Tutorial 9: Facial Keypoint Detection

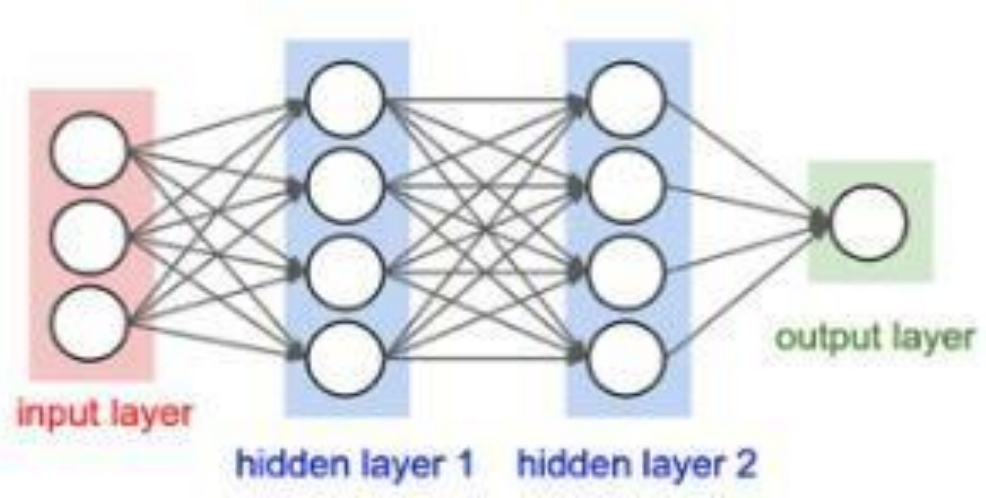
Overview

- Fully Connected & Convolutional Layers
  - Repaso
  - Cambios a Dropout & BatchNorm

- Ejercicio 9: Facial Keypoint Detection

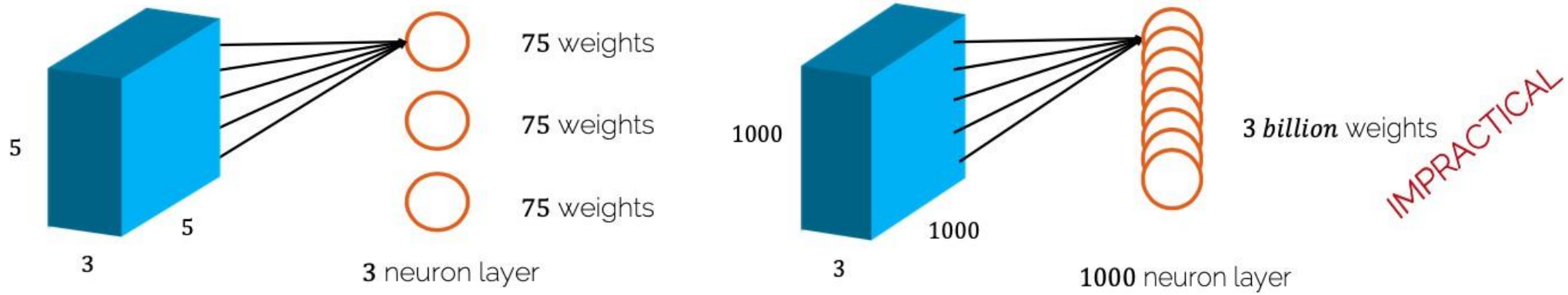
Repaso: Fully Connected Layers

- Fully Connected Layers: Cada capa está formada por un conjunto de neuronas, donde cada neurona individual está conectada a todas las neuronas de la capa anterior.



Computer Vision - FN

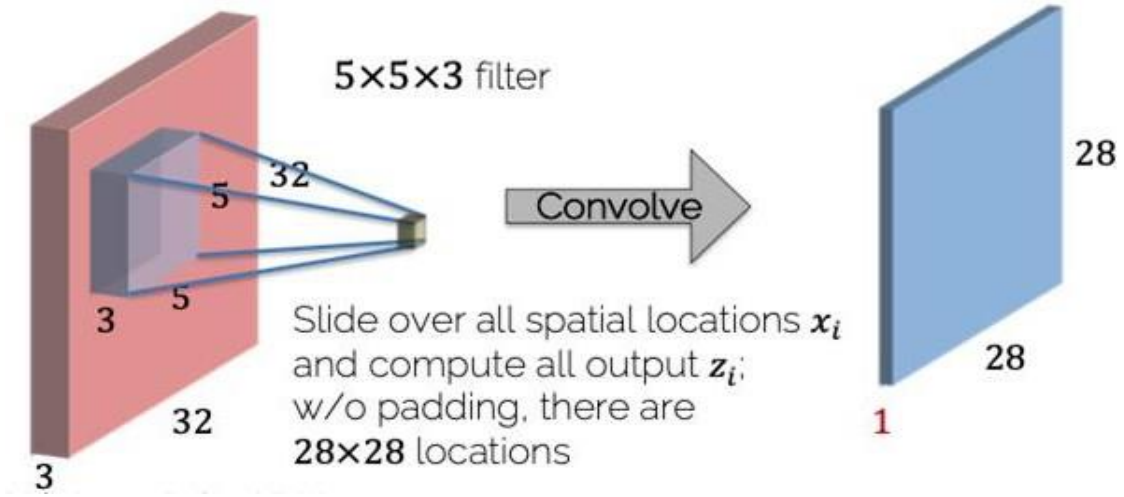
- Suposición: La entrada a la red son imágenes
- Inconveniente: las imágenes deben tener cierta resolución para contener suficiente información.



- ¿Podemos reducir el número de parámetros de nuestra arquitectura?

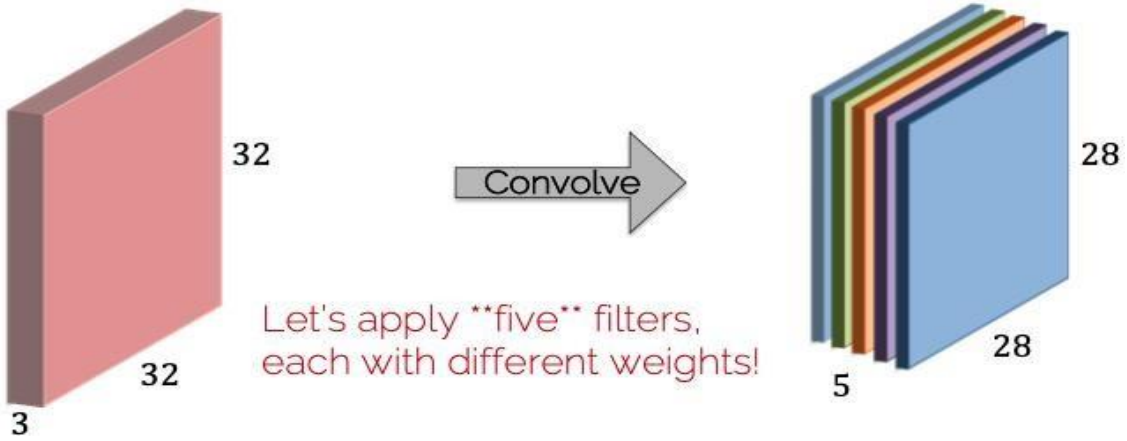
Computer Vision - CNN

- Suposición: La entrada a la red son imágenes
- Idea: Filtro deslizante sobre la imagen de entrada (convolución) en lugar de pasar toda la imagen por todas las neuronas individualmente.



Computer Vision - CNN

- Suposición: La entrada a la red son imágenes
- Filtros: Ventana deslizante con los mismos parámetros de filtro para extraer características de la imagen.
- Ventaja: aprender "conceptos" invariantes de la posición en la imagen y reparto de parámetros



Convolution: Hard-coded

3x3 kernel

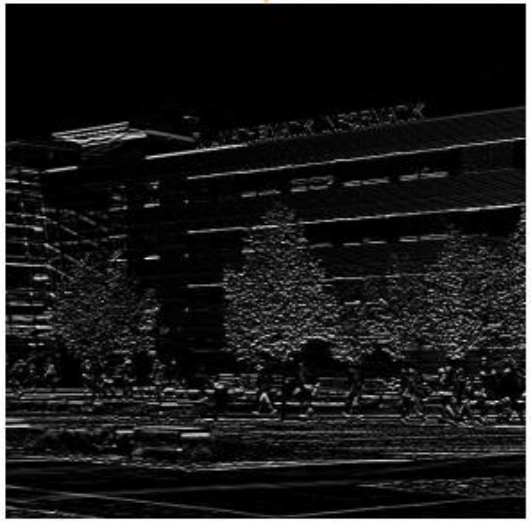
```
[-1, -1, -1]
[ 0,  0,  0]
[ 1,  1,  1]
```

*

*

3x3 kernel

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



Convolutional Layers: BatchNorm y Dropout

Repaso: Batch Normalization

- Batch Norm para redes neuronales FC
  - Tamaño de entrada (N, D)
  - Calcular la media y la varianza de los minibatches en N (es decir, nosotros calcular media/var para cada dimensión de característica)

Input: $x : N \times D$

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

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

Output: $y : N \times D$

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

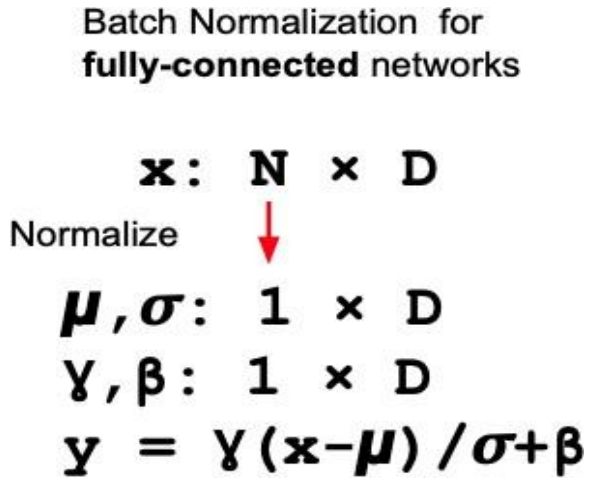
$$\sigma_j^2 = \frac{1}{N} \sum_{i=1}^{N} (x_{i,j} - \mu_j)^2$$

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

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

Repaso: Batch Normalization

- Batch Norm para redes neuronales FC
  - Tamaño de entrada (N, D)
  - Calcular la media y la varianza de los minibatches en N (es decir, nosotros calcular media/var para cada dimensión de característica)



Spatial Batch Normalization

- BatchNorm para Convolutional NN = spacial batchnorm
  - Tamaño de entrada (N, C, W, H)
  - Calcular la media y la varianza de los minilotes en N, W, H (es decir, calculamos la media/var para cada canal C).

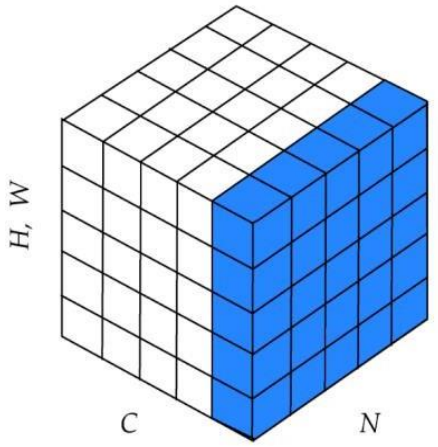
$$x: \ N \times C \times H \times W$$

Normalize

$$\mu, \sigma: \ 1 \times C \times 1 \times 1$$

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

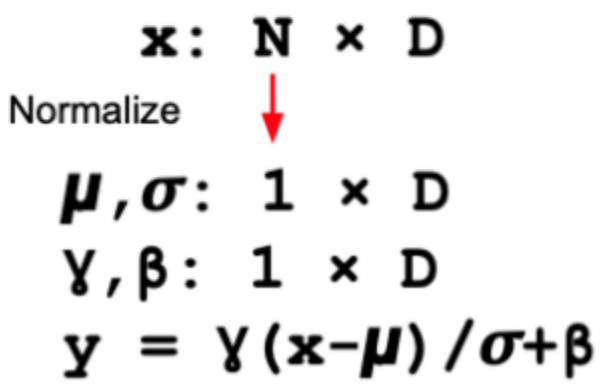
$$y = \gamma(x - \mu)/\sigma + \beta$$



Spatial Batch Normalization

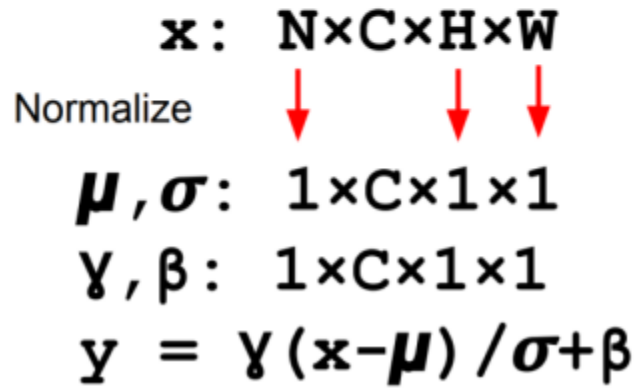
Fully Connected

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



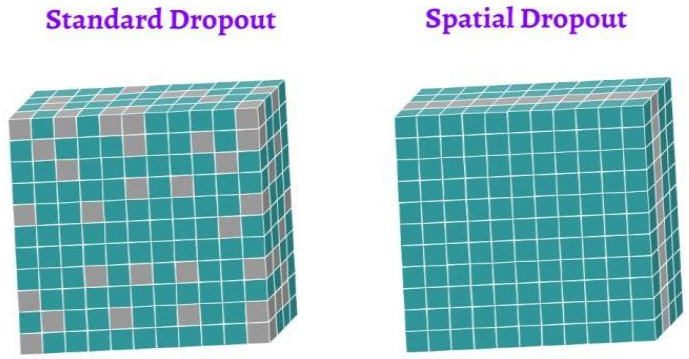
Convolutional = spatial BN

- 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)



Dropout para convolutional layers

- **Regular Dropout**: Desactivación de neuronas específicas en el redes (una neurona "mira" toda la imagen)
- **Dropout Convolutional Layers**: El Dropout estándar a nivel de neurona (es decir, el Dropout aleatorio de una unidad con una probabilidad determinada) no mejora el rendimiento en NN convolucionales.
- **Spatial Dropout**: aleatoriamente pone en cero completos feature maps



Dropout for convolutional layers

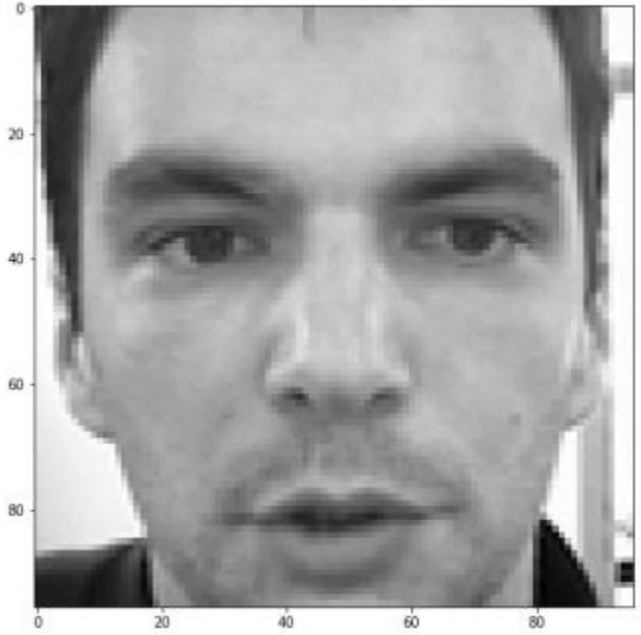
```
def dropout_mlp():
    m = nn.Dropout(p=0.5)
    batch_size = 1
    inputs = torch.randn(batch_size, 3 * 5 * 5)
    outputs = m(inputs)

    print(outputs)

    tensor([[
        -0.89,  0.37, -0.00,  0.00, -0.08, -0.00,
        0.00, -3.55,  0.00,  0.47, -0.00,  5.08,
        -0.00, -0.00,  2.63,  0.00,  0.00,  0.00,
        2.18,  1.92, -0.00,  0.66,  1.96,  0.00,
        -0.00,  -0.00,  0.00,  1.31, -1.95, -0.00,
        0.00, -4.44,  0.00, -1.07, -0.90,  -0.07,
        -3.81,  0.00,  0.23,  2.38,-2.27, -0.51,
        -3.32, -0.00, -0.65,  0.00, -0.00, -0.00,
        -0.00, -0.00, -0.61,  0.00,  0.00,  0.00,
        -1.85, -0.40,  0.00,  0.68, -0.00, -1.96,
        -0.00, -1.65,  0.00, -0.66,  3.10,  0.00,
        -0.00,  1.89,  0.00, -1.28,1.62, -0.56,
        -0.00, -0.00, -0.99]])
```

```
def dropout_cnn():
    m = nn.Dropout2d(p=0.5)
    batch_size = 1
    inputs = torch.randn(batch_size, 3, 5 * 5)
    outputs = m(inputs)

    print(outputs)

    tensor([[
        [ 0.03,  1.40,  1.76, -4.34, -0.63,
         -0.31,  2.80,  2.72, -3.00,  2.67,
          -2.31, -3.45,  0.95,  1.18,  1.18,
          -1.05,  0.74,  3.56,  0.55, -1.19,
          -0.28,  0.89, -3.36, -2.00, -0.29],
        [ 0.00, -0.00, -0.00, -0.00, -0.00,
         0.00, -0.00, -0.00, -0.00,  0.00,
         -0.00,  0.00,  0.00, -0.00, -0.00,
         0.00, -0.00,  0.00,  0.00, -0.00,
         -0.00,  0.00, -0.00,  0.00,  0.00],
        [ 0.00, -0.00, -0.00, -0.00,  0.00,
         0.00,  0.00,  0.00, -0.00, -0.00,
         -0.00, -0.00,  0.00, -0.00, -0.00,
         0.00,  0.00,  0.00, -0.00,  0.00,
         -0.00, -0.00,  0.00,  0.00, -0.00]]])
```

Ejercicio 9:
Facial Keypoint Detection

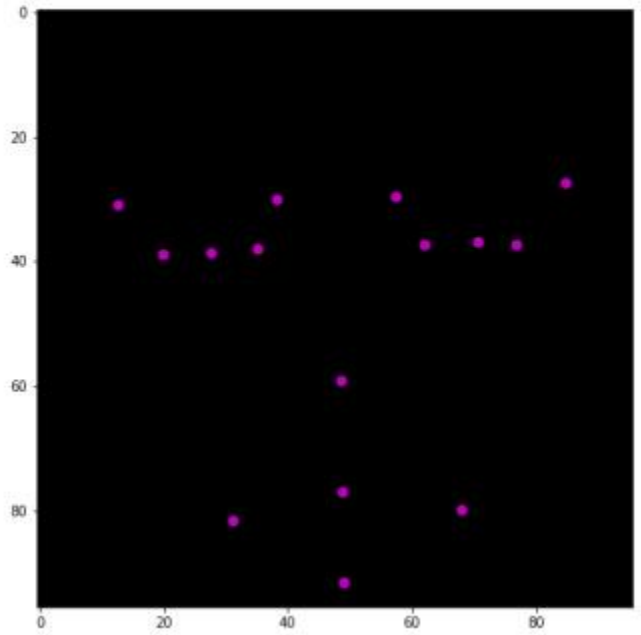
Submission: Facial Keypoints

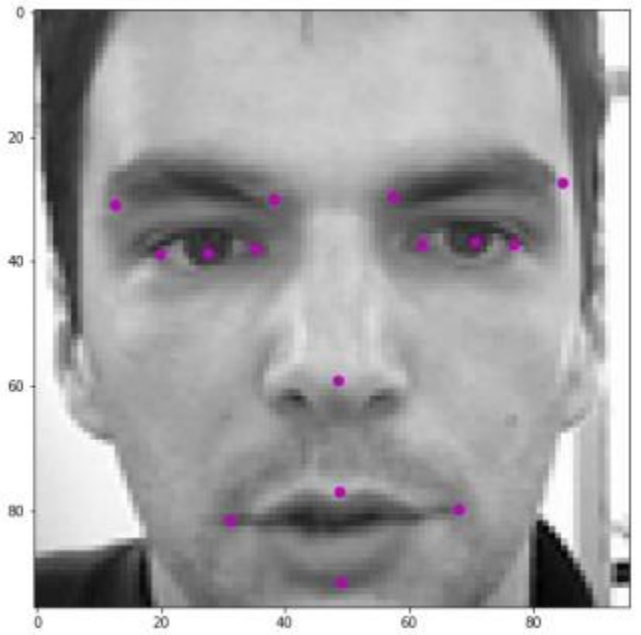
Input:
(1, 96, 96) grayscale image



CNN

Output:
(2, 15) keypoint coordinates





Submission: Metric

Accuracy (Classification) → Score (Regression)

```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 Requirement: Score >= 100

Nos vemos el próximo lunes ☺