Deep Learning for Image Classification

Instructor: Stephin Rachel Thomas

February 12, 2026

Today’s Topics

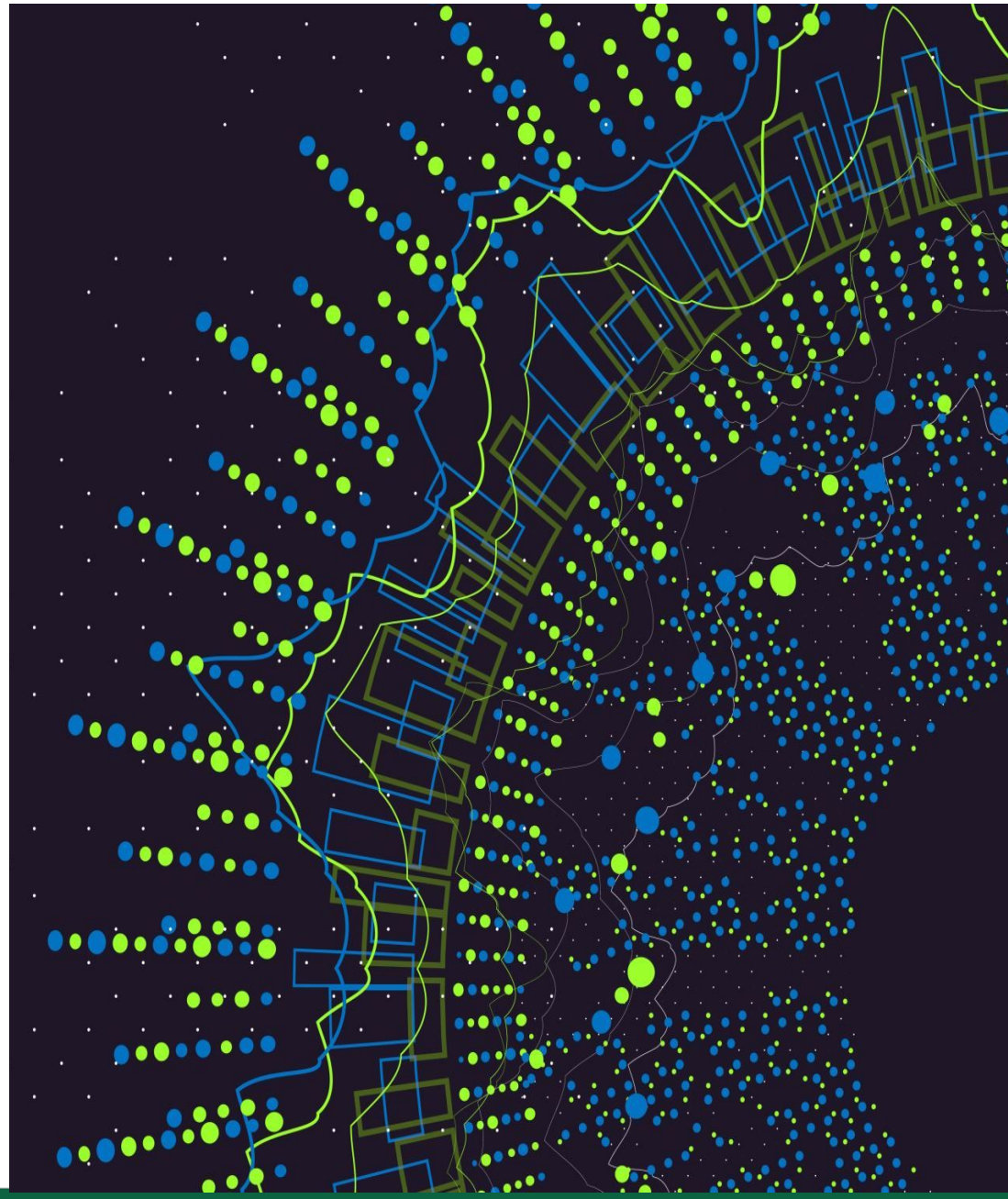
- Fundamentals of Image classification with CNN
- Dataset Preparation
- Discussion
- Data augmentation Strategy
- Designing CNN architecture
- Activation Functions
- Loss Functions
- Back propagation
- Optimizers
- Training CNN
- Common issues in Computer vision
- Troubleshooting in CNN Training
- Midterm test details

INTRODUCTION

Deep Learning, a subset of machine learning, involves neural networks with many layers.

In computer vision, deep learning powers tasks such as image classification, object detection, and semantic segmentation.

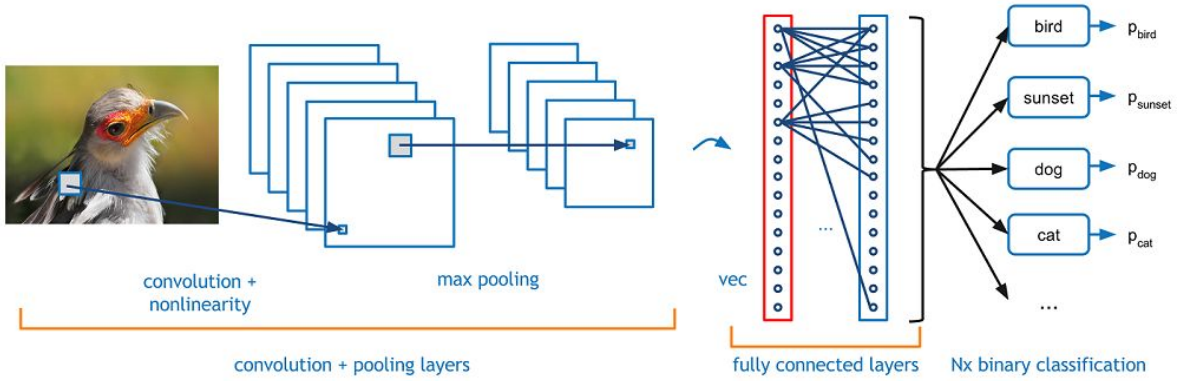
These tasks are accomplished through models that can identify patterns and features in images, mimicking human vision.



Fundamentals of Image Classification with CNNs

Image classification with CNNs involves categorizing and labeling images into predefined classes.

CNNs process images through layers that detect features, reduce dimensions, and classify images based on learned patterns.



Key components include convolutional layers for feature extraction, pooling layers for dimensionality reduction, and fully connected layers for classification.

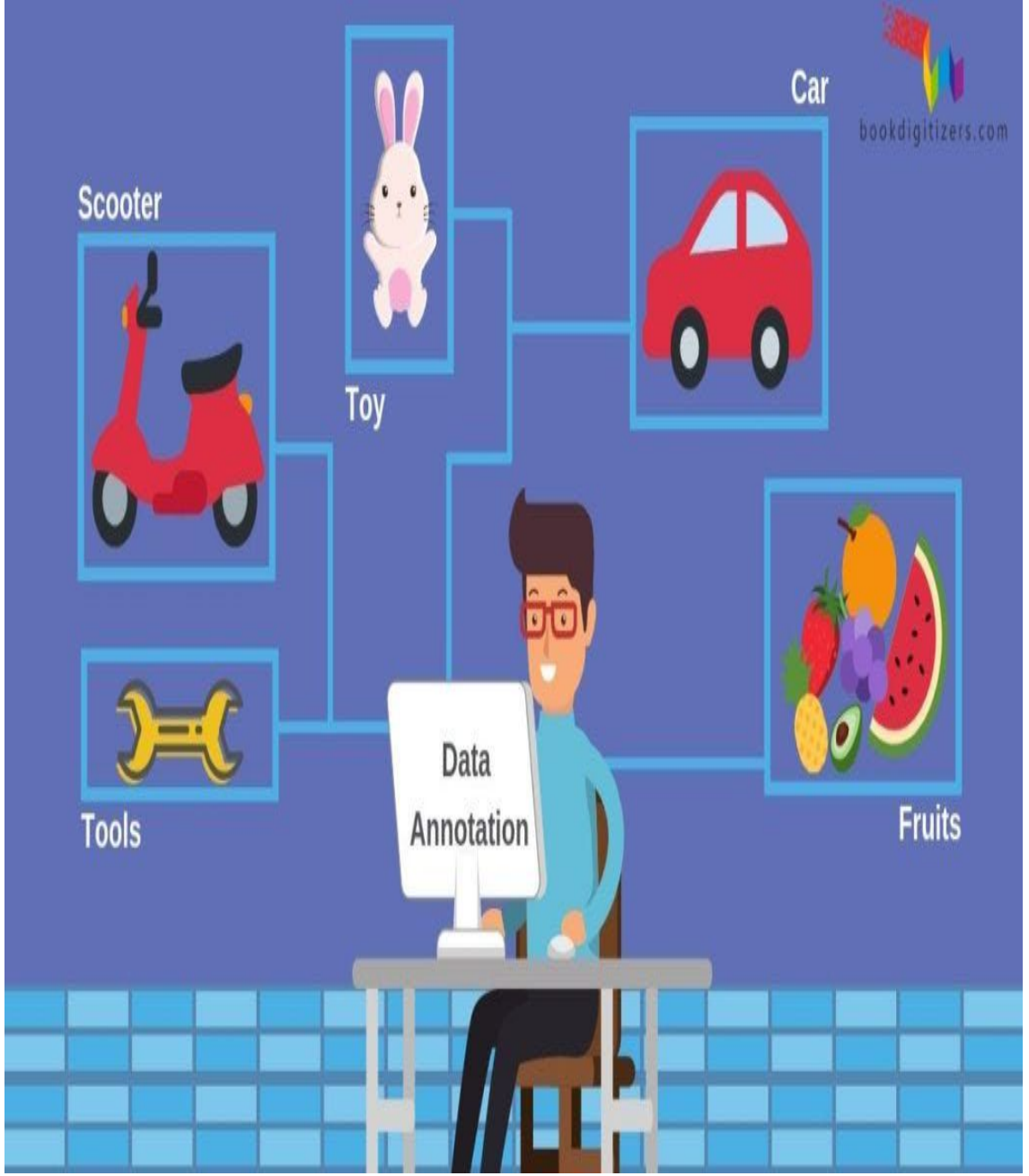
Dataset Preparation: Collection and Annotation

Dataset preparation is a vital step in image classification.

It involves collecting a diverse set of images representing different classes.

Annotation, the process of labeling images with class names, is essential for supervised learning.

Quality and diversity of the dataset directly impact the model's ability to learn and generalize to new, unseen images.



Data Preprocessing Techniques for Image Data

Preprocessing is crucial for preparing images for CNNs.

It includes resizing images to a uniform size, normalizing pixel values (typically to a 0-1 range), and converting images to grayscale or other color spaces if needed.

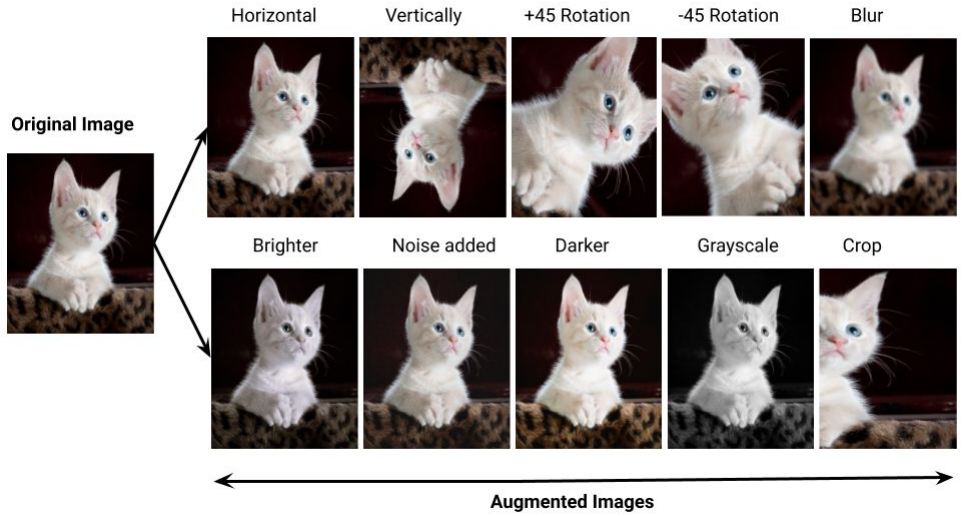
These steps ensure consistent input for the CNN, aiding in effective learning and reducing computational load.

Discussion – Why do we split our data into train, validation, and testing sets?

Data Augmentation Strategies in Image Classification

Data augmentation artificially expands the training dataset by applying random transformations like rotation, scaling, flipping, and cropping to the images.

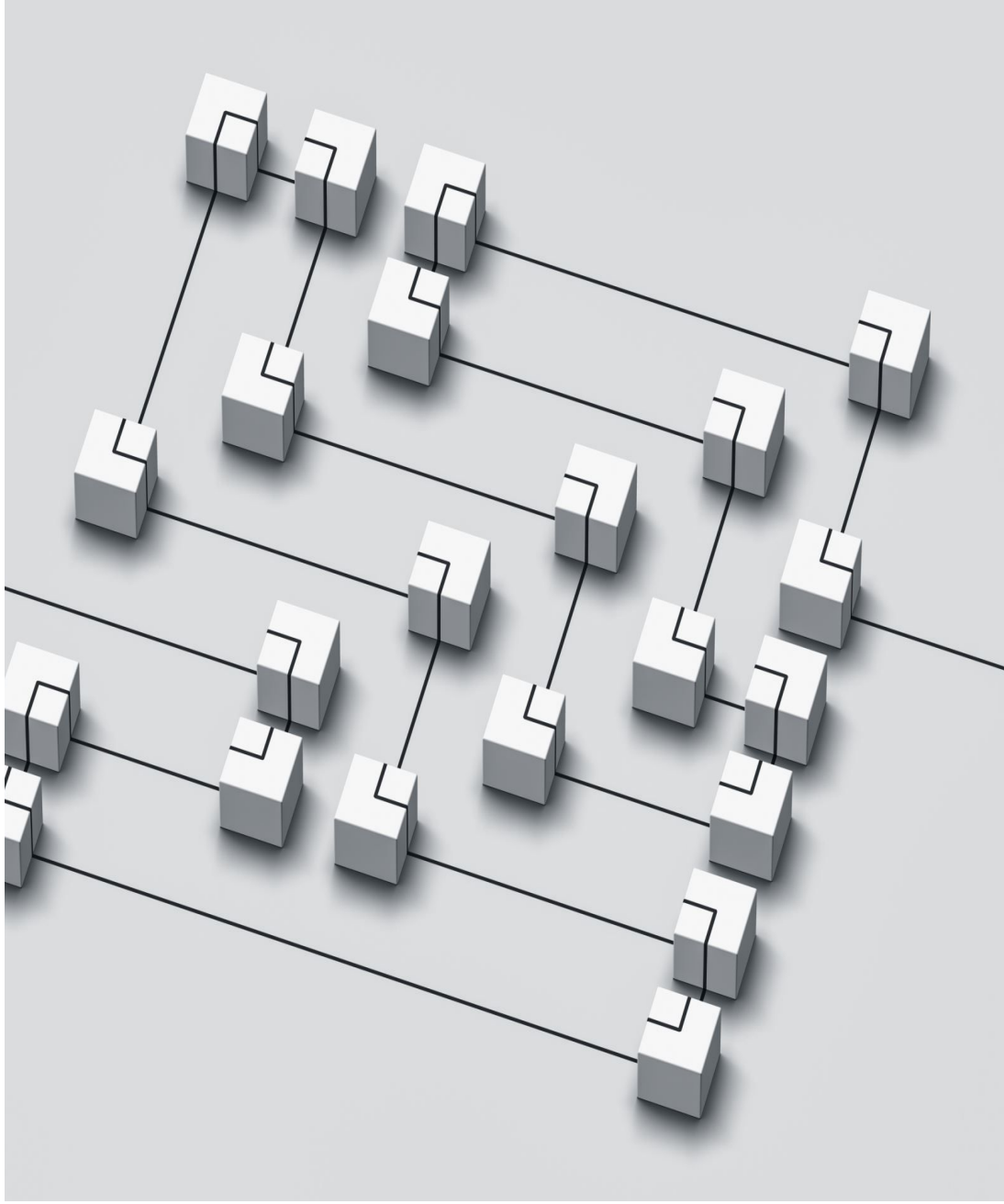
This process helps in reducing overfitting, as it exposes the model to a wider variety of features and scenarios, making it more robust and improving generalization.



Designing a CNN Architecture: Key Considerations

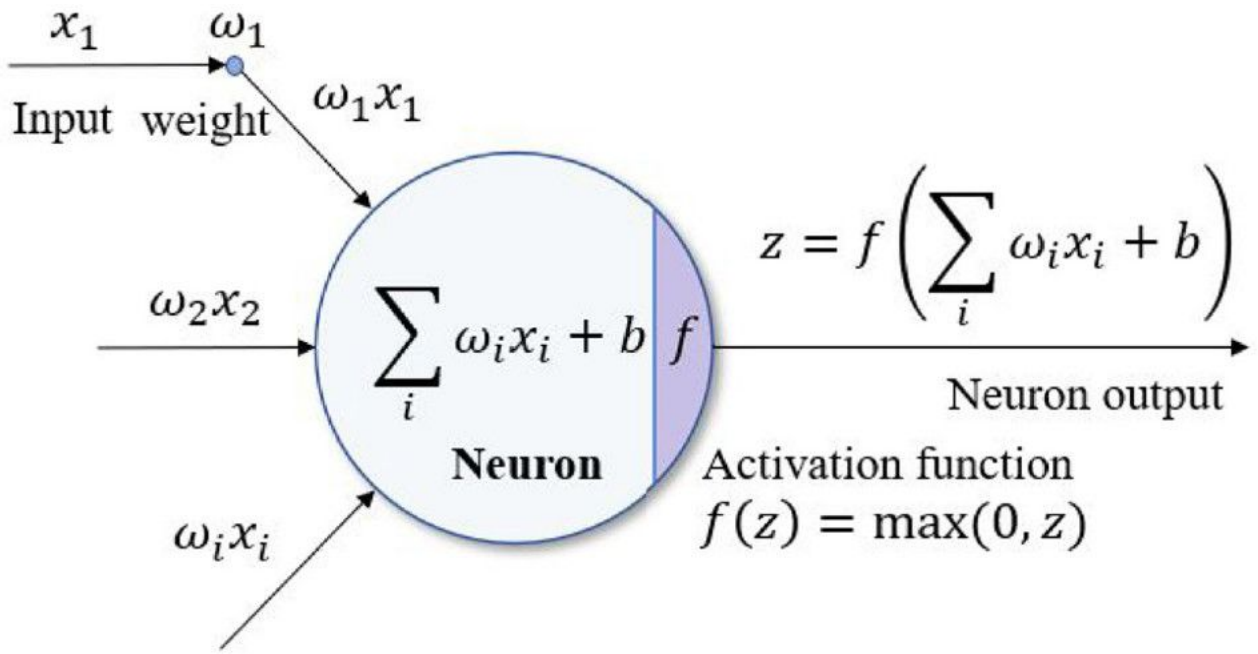
Designing a CNN involves decisions about the number of layers, types of layers (convolutional, pooling, fully connected), and their parameters (like filter size, stride, and activation functions).

The architecture should match the complexity of the task; deeper networks for more complex tasks, and consideration of computational efficiency and overfitting risks.



Activation Functions

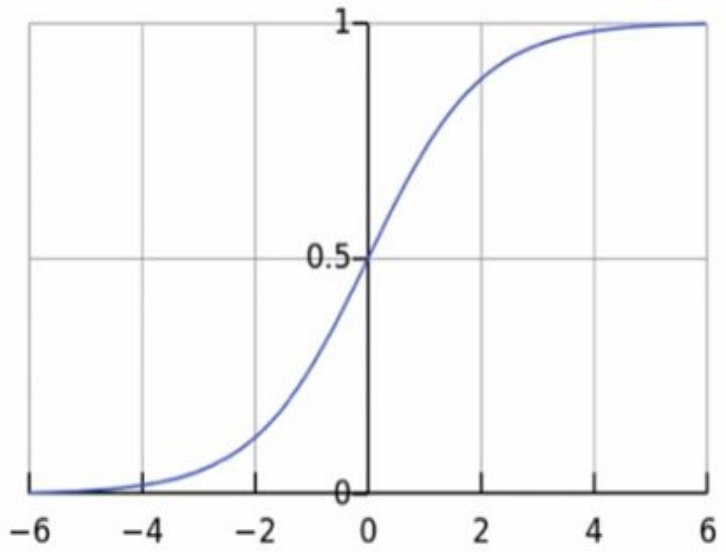
- Activation function determines if a neuron fires
- Introduces nonlinearity to the network
- Applied after convolution layer, after each fully conncted later and output layer allowing the network to learn and represent complex patterns in the data



Different types of Activation Functions

Sigmoid

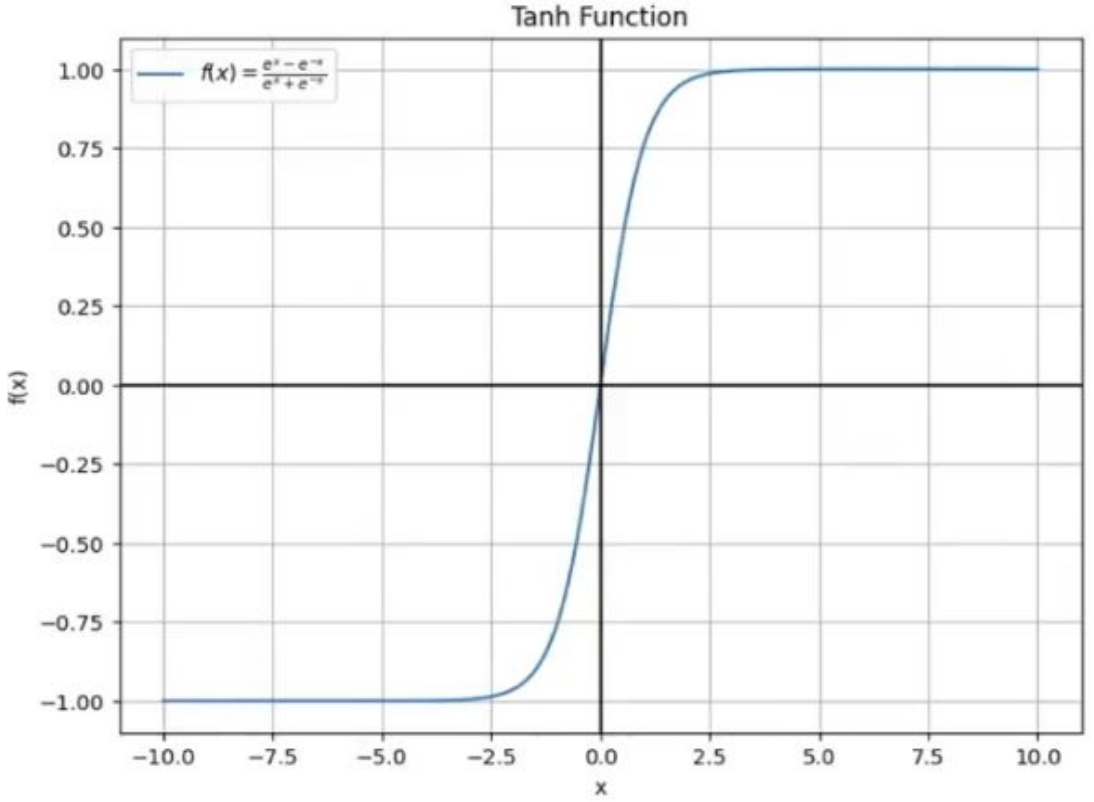
- Output of activation function between 0 and 1
- Suitable for binary classification tasks
- Vanishing gradient problem – near boundaries, the network doesn't learn quickly
- Used for output layer activation in binary classification



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

Tanh

- Maps inputs to a range between -1 and 1
- Provides a more balanced output with zero-centered data
- Smooth and differentiable activation function
- Shares vanishing gradient problem with sigmoid
- Used for handling negative input values

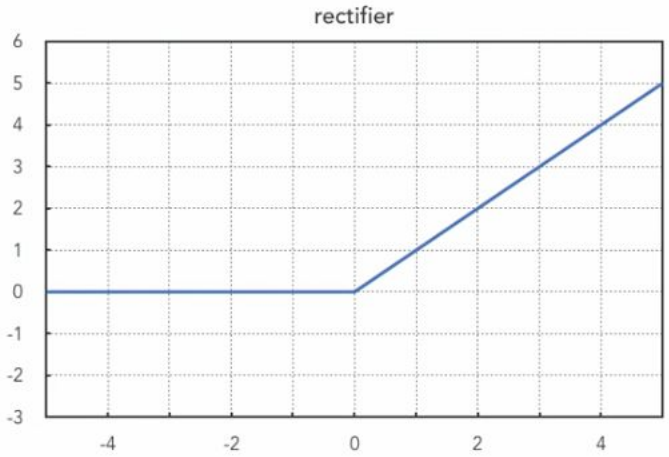


ReLU – Rectified Linear Unit

- Only input values > 0 are kept
- Range [0, ∞]
- f(x)= max(0, x)
- While keeping positive values unchanged, it promotes sparse representations, reducing overfitting
- Mitigates vanishing gradient problem, enabling faster learning
- Most commonly used for efficiency and in the hiddenlayers of feed forward neural networks

ReLU

$$f(x)= \begin{cases} 0 & \text{for } x<0 \\ x & \text{for } x=>0 \end{cases}$$



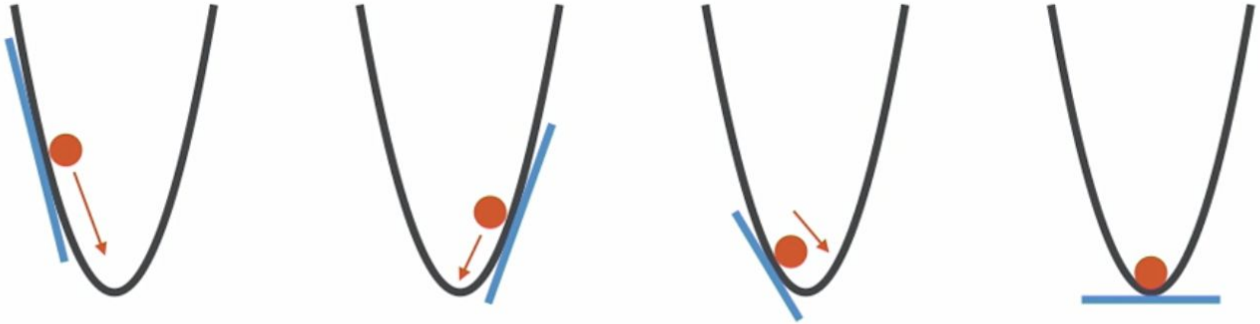
Loss Functions

- A loss function is a mathematical function that measures how well a model's predictions match the true outcomes
- Quantify the difference between model predictions and true outcome
- The goal of a loss function is to guide optimization algorithms in adjusting model parameters (weight, bias, convolutional filter values etc) to reduce this loss over time
- Appropriate loss function is vital for successful CNN training
- Example for loss function that are commonly used include:
  - Mean Squared Error
    - $MSE = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - y'_i)^2$
  - Cross-Entropy loss
    - Binary Cross-Entropy Loss (log loss) - For binary classification, whose output is probability value between 0 and 1
    - Categorical Cross-Entropy Loss – For multiclass classification problems, whose output is probability distribution over multiple classes

.

Gradient Descent

- Most models use gradient descent or its variants to minimize the loss
- It is an optimizing algorithm which is used to iterate through different combinations of weights to find the best combination of weights that minimizes the error
- The algorithm calculates the gradient of the loss function with respect to the model parameters and updates the parameters in the opposite direction of the gradient.



- Calculate the gradient at current position

```
while (gradient ! = 0)
if (gradient < 0)
      move right
else if (gradient > 0)
      move left
```

We have reached the lowest/optimal position

Back Propagation

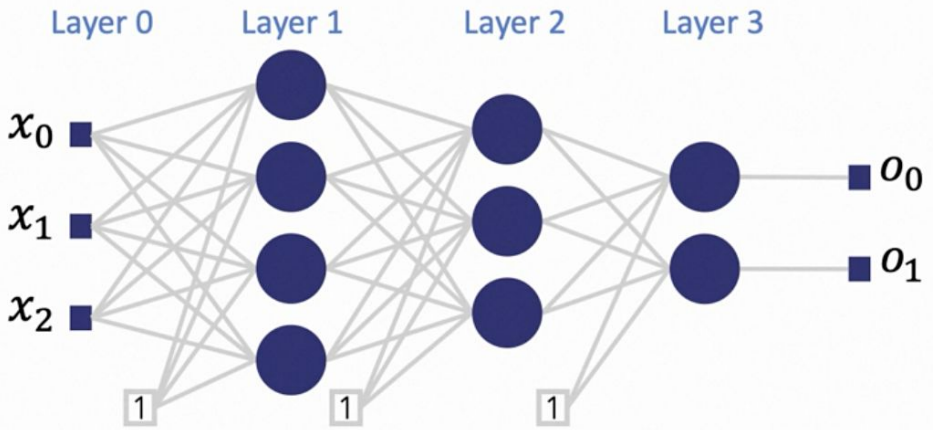
- A critical algorithm in training CNN used to compute gradients of the loss function with respect to the weights in a neural network.
- It happens only during training
- Basic Steps are;

    1. Feed a sample to the network
    2. Calculate the mean squared error
    3. Calculate the error term of each output neuron
    4. Iteratively calculate the error terms In the hidden layers
    5. Apply the delta rule
    6. Adjust the weights

Step 1: Feed a sample to the Network

$$y = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$x = \begin{bmatrix} 2 \\ 5 \\ 1 \end{bmatrix}$$

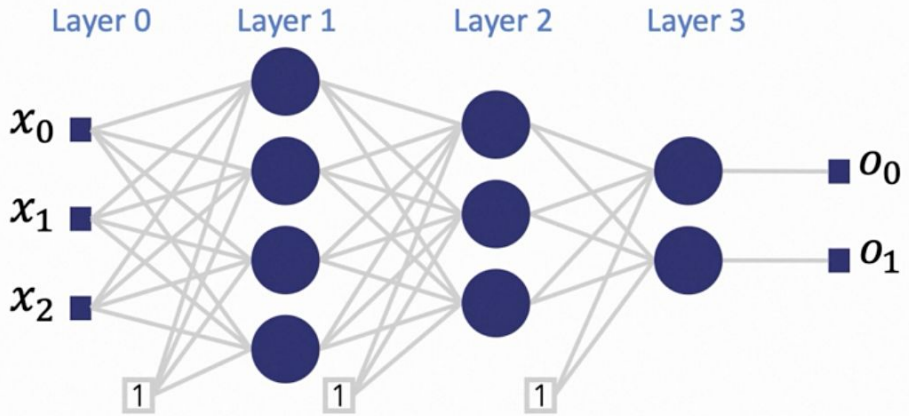


$$o = \begin{bmatrix} 0.2 \\ 0.49 \end{bmatrix}$$

Step 2: Calculate Mean Squared Error

$$\text{MSE} = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - o_i)^2 = 0.15 \qquad (y-o)^2 = \begin{bmatrix} 0.04 \\ 0.2601 \end{bmatrix}$$

$$x = \begin{bmatrix} 2 \\ 5 \\ 1 \end{bmatrix}$$

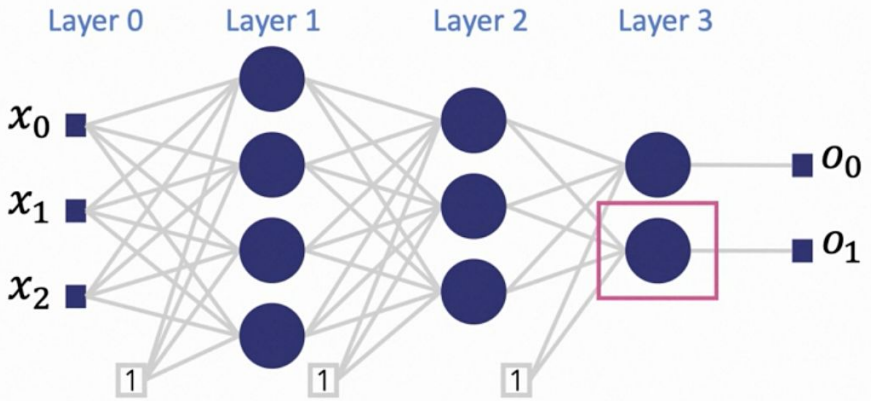


$$o = \begin{bmatrix} 0.2 \\ 0.49 \end{bmatrix}$$

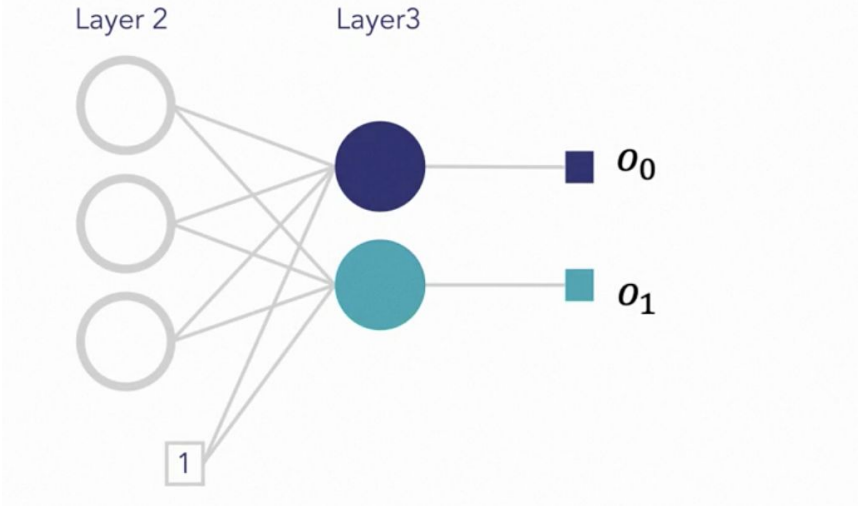
$$y = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Step 3: Calculate the Output Error Terms

$$\delta_k = o_k * (1 - o_k) * (y_k - o_k)$$

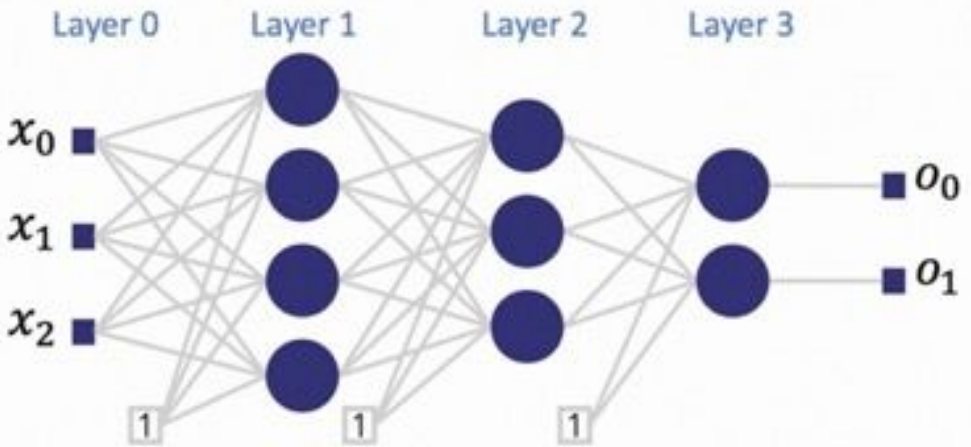


$$\delta_1 = o_1 * (1 - o_1) * (y_1 - o_1)$$

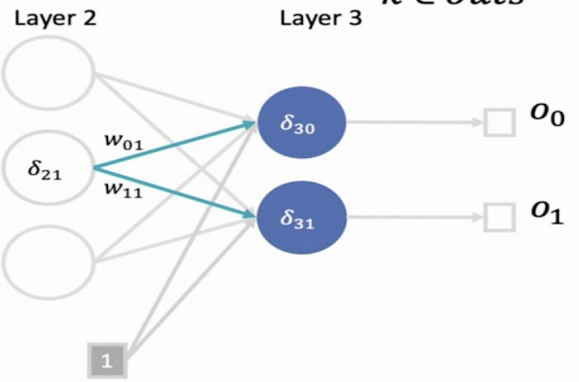


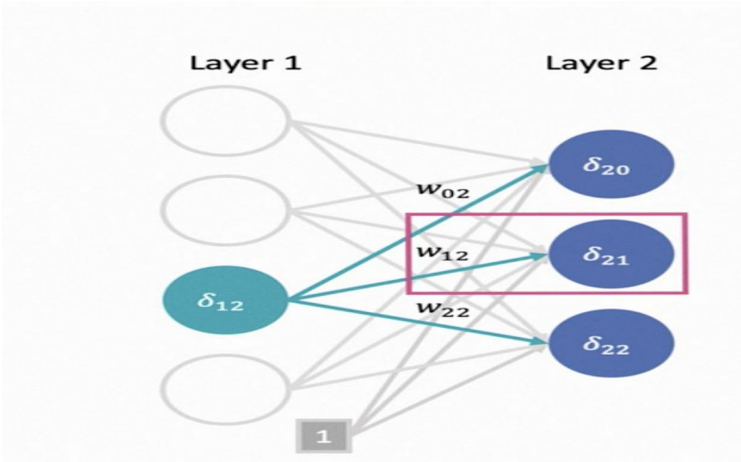
Step 4: Calculate the Hidden Layer Error Terms

$$\delta_h = o_h * (1 - o_h) * \sum_{k \in outs} w_{kh}\delta_k$$



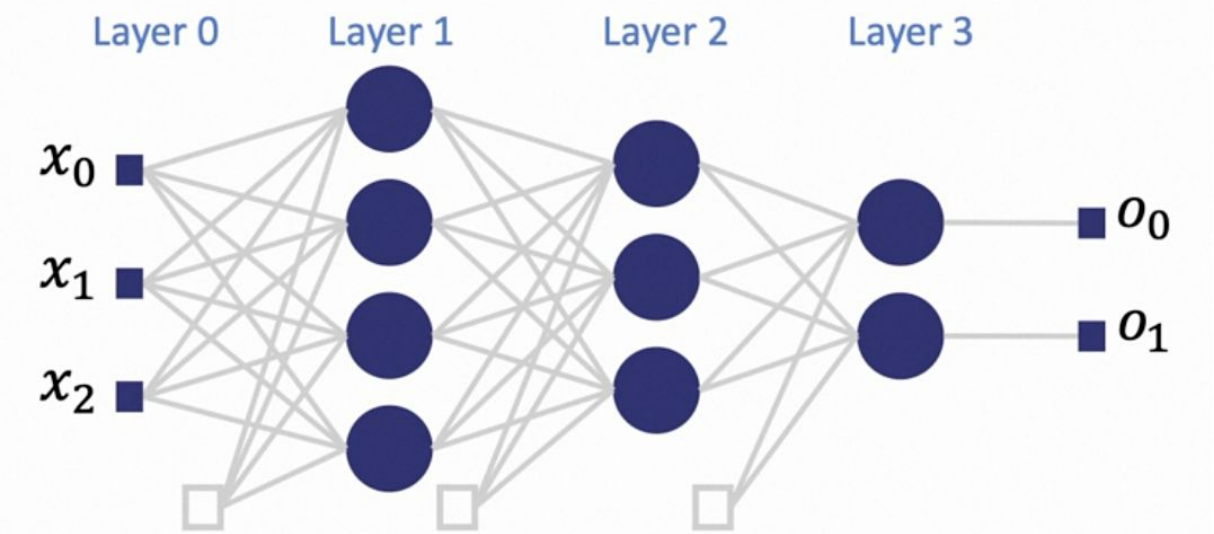
$$\delta_1 = o_1 * (1 - o_1) * \sum_{k \in outs} w_{k1}\delta_k$$





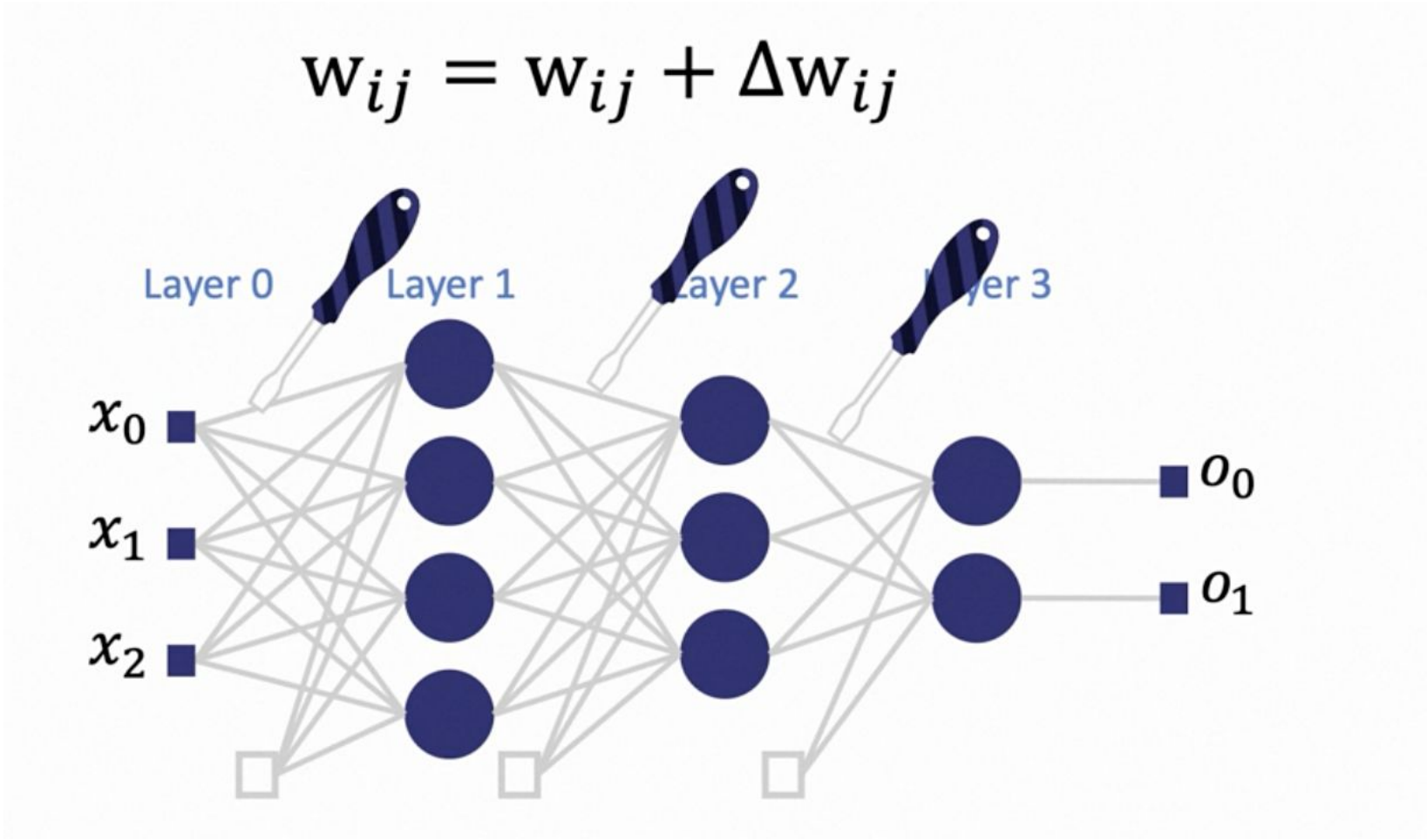
Step 5: Apply the Delta Rule

$$\Delta w_{ij} = \eta * \delta_i * x_{ij}$$



Step 6: Adjust the Weights

$$w_{ij} = w_{ij} + \Delta w_{ij}$$



Optimizers: Types and Their Impact on Training

Optimizers in CNNs are algorithms used to adjust the weights of the network to minimize loss.

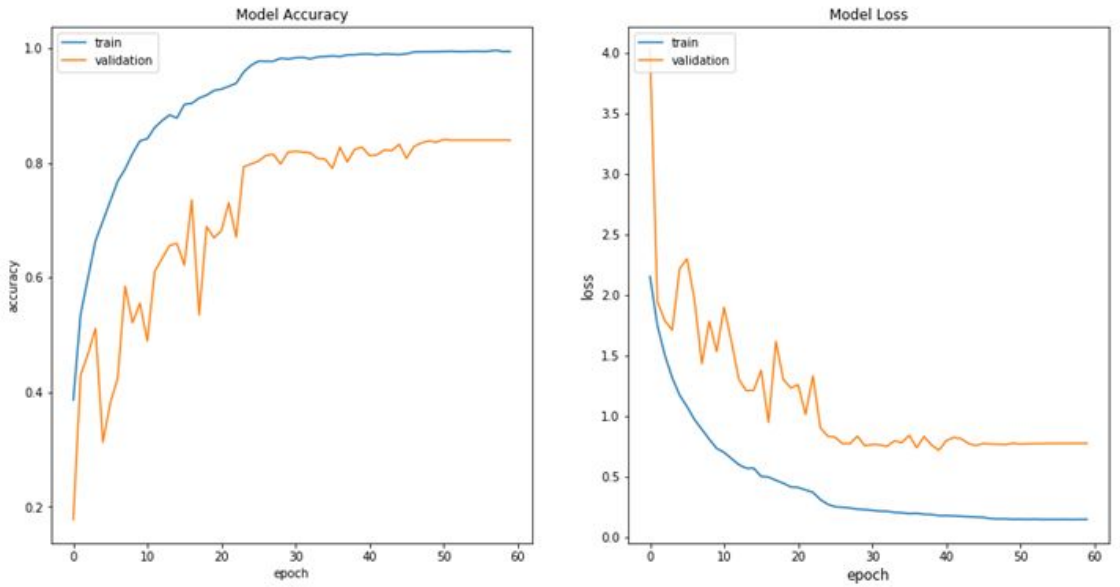
Key types include SGD (Stochastic Gradient Descent), which is simple yet effective; Adam, known for its adaptiveness to different problems; and RMSprop, which adjusts the learning rate during training.

The choice of optimizer affects the speed and quality of training, and sometimes a combination of optimizers is used for different stages of training to achieve better results.

Training a CNN: Steps and Best Practices

Training a CNN involves initializing weights, forward propagation to get predictions, calculating loss, and backpropagation to calculate gradients and optimizers adjust weights.

Best practices include using a validation set for hyperparameter tuning, applying early stopping to prevent overfitting, and periodically saving the model state for recovery. Monitoring training progress with metrics like loss and accuracy, both on training and validation sets, helps in understanding model performance and making necessary adjustments.

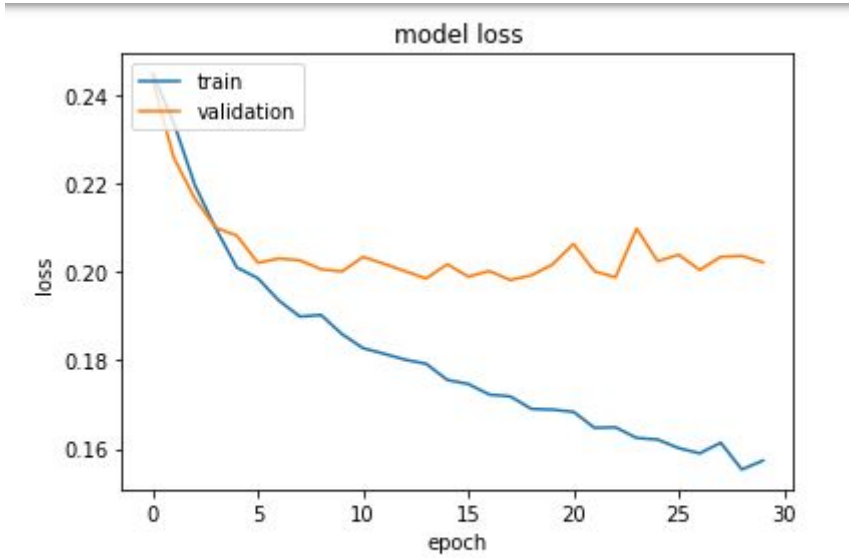


Understanding Overfitting in Deep Learning

Overfitting occurs when a CNN model learns the training data too well, including its noise and outliers, leading to poor performance on new, unseen data.

This usually happens in overly complex models with too many parameters.

Symptoms of overfitting include much higher accuracy on training data compared to validation data.



Strategies to Prevent Overfitting

To prevent overfitting:

1) Use dropout layers which randomly deactivate certain neurons during training, preventing co-adaptation of features.
2) Apply regularization methods like L1 (lasso) and L2 (ridge) which penalize large weights.
3) Augment the dataset to provide more varied training examples.
4) Simplify the model by reducing the number of layers or neurons.
5) Early stopping halts training when performance on a validation set starts to degrade.

Hardware Resources for Deep Learning: CPUs vs GPUs vs TPUs

Deep learning, particularly CNNs, requires significant computational resources.

CPUs, with fewer cores, are versatile but slower for this task.

GPUs, with thousands of cores, are ideal for the parallel processing needs of deep learning.

TPUs, designed specifically for neural network operations, provide even faster computations. Choice of hardware can significantly impact training time, cost, and scalability of deep learning models.

Optimizing CNNs for Efficient Resource Use

Efficient resource use in CNNs involves techniques like pruning (removing redundant neurons), quantization (reducing the precision of the numbers used), and using efficient architectures like MobileNets. These optimizations are crucial for deploying models in resource-constrained environments like mobile devices, ensuring a balance between performance and resource use.

Integrating CNNs with Other Deep Learning Techniques

Integrating CNNs with other deep learning techniques like Recurrent Neural Networks (RNNs) for video classification or Natural Language Processing (NLP) models for image captioning enhances their application scope. These integrations allow for multimodal learning, where CNNs process visual data while other models handle different data types like sequential data in videos or text in captions, leading to more comprehensive AI solutions.

Troubleshooting Common Issues in CNN Training

Common issues in CNN training include overfitting, underfitting, and convergence problems.

Strategies to troubleshoot include adjusting learning rates, modifying network architectures, and using techniques like batch normalization and dropout.

Ensuring high-quality and diversified training data is also crucial, as is regular monitoring of performance metrics during training to identify and address issues early.

Techniques to Address Underfitting

Underfitting, where a model fails to capture the underlying trend of the data, can be addressed by increasing the model complexity (adding more layers/neurons), training for longer durations, or using more powerful and diverse feature extraction methods. Another approach is to revisit data preprocessing and augmentation techniques to ensure the model receives sufficient and varied information during training.

CST8508_26W - Midterm Test

- Paper based exam on Feb 19 – 7.00pm – 8.00 pm
- Total Marks : 25
- Duration: 60 min
- Calculators allowed
- No other personal electronic devices allowed in the classroom during test
- Contributes to 15% of final grade
- Test Format
    - Multiple Choice Questions
    - Fill in the blanks Questions
    - Short answer Questions
    - Mathematical Questions