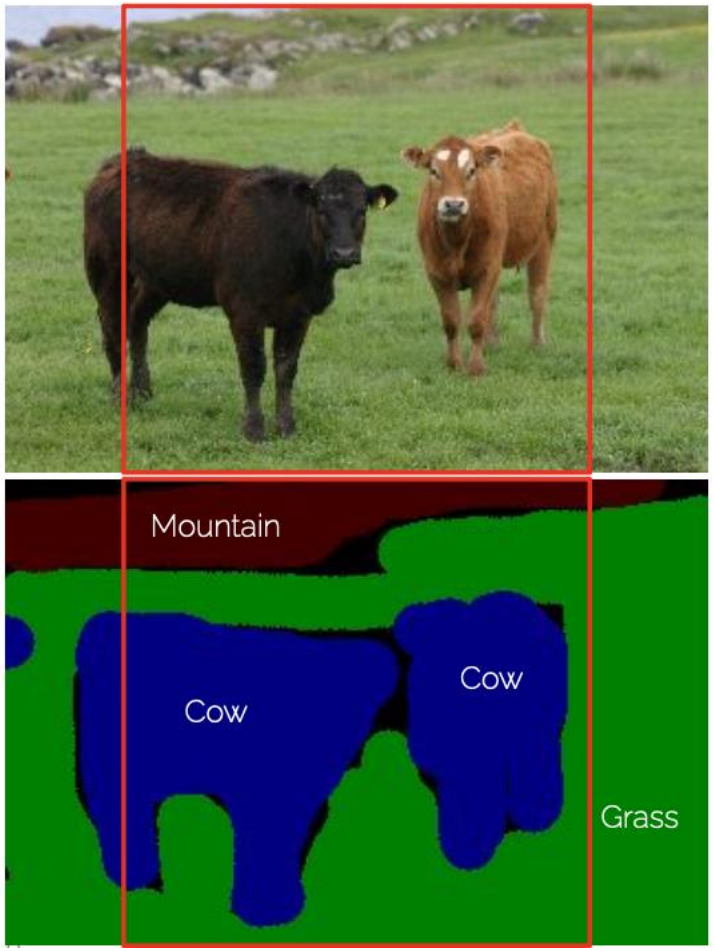
Tutorial 10: Segmentación Semántica

Segmentación Semántica



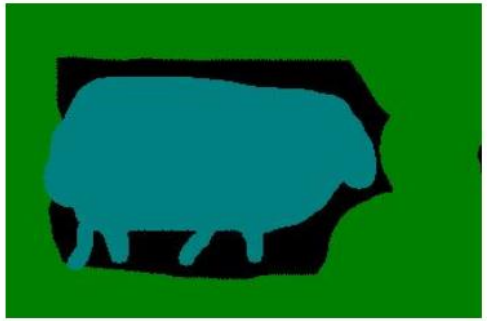
Input:
(3xWxH) RGB image

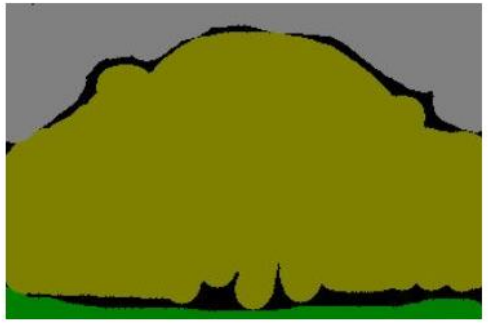
Output:
(23xWxH) segmentation map with scores for every class in every pixel

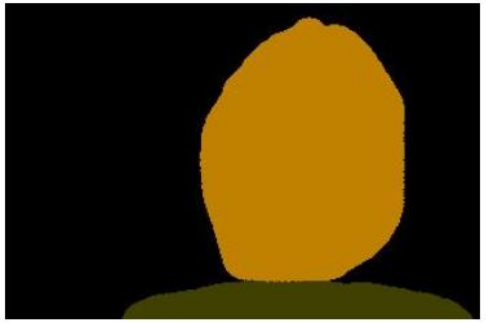
Labels en la Segmentación Semántica

object class	R	G	B	Colour
void	0	0	0	
building	128	0	0	
grass	0	128	0	
tree	128	128	0	
cow	0	0	128	
horse	128	0	128	
sheep	0	128	128	
sky	128	128	128	
mountain	64	0	0	

"void" for unlabelled pixels







Métricas: Loss Function

- Cross-entropy loss promediado por pixel

```
for (inputs, targets) in train_data[0:4]:
    inputs, targets = inputs, targets
    outputs = dummy_model(inputs.unsqueeze(0))
    loss = torch.nn.CrossEntropyLoss(ignore_index=-1, reduction='mean')
    losses = loss(outputs,targets.unsqueeze(0))
    print(losses)
```

Métricas: Accuracy

- Solo se consideran pixeles que no son “void”

```
def evaluate_model(model):
    test_scores = []
    model.eval()
    for inputs, targets in test_loader:
        inputs, targets = inputs.to(device), targets.to(device)

        outputs = model.forward(inputs)
        _, preds = torch.max(outputs, 1)
        targets_mask = targets >= 0
        test_scores.append(np.mean((preds == targets)[targets_mask].data.cpu().numpy()))


    return np.mean(test_scores)
print("Test accuracy: {:.3f}".format(evaluate_model(dummy_model)))
```

Arquitecturas de Modelos

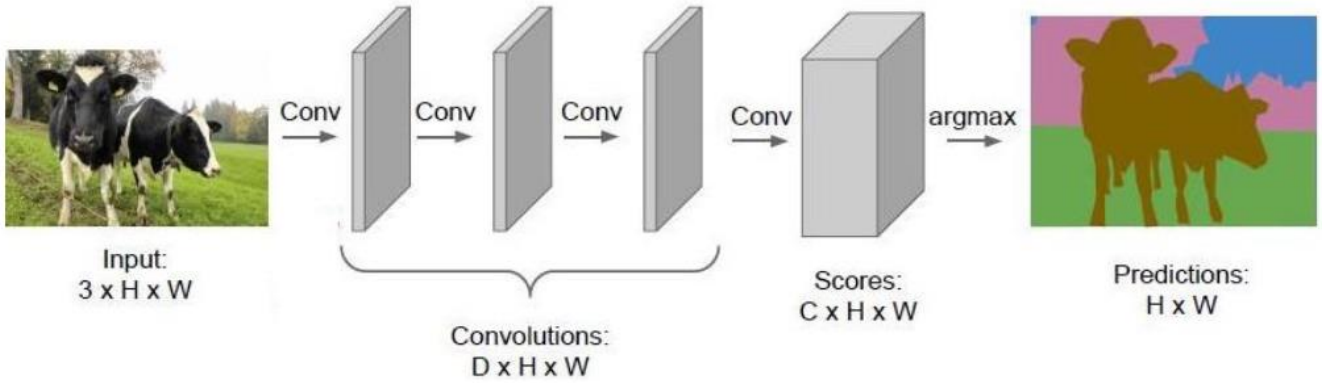
Task de Segmentación Semántica

- Shape del Input: (N, num_channels, H, W)
- Shape del Output: (N, num_classes, H, W)

- Queremos:
  - Mantener la dimensionalidad (H, W)
  - Obtener características en diferentes resoluciones espaciales

Solución Naive

- Mantener la dimensionalidad constante a lo largo de la red
- Usar tamaños de filtro incrementalmente

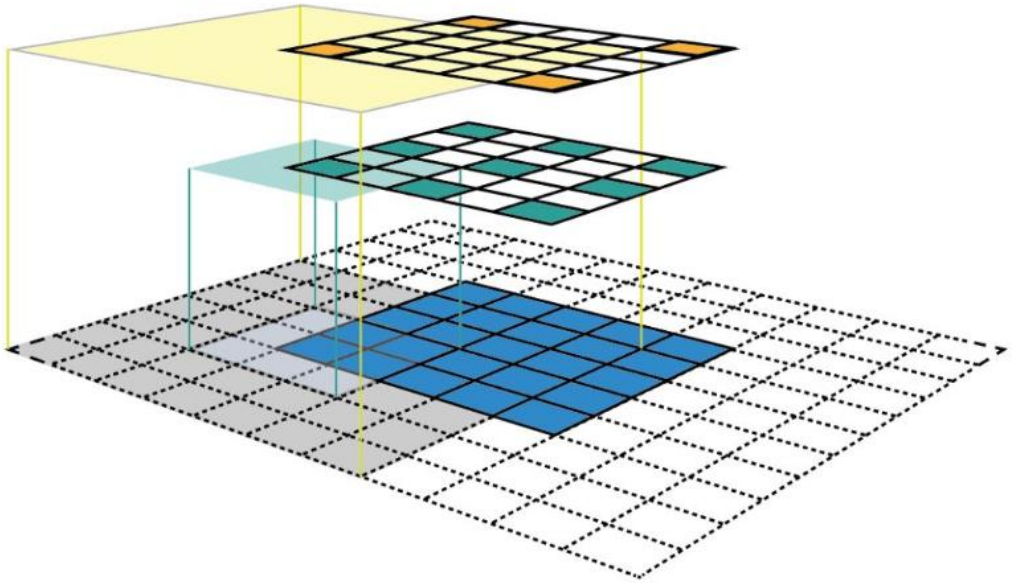


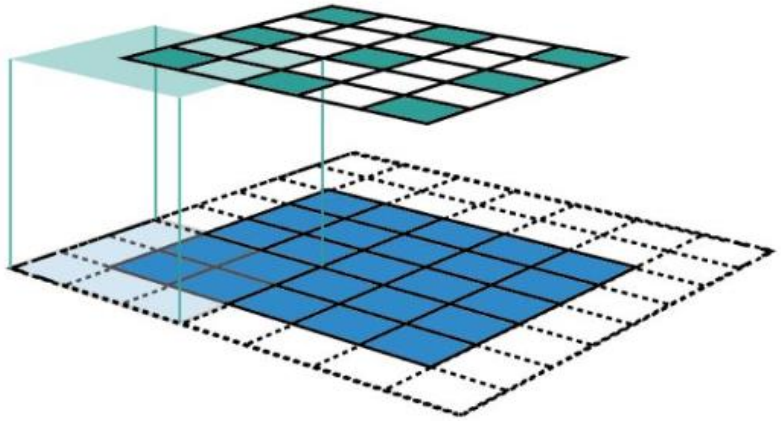
- Problema:
  - Consumo de memoria
    - Por ejemplo, tenemos que guardar inputs y outputs para cada capa:

      128 filtros a (64xWxH) -> millones de parámetros

Entonces: Receptive Field (RF)

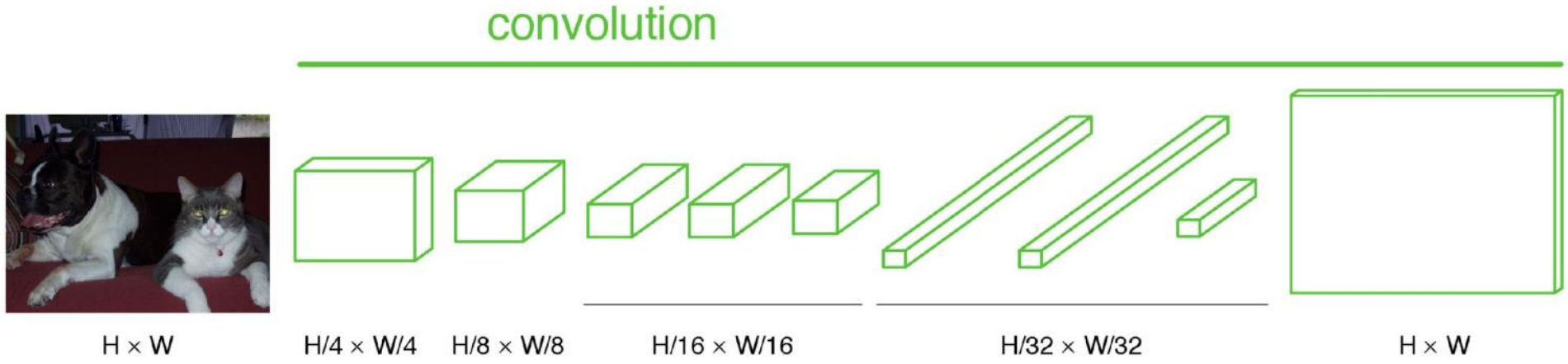
- Región en el input space
- Se reduce dimensionalidad





Como veníamos viendo

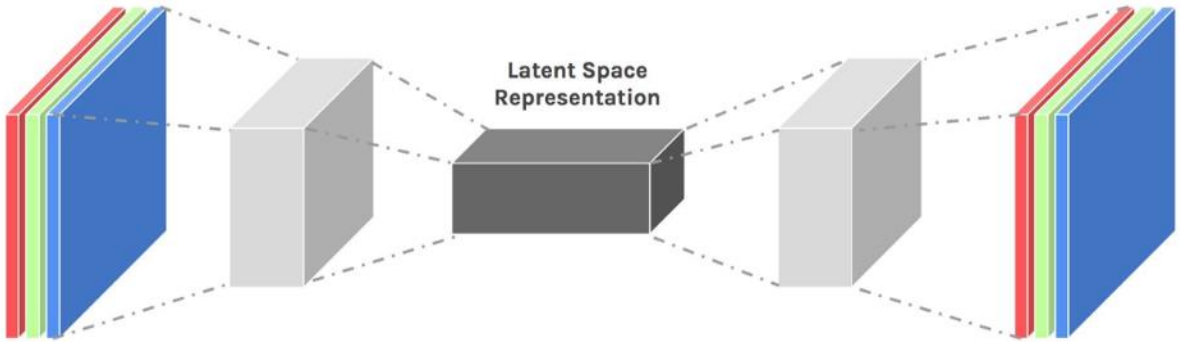
- Usamos convoluciones y pooling para aumentar el receptive field
- Y hacemos un Upsampling al final para recuperar la resolución del input



La mejor solución

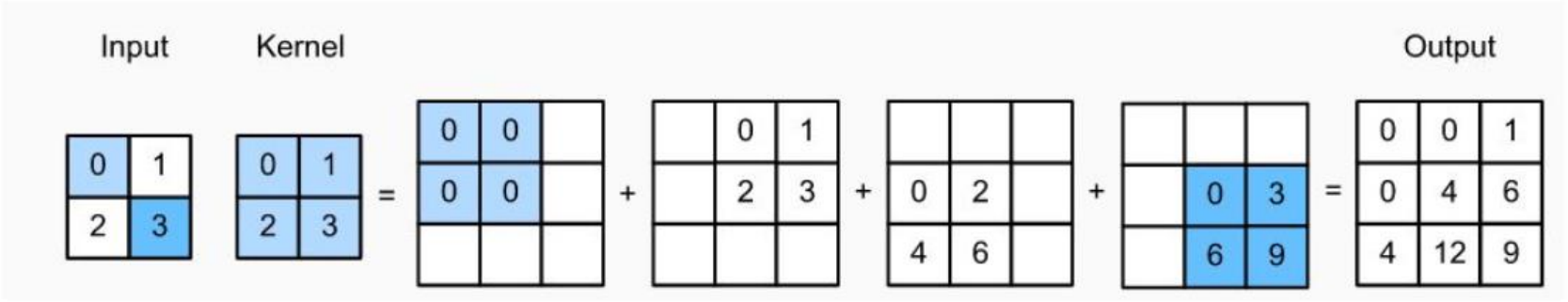
- Reducir el tamaño lentamente -> y luego aumentar el tamaño lentamente
  - Pooling -> Upsampling
  - Strided Convolution -> Transposed Convolution

- Combinar con convoluciones normales, bn, dropout, etc.



Transposed Convolutions

- Upsampling con parámetros entrenables

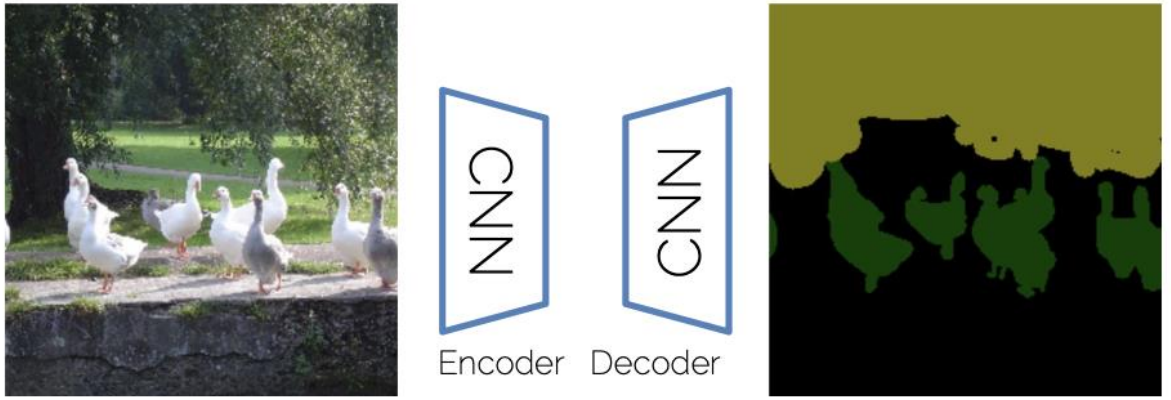


- Cálculo del output en las Transposed Convolutions

$$out = (in - 1) * stride - 2 * pad + kernel$$

Cómo obtener resultados más rápido?

- Transfer Learning



- Posibles soluciones
  - “The Oldschool”
    - Agarrar un Encoder preentrenado, setear el decoder, y solo entrenar el decoder
    - Candidatos de Encoders: AlexNet, MobileNets
  - “The Lazy”
    - Agarrar una red totalmente pre entrenada y ajustar solo los outoputs

Nos vemos el próximo lunes ☺