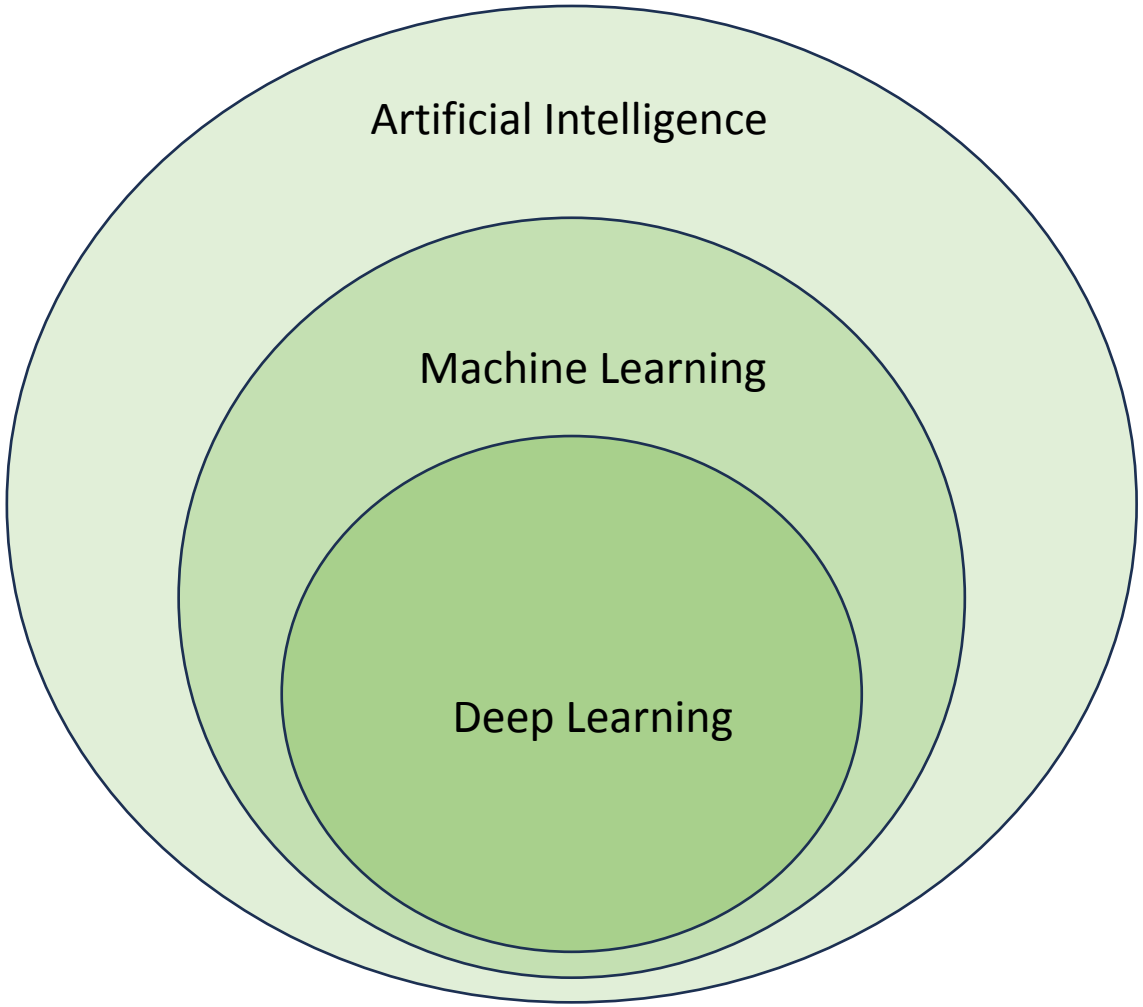
Deep Learning for Computer Vision

Aishwarya Venkataramanan

What is Deep Learning?



AI - Simulation of human intelligence in machines that are capable of performing tasks that typically require human cognitive processes.

ML - A subset of AI that focuses on creating algorithms that enable machines to learn from data.

DL - A subfield of machine learning that utilizes neural networks with multiple layers to learn hierarchical representations of data.

Why do we need Deep Learning?

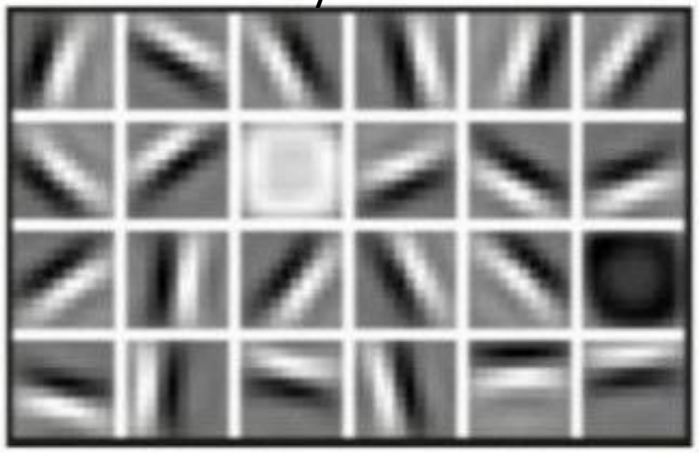
- **Feature Learning** - Automatically learns the relevant features from the data, reducing the need for manual feature engineering.

- **Complex Pattern Recognition -** Excels at recognizing intricate patterns and relationships in data.

- **Representation Learning** - Creates hierarchical representations of data, enabling models to learn features at different levels of abstraction.

Hierarchy of Features Learnt by Deep Neural Networks

Low Level Features – Shallow Layers



Lines & Edges

Mid Level Features – Mid Layers

Eyes & Nose & Ears

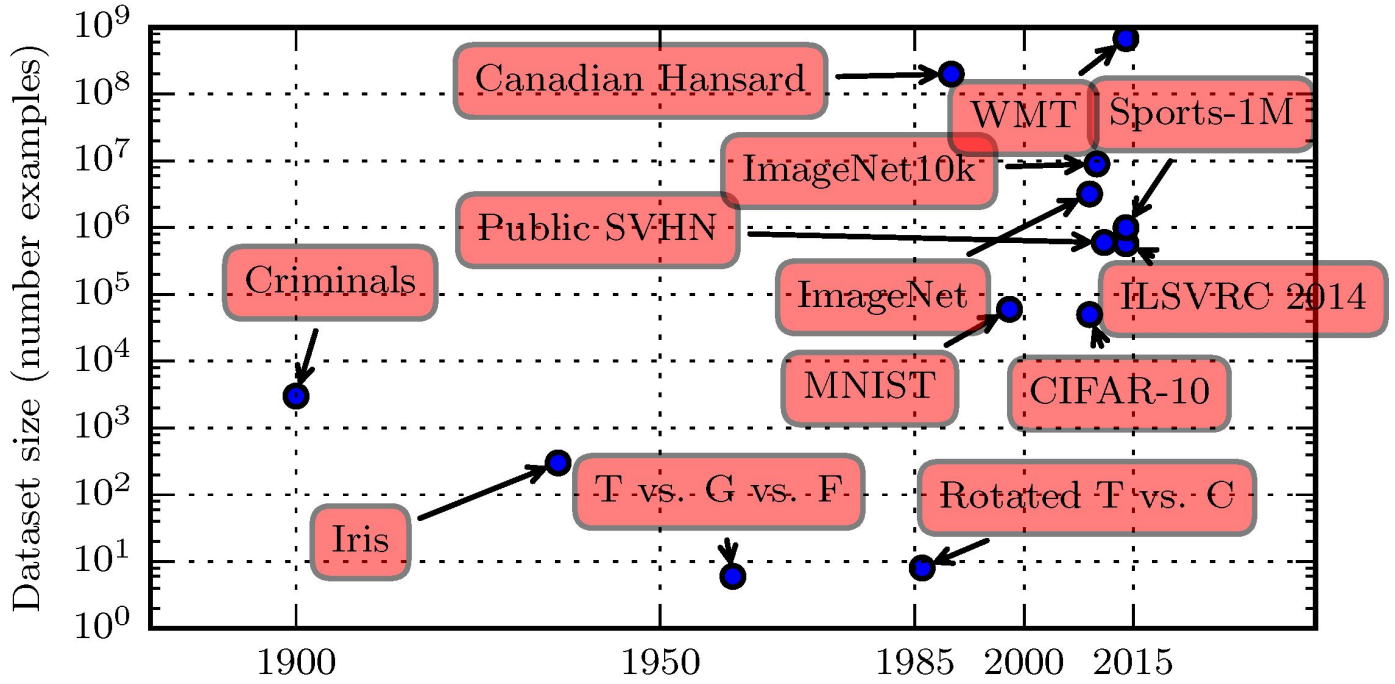
High Level Features – Deep Layers

Facial Structure

Emergence of Deep Learning

Neural Networks have been existing since 1950s. The recent rise in deep learning can be attributed to 3 main reasons:

1. Large Dataset Sizes



ImageNet – 14 million images

Current Large Language Models are trained on images and texts scrapped from the web.

E.g. DALLE-2 – trained on approximately 650 million image-text pairs.

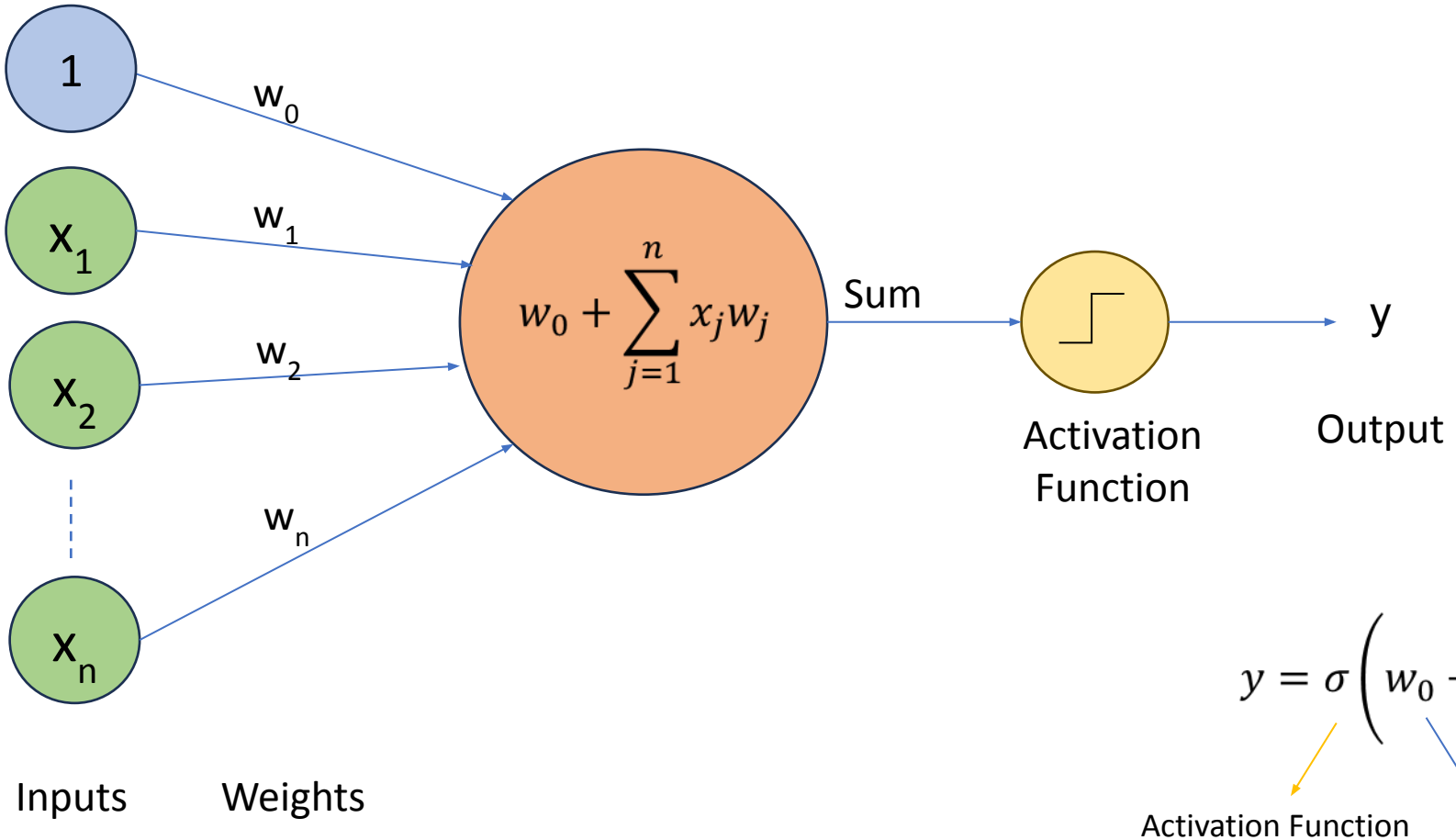
2. Improved Hardware Capabilities

- The development of powerful GPUs (Graphics Processing Units) enabled researchers to train deep neural networks more efficiently.
- GPUs excelled at parallel processing, a crucial aspect of training neural networks.

3. Software

- Software libraries like TensorFlow and PyTorch provide a range of features and capabilities that make it easier for researchers and developers to design, train, evaluate, and deploy complex models.
- They offer pre-built components and models, enabling quick experimentation.

Neurons - Building Blocks of Neural Nets

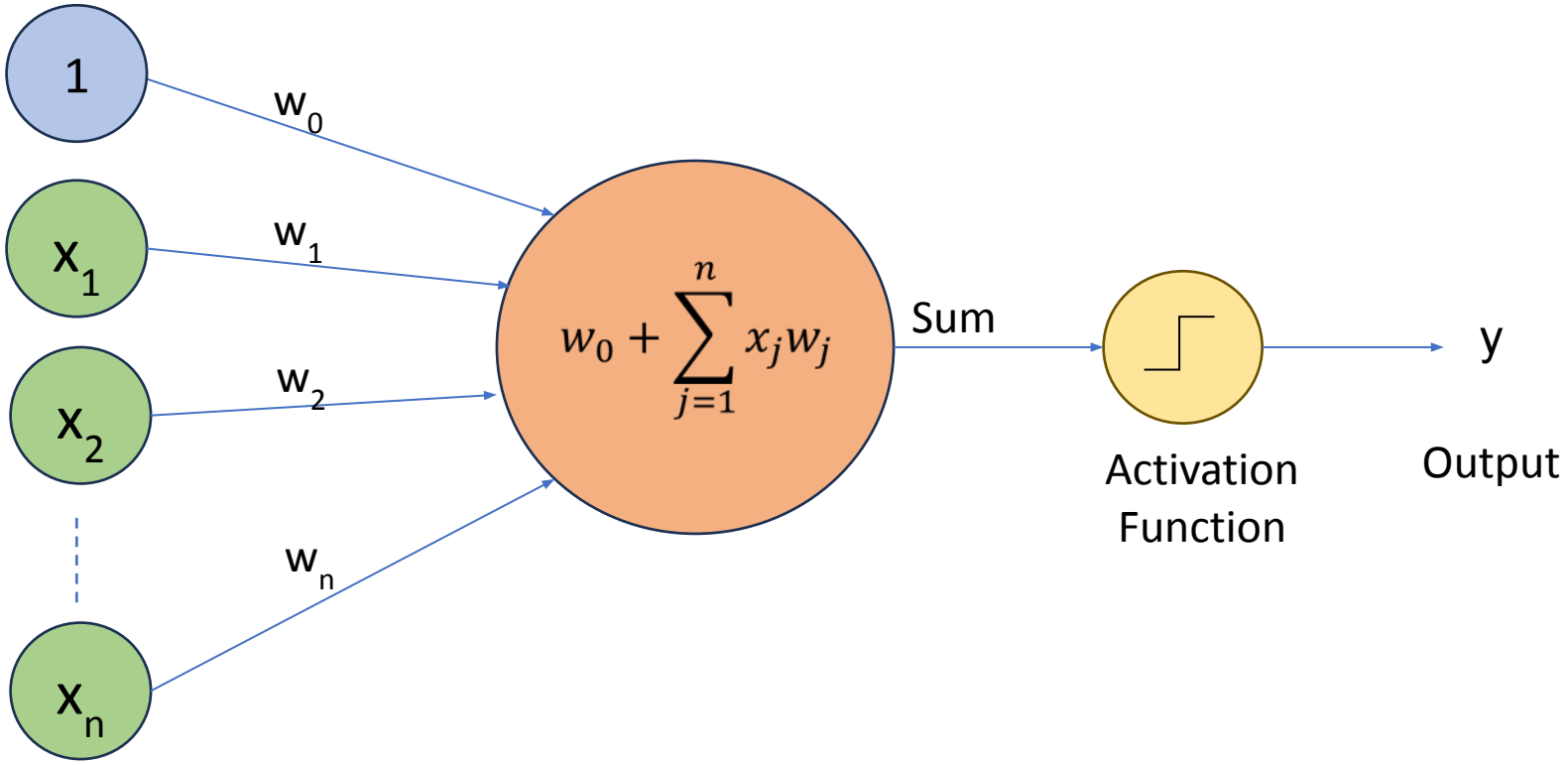


$$y = \sigma\left(w_0 + \sum_{i=1}^{n} x_i w_i\right)$$

Activation Function (Non-linear function)

Bias

Linear Combination of Inputs



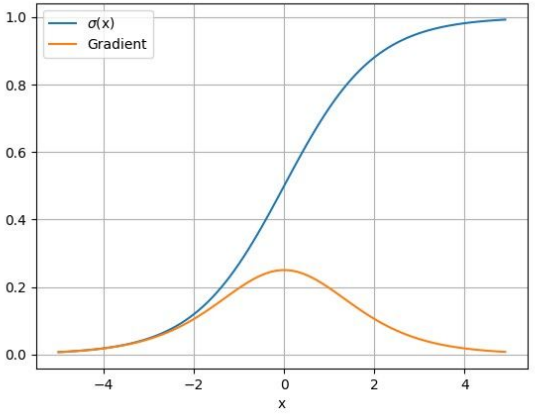
$$y = \sigma(w_0 + \mathbf{X}^{\mathrm{T}}\mathbf{W})$$

$$\boldsymbol{X} = \begin{bmatrix} x_0 \\ \vdots \\ x_n \end{bmatrix} \qquad \mathbf{W} = \begin{bmatrix} w_0 \\ \vdots \\ w_n \end{bmatrix}$$

Activation Functions

- Introduces non-linearity in the neuron output which allows the network to learn complex representations

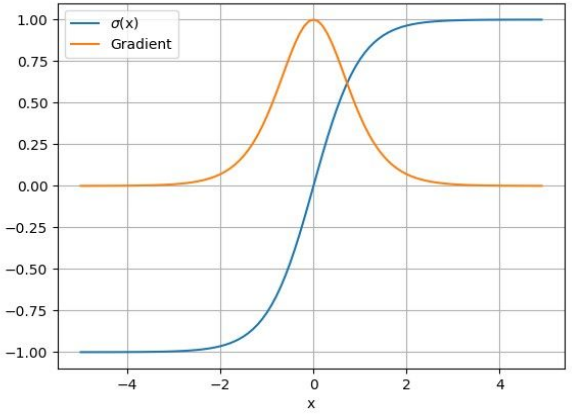
Sigmoid



$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$

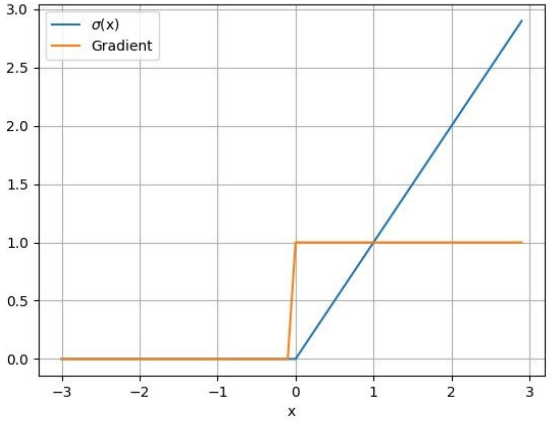
Hyperbolic Tangent (Tanh)



$$\sigma(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

$$\sigma'(x) = 1 - \sigma(x)^2$$

Rectified Linear Unit (ReLU)



$$\sigma(x) = \max(0, x)$$

$$\sigma'(x) = \begin{cases} 1, & x > 0 \\ 0, & x \leq 0 \end{cases}$$

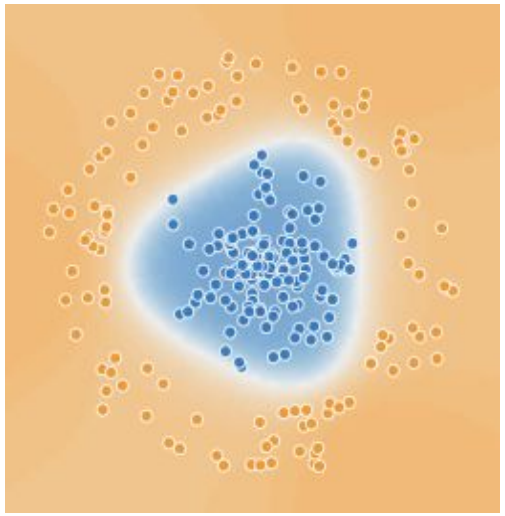
Other commonly used activation functions

- Leaky ReLU
- Parametric ReLU
- ELU
- SELU
- SiLU
- Swish

Simple example demonstrating the importance of activation functions:
We want to build a neural network for binary classification that can distinguish between the orange and blue points.

No Activation Function

Sigmoid Activation

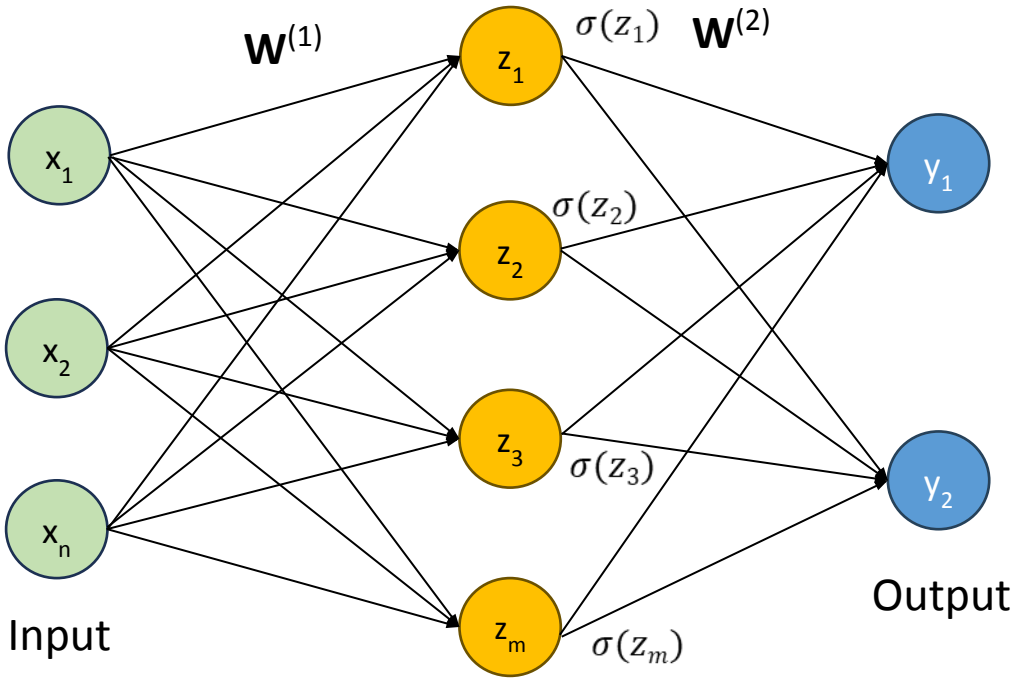


The data is non-linear and a linear 2D line cannot accurately distinguish the two classes. Whereas, when we introduce non-linearity through a sigmoid activation function, the network learns a non-linear function to distinguish the two classes effectively.

Try it on https://playground.tensorflow.org/

Single Layer Neural Network

Commonly referred to as dense layer / linear layer

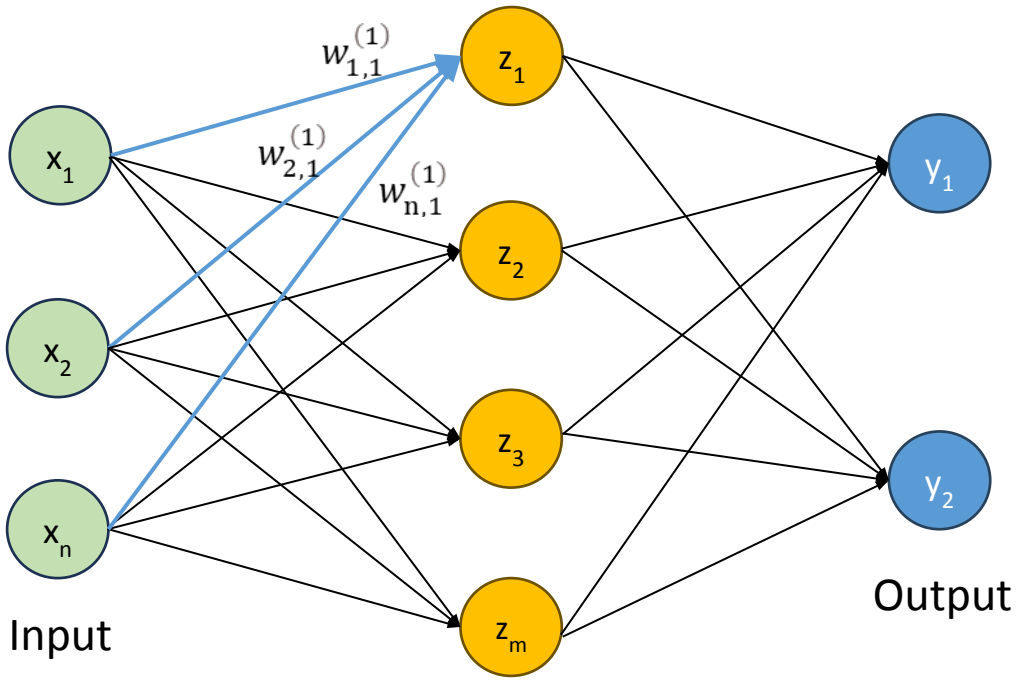


Hidden layer

$$z_i = w_{0,i}^{(1)} + \sum_{j=1}^{n} x_j w_{j,i}^{(1)}$$

Final Output

$$y_i = \sigma\left(w_{0,i}^{(2)} + \sum_{j=1}^{m} \sigma(z_j) w_{j,i}^{(2)} \right)$$

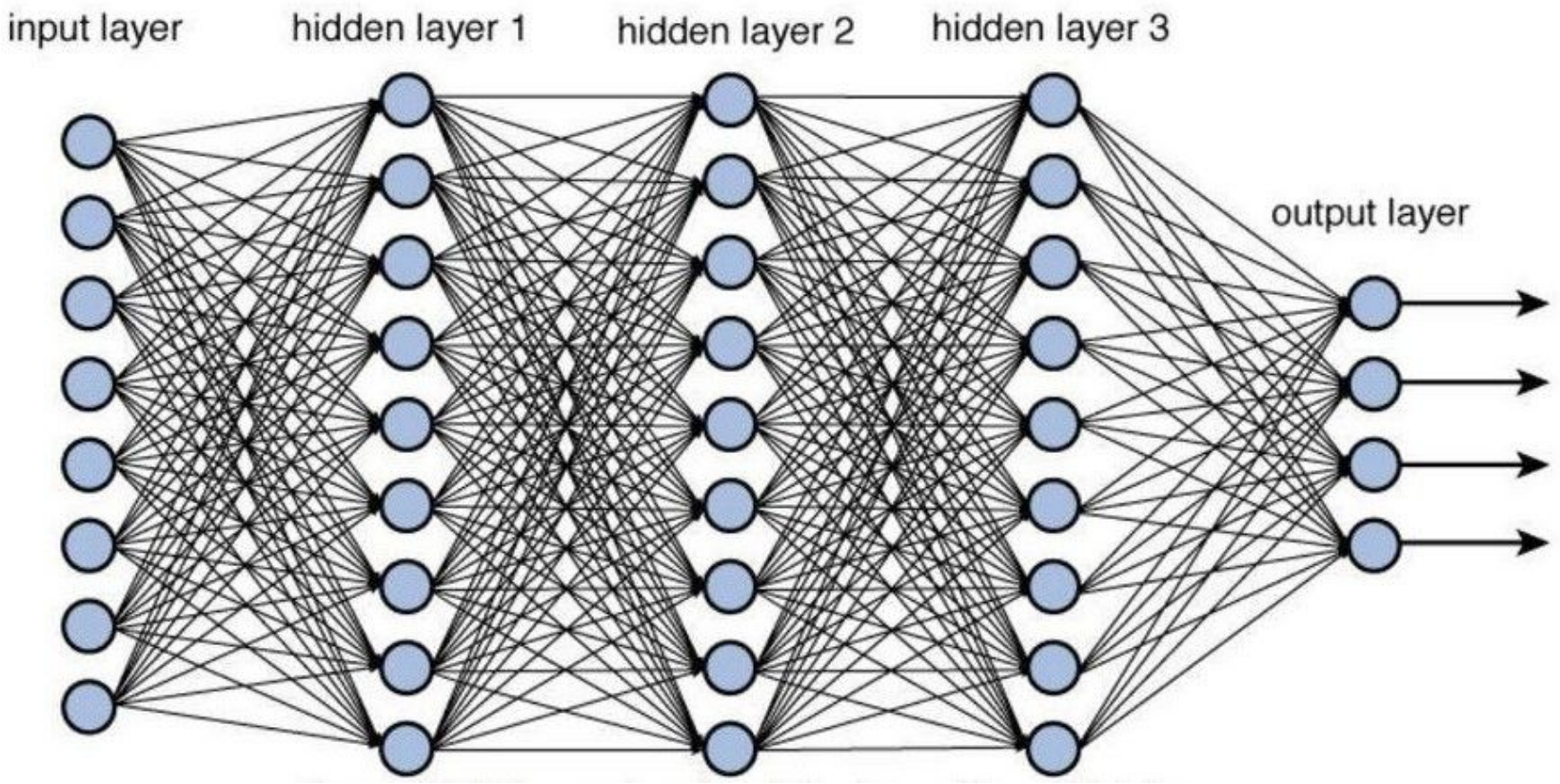


$$z_1 = w_{0,1}^{(1)} + \sum_{j=1}^{n} x_j w_{j,1}^{(1)}$$

$$= w_{0,1}^{(1)} + x_1 w_{1,1}^{(1)} + x_2 w_{2,1}^{(1)} + x_n w_{n,1}^{(1)}$$

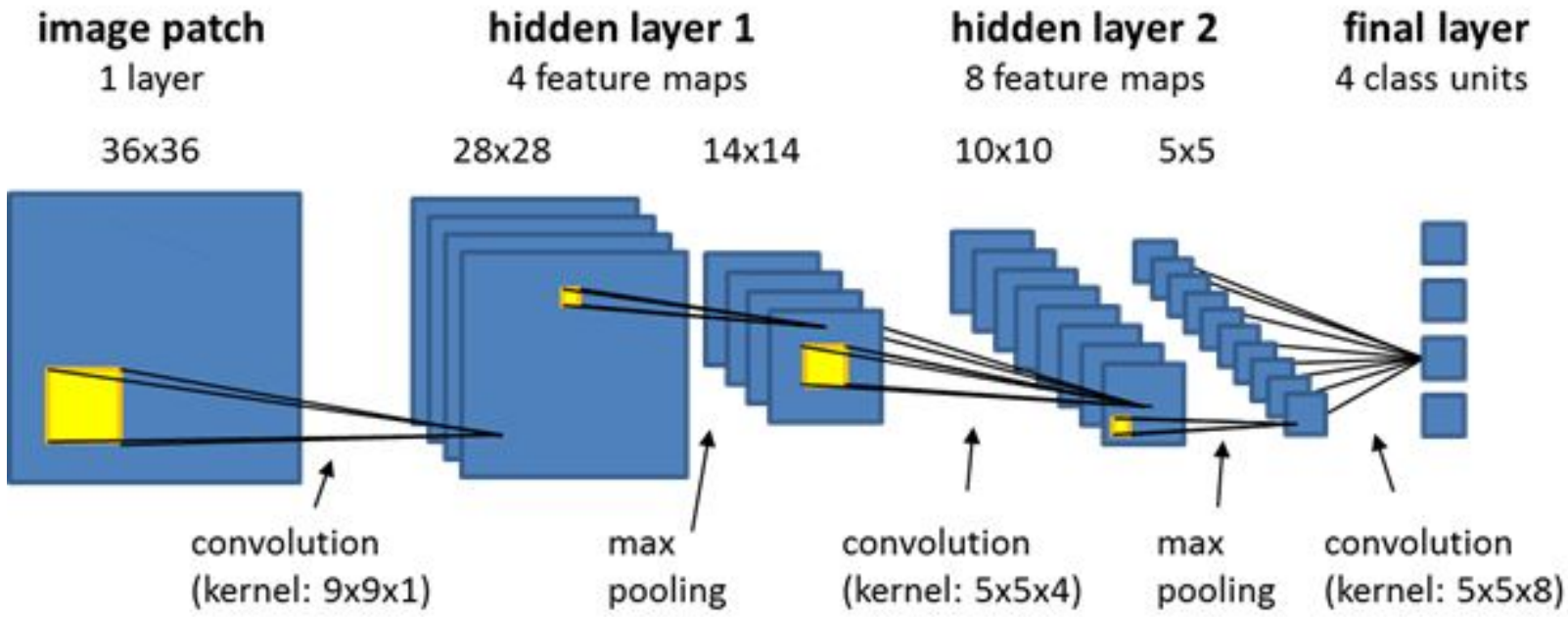
Deep Neural Network

Contains several dense layers connected together



Convolutional Neural Networks (CNNs)

Used to connect neurons to grid-structure inputs such as images, videos.



Convolution

- A convolution is a linear operation between two functions.
- In images, the functions are discrete and defined over 2D spaces.
- The convolution is written as follows: $(f * g)(u_0, v_0) = \sum_{u=-\infty}^{\infty} \sum_{v=-\infty}^{\infty} f(u, v) \,.\, g(u_0 - u, v_0 - v)$
- The convolution layers act as feature extractors. By using various filters, a CNN can learn to detect different features, such as edges, corners, textures, and more complex patterns.

Convolution calculation requires the following:

1. Filter/Kernel: A small filter with learnable weights is defined. The size of this filter is much smaller than the input image or feature map.

Convolution

2. Sliding Window: The filter is placed at the top-left corner of the input image or feature map. It starts sliding (or convolving) across the input, moving step by step according to a defined stride.

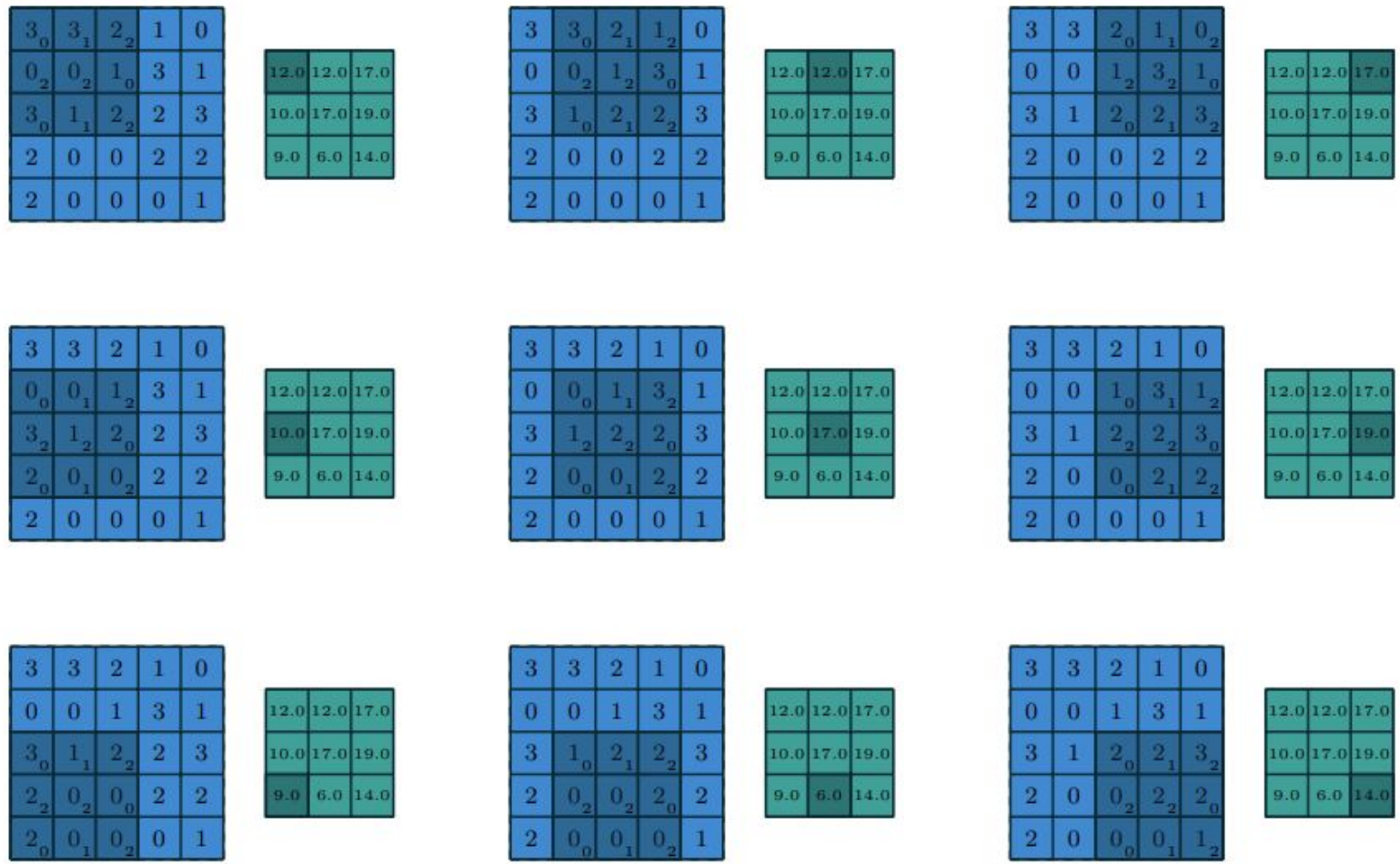
3. Element-Wise Multiplication: At each position of the filter, element-wise multiplication is performed between the filter's values and the values of the overlaid region of the input.

4. Summation: The results of the element-wise multiplications are summed up to produce a single value. This value is placed in the corresponding position of the output (feature map).

The filter continues sliding with a defined stride until it has covered the entire input.

Kernel

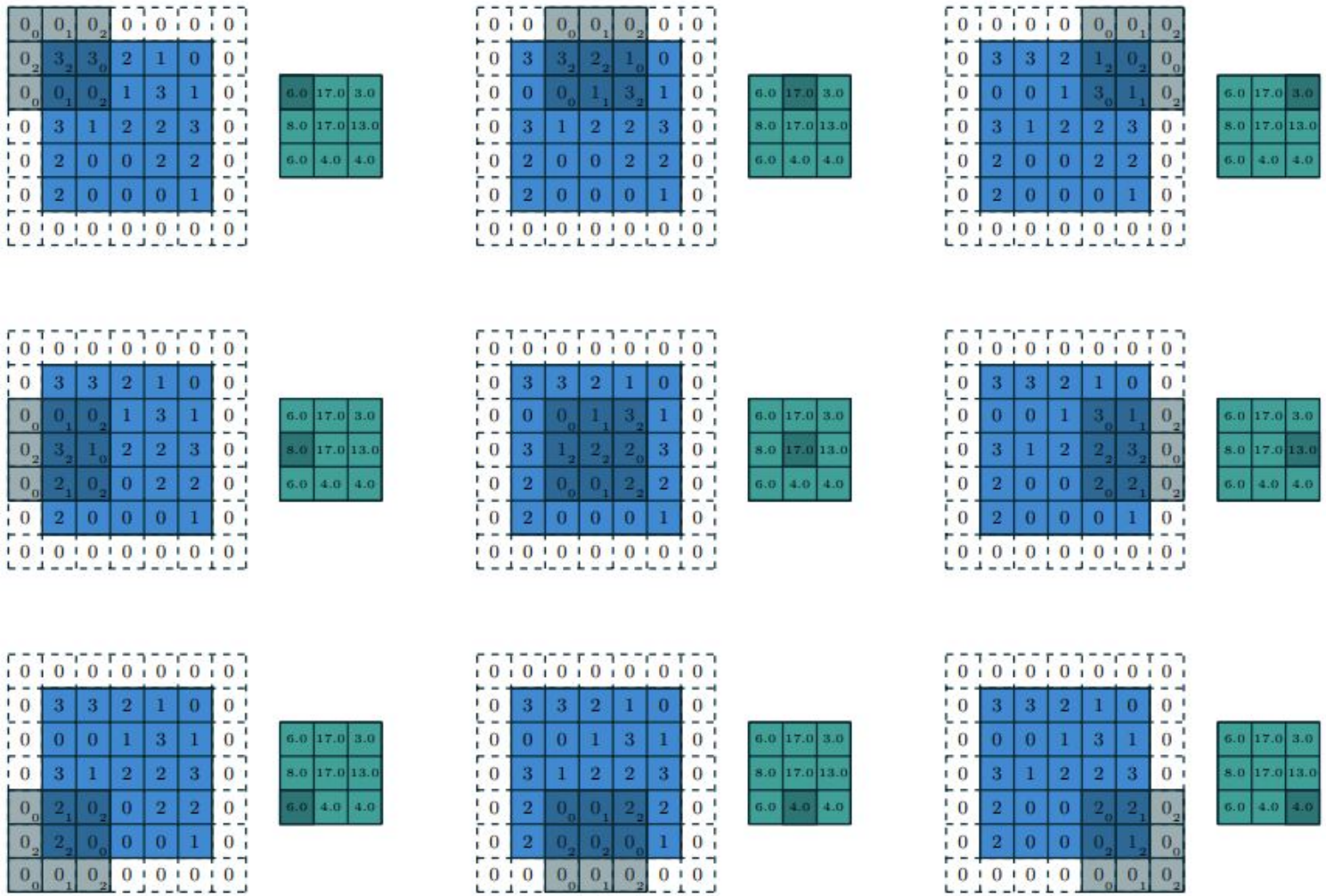
0	1	2
2	2	0
0	1	2



Sometimes, convolution is performed on padded images to control the spatial dimensions of the output.

Kernel

0	1	2
2	2	0
0	1	2

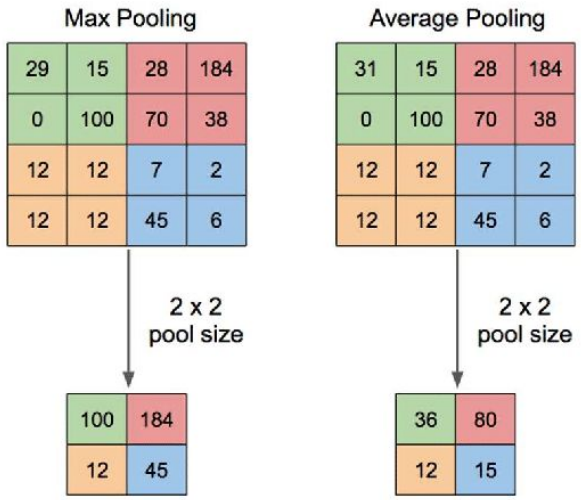


Ref: https://arxiv.org/abs/1603.07285

Pooling

- Pooling layers are another important component in CNNs.
- They are used to downsample the spatial dimensions of feature maps while retaining important information.
- Two commonly used pooling layers in deep learning are Max Pooling and Average Pooling

Max pooling operates by dividing the input feature map into non-overlapping regions (often called pooling windows or filters) and selecting the maximum value within each region.



Average pooling, similar to max pooling, divides the input feature map into non-overlapping regions and calculates the average value within each region.

Benefits of Pooling

- **Reduced Computational Complexity:** By reducing the spatial dimensions, pooling layers decrease the number of parameters and computation needed in subsequent layers.

- **Translation Invariance:** Pooling layers help make the network more robust to small translations and local variations in the input data, as the most important features are retained.

- **Feature Reduction:** Downsampling leads to a decrease in the amount of information passed forward, aiding in the selection of the most important features.

Why CNNs?

- Captures spatial relation in the data – allows network to learn features like edges, corners, and textures.
- Translation Invariant - can recognize patterns regardless of their position in the input.
- Parameter Sharing - the same set of weights (kernel) is shared across different spatial locations.
- Sparse Connectivity - CNNs have sparse connectivity due to the small receptive fields of their convolutional kernels.

Training Neural Networks

- Dataset
- Loss Function / Objective Function
- Model Architecture
- Optimizer
- Hyperparameters
- Backpropagation
- Model Evaluation
- Regularization Techniques

Dataset

- In a supervised learning, the dataset consists of input X and output y. For example, X can be a set of images and y is the corresponding ground truth predictions like classification labels, segmentation masks.

- In machine learning and deep learning, a dataset is typically divided into three main subsets: the training set, the validation set, and the test set. This is to obtain an unbiased evaluation of model prediction.
  - **Train set** – Used for training the model.
  - **Validation set** – Used for fine-tuning the hyperparameters, monitoring the model's performance, early stopping.
  - **Test set** – Used to assess the model's performance after it has been fully trained. gives an unbiased estimate of how well the model is likely to perform on new, unseen data.

Note: The three datasets should be kept independent from each other. Mixing the data from these subsets can lead to metrics that are not representative of the real performance.

Typical proportion of dataset for the subsets: 70-15-15 or 80-10-10 (train-validation-test).

Dataset

Dataset Standardization

Before training, it is a good practice to scale the data to a common range to ensure that all features have a similar impact on the model's learning process.

The standardization process transforms pixel values of an image so that they have a mean of 0 and a standard deviation of 1.

$$\text{Standardized pixel value} = \frac{\text{Original pixel value} - \text{Mean}}{\text{Standard Deviation}}$$

Where the mean and standard deviation of pixel is calculated across all the images in the training set.

Loss Functions / Objective Functions

What are loss functions?

It quantifies the difference between the predicted outputs of a model and the actual ground truth values.

The goal of a learning algorithm is to minimize this loss function, which in turn helps the model learn the best possible parameters for making accurate predictions.

Depending on the task at hand, the loss function differs. For example, classification networks typically optimize a cross-entropy loss, regression networks optimize a mean-squared error (MSE) loss.

Binary Cross-Entropy Loss

- Also known as log loss, it is commonly used in binary classification problems.
- The loss is given as:

$$L(y, \hat{y}) = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)]$$

N is the number of samples in the dataset.

y_i is the true label (either 0 or 1) for the i^{th} sample.

$\hat{y}_i$ is the predicted probability of the positive class (class 1) for the i^{th} sample.

An Example:

Assume that you have 3 samples in your dataset with the following values:

1. True label: 1, Predicted probability: 0.8
2. True label: 0, Predicted probability: 0.4
3. True label: 1, Predicted probability: 0.6

Binary Cross-Entropy Calculation:

1. 1.log(0.8) = -0.223
2. (1-0)log(1-0.4) = -0.511
3. 1.log(0.6) = -0.511

$$L = -(-0.223-0.511-0.511)/3 = 0.415$$

Notice that the more the predicted value deviates from the true value, the larger is the loss.

Categorical Cross-Entropy Loss

- Measures the difference between the predicted class probabilities and the true one-hot encoded class labels.
- It is commonly used in multiclass classification.

$$L(y, \hat{y}) = -\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{C} y_{ij} \cdot \log(\hat{y}_{ij})$$

C is the number of classes

An Example:

- Suppose you have a multiclass classification problem with three classes (Class A, Class B, Class C). You have four samples in your dataset:

1. True labels: [1, 0, 0], Predicted probabilities: [0.7, 0.2, 0.1]
2. True labels: [0, 1, 0], Predicted probabilities: [0.4, 0.5, 0.1]
3. True labels: [0, 0, 1], Predicted probabilities: [0.2, 0.3, 0.5]
4. True labels: [1, 0, 0], Predicted probabilities: [0.8, 0.1, 0.1]

- For each sample, you calculate the individual categorical cross-entropy terms:

1. $1 \cdot \log(0.7) = -0.357$
2. $1 \cdot \log(0.5) = -0.693$
3. $1 \cdot \log(0.5) = -0.693$
4. $1 \cdot \log(0.8) = -0.223$

$$L = -(-0.357 - 0.693 - 0.693 - 0.223)/4 = 0.4915$$

Mean-Squared Error Loss

- Commonly used in regression tasks.
- It quantifies the average squared difference between the predicted values and the actual target values.

$$L(y, \hat{y}) = \frac{1}{N}\sum_{i=1}^{N}\left(y_i - \hat{y}_i\right)^2$$

N is the number of samples in the dataset.

y_i is the true target value for the i^{th} sample.

$\hat{y}_i$ is the predicted value for the i^{th} sample.

Model Architecture

The model architecture used for training depends on the problem at hand and the number of training samples.

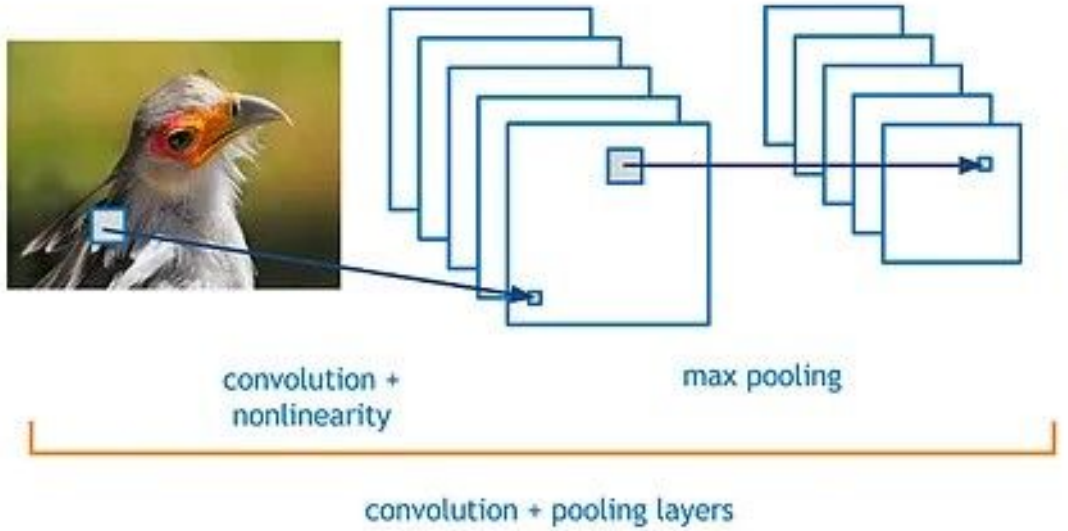
However, there exists several standard architectures for different tasks, which can be adapted to train on our own data.

We will discuss the overall principle behind the working of the following tasks:

- Classification
- Object Detection
- Image Segmentation

Classification

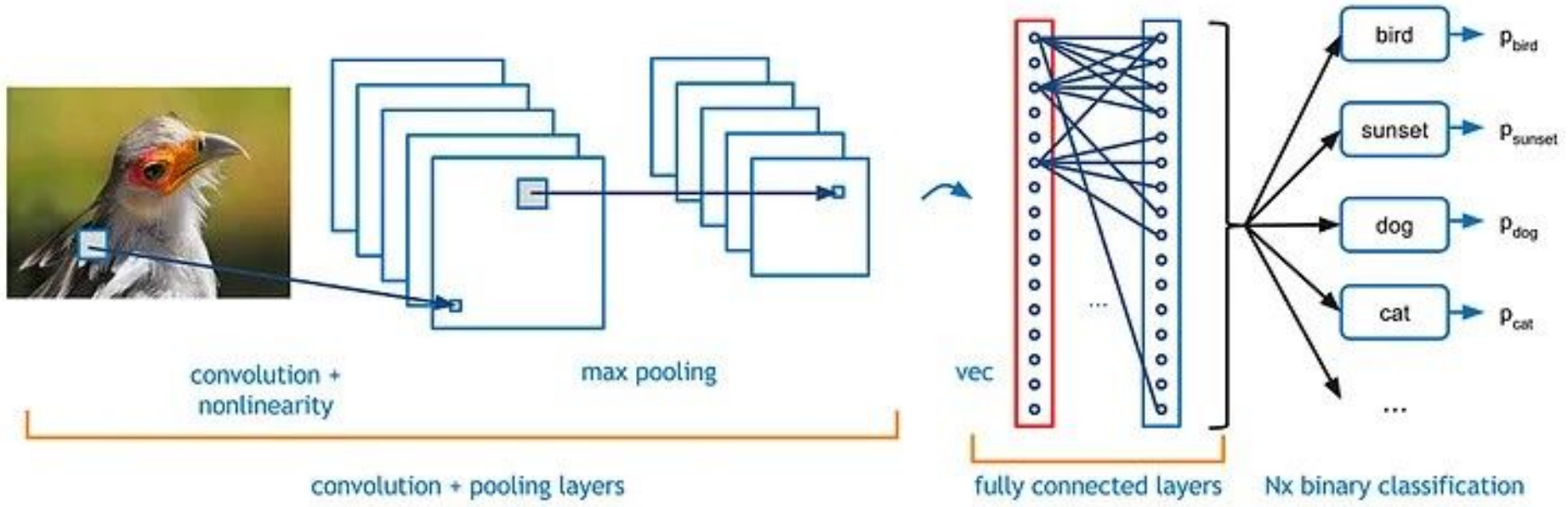
- **Input and Feature Extraction** - A CNN backbone acts as a feature extractor. The convolutional layers and dense layers capture hierarchical representations of the data. With increasing depth, the network captures increasingly complex features needed to effectively classify the data.



- **Classification -** The classification layer is usually implemented as one or more fully connected layers (also known as dense layers) followed by an activation function.
- The final output of the classification layer is typically passed through a softmax activation function, which converts the output into a probability distribution over the different classes or categories.

Softmax: $$s\left(x_i\right) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

- Finally, the class with the highest probability is predicted as the class label for the input data.



Some standard architectures

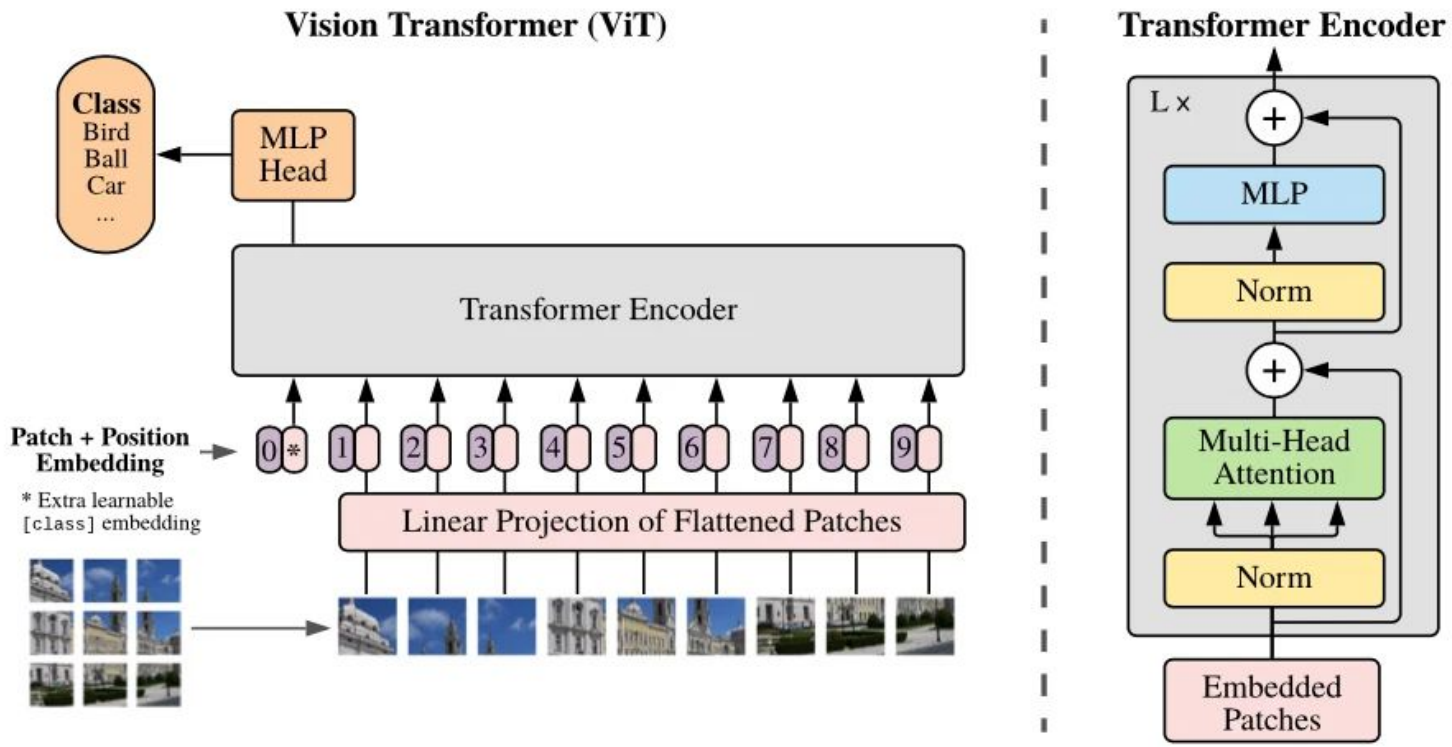
- AlexNet
- VGGNet
- Inception
- ResNet
- DenseNet
- MobileNet
- EfficientNet

Tip: ‘timm’ is a library that implements several standard architectures in PyTorch. It also includes the model weights trained on the ImageNet dataset.

https://huggingface.co/docs/timm/index

Vision Transformers (ViT)

ViT model uses a transformer architecture, initially introduced for natural language processing. It treats images as sequences of patches.



The working principle of a ViT is as follows:

1. Input Patches:

- The input image is divided into fixed-size non-overlapping patches. Each patch is typically a small square region of pixels.
- These patches are then linearly flattened to form a sequence of vectors.

2. Patch Embeddings:

- Each patch vector is passed through an initial linear layer (embedding layer) to transform it into a higher-dimensional embedding space.
- The resulting embeddings now represent the visual information from the input patches.

3. Positional Encodings:

- As transformers don't have built-in spatial information like CNNs, positional encodings are added to the patch embeddings to give the model information about the positions of the patches.

4. Transformer Encoder:

- The patch embeddings, along with positional encodings, are fed into a standard transformer encoder.
- The encoder consists of multiple layers of self-attention mechanisms and feedforward neural networks.
- The self-attention mechanism allows the model to capture relationships between different patches regardless of their positions.

5. Global Information:

- The self-attention mechanism in the transformer allows each patch to attend to all other patches, capturing global contextual information.
- This enables the model to learn both local and global relationships between patches.

6. Classification Head

- After the transformer encoder, the output embeddings are often passed through a pooling operation, such as mean pooling, to aggregate information from all patches into a fixed-size vector.
- This vector is then fed into a fully connected classification head to make class predictions.

Object Detection

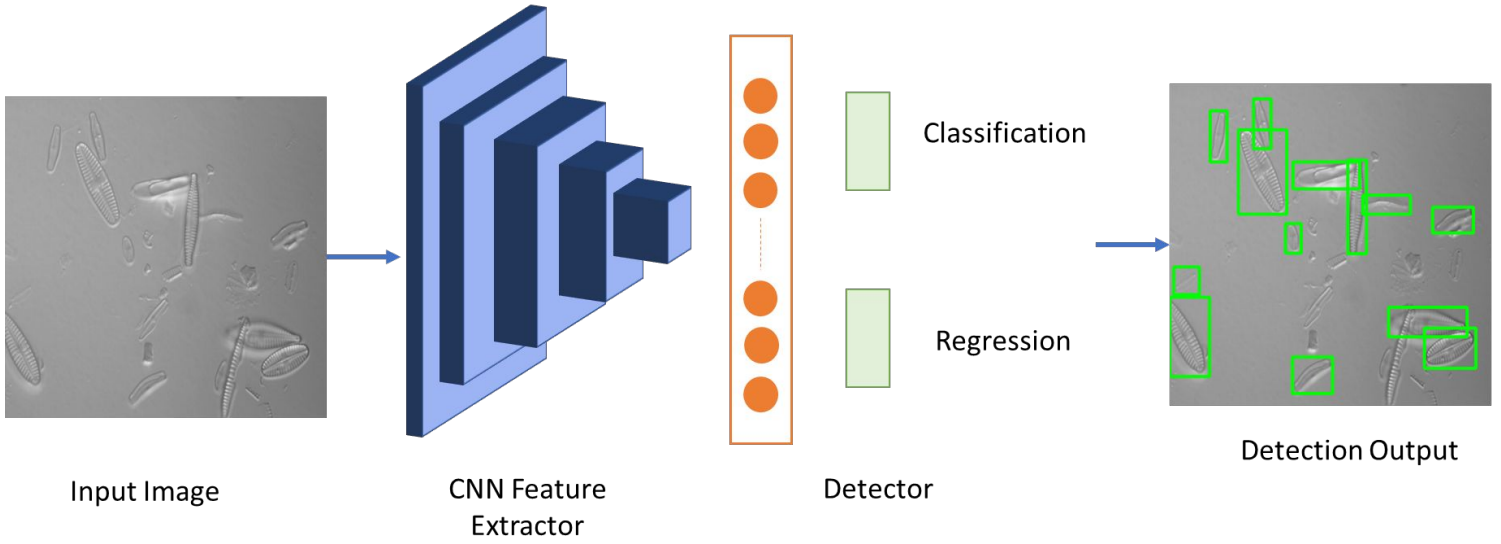
- The object detection networks in deep learning can be broadly divided into two types:
  - One Stage Detection Networks - aim to directly predict bounding boxes and class probabilities for objects in a single pass through the network.
  - Two Stage Detection Networks - break down the object detection task into two distinct stages: region proposal generation and object classification/refinement.

Tip: Deep Learning frameworks such as Tensorflow and PyTorch have several of the standard object detection networks already implemented that can be used off the shelf.

PyTorch: **Detectron2** - https://github.com/facebookresearch/detectron2/tree/main

Tensorflow: **Object Detection API** - https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/

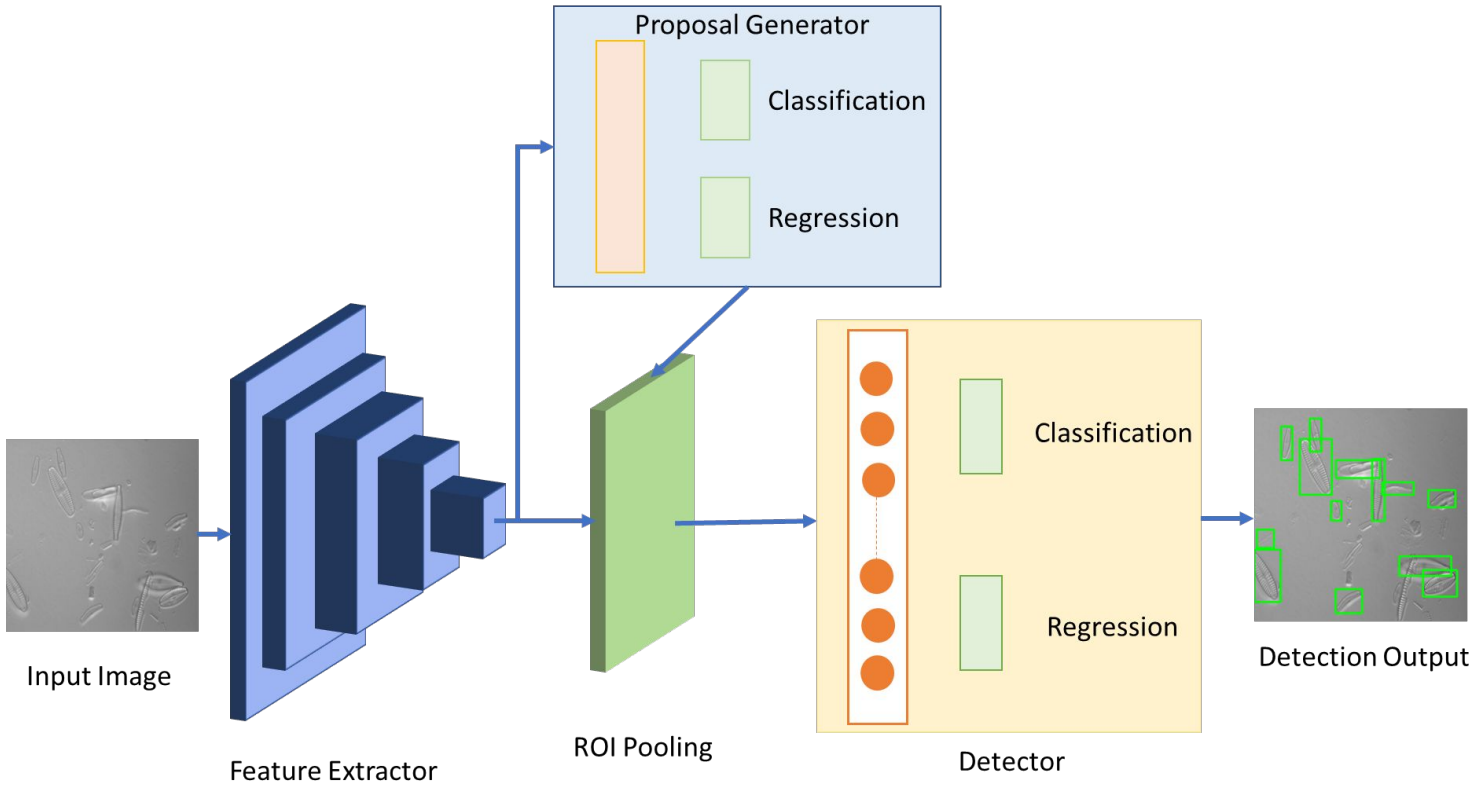
One Stage Object detection



Examples: Single Shot Detector (SSD), YOLO family of detectors

- **Input and Feature Extraction** - A pre-trained CNN backbone on ImageNet acts as a feature extractor and extracts high level features from the input image.
- **Object Localization**
  - The feature maps obtained from the CNN are used to predict a set of bounding boxes of potential objects in the image.
  - In addition, the network also outputs a probability value for each box p_{obj} , called the objectness score, which indicates the likelihood of presence of an object in the box region.
- **Object Classification**
  - For each bounding box, the network predicts the class of object it contains.
  - The CNN is connected with fully connected layers, which performs a multi-class classification task to identify the objects present in the image
- **Non Maximum Suppression (NMS)**
  - Most of the time, the network outputs multiple box predictions for every detected object, resulting in redundant detections.
  - To remove them, a technique called non-maximum suppression is used, where only the box corresponding to the most confident prediction for each object is retained.

Two-stage Object detection

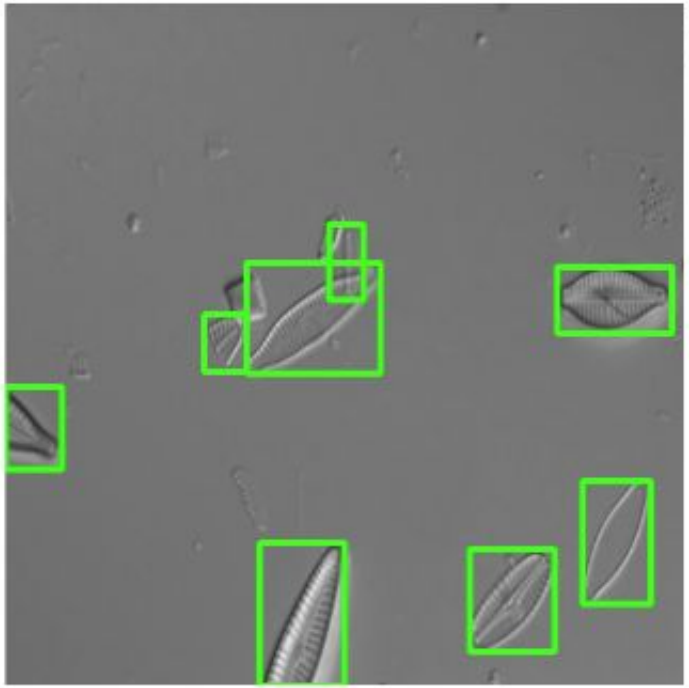


Examples: RCNN, Fast RCNN, Faster RCNN

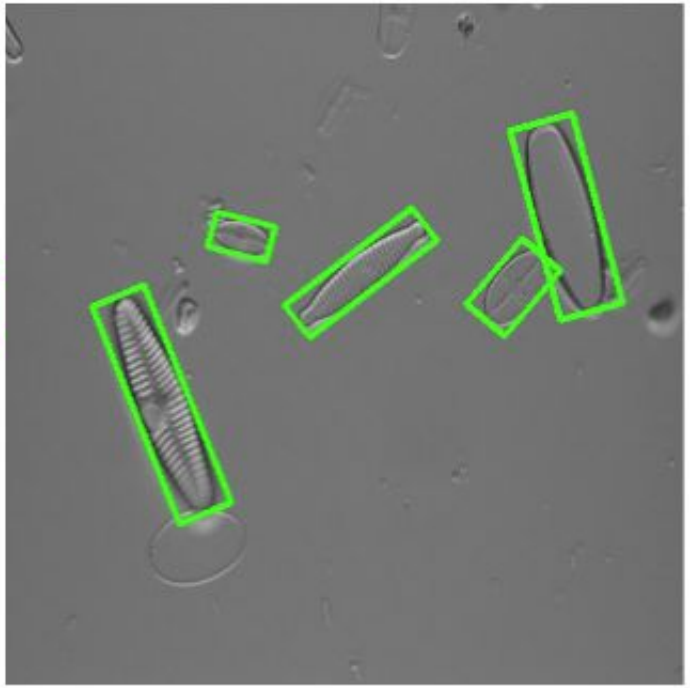
- **Input and Feature Extraction** - A pre-trained CNN backbone on ImageNet acts as a feature extractor and extracts high level features from the input image.
- **Region Proposal Network (RPN)**
  - The feature maps from the CNN is used to produce a set of bounding boxes that potentially contain objects.
  - The RPN applies a sliding window over these feature maps to generate a set of object proposals.
- **Object detection and classification**
  - This stage of the network takes the object proposals generated by the RPN and performs object detection.
  - It uses a separate CNN to classify the objects and refine their bounding boxes.
  - The bounding box regression step involves predicting the offsets between the proposal box and the ground truth box.
- **Non Maximum Suppression (NMS)**
  - Similar to one stage networks, NMS is used to removed redundant detections.

Diatom Detection using YOLOv5

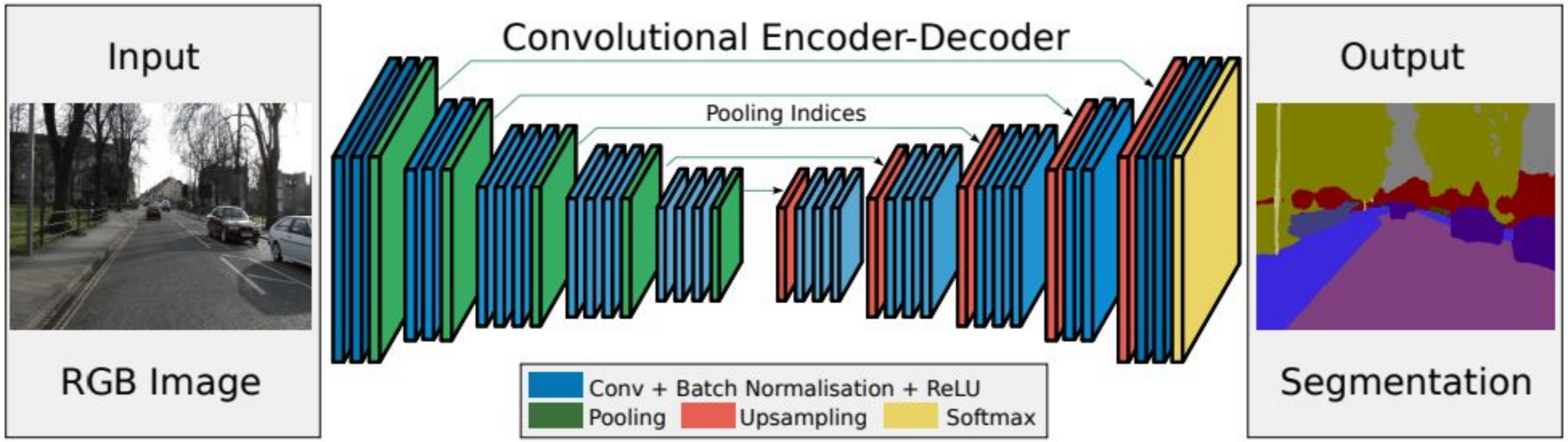
Axis-aligned bounding box detection



Rotated bounding box detection



Semantic Segmentation

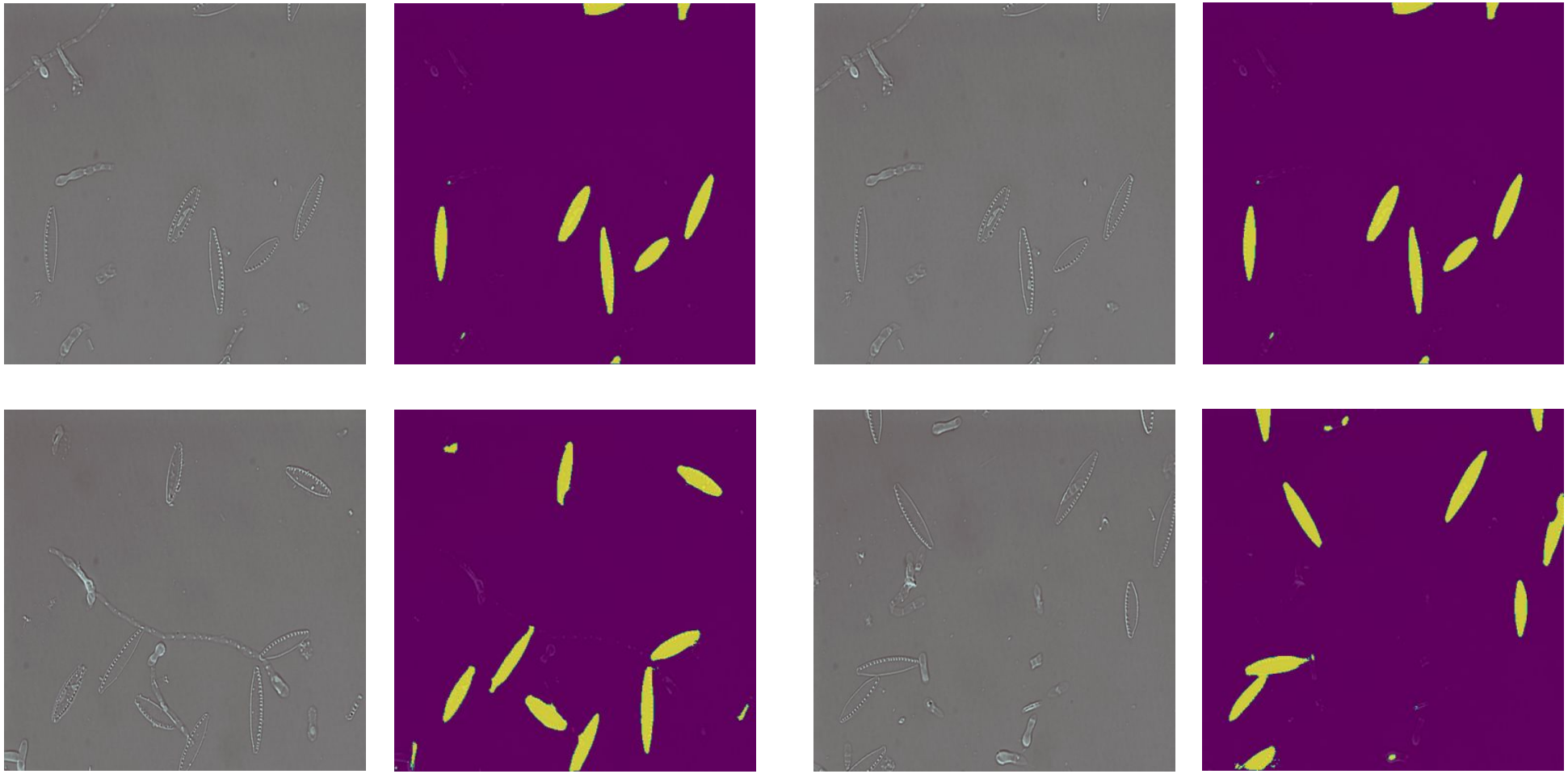


- **Encoder**
  - The encoder consists of a set of convolution layers to learn feature representations of the input image.
  - These features are then downsampled through a series of pooling or strided convolutional layers to reduce the spatial resolution of the feature maps while increasing their depth.
- **Decoder**
  - The decoder is made up of several upsampling layers that gradually increase the spatial resolution of the feature maps while decreasing their depth.
  - The final layer outputs feature maps, whose spatial resolution is the same as the input, and the number of channels equal the number of segmentation labels.
- **Pixelwise Classification**
  - The feature maps obtained from the decoder are passed through a softmax layer, which outputs the class probability values for every pixel.
  - The class corresponding to the largest probability is assigned to the pixel.

Standard segmentation architectures

- Fully Connected Networks
- U-Net
- SegNet
- PSPNet
- DeepLab family of networks

Semantic Segmentation of Diatoms using DeepLabV3+



Instance Segmentation

This type of method combines object detection and semantic segmentation to identify and segment individual objects in an image.

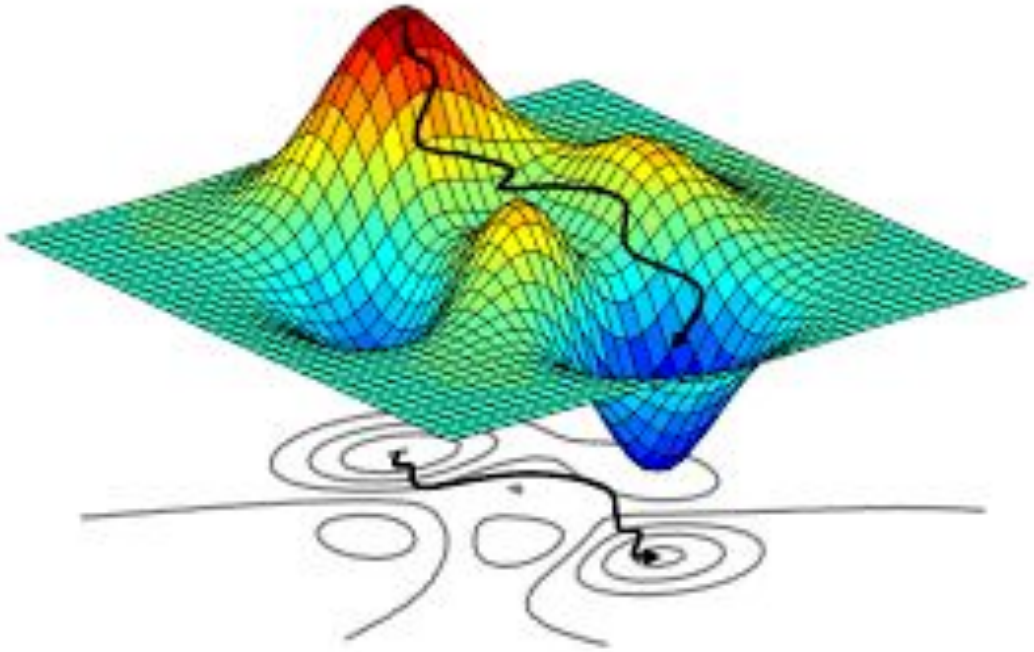
E.g. Mask RCNN

Semantic Segmentation

Instance Segmentation

Optimizer

An optimization algorithm adjusts the parameters (weights) values of the model to minimize the loss function.



Gradient Descent - Algorithm

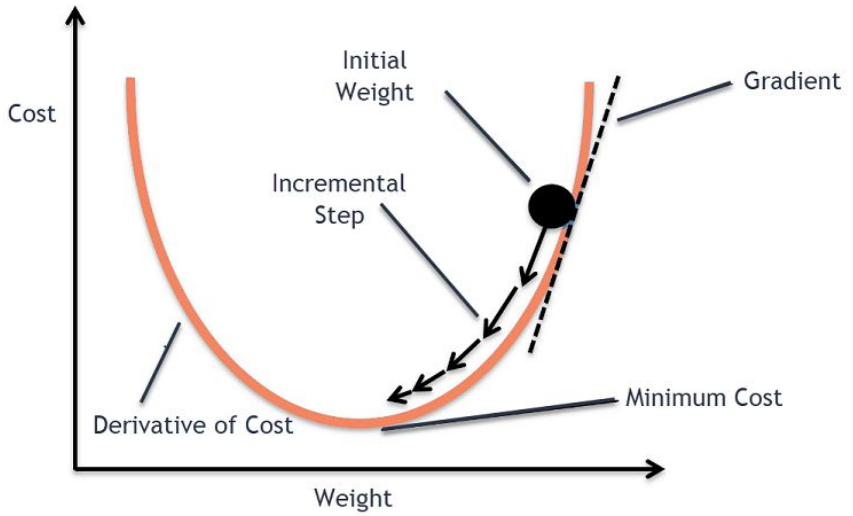
- Initialize model parameters θ (weights and biases) with some values. The initialization can be done randomly. Next define the loss function J(θ).
- Compute the gradients of the loss function with respect to each parameter, and take a small step in the opposite direction to the gradient:

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

Here, α is a hyperparameter called the **learning rate.**

- To give an intuition, we look at surrounding points from the parameter values and see where to go so that the loss function decreases the most.
- The process is repeated for until we reach a local minimum value of J(θ):

$$\theta^* = \text{argmin}_\theta J(\theta)$$



Batch Gradient Descent

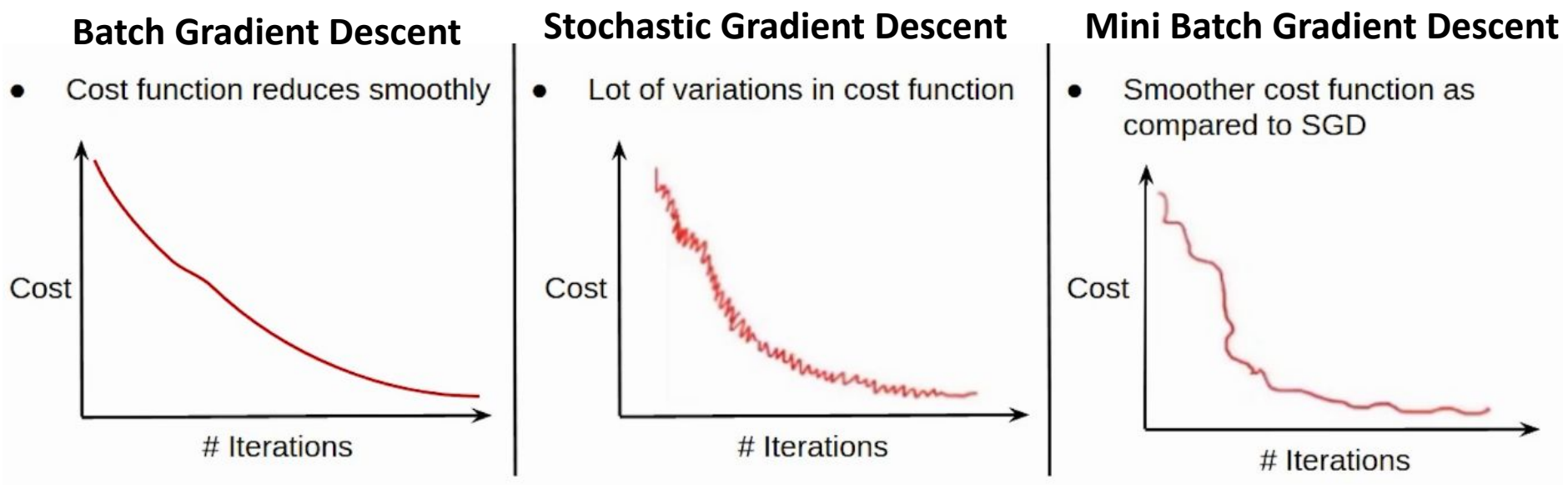
- In batch gradient descent, all the training examples are considered a single batch.
- The loss function is calculated for all the training images and the gradient value is calculated for this and the parameter is updated.
- A drawback is that for large datasets, this takes a huge amount of time since you have to calculate the loss for all the training examples in order to make a single step of parameter update.

Stochastic Gradient Descent (SGD)

- In this algorithm, the loss calculation and parameter update is performed for each training sample.
- Since this is performed per sample, the parameter update can be noisy (Due to noise in the data).

Mini Batch Gradient Descent

- In this algorithm, training dataset is divided into small batches, and parameter updates are computed based on the average gradient of the loss function calculated over each mini-batch.
- This provides smoother parameter updates than SGD and is also faster than batch gradient descent.

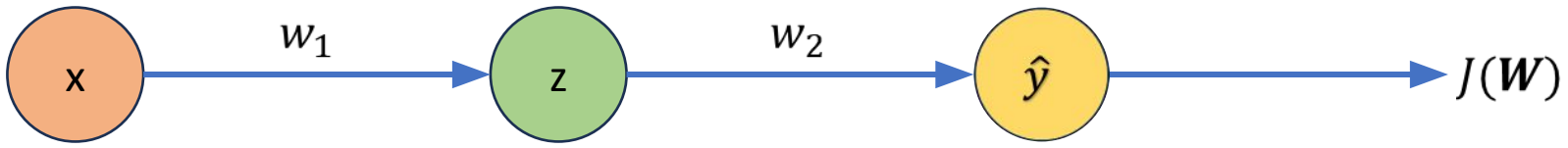


Backpropagation

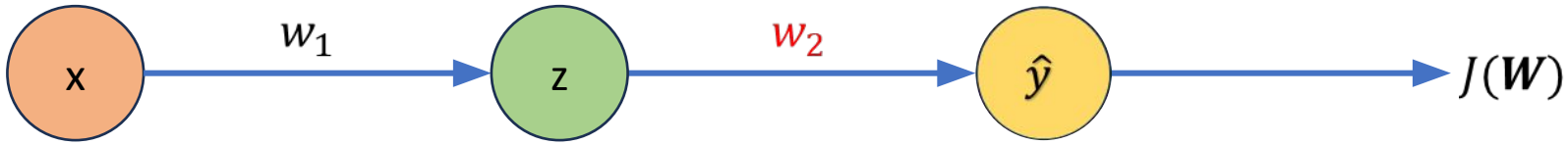
One of the key requirements for optimization is the **gradient calculation.**

Backpropagation allows us to calculate the gradients required for our optimization algorithm.

Backpropagation relies on **chain rule** for computing the gradients.

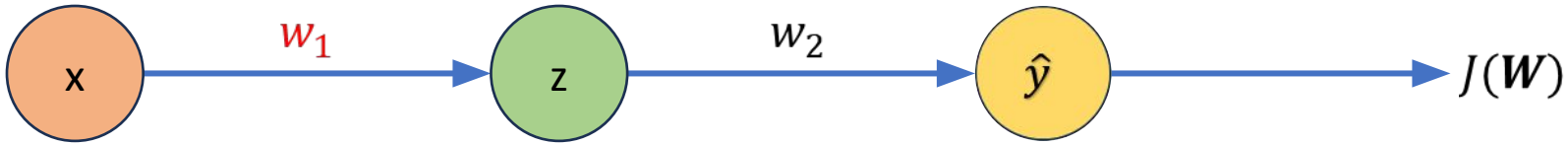


Backpropagation



$$\frac{\partial J(W)}{\partial w_2} = \frac{\partial J(W)}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w_2}$$

Backpropagation



$$\frac{\partial J(W)}{\partial w_1} = \frac{\partial J(W)}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w_1} = \frac{\partial J(W)}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z} * \frac{\partial z}{\partial w_1}$$

Apply Chain rule

This process is repeated for every weight in the network using gradients from the later layers.

Hyperparameters

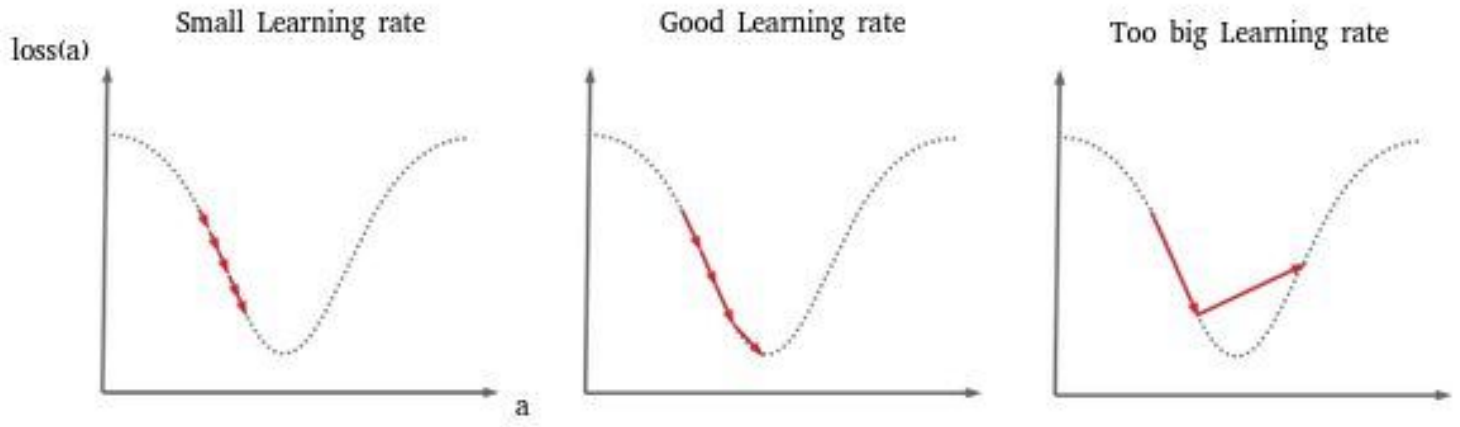
- Hyperparameters are parameters that are set before the training process begins and control various aspects of the training process itself.
- They are chosen by the user or the developer based on prior knowledge, experimentation, or best practices.

Hyperparameters

Learning Rate:

The learning rate determines the step size at which the optimization algorithm (such as gradient descent) updates the model's parameters in the direction that minimizes the loss function.

A high learning rate might lead to fast convergence but risk overshooting the optimal solution, while a low learning rate might lead to slow convergence.



Batch Size:

- The batch size specifies the number of training examples used in each iteration of the optimization algorithm.
- The batch size value is highly dependent on the hardware used for training. A larger RAM allows a larger batch size.

Number of Epochs:

- An epoch is a complete pass through the entire training dataset.
- The number of epochs determines how many times the optimization algorithm iterates over the entire dataset.

Learning Rate Scheduling:

- Involves dynamically changing the learning rate during the training process.
- Learning rate scheduling methods aim to overcome challenges such as convergence difficulties and oscillations in the loss

Model Evaluation

Once we have trained our model, we need to assess it's performance to understand how well it generalizes to new, unseen data.

The model is evaluated on the **test dataset**.

The model evaluation method varies from task to task. To evaluate the models quantitatively, we use performance metrics. Whenever relevant, it can also be evaluated qualitatively (E.g qualitative comparison of images generated by generative networks.)

Performance Metrics

The performance metrics give us a quantitative value, that indicates how well a model's predictions match the ground truth values.

Commonly used metrics:

1. **Classifiction:**
    1. Accuracy
    2. Precision
    3. Recall
    4. F1-Score
    5. AUC-ROC Score

While accuracy is the commonly used metric for evaluating classifiers, it is also an unreliable metric when dealing with imbalanced dataset. F1 Score is commonly preferred in this case.

2. Object detection:

1. Average Precision
2. Intersection over Union (IoU) / Mean IoU (mIoU)
3. Mean Average Precision (mAP)
4. ROC Curve
5. Precision and Recall

3. Segmentation

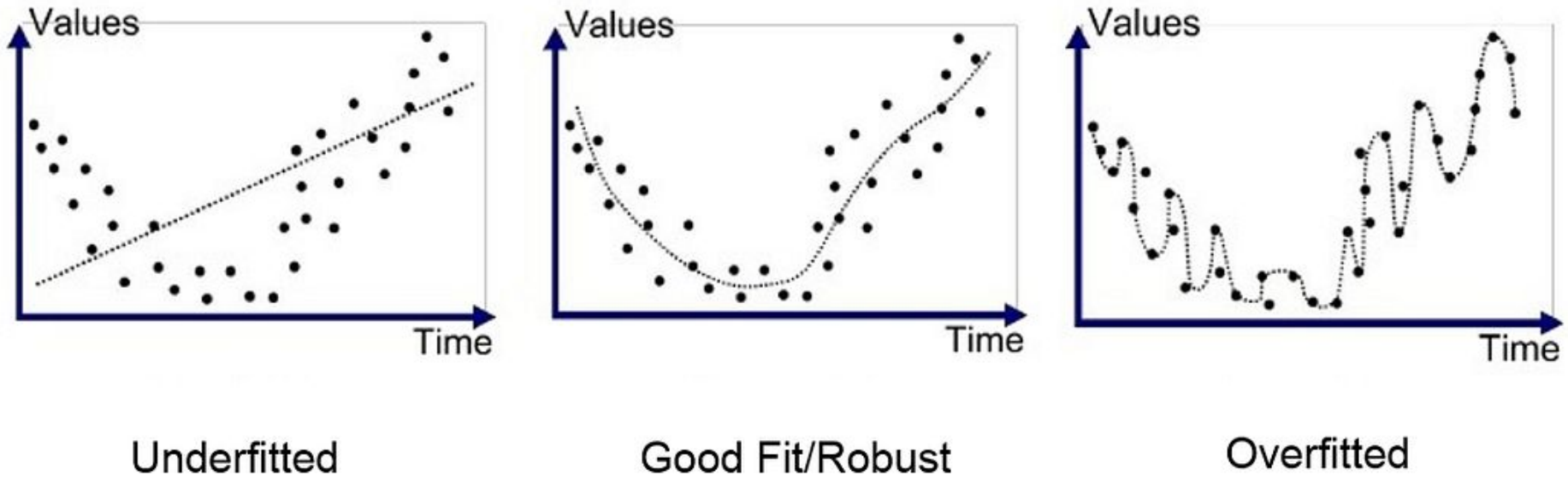
6. Accuracy
7. mIoU score
8. Dice coefficient

Challenges in Training

Overfitting and Underfitting

A condition where the model learns to perform extremely well on the training data but fails to generalize to new, unseen data is called **overfitting**. Overfitting captures noise and intricacies of the training data .

The opposite situation is referred to as **underfitting**, where a model is too simple to capture the underlying patterns in the data. An underfitted model performs poorly not only on the training data but also on new, unseen data.



Mitigating Overfitting

- **Regularization:** Adding penalty terms to the loss function to discourage the model from learning overly complex patterns.
- **Cross-Validation:** Splitting the data into multiple subsets for training and validation to get a more robust estimate of the model's performance.
- **Early Stopping:** Monitoring the validation performance and stopping training when validation error starts to increase.
- **Data Augmentation:** Increasing the diversity of the training data by applying transformations like rotation, cropping, or adding noise.
- **Reducing Model Complexity:** Using simpler model architectures or reducing the number of layers/neurons.

Mitigating Underfitting

- **Increasing Model Complexity:** Using more complex model architectures, such as deep neural networks with more layers and neurons, to capture the intricate patterns in the data.
- **Tuning Hyperparameters:** Adjusting hyperparameters, such as learning rate, batch size, and regularization strength, to improve the model's learning process.
- **Using More Data:** Increasing the size of the training dataset can help the model learn from a broader range of examples and patterns.
- **Reducing Regularization:** If the model is over-regularized, it might lead to underfitting. Adjusting regularization strength might help alleviate this issue.

Regularization

- Regularisation techniques are used to avoid overfitting in models.
- Several types of regularisation methods exist. The common ones are:

L1 Regularization (Lasso Regularization):

L1 regularization adds a penalty term to the loss function that is proportional to the absolute values of the model's weights.

$$L_{regularized} = L_{original} + \lambda \sum_{i=1}^{n} |w_i|$$

- $L_{original}$ is the original loss function without regularization.
- n is the number of model weights.
- w_i represents the i^{th} weight of the model.
- λ is the regularization parameter controlling the strength of the L2 penalty.

L2 Regularization (Ridge Regularization):

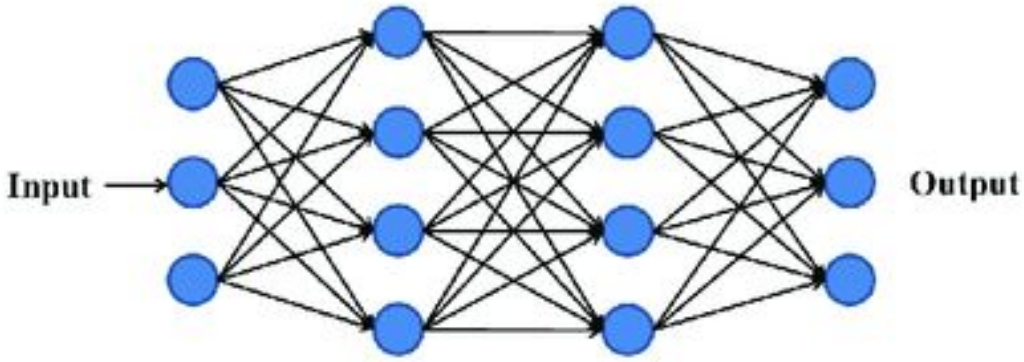
L1 regularization adds a penalty term to the loss function that is proportional to the absolute values of the model's weights.

$$L_{regularized} = L_{original} + \lambda \sum_{i=1}^{n} w_i^2$$

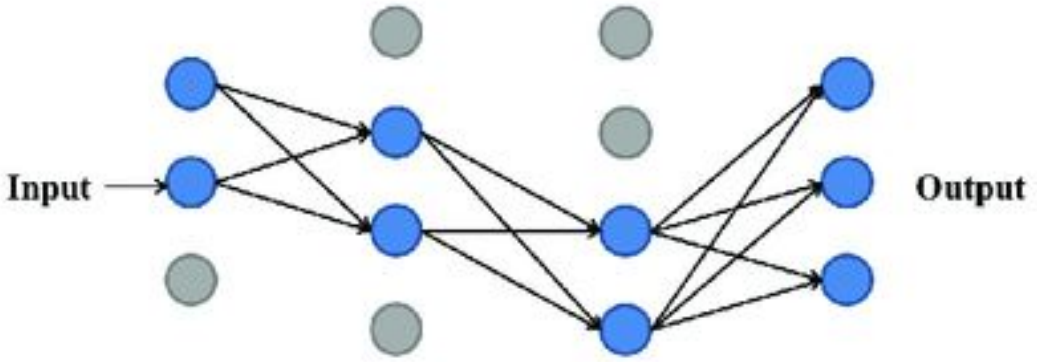
Dropout

- Dropout is a regularization technique specifically designed for neural networks, often used to prevent overfitting.
- It involves randomly deactivating (or "dropping out") a proportion of neurons in a neural network layer during each training iteration.
- Dropout helps prevent neurons from relying too much on each other and encourages the network to learn more robust and generalizable features.
- During inference (i.e., when making predictions on new data), dropout is turned off, and all neurons are used.

Dropout Illustration



(a) Structure without Dropout

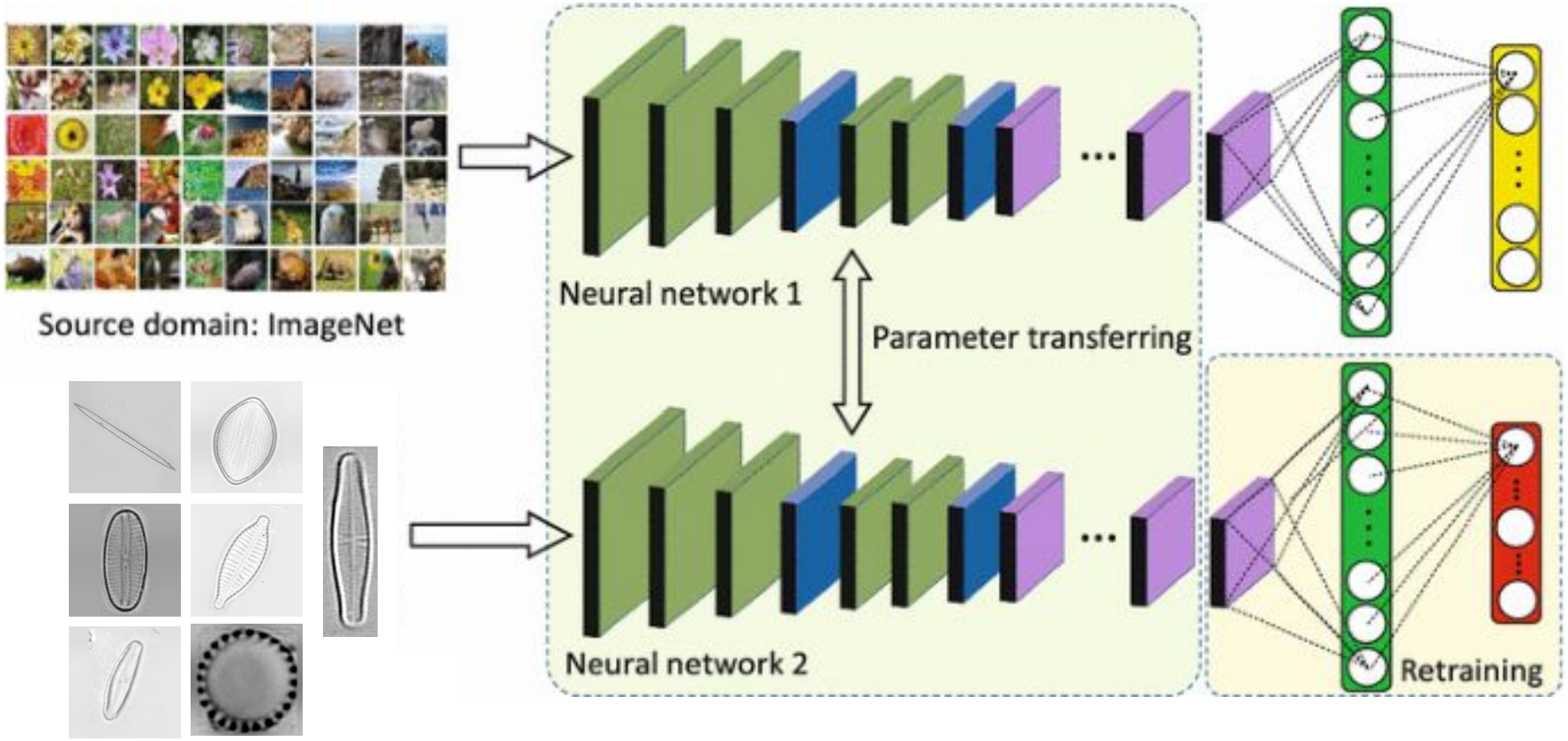


(b) Structure with Dropout

Transfer Learning for Data Scarcity

- Transfer learning involves using pre-trained models (often trained on large datasets) as a starting point and fine-tuning them on a new task or dataset.
- This approach is particularly useful when you have limited data for the new task or when training from scratch would be computationally expensive.

Transfer Learning



Dataset Augmentation

- It is a technique used to artificially increase the diversity and size of training dataset.
- It involves applying various transformations and modifications to the original data in order to create new examples that are still relevant to the problem.
- This helps improve the generalization and robustness of models by exposing them to a wider range of variations that they might encounter during deployment.
- Some commonly used augmentation techniques for images: Rotation, Flipping, Translation, Scaling, Shearing, Color Jittering, Noise Addition.

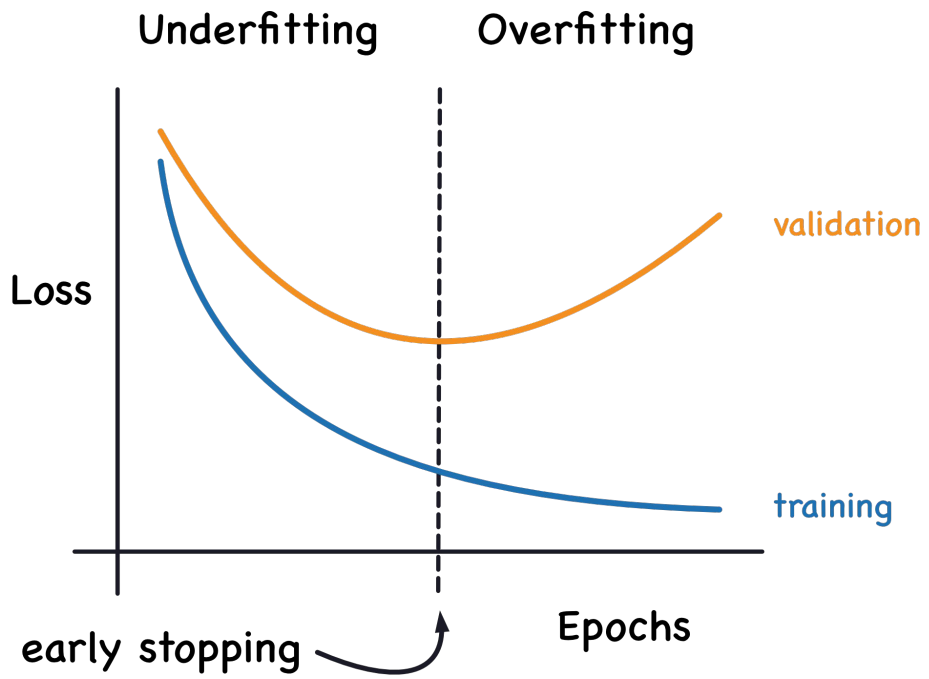
Data Augmentation Examples

Tip: In python, a library called **Albumentations** implements a wide variety of augmentation techniques for different computer vision tasks.

https://albumentations.ai

Loss Curves

Loss curves are commonly used to gain insight into the generalisation capabilities of the model. The X-axis is the number of epochs and the Y-axis is the training loss and the validation loss.
It can be used to diagnose the model for overfitting and underfitting.



How to identify an underfit model?

There are two scenarios which indicates that a model is underfit:

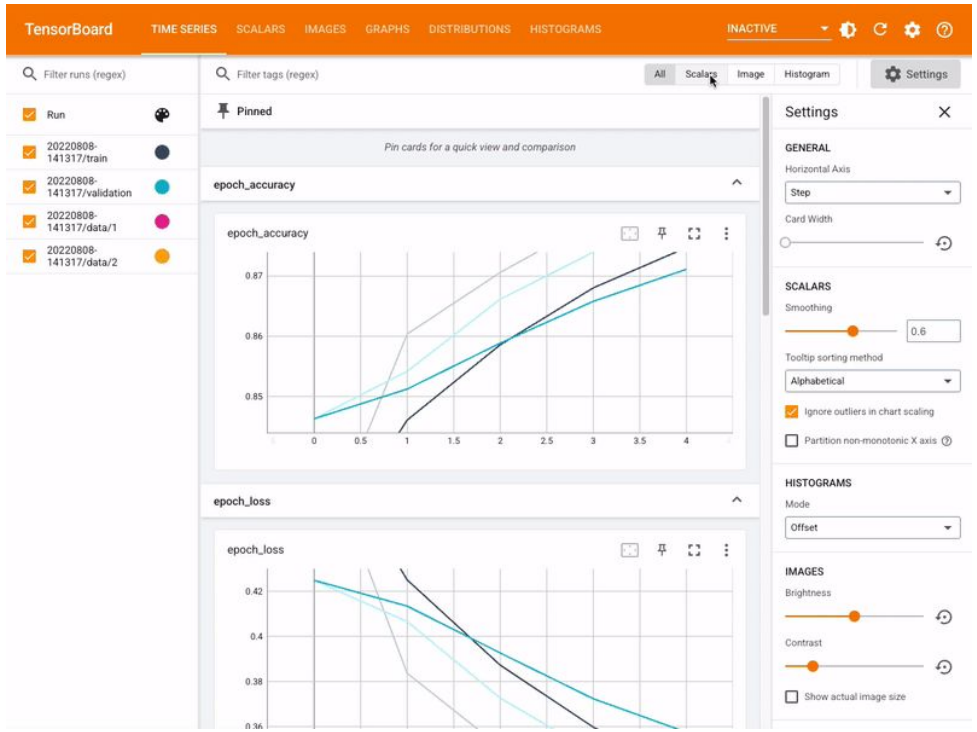
1. If the training loss curve shows noisy values with high loss, it means that the model did not learn anything from the training data.
2. If the training loss that is decreasing and continues to decrease at the end of the plot, it means that the model needs to be trained further and it has not reached.

How to identify an overfit model?

When the training loss decreases with the number of epochs but the validation loss increases, it indicates an overfit model.

TensorBoard is a useful tool to visualize loss curves and other plots.

It is available in both TensorFlow and PyTorch.



Conclusion

- Neural networks are the heart of deep learning, allowing us to model complex relationships in data. Their potential is boundless, extending from image classification to natural language understanding.
- Convolutional Neural Networks (CNNs) have revolutionized computer vision. They capture intricate features and hierarchies, enabling machines to see the world much like we do.
- Proper data preparation, augmentation, and quality are crucial for training reliable models. The strength of a deep learning model lies in the data it learns from.
- Challenges like overfitting, limited data, and interpretability drive research and innovation. These challenges open doors to exciting opportunities for improvement.
- Deep learning for computer vision is behind several real-world applications such as self-driving cars, medical diagnoses, and even enhancing our creative expressions through art.