Student Name: Madhushree Sannigrahi
Instructor: Arnaud PIETROSEMOLI

Deep Learning

Horizon Segmentation Lab Report

Dataset

1. The data used to develop a neural network can be split into 3 categories. What are they?

The overall dataset used to develop a neural network is usually split into 3 parts:

- training data
- validation data
- testing data

2. Detail the role of each data category?

- **Training data** is the primary data used for the training of the model. The model learns patterns and relationships from this data, adjusting its weights and biases to minimize the difference between its predictions and the actual target values. Training data should cover a wide range of scenarios and represent the overall data to ensure the model generalizes well to unseen data.
- **Validation data** is mostly used to test how the model behaves with unseen data. The model is not trained on the validation dataset, however, it is used to tune the hyperparameters and prevent overfitting. This further improves the performance of the model.
- **Testing data** is used to assess the model's performance on completely new and unseen data. The test data is a completely independent set that the model has never encountered during training or validation.

3. How are they distributed? Give approximately the percentages for each category relative to the total data.

- Training data : Around 60-80 % of the total dataset.
- Validation data : Around 10-20 % of the total dataset.
- Testing data : The remaining 10-20 % of the total dataset.

4. We distinguish 3 types of segmentation: Semantic vs. Instance vs. Panopticon. What is the difference between these three types of segmentation?

Semantic segmentation, instance segmentation, and panoptic segmentation are three different image segmentation techniques used in computer vision.

- **Semantic segmentation** assigns each pixel in an image to a predefined class or category. Semantic segmentation is a powerful tool for understanding the layout of a scene and identifying objects in general, without distinguishing between individual instances.
- **Instance segmentation** goes a step further than semantic segmentation by not only classifying pixels but also distinguishing between individual instances of the same class. Instance segmentation is more computationally expensive than semantic segmentation but provides more detailed information about the scene.
- **Panoptic segmentation** combines the best of both semantic and instance segmentation by creating an individual segmentation mask for each object. It assigns a unique identifier to each pixel and the semantic class of the object but it is also the most computationally expensive.

For example: In a traffic perception scenario,

- *semantic segmentation* could be used to classify each pixel as either road, sidewalk, car, pedestrian, or cyclist. This allows for tasks such as understanding the layout of a scene and identifying objects.
- *instance segmentation* could be used to identify and segment individual cars, pedestrians, and cyclists. This allows for tasks such as counting objects and tracking their movements.
- *panoptic segmentation* can be used to label the car in the foreground as çar-0", and the pedestrians crossing the road as "pedestrian-1.ªnd "pedestrian-2". This allows for tasks such as scene understanding, object detection, object tracking, etc. This makes panoptic segmentation the best-suited segmentation for traffic perception.

5. What type of segmentation concerns us for TP? Semantic? Instance? Panopticon?

In our practical, we are working with semantic segmentation of horizon.

6. The data is usually first labelled/annotated in order for the algorithm to understand what the outcome needs to be. For the classification problem, each image is annotated according to its class. For the detection problem, each image is associated with bounding boxes positions and sizes. For the segmentation problem, what form does this labelling take?

For the segmentation problem, the data is typically labelled using pixel-level annotations. This means that each pixel in the image is assigned to a specific class or category.
For example, to segment different types of rocks in a geological sample, each pixel in the image has to be labelled as either quartz, mica, calcite, or background.
There are two main types of pixel-level annotations:

- **Binary segmentation masks:** Masks are simply binary images(objects as 1 and background as 0) that represent the semantic segmentation of an object.
- **Instance segmentation masks:** They not only identify the objects in an image, but they also identify individual instances of the same object.

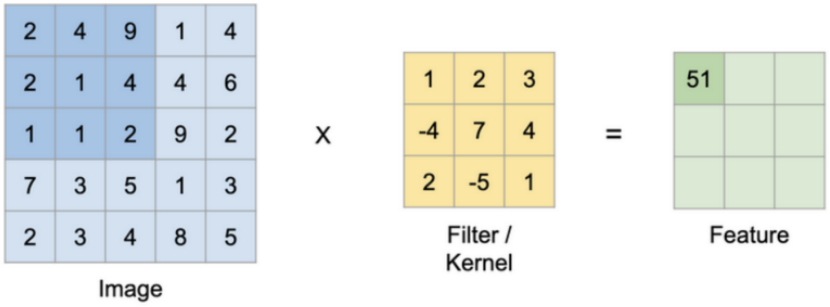
Anonymized | Binary Segmentation | Instance Segmentation

Model

1. What is the main function of the convolution layer?

The main function of the convolution layer in deep learning is feature extraction. It applies filters to input data, allowing the network to learn and detect patterns, edges, and textures, which helps in capturing hierarchical features crucial for tasks like image recognition.

2. Complete the feature matrix by detailing the calculations.



For calculating the 2D Convolution between those two given matrices we will calculate the Hadamard product between the Image patches (deep blue selected part) with the kernel. We will sum all the values of the matrix and we will get the first value of the feature matrix. Then we will loop through the whole matrix patch by patch.
To illustrate this we can do the step-by-step operation for the initial value.
Step 1: We have two matrices for the convolution operation. The first matrix is the kernel projected onto the original matrix. The second matrix is the filter/kernel. For the first case:

$$\text{Image Patch} = \begin{bmatrix} 2 & 4 & 9 \\ 2 & 1 & 4 \\ 1 & 1 & 2 \end{bmatrix} \quad \text{Kernel} = \begin{bmatrix} 1 & 2 & 3 \\ -4 & 7 & 4 \\ 2 & -5 & 1 \end{bmatrix}$$

Step 2: To calculate the Hadamard product between the Image Patch and Kernel, we perform element-wise multiplication:

$$\text{Image Patch}\odot\text{Kernel} = \begin{bmatrix} 2 & 4 & 9 \\ 2 & 1 & 4 \\ 1 & 1 & 2 \end{bmatrix} \odot \begin{bmatrix} 1 & 2 & 3 \\ -4 & 7 & 4 \\ 2 & -5 & 1 \end{bmatrix} = \begin{bmatrix} 2 \cdot 1 & 4 \cdot 2 & 9 \cdot 3 \\ 2 \cdot (-4) & 1 \cdot 7 & 4 \cdot 4 \\ 1 \cdot 2 & 1 \cdot (-5) & 2 \cdot 1 \end{bmatrix} = \begin{bmatrix} 2 & 8 & 27 \\ -8 & 7 & 16 \\ 2 & -5 & 2 \end{bmatrix}$$

3. The convolution layer allows weight sharing. Explain what this involves. What is his interest?

Weight sharing means using the same set of weights across multiple neurons within a particular layer. This means that instead of each neuron having its own unique weights to connect to

neighbouring neurons, it shares the same weights with its counterparts. This sharing strategy effectively reduces the number of parameters by a significant factor, making CNNs more computationally efficient and less prone to overfitting.

4. For deep learning, we stack a significant number of convolution layers, why do we do this?

Stacking convolution layers is a fundamental strategy in deep learning architectures due to its ability to extract progressively higher-level features, learn hierarchical representations, and introduce non-linearities. The hierarchical nature of stacked convolution layers allows the network to capture the underlying structure and patterns. Each layer introduces non-linearity to the system. These factors contribute to the remarkable ability of CNNs to handle complex tasks in image recognition, computer vision, and other domains.

5. What is the point of the batch normalisation layer? Explain the batch normalization process.

Batch normalization (BN) is a regularization technique that aims to improve the training of deep neural networks by normalizing the activations of each layer. This normalization helps to stabilize the training process by reducing internal covariate shift, which is a phenomenon where the distribution of activations changes throughout training. BN also helps to improve the generalization performance of deep neural networks by making them less sensitive to initialization and input distribution.
The batch normalization process can be summarized as follows:

1. **Compute mean and standard deviation**: For each activation in the current layer, compute the mean and standard deviation across the current batch of inputs.
2. **Normalize activations**: Subtract the mean and divide by the standard deviation of each activation.
3. **Scale and shift**: Apply learnable scaling and shifting parameters, gamma and beta, respectively, to the normalized activations.
4. **Output**: Return the scaled and shifted activations.

By normalizing the activations in this way, BN helps to stabilize the training process and improve the generalization performance of deep neural networks.

6. What is the point of the layer normalisation layer? Explain the layer normalization process.

The purpose of Layer Normalization is to address the issue of internal covariate shift, aiming to improve the training stability and convergence of deep neural networks. Internal covariate shift

refers to the change in the distribution of the input to a network's activation functions across training iterations, which can slow down the training process.
Layer Normalization helps in training deeper networks and allows for more straightforward application of gradient-based optimization algorithms.
The layer normalization process involves the following steps for each input feature in a given layer:
1. **Compute the Mean and Variance:**
Calculate the mean μ and variance σ^2 of the input values across the training data.
2. **Normalize the Inputs:**
Normalize the input values x using the mean and variance:

$$\hat{x} = \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

where ϵ is a small constant added for numerical stability, making the distribution of each feature more stable and reducing the internal covariate shift.
3. **Scale and Shift:**
Introduce learnable parameters γ (scaling) and β (shifting):

$$y = \gamma\hat{x} + \beta$$

This step allows the network to learn the optimal scale and shift for each feature.
The layer normalization process is applied independently to each input feature, making it different from batch normalization, which operates over entire mini-batches.

7. Why do we use batch normalization in our case rather than layer normalization?

8. Give some types of activation functions.

Here are some common types of activation functions used in neural networks:

- **Sigmoid:** Outputs values between 0 and 1, suitable for binary classification tasks (e.g., is it a cat or not?).
- **ReLU (Rectified Linear Unit):** Outputs the input directly if positive, otherwise 0. Popular for its speed and efficiency, often used in hidden layers of deep networks.
- **Leaky ReLU:** Similar to ReLU, but allows a small positive gradient for negative inputs, addressing a vanishing gradient issue. Like a leaky faucet, it lets a tiny bit of information through even when negative.
- **Tanh (Hyperbolic Tangent)**: Outputs values between -1 and 1, often used for regression tasks (e.g., predicting house prices).
- **Softmax**: Outputs a probability distribution across multiple classes, commonly used in multi-class classification (e.g., recognizing different handwritten digits).

These are just a few examples, and the choice of activation function depends on the specific task and network architecture.

9. What are activation functions used for?

Activation functions in neural networks are like decision makers. They add non-linearity, allowing the network to learn complex patterns, and prevent vanishing gradients which is a huge training roadblock. We can imagine them as gates, letting only important information flow forward in the network.

Loss function

1. In the formula below of binary cross enentropy , what do $N,\ y_i,\ p(y_i)$ represent? $-\frac{1}{N}\sum_{i=1}^{N}[y_i log(p(y_i)) + (1-y_i) log(1-p(y_i))]$

In the binary cross-entropy formula:

- N represents the total number of samples or data points in the dataset.
- y_i is the true label (ground truth) for the i-th sample, indicating whether it belongs to class 1 (positive) or class 0 (negative).
- $p(y_i)$ is the predicted probability that the i-th sample belongs to class 1 according to the model.

The formula calculates the binary cross-entropy loss, measuring the dissimilarity between the true labels (y_i) and the predicted probabilities ($p(y_i)$) for each sample in the dataset.

2. What is $-y_i log(p(y_i))$ equal to in the following cases?
2.1. For the pixel whose target is equal to 1 and prediction is equal to 1?
2.2. For the pixel whose target is equal to 1 and prediction tend to 0?

In the binary cross-entropy loss term $-y_i \log(p(y_i))$:

- **Case 2.1: Target is 1, and Prediction is 1**

  $$-y_i \log(p(y_i)) = -1 \log(1) = 0$$

  In this case, the loss is 0, indicating a perfect match between the predicted probability and the true label.

- 2.2: Target is 1, and Prediction tends to 0

  $$-y_i \log(p(y_i)) = -1 \log(0)$$

  This expression approaches infinity as $p(y_i)$ approaches 0, indicating a high loss due to a significant mismatch between the predicted probability and the true label.

3. What do $-y_i log(p(y_i))$ represent?

The term $-y_i \log(p(y_i))$ in the binary cross-entropy loss function represents the contribution of a single data point to the overall loss. The negative logarithm $\log(p(y_i))$ penalizes predictions that are far from 1, encouraging the model to assign higher probabilities to true positive instances and lower probabilities to true negative instances.

4. What is $-(1-y_i)log(1-p(y_i))$ equal to in the following cases?
4.1. For the pixel whose target is equal to 0 and prediction is equal to 1?
4.2. For the pixel whose target is equal to 0 and prediction tend to 0?

In the binary cross-entropy loss term $-(1-y_i)\log(1-p(y_i))$, the expression evaluates differently based on the cases you mentioned:

- Case 4.1: For the pixel whose target is equal to 0 and prediction is equal to 1:

  $$-(1-y_i)\log(1-p(y_i)) = -(1-0)\log(0)$$

  . The loss becomes undefined (infinity) because the model incorrectly predicts a probability of 1 for the negative class.

- Case 4.2: For the pixel whose target is equal to 0 and prediction $p(y_i)$ tends to 0:

  $$-(1-y_i)\log(1-p(y_i)) \quad tends \quad to \quad 0$$

  The loss becomes small, indicating a correct prediction for the negative class.

5. What do $-(1-y_i)log(1-p(y_i))$ représent?

The term $-(1-y_i)\log(1-p(y_i))$ in the binary cross-entropy loss function represents the contribution of a single data point to the overall loss. Breaking it down:

- y_i is the true label (ground truth) for the data point, taking a value of 0 or 1.
- $p(y_i)$ is the predicted probability that the data point belongs to class 1 according to the model.

The negative logarithm $\log(1-p(y_i))$ penalizes predictions that are far from 0, encouraging the model to assign lower probabilities to true negative instances and higher probabilities to true positive instances. This term is summed and averaged over the entire dataset to compute the binary cross-entropy loss.

6. What is binary cross entropy supposed to measure?

Binary cross-entropy measures how well predicted probabilities align with true binary labels in a classification task, guiding the model to minimize discrepancies during training.

7. The code uses BCEWithLogitsLoss instead of BCELoss. What is the difference between the two functions? why should we use BCEWithLogitsLoss?

BCELoss (Binary Cross-Entropy Loss) expects raw scores (logits) as input and requires an additional sigmoid activation layer in the network to convert logits to probabilities.
BCEWithLogitsLoss is a numerically stable and efficient implementation of binary cross-entropy loss for raw model outputs(logits).

- **BCELoss requires probabilities as input:** It's typically used with models that already have a sigmoid activation layer to ensure inputs are between 0 and 1.
- **BCEWithLogitsLoss handles logits directly:** It applies the sigmoid function internally, eliminating the need for an extra activation layer and enhancing numerical stability.
- **Enhanced stability and efficiency:** It's less prone to numerical issues and can be slightly faster due to internal optimizations.
- **Gradient clipping compatibility:** It supports gradient clipping techniques, further stabilizing training and preventing exploding gradients.

8. What are the drawbacks of binary cross enntropy?

Drawbacks of binary cross-entropy include:

- Sensitivity to class imbalance.
- Difficulty in handling datasets with noisy or mislabeled examples.
- May struggle with highly imbalanced datasets or rare events.
- Lack of explicit consideration for misclassification costs or varying importance of different instances.

9.Cite other loss functions suitable for image segmentation.

While BCEWithLogitsLoss is effective, several other loss functions offer potential advantages depending on your specific image segmentation task and data characteristics:
For imbalanced classes:

- **Weighted BCE:** Assigns higher weights to underrepresented classes for balanced penalties.
- **Focal Loss:** Focuses on hard-to-classify pixels, reducing sensitivity to outliers.
- **Dice Loss:** Measures overlap between prediction and ground truth, less affected by class imbalance.

For better accuracy and interpretability:

- **IoU Loss (Intersection over Union):** Penalizes based on overlapping area, directly measuring accuracy.
- **Boundary Loss:** Focuses on accurately predicting object boundaries.
- **Lovasz-Softmax Loss:**Optimizes directly for IoU, offering good interpretability.

Training

1. What is the difference between the training phase and the inférence phase?

Training Phase:

- **Goal:** To Learn! Feed the model tons of data and train it to perform a task (eg- classification, regression, etc).
- **Focus:** Adjusting internal weights & connections (absorbing knowledge).
- **Output:** An "trained"model (hopefully!).

Inference Phase:

- **Goal:** Apply your learning! Present new data (the actual test).
- **Focus:** Using learned weights & connections to make predictions (answering the questions).
- **Output:** Predictions based on the new data (test score).

Key Differences:

- **Data:** Training uses large datasets, inference sees single new data points.
- **Goal:** Training builds knowledge, inference uses that knowledge.
- **Output:** Training refines the model, inference produces predictions.

2. The training task of a network is composed of gradient descent and back propagation. What is the goal of gradient descent?

In a CNN, gradient descent serves as the optimizer, guiding the training process towards better performance. Its goal is to minimize the loss function, which measures how well the network's predictions align with the actual data. The lower the loss, the better the network learns the underlying patterns in the data and makes accurate predictions.

3. What is the goal of back propagation?

4. The new weights are computed with previous weights and gradients. Give the relation between the new weigth, the previous weights and the cost function $Q : w \to Q(w)$.

The new weights in a neural network are indeed computed based on the previous weights, gradients, and the cost function. Here's the relationship:
New weight (w_new) = previous weight (w_old)- learning_rate * gradient
where:

- **w_new** is the updated weight after the training step.
- **w_old** is the weight before the update.
- **learning_rate** is a hyperparameter that controls the step size of the update.
- **gradient** is the partial derivative of the cost function Q with respect to the weight w_old, which indicates how much changing that weight will affect the overall cost.

This update rule essentially moves the weight in the direction opposite to the gradient, with the size of the step determined by the learning rate. Gradients that point towards higher cost values have a negative sign, so subtracting them pushes the weight in the direction that reduces the cost.

5. Apply the gradient descent algorithm to the following simple function. Detail the calculation steps. $f : x \to 3x^2 + 5x + 4$

Applying the gradient descent algorithm to minimize the given function $f(x) = 3x^2 + 5x + 4$, where the update rule is defined as:

$$x_{\text{new}} = x_{\text{old}} - \alpha \cdot \nabla f(x_{\text{old}})$$

Here, $\nabla f(x)$ represents the gradient of $f(x)$ with respect to x, and α denotes the learning rate.
Initialization:

- Choose a different initial guess for x_{old}.
- Select a learning rate α.

Gradient Calculation:

$$\nabla f(x) = \frac{df}{dx} = 6x + 5$$

Parameter Update:

$$x_{\text{new}} = x_{\text{old}} - \alpha \cdot (6x_{\text{old}} + 5)$$

Iteration Process: Repeat steps 2 and 3 until convergence or a specified number of iterations. For the first iteration with a different initial guess $x_{\text{old}} = -2$ and learning rate $\alpha = 0{,}01$:

Iteration 1:

$$\nabla f(-2) = 6(-2) + 5 = -7$$

$$x_{\text{new}} = -2 - 0{,}01 \cdot (-7) = -1{,}93$$

Iteration 2:

$$\nabla f(-1{,}93) = 6(-1{,}93) + 5 = -6{,}58$$

$$x_{\text{new}} = -1{,}93 - 0{,}01 \cdot (-6{,}58) = -1{,}8742$$

Iteration 3:

$$\nabla f(-1{,}8742) = 6(-1{,}8742) + 5 = -6{,}2432$$

$$x_{\text{new}} = -1{,}8742 - 0{,}01 \cdot (-6{,}2432) = -1{,}8118$$

Subsequent iterations repeat steps 2 and 3 using x_{new} as the updated x_{old} until convergence or reaching the maximum number of iterations. Adjust the learning rate for convergence and stability.

6. In deep learning, gradient descent is said to be stochastic. Explain why we call the algorithm stochastic?

In deep learning, gradient descent is referred to as *stochastic* due to the random or probabilistic nature introduced during the training process. This term emphasizes that the algorithm does not rely on the entire dataset for computing the gradient during each iteration but rather operates on a subset of the data. Some of the major reasons for stochasticity is:

1. **Computational Efficiency:**
   - Processing the entire dataset in each iteration can be computationally expensive, especially for large datasets.
   - Stochasticity allows the model to make frequent updates based on smaller batches, speeding up the training process.
2. **Avoidance of Local Minima:**
   - Stochasticity introduces variability in the updates, helping the algorithm to escape local minima and explore a larger portion of the parameter space.
   - This exploration improves the chances of finding a more globally optimal solution.
3. **Regularization Effect:**
   - The inherent noise introduced by using different mini-batches in each iteration can act as a form of regularization, preventing overfitting and enhancing generalization.
4. **Parallelization:**
   - Stochastic updates enable parallel processing of mini-batches, facilitating distributed and parallel computing, which is essential for training large models on parallel hardware.

7. What is the batch size?

The batch size in the context of deep learning refers to the number of training examples utilized in one iteration. During each iteration of the training process, the neural network processes a subset of the entire training dataset, and this subset is the batch.

8. Why is the batch size limited in practice?

The batch size in deep learning is limited due to constraints in memory availability, computational efficiency considerations, and the need to balance stochasticity for effective optimization and generalization. Smaller batch sizes allow for more frequent updates and exploration of the parameter space but may result in increased noise during training. Memory limitations, learning rate sensitivity, and the trade-off between parallelization and gradient estimation also influence the practical choice of batch size. Experimentation is crucial to finding an optimal batch size that balances these factors for a given model and dataset. Common batch sizes typically range from a few samples to larger values, depending on specific requirements and hardware resources.

9. What is the epochs number?

The number of epochs is the total count of passes the entire training dataset undergoes during the training of a machine learning model.

10. What is the steps per epochs?

The steps per epoch refers to the number of batches processed per complete iteration through the training dataset during one epoch in machine learning training.

11. What is the relation between the size of training dataset, the batch size and the steps number per epoch?

In short, dataset size and batch size determine steps per epoch:

- **Larger dataset:** Allows larger batch sizes without seeing all data per epoch.
- **Smaller dataset:** Needs smaller batches to avoid overfitting.
- **Steps per epoch:** Either automatically calculated or manually set to limit batches per epoch.

The relationship between the size of the training dataset (N), the batch size (B), and the steps number per epoch (S) is given by the formula:

$$S = \frac{N}{B}$$

This formula reflects how many batches are needed to complete one pass through the entire dataset during one epoch of training.

12. What is the effect of the learning rate on the evolution of the cost function?

The learning rate affects the evolution of the cost function during training by determining the size of the steps taken in the parameter space;
Learning Rate Impact on Cost Evolution:

- - Determines step size in parameter space during optimization.
- - Too high may cause divergence, overshooting; too low may result in slow convergence or getting stuck.
- - Crucial for stable and efficient model training.

The choice of an appropriate learning rate is crucial for achieving efficient and stable model training.

13. What is the effect of the batch size on the evolution of the cost function?

Batch Size Impact on Cost Evolution:

- - Influences frequency of parameter updates during training.
- - Larger batches provide smoother convergence but may converge to suboptimal solutions.
- - Smaller batches can lead to faster convergence but introduce more noise and variability.
- - Involves a trade-off between computational efficiency and convergence behavior.

14. What happens when the number of epochs is too large?

When the number of epochs is too large:

- Risk of overfitting increases.
- Model may memorize noise and specifics in the training data.
- Reduced generalization to new, unseen data.
- Increased computational costs without significant learning benefits.
- Regularization techniques and validation monitoring can address overfitting concerns.