

Evaluating Neural Network

Beurton-Aimar

September 29, 2025

Results Evaluation

What to do?

- Checking the results at each epoch allows to:
 - verify if the network learns something,
 - stop overfitting,
 - extract measures at different levels.
- Testing the generalization level of the final configuration with the test data set.
- Display and interpret the output values.
- Reconstruct data from output - example of images.

Running a Model

```
import time
def train(model, optimizer, loss_fn, train_dl,
          val_dl, epochs=100, device='cpu'):

    print( 'train() called: model=%s, opt=%s (lr=%f),
    .....epochs=%d, device=%s\n' %
          (type(model).__name__, type(optimizer).__name__,
           optimizer.param_groups[0]['lr'],
           epochs, device))
    history = {} # Collects per-epoch loss and acc.
    history['loss'] = []
    history['val_loss'] = []
    history['acc'] = []
    history['val_acc'] = []
    start_time_sec = time.time()
```

```

for epoch in range(1, epochs+1): # TRAIN AND EVALUATE
    model.train()
    train_loss = 0.0
    num_train_correct = 0
    num_train_examples = 0
    for batch in train_dl:
        optimizer.zero_grad()
        x = batch[0].to(device)
        y = batch[1].to(device)
        yhat = model(x)
        loss = loss_fn(yhat, y)
        loss.backward()
        optimizer.step()
        train_loss += loss.data.item() * x.size(0)
        num_train_correct +=
            (torch.max(yhat, 1)[1] == y).sum().item()
        num_train_examples += x.shape[0]
    train_acc = num_train_correct / num_train_examples
    train_loss = train_loss / len(train_dl.dataset)

```

——— EVALUATE ON VALIDATION SET ———

```
model.eval()
val_loss      = 0.0
num_val_correct = 0
num_val_examples = 0
for batch in val_dl:
    x      = batch[0].to(device)
    y      = batch[1].to(device)
    yhat = model(x)
    loss = loss_fn(yhat, y)
    val_loss      += loss.data.item() * x.size(0)
    num_val_correct += (torch.max(yhat, 1)[1] == y)
    num_val_examples += y.shape[0]
val_acc = num_val_correct / num_val_examples
val_loss = val_loss / len(val_dl.dataset)
```

```

if epoch == 1 or epoch % 10 == 0:
    print( 'Epoch_%3d/%3d, _train_loss:_%5.2f,
    _train_acc:_%5.2f, _val_loss:_%5.2f,
    _val_acc:_%5.2f'%(epoch, epochs, train_loss ,
                        train_acc , val_loss , val_acc))
    history['loss'].append(train_loss)
    history['val_loss'].append(val_loss)
    history['acc'].append(train_acc)
    history['val_acc'].append(val_acc)

```

END OF TRAINING LOOP

```

end_time_sec      = time.time()
total_time_sec    = end_time_sec - start_time_sec
time_per_epoch_sec = total_time_sec / epochs
print()
print( 'Time_total:%5.2f_sec' %(total_time_sec))
print( 'Time_per_epoch:%5.2f_sec'%(time_per_epoch_sec)
return history

```

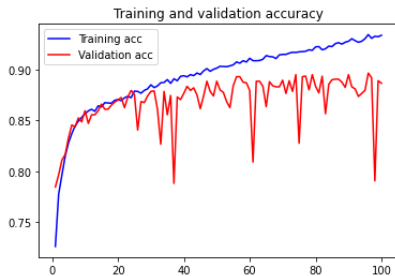
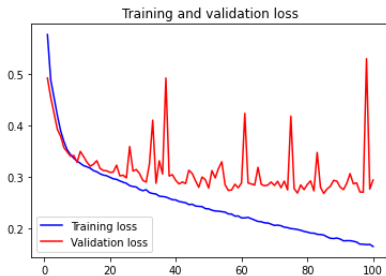
Model Calling

- model is a `torch.nn.Module`
- optimizer is a `torch.optim.Optimizer`

```
history = train(  
    model = model,  
    optimizer = optimizer ,  
    loss_fn = loss_fn ,  
    train_dl = train_loader ,  
    val_dl = val_loader ,  
    device='cuda')
```

Loss/Accuracy Visualization

```
import matplotlib.pyplot as plt
acc = history['acc']
val_acc = history['val_acc']
loss = history['loss']
val_loss = history['val_loss']
epochs = range(1, len(acc) + 1)
plt.plot(epochs, acc, 'b', label='Training_acc')
plt.plot(epochs, val_acc, 'r', label='Validation_acc')
plt.title('Training_and_validation_accuracy')
plt.legend()
plt.figure()
plt.plot(epochs, loss, 'b', label='Training_loss')
plt.plot(epochs, val_loss, 'r', label='Validation_loss')
plt.title('Training_and_validation_loss')
plt.legend()
plt.show()
```

Que peut-on conclure ?

Evaluation Metrics

Precision

- Measures the proportion of True Positive among all positive predictions.

$$\text{Precision} = (\text{True Positive}) / (\text{True Positive} + \text{False Positive})$$

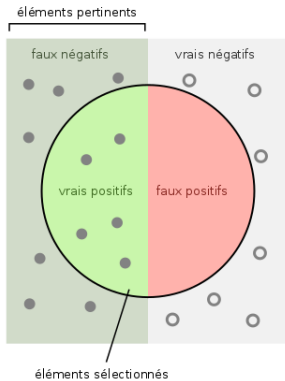
Example: if the model has predicted 100 trees in an image which contains 90 - the precision is equal to 90%.

Recall

- Measures the proportion of True Positive among all items.

$$\text{Recall} = (\text{True Positive}) / (\text{True Positive} + \text{False Negative})$$

Example: if the model has predicted 75 trees in an image containing 100 trees, Recall is equal to 75%



Combien de candidats sélectionnés sont pertinents ?

$$\text{Précision} = \frac{\text{vrais positifs}}{\text{vrais positifs} + \text{faux positifs}}$$

Combien d'éléments pertinents sont sélectionnés ?

$$\text{Rappel} = \frac{\text{vrais positifs}}{\text{vrais positifs} + \text{faux négatifs}}$$

Evaluation Metrics

F1 Score

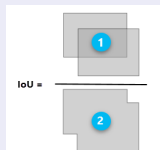
- Harmonic mean of precision and recall, providing a balanced measure of a performance model.

$$\text{F1 Score} = (\text{Precision} \times \text{Recall}) / ((\text{Precision} + \text{Recall}) / 2)$$

Values are contained between 0 and 1 - this score has to be maximized.

IoU

- The ratio of Intersection under Union.
Used as a threshold to set if a result is a true positive or a false positive.



Evaluation Metrics

Average Precision

- Computed on recall values - between 0 and 1.

Mean Average Precision

- Average precision with several threshold.

Example: $\text{mAP}[0.5:0.05:0.95]$ corresponds to an average precision of IoU values between 0.5 and 0.95 with an increment of 0.05 averaged on all classes.

Visualizing Results

Contingency Table

- Presents classifier results.
- Can be used both for binary and multi-class classifier.

		Classe réelle	
		-	+
Classe prédite	-	True Negatives <i>(vrais négatifs)</i>	False Negatives <i>(faux négatifs)</i>
	+	False Positives <i>(faux positifs)</i>	True Positives <i>(vrais positifs)</i>

Matrice de confusion

```
y_true = []  
y_pred = []
```

```
for images, labels in test_loader:  
    images = images.to(device)  
    outputs = model(images)  
    _, predicted = torch.max(outputs, 1)  
    y_pred.extend(predicted.cpu().numpy())  
    y_true.extend(labels.cpu().numpy())
```

```
# Convert lists to tensors for calculation  
y_true_tensor = torch.tensor(y_true)  
y_pred_tensor = torch.tensor(y_pred)
```

Calculating precision, recall, and F1 score using Py

```
TP = ((y_pred_tensor == 1) &  
      (y_true_tensor == 1)).sum().item()
```

```
FP = ((y_pred_tensor == 1) &  
      (y_true_tensor == 0)).sum().item()
```

```
FN = ((y_pred_tensor == 0) &  
      (y_true_tensor == 1)).sum().item()
```

```
precision = TP / (TP + FP) if TP + FP > 0 else 0
```

```
recall = TP / (TP + FN) if TP + FN > 0 else 0
```

```
f1 = 2 * (precision * recall) / (precision + recall)  
    if (precision + recall) > 0 else 0
```

```
print(f'Precision:{ precision}')
```

```
print(f'Recall:{ recall}')
```

```
print(f'F1_Score:{ f1}')
```


Drawing a Contengency Matrix

```
from sklearn.metrics import confusion_matrix
import seaborn as sn
import pandas as pd
# constant for classes
classes = ('T-shirt/top', 'Trouser', 'Pullover',
          'Dress', 'Coat', 'Sandal', 'Shirt',
          'Sneaker', 'Bag', 'Ankle_Boot')
# Build confusion matrix
cf_matrix = confusion_matrix(y_true, y_pred)
df_cm = pd.DataFrame(
    cf_matrix / np.sum(cf_matrix, axis=1)
    [:, None],
    index=[i for i in classes],
    columns=[i for i in classes])
plt.figure(figsize = (12,7))
sn.heatmap(df_cm, annot=True)
plt.savefig('output.png')
```

Drawing a Contengency Matrix



Computer Vision

Context

- Dedicated to semantic segmentation that is considered as a way for the computer to understand what it is seeing.
- Classification is one of the main task.
- Goal: the main object is identified by the computer and labeled.
- Computer is able to know the position of the object.
- Most often a **bounding box** is set by using object contours.
- Segmentation is the process of dividing an image into multiple segments or regions to simplify its representation and make it easier to analyze.

Computer Vision

Semantic Segmentation

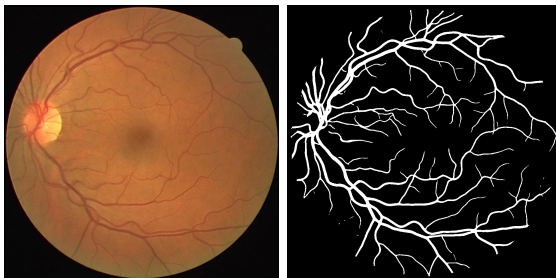
- Segmentation provides context and meaning to individual pixels, transforming raw images into structured data that machines can interpret.
- Segmenting images allows to identify and extract specific objects, delineate boundaries, and even classify regions based on their content.
- Creating models that excel at segmentation is not just about accurate delineation; it's about empowering machines to understand and interpret visual data with precision and efficiency.
- A well-designed segmentation model can significantly enhance the performance of downstream tasks, leading to more robust and intelligent systems^a

^a<https://medium.com/@fernandopalominocobo/mastering-u-net-a-step-by-step-guide-to-segmentation-from-scratch-with-pytorch-6a17c5916114>

Semantic Segmentation

Application

- Medical images analysis: tumor segmentation, disease diagnosis, tissues analysis ...

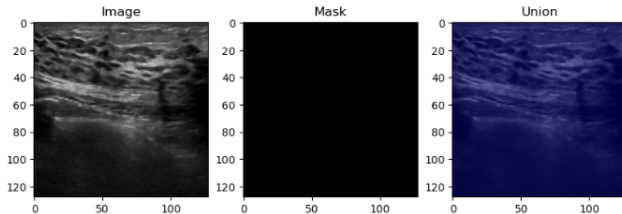


Semantic Segmentation

Application

- Medical images analysis: tumor segmentation, disease diagnosis, tissues analysis ...

Normal class

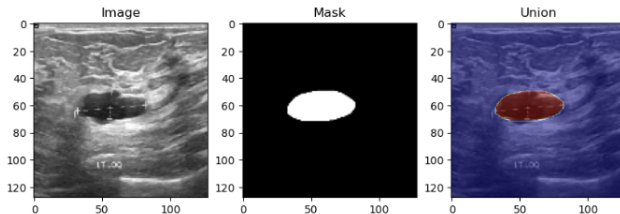


Semantic Segmentation

Application

- Medical images analysis: tumor segmentation, disease diagnosis, tissues analysis ...

Benign class

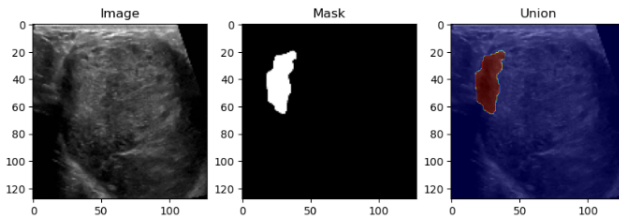


Semantic Segmentation

Application

- Medical images analysis: tumor segmentation, disease diagnosis, tissues analysis ...

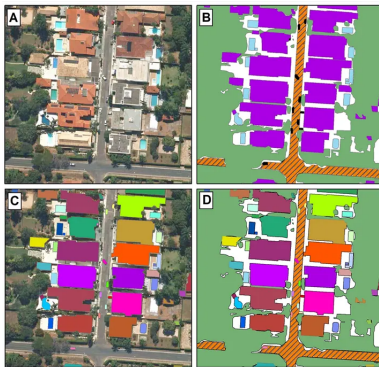
Malignant class



Semantic Segmentation

Application

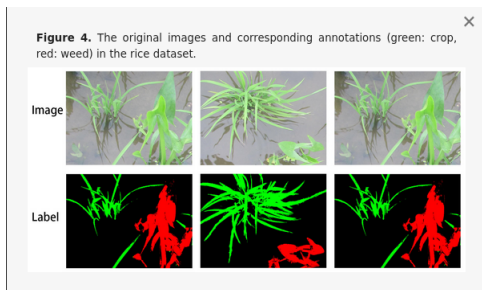
- Object detection: autonomous car, satellite mapmaking,



Semantic Segmentation

Application

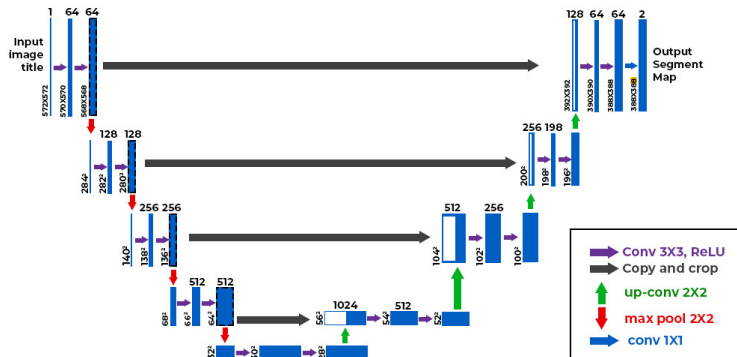
- Robot management: agriculture monitoring



from : Yang et al. “MSFCA-Net: A Multi-Scale Feature Convolutional Attention Network for Segmenting Crops and Weeds in the Field “ (2023)

Segmentation - U-Net

State of the art for semantic segmentation



from : Ronneberger et al. "UNet: Convolutional Networks for Biomedical Image Segmentation" MICCAI (2015)

Segmentation - U-Net

Encoder-Decoder

- CNN architecture dedicated to image segmentation.
- Characterized by its U-shaped structure.
- Encoding path:
 - captures the context of the input image by using a series of convolutional and max-pooling layers to downsample the spatial dimensions. It “contracts” the original images.
- Decoding path:
 - uses upsampling and convolutional layers to produce a segmentation map that has the same spatial dimensions as the input image. It “expands” the contracted images.

Segmentation - U-Net

Encoder-Decoder

- Skip connections:
 - connect the encoding and decoding paths by merging features.
 - help to retain spatial details lost during downsampling, preserving the image's local and global context.
 - By maintaining this spatial information, U-Net achieves more accurate segmentation masks.
- The skip connections assist the network in grasping the relationships between image parts, leading to improved segmentation results.

Segmentation - U-Net

First step - Double convolution

- Repeats at each step, consists on two convolutions of 3x3 followed by ReLU activation,

```
class DoubleConv(nn.Module):
    def __init__(self, in_channels, out_channels):
        super().__init__()
        self.conv_op = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(out_channels, out_channels, kernel_size=3, padding=1),
            nn.ReLU(inplace=True)
        )

    def forward(self, x):
        return self.conv_op(x)
```

Segmentation - U-Net

Encoder path

- Downsample part: Double convolution and Maxpooling,

```
class DownSample(nn.Module):  
    def __init__(self, in_channels, out_channels):  
        super().__init__()  
        self.conv = DoubleConv(in_channels, out_channels)  
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)  
  
    def forward(self, x):  
        down = self.conv(x)  
        p = self.pool(down)  
  
        return down, p
```

Segmentation - U-Net

Skip connections

- Skip connections allow for the fusion of low-level and high-level features.
- Before doing the MaxPooling, the convolutioned tensor is saved.
- That convolutioned tensor is later on concatenated with an upsampled tensor with its own dimension.

Segmentation - U-Net

Decoder

- Upsampling part (decoding path) consists on a deconvolution followed by a double convolution.

```
class UpSample(nn.Module):
    def __init__(self, in_channels, out_channels):
        super().__init__()
        self.up = nn.ConvTranspose2d(in_channels, in_channels//2, kernel_size=2, stride=2)
        self.conv = DoubleConv(in_channels, out_channels)

    def forward(self, x1, x2):
        x1 = self.up(x1)
        x = torch.cat([x1, x2], 1)
        return self.conv(x)
```

Evaluating Segmentation Performance

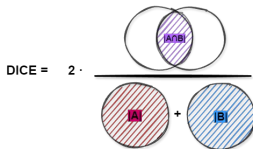
Dice metric

- DICE metric provides a measure of the similarity between two sets.
- In segmentation domain the two sets are: the predicted segmentation and the ground truth segmentation.
- Mathematically, the DICE score is defined as:

$$DICEscore = 2 * |A \cap B| / (|A| + |B|)$$

i.e.





Dice score = $2 * (\text{number of common elements}) / (\text{number of elements in set A} + \text{number of elements in set B})$



Dice Score

- Dice coefficient ranges from 0 to 1 .
- 1 indicates a higher degree of overlap.
- A DICE score of 1 would mean a perfect overlap between the predicted and ground truth segmentations.
- In segmentation task, matrix A could be the predicted mask and matrix B the groundtruth.
- The resulting matrix will have a value of 1 at positions i,j only if both matrix A and matrix B have a value of 1 at that same position i,j .
- **Caution:** in Python the operator `*` multiplies element by element, thus achieving that we want to do. This multiplication **is not a standard matrix multiplication!**

Dice Score

Predicted mask	Reference mask	DICE
		0.989
		0.540

Dice Score

```
epochs_list = list(range(1, EPOCHS + 1))

plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(epochs_list, train_losses, label='Training_Loss')
plt.plot(epochs_list, val_losses, label='Validation_Loss')
plt.xticks(ticks=list(range(1, EPOCHS + 1, 1)))
plt.title('Loss_over_epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.grid()
plt.tight_layout()

plt.legend()

plt.subplot(1, 2, 2)
plt.plot(epochs_list, train_dcs, label='Training_DICE')
plt.plot(epochs_list, val_dcs, label='Validation_DICE')
plt.xticks(ticks=list(range(1, EPOCHS + 1, 1)))
plt.title('DICE_Coefficient_over_epochs')
plt.xlabel('Epochs')
plt.ylabel('DICE')
plt.grid()
plt.legend()

plt.tight_layout()
plt.show()
```

Dice Score

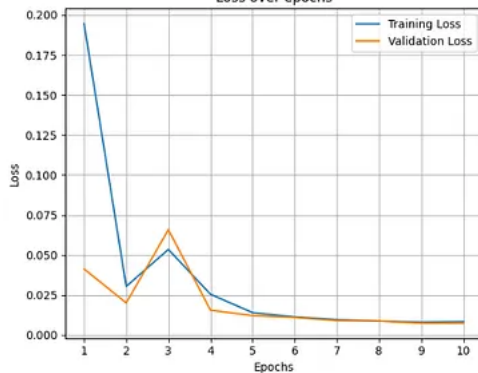
```
epochs_list = list(range(1, EPOCHS + 1))

plt.figure(figsize=(12, 5))
plt.plot(epochs_list, train_losses, label='Training_Loss')
plt.plot(epochs_list, val_losses, label='Validation_Loss')
plt.xticks(ticks=list(range(1, EPOCHS + 1, 1)))
plt.ylim(0, 0.05)
plt.title('Loss_over_epochs_(zoomed)')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.grid()
plt.tight_layout()

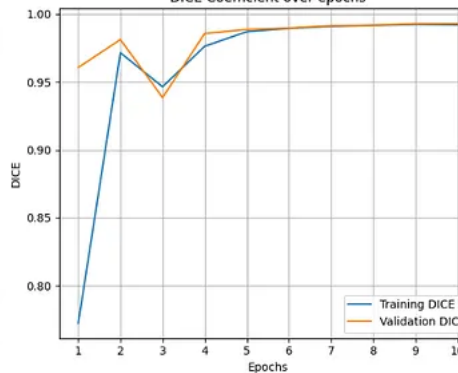
plt.legend()
plt.show()
```

Dice Score

Loss over epochs



DICE Coefficient over epochs



Results

