Deep Learning for Computer Vision: Comprehensive Study Guide

Section 1: Recurrent Neural Networks (RNNs) and Sequence Models

Recurrent Neural Networks (RNNs) are a special class of neural networks designed for processing sequential data. Unlike traditional feedforward networks, which assume all inputs are independent, RNNs maintain a **hidden state** that allows them to remember past information. This makes RNNs particularly powerful for tasks where the order and context of data matter.

1.1 Why Use RNNs? Sequential Data Problems

Sequential data occurs in many real-world problems, such as:

- **Time Series Data:** Stock prices, weather forecasting, or sensor data, where predictions depend on past trends.
- **Speech Recognition:** Understanding spoken language requires context from previous words or sounds.
- **Natural Language Processing (NLP):** Tasks like text generation, translation, or sentiment analysis depend on understanding the sequence of words.

Example: If a model sees the sentence *"The cat sat on the ..."*, it should predict *"mat"* based on previous words.

1.2 How RNNs Work: Hidden States and Time Steps

In an RNN, the input is processed one time step at a time, updating the hidden state h_t based on the current input x_t and the previous hidden state h_{t-1}:

$$h_t = \tanh(W_x x_t + W_h h_{t-1} + b),$$

where:

- W_x: Weights applied to the current input x_t.
- W_h: Weights applied to the previous hidden state h_{t-1}.
- b: Bias term.

- tanh: Activation function that outputs values between -1 and 1.

Key Insight: RNNs "remember" past information by passing hidden states forward through time. However, they only have short-term memory due to the **vanishing gradient problem**.

1.3 The Vanishing Gradient Problem

The vanishing gradient problem arises during backpropagation through time (BPTT). In long sequences, gradients become extremely small as they are multiplied repeatedly at each time step. This causes the network to "forget" long-term dependencies.
Real-World Analogy: Imagine trying to recall a conversation from last week but struggling to remember the details because too much information has faded away.

1.4 Long Short-Term Memory (LSTM): Solving the Problem

LSTMs were designed to address the vanishing gradient issue by introducing a **cell state** and a set of **gates** that regulate information flow. These gates allow LSTMs to selectively remember, forget, and update information across long time steps.
How It Works:

- **Forget Gate:** Determines what part of the previous cell state C_{t-1} to forget.
$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f),$$
where σ is the sigmoid function.
- **Input Gate:** Decides what new information to add to the cell state.
$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i), \quad \tilde{C}_t = \tanh(W_C x_t + U_C h_{t-1} + b_C).$$
- **Update Cell State:** Combines the forget gate and input gate to update the cell state:
$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t,$$
where $\odot$ is element-wise multiplication.
- **Output Gate:** Controls what information to output at time t:
$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o), \quad h_t = o_t \odot \tanh(C_t).$$

Key Insight: LSTMs can learn to retain relevant information for longer periods, making them suitable for tasks with long-term dependencies, such as language modeling and translation.

1.5 Gated Recurrent Units (GRUs): A Simplified LSTM

GRUs are a variant of LSTMs that combine the forget and input gates into a single **update gate**, reducing computational complexity while achieving similar performance.
GRU Update Equations:

- **Reset Gate:** Determines how much of the previous hidden state to ignore:
$$r_t = \sigma(W_r x_t + U_r h_{t-1}).$$
- **Update Gate:** Balances retaining the old hidden state and updating it with new information:
$$z_t = \sigma(W_z x_t + U_z h_{t-1}).$$
- **New Hidden State:**
$$\tilde{h}_t = \tanh(W_h x_t + U_h (r_t \odot h_{t-1})).$$
- **Final Hidden State:**
$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \tilde{h}_t.$$

Key Difference: GRUs are computationally faster and easier to implement compared to LSTMs, but they lack explicit control over the cell state.

Section 2: Attention Mechanisms

2.1 Why Do We Need Attention?

RNNs and LSTMs process sequences step-by-step, encoding the input into a fixed-size vector. This bottleneck causes problems when dealing with long sequences:

- The fixed-size vector cannot capture all the relevant information from long inputs.
- Critical parts of the input may be ignored or diluted.

Solution: Attention mechanisms dynamically focus on the most relevant parts of the input sequence during each step of processing.

2.2 How Attention Works

Given an input sequence, attention computes a set of **weights** to determine how much focus each input token should receive. The steps are:

1. Compute **alignment scores** $e_{t,i}$ that measure the relevance of the input x_i to the current output y_t.
2. Apply the **softmax** function to normalize alignment scores into attention weights:
$$a_{t,i} = \frac{\exp(e_{t,i})}{\sum_j \exp(e_{t,j})}.$$
3. Use these weights to compute a weighted sum of the inputs, which becomes the context vector:
$$c_t = \sum_i a_{t,i} x_i.$$

Intuition: Attention works like highlighting key parts of a textbook while studying. The model emphasizes relevant inputs and ignores less useful ones.

2.3 Types of Attention

- **Soft Attention:** Weights are applied to all inputs (continuous).
- **Hard Attention:** Focuses on specific inputs (discrete) but is harder to train.
- **Self-Attention:** Allows each token in a sequence to attend to all other tokens. This is the core of the Transformer model.

Real-World Applications: Attention mechanisms are critical in tasks such as machine translation, image captioning, and summarization.

Section 3: CNNs and Image Processing

3.1 Convolutional Neural Networks (CNNs)

CNNs are designed for image data. Unlike dense neural networks, CNNs exploit spatial hierarchies in images:

- **Convolutional Layers:** Learn filters (small matrices) that detect patterns like edges, textures, and shapes.
- **Pooling Layers:** Reduce the spatial dimensions of the feature maps, making the network efficient.

How Convolution Works: A small filter (kernel) slides over the input image and computes dot products at each position. This produces a feature map highlighting specific patterns detected by the filter.
Example: In an image of a cat:

- Lower layers may detect simple patterns like edges, corners, or textures.
- Middle layers may combine these patterns to detect parts of the cat's face (e.g., eyes, ears).
- Deeper layers may recognize the entire cat.

3.2 Semantic vs Instance Segmentation

Segmentation tasks aim to classify pixels in an image into meaningful categories:

- **Semantic Segmentation:** Assigns each pixel a class label (e.g., road, building, tree). However, it does not differentiate between individual objects of the same class.
- **Instance Segmentation:** Differentiates between individual objects of the same class (e.g., identifying multiple cars in a parking lot).

Example: In a street scene:

- Semantic segmentation labels all cars as **"car"**.
- Instance segmentation identifies **each car** as a separate object.

Key Challenge: Background pixels often dominate the dataset, leading to class imbalance.
Fixes:

- Use weighted loss functions to penalize errors on underrepresented classes.
- Data augmentation to generate examples of minority classes.
- Oversampling or generating synthetic data for rare classes.

Section 4: Object Detection

4.1 Faster R-CNN: Improving Efficiency

Faster R-CNN is an object detection model that combines region proposal and object classification into a single, efficient pipeline.
Steps:

1. A **Region Proposal Network (RPN)** generates candidate bounding boxes (regions of interest) directly from the convolutional feature maps.
2. The shared convolutional features are reused for both region proposals and object classification, reducing computation time.
3. A classifier predicts the object class and refines the bounding box coordinates.

Key Insight: Faster R-CNN improves efficiency by learning to propose regions instead of relying on external methods like selective search.

4.2 Anchor Boxes

Object detection models like Faster R-CNN and YOLO use **anchor boxes** to detect objects of different scales and aspect ratios.
The total number of anchor boxes can be calculated as:

$$\text{Total anchor boxes} = \text{Feature Map Height} \times \text{Feature Map Width} \times K,$$

where K is the number of anchor boxes per pixel.
Why Anchor Boxes? They allow the model to detect objects of various shapes and sizes efficiently within a single forward pass.

Section 5: Generative Models

5.1 Variational Autoencoders (VAEs) vs GANs

Generative models aim to create new data samples that resemble the training data. Two popular approaches are **VAEs** and **GANs**:

Property	**VAEs**	**GANs**
Training Objective	ELBO (Reconstruction + KL)	Adversarial loss (minimax)
Latent Space	Structured, continuous	Implicit, unstructured

Mode Collapse in GANs: GANs can suffer from **mode collapse**, where the generator produces limited outputs that fool the discriminator.
Fixes:

- Use Wasserstein GAN (WGAN) for stable training.
- Add minibatch discrimination to encourage diversity in outputs.
- Apply spectral normalization to the discriminator to improve stability.

Section 6: Transformers

6.1 Why Transformers?

Transformers revolutionized sequence modeling by replacing RNNs and LSTMs with a fully **self-attention** mechanism. Unlike RNNs, Transformers process all tokens in parallel, which significantly speeds up training and allows better handling of long-range dependencies.
Key Idea: Self-attention allows the model to relate each token in the input to every other token, regardless of their position in the sequence.

6.2 Multi-Head Attention

Multi-head attention enhances the expressiveness of self-attention by allowing the model to focus on different parts of the input simultaneously.
Example: For input dimension $D = 64$ and 8 attention heads:

- Each head processes $D/8 = 8$-dimensional subspaces.
- Attention operates on smaller feature spaces, enabling the model to learn diverse relationships.