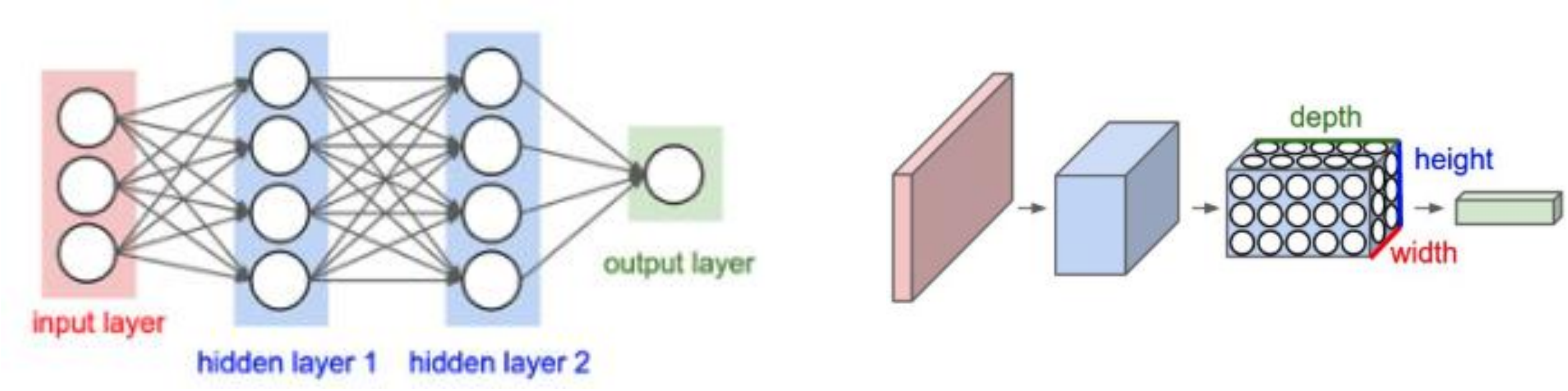


http://graphics.stanford.edu/courses/cs468-17-spring/LectureSlides/L10%20-%20intro_to_deep_learning.pdf

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 regular 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: $y : N \times D$

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

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

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

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

Batch Normalization for **fully-connected** networks

x: N × D

Normalize ↓

μ,σ: 1 × D

γ,β: 1 × D

y = γ(x−μ)/σ+β

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)

Batch Normalization for **convolutional** networks (Spatial Batchnorm, BatchNorm2D)

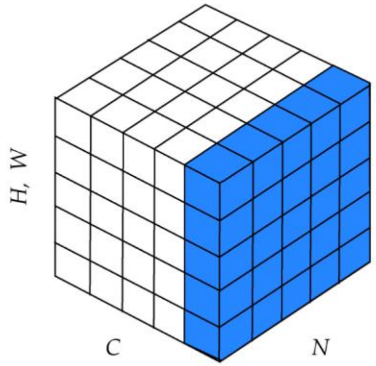
$$\mathbf{x}: \ N \times C \times H \times W$$

Normalize

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

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

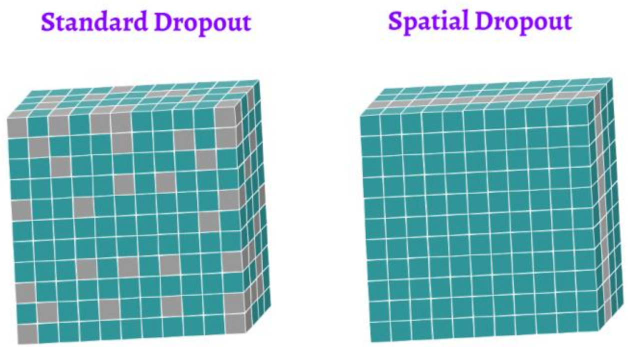
$$\mathbf{y} = \gamma(\mathbf{x} - \boldsymbol{\mu}) / \boldsymbol{\sigma} + \beta$$



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
- **Variant:** Spatial Dropout randomly sets entire feature maps to zero



Exercise 9: Facial Keypoints Detection

Exercise 9: Facial Keypoints

Input:

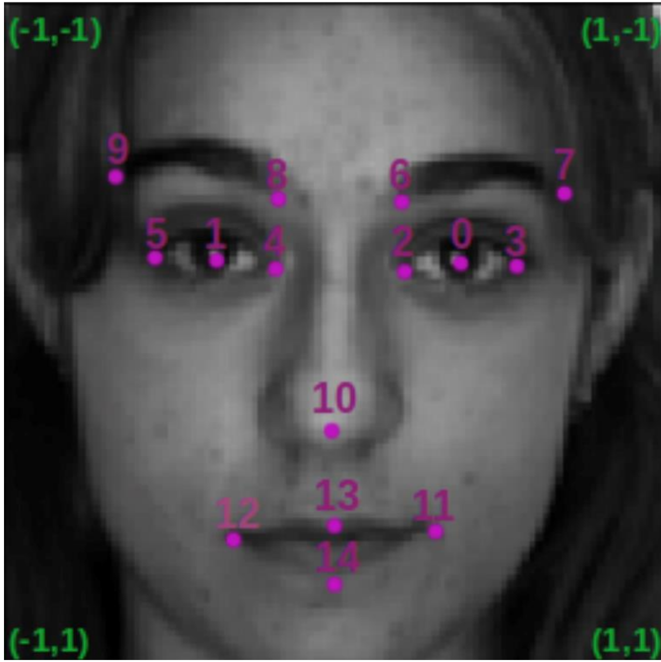
(1, 96, 96) image

- Grayscale, not RGB

Output:

coordinates of facial keypoints

(2, 15)



Submission: Metric

```python
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: Details

- Submission **Start:** 12.01.2023 13.00
- Submission **Deadline :** 18.01.2023 15.59
- Your model's **evaluation score** is all that counts!
  - Evaluation score: 1 / (2 * MSE)
  - A **score of at least 100** to pass the submission

Good luck &
see you next Week
☺