Homework 3

‘‘Computer vision’’

professor : Dr. Mahmoudi Aznaveh

Spring 2024

By Katayoun Kobraei (99222084)

Theory questions:

1. Faster RCNN and Fast RCNN are two of powerful models in object detection tasks which are the improved form of RCNN in speed and accuracy. Here we analyze their loss function separately:

   Fast RCNN:

   Fast RCNN uses the region proposal network (RPN) to accurately detect objects by combining classification and localization in a single framework. It improves the training and testing speed as well as increasing the detection accuracy. Since Fast RCNN is an end-to-end learning architecture (Except the region proposal generation part) to learn the class of object as well as the associated bounding box position and size, the loss is multi-task loss. The multi-task loss that consists of two main components: classification loss and regression loss.

   - Classification Loss:
     Classification loss measures how well the network classifies each proposed region into one of object classes or the background. It typically uses a softmax loss over multiple classes. This is its formula:

     $$L_{cls}(p, u) = -\ log(p_u)$$

     where p is the predicted probability over classes, and u is the ground-truth class and p_u is the predicted probability of the true class u.

   - Regression Loss:
     This loss measures how accurately the network predicts the bounding box coordinates for each proposed region. It usually employs a smooth L1 loss, which is less sensitive to outliers than the L2 loss. This is its formula:

     $$L_{reg}(t^u, v) = smooth_{L1}(t_i^u - v_i)$$

Where t^u are the predicted bounding box coordinates for class u, and v are ground-truth bounding box coordinates. The sum is taken over the four coordinates: x(center x), y(center y), w(width), and h(height).

- Smooth L1 Loss

  The smooth L1 loss function is used for bounding box regression:

$$smooth_{L1}(x) = \begin{cases} 0.5x^2 & if\ |x| \geq 1 \\ |x| - 0.5 & otherwise \end{cases}$$

  This function is less sensitive to outliers compared to the L2 loss (squared loss), as it behaves like L2 loss for small errors (where $|x|<1$) and like L1 loss for larger errors (where $|x| \geq 1$). This property helps in stabilizing training and avoiding exploding gradients.

The total loss $L(p, u, t^u, v)$ in the Fast R-CNN network is a combination of the classification loss and the localization (regression) loss. The formula is:

$$L(p, u, t^u, v) = L_{cls}(p, u) + \lambda[u \geq 1]L_{reg}(t^u, v)$$

Where $L_{cls}(p, u)$ is the classification loss, $L_{reg}(t^u, v)$ is the regression loss and λ is a weight parameter to balance the classification and localization losses. $[u \geq 1]$ is an indicator function that is 1 if $u \geq 1$ and 0 otherwise, ensuring that the localization loss is only applied to non-background classes.

Faster RCNN:

Faster RCNN is the extension of Fast RCNN which integrates the Region Proposal Network (RPN) directly into the object detection pipeline. This eliminates the need for an external region proposal algorithm. The loss function in Faster RCNN also includes multi-task losses for both the RPN and the Fast R-CNN stages so it comes with 4 losses
1.RPN classification (Object foreground/background)
2.RPN regression (Anchor → ROI)
3.Fast RCNN Classification (object classes).
4.Fast RCNN Regression (ROI → Bounding Box)
We already explained Fast RCNN losses so we now we explain RPN loss:

- RPN Loss:
  The RPN proposes candidate object bounding boxes. The RPN loss is composed of classification loss and regression loss similar to Fast R-CNN.

  RPN Classification Loss (L_rpn_cls): Binary classification loss (object vs. not object) using cross-entropy loss.

  RPN Regression Loss (L_rpn_reg): Smooth L1 loss for bounding box regression.

  So the RPN loss would be:

  $$L(\{p_i\},\{t_i\}) = \frac{1}{N_{cls}}\Sigma_i L_{cls}(p_i, p_i^*) + \lambda\frac{1}{N_{reg}}\Sigma_i p_i^* L_{reg}(t_i, t_i^*)$$

  Where $\{p_i\}$ are the predicted probabilities of anchor i being an object. $\{t_i\}$ are the predicted bounding box coordinates for anchor i. p_i^* is the ground-truth label (1 for object, 0 for background) for anchor i. t_i^* are the ground-truth bounding box coordinates for anchor i. N_{cls} is the number of anchors. N_{reg} is the number of anchors that are labeled as objects (where

$p_i^* = 1$). λ is a weight parameter to balance the classification and regression losses.

Components of the Loss Function

1. Classification Loss L_{cls}:

   It measures the accuracy of the predicted probabilities $\{p_i\}$. The classification loss is typically a log loss (cross-entropy loss):

   $$L(p_i, p_i^*) = -p_i^* log(p_i) - (1 - p_i^*) log(1 - p_i)$$

   This loss is summed over all anchors and normalized by the total number of anchors N_{cls}.

2. Regression Loss L_{reg}:

   It measures how accurately the network predicts the bounding box coordinates $\{t_i\}$. The regression loss is typically a smooth L1 loss which was explained in previous network.

The total RPN loss $L(\{p_i\}, \{t_i\})$ is a weighted sum of the classification and regression losses, where the classification loss is averaged over all anchors, the regression loss is averaged only over anchors labeled as objects and the parameter λ balances the importance of the classification and regression terms.

Relationship Between the RPN and the Given Image

The image shows how the Region Proposal Network (RPN) generates region proposals using anchor boxes. Now we explain image in detail and its relation to the RPN in the Faster R-CNN framework:

1. **Anchor Box**: The initial reference box around which region proposals are made. Anchors serve as a starting point for predicting bounding boxes. The RPN generates multiple anchor boxes centered at each sliding window location on the feature map. Each anchor has different scales and aspect ratios to accommodate various object sizes and shapes. In the image, the anchor box is represented by the blue box.

2. **Predicted Bounding Box**: The refined bounding box that the RPN predicts based on the anchor box. The RPN uses a sliding window mechanism with a 3×3 kernel to traverse the feature map. The center of each sliding window is considered the anchor point. For each anchor point, k anchor boxes are generated with different scales and aspect ratios.

3. **Offsets (Δ)**: These are the adjustments applied to the anchor box coordinates to predict the bounding box. The RPN predicts offsets for each anchor box. These offsets adjust the anchor box coordinates to better fit the object, resulting in the predicted bounding box. The red box in the image represents the new predicted bounding box after applying the offsets.

Improvement Provided by RPN:

1. **Efficiency**: By generating region proposals directly within the network, the RPN eliminates the need for an external region proposal algorithm like Selective Search, significantly speeding up the detection process.
2. **Unified Framework**: The RPN allows Faster R-CNN to be trained end-to-end, enabling unified optimization and a single loss function for the entire network. This integration results in a more streamlined and efficient training process.
3. **Adaptability**: The use of multiple scales and aspect ratios for anchor boxes allows the RPN to handle objects of various sizes and shapes, improving detection performance across diverse datasets.

2. **a) Differences Between Lucas & Kanade and Horn & Schunck Algorithms:**

The assumption in Lucas & Kanade optical flow is constant within a small window of pixels but The assumption in Horn & Schunck Algorithms is that the optical flow varies smoothly over the entire image.
Lucas & Kanade Solves the optical flow constraint equation within a local window using a least squares approach. This involves solving a set of linear equations derived from the brightness constancy assumption.
The algorithm is local, operating on a small neighborhood of pixels, which makes it computationally efficient for small displacements.
However, for Horn & Schunck, it uses a global approach by introducing a smoothness constraint to the optical flow field which involves solving a set of partial differential equations that balance the brightness constancy assumption with the smoothness constraint. The algorithm is iterative and considers the entire image, making it computationally more intensive but capable of handling large displacements.

Noise Resistance:

The local approach of Lucas & Kanade where the optical flow is estimated in small neighborhoods, tends to average out the noise over the window. This local averaging effect makes the Lucas & Kanade method more robust to noise compared to Horn & Schunck. The global nature of the Horn & Schunck method makes it sensitive to noise. The smoothness constraint can propagate noise throughout the optical flow field, leading to less accurate results in noisy conditions.

In Lucas & Kanade the main assumption is that within a small window, the flow is approximately constant. This assumption is often valid even in the presence of noise, as noise can be smoothed out by the local averaging. But for the other one the assumption of smooth flow across the entire image can be problematic in the presence of noise, as the noise can cause the flow field to be inaccurately smoothed.

Comparison of Assumptions:

In summary Lucas & Kanade assumes a constant flow in a small neighborhood and More robust to noise due to local averaging. So it will be more suitable for small displacements. But Horn & Schunck assumes a globally smooth flow across the entire image and it is more sensitive to noise due to the global nature. So it will be more suitable for large displacements and detailed flow fields.

b) The Role of the Regularization Parameter α in Horn & Schunck Method

In the Horn & Schunck optical flow method, the regularization parameter α plays a crucial role in balancing the two main components of the algorithm: the data term and the smoothness term.

1. **Data Term**: Here it ensures that the estimated optical flow is consistent with the image brightness constancy assumption. It is derived from the optical flow constraint equation: →

   $$E_d = (\nabla I . \vec{V} + I_t)^2$$

   where ∇I is the spatial gradient of the image, vector v is the optical flow vector, and I_t is the temporal gradient of the image.

2. **Smoothness Term**: Here it imposes a smoothness constraint on the flow field to ensure that the flow varies smoothly across the image. It is represented by the sum of the squared magnitudes of the flow gradients:

   $$E_s = (||\nabla u||^2 + ||\nabla v||^2)$$

   where u and v are the horizontal and vertical components of the optical flow.

The Role of α: The overall energy function that the algorithm minimizes is a weighted sum of the data and smoothness terms:

$$E = \int [\nabla I . \vec{v} + I_t)^2 + \alpha(||\nabla u||^2 + ||\nabla v||^2) dx\, dy$$

Here α is the regularization parameter that balances the importance of the data term and the smoothness term.A high Alpha places more emphasis on the smoothness term, enforcing a smoother flow field. However a low alpha places more emphasis on the data term, allowing the flow field to follow the brightness constancy assumption more closely, potentially at the cost of smoothness.

Effect of Changing α: If we increase alpha the flow field becomes smoother, so small variations in flow are suppressed, leading to a more uniform flow. This leads to inaccuracies in areas with rapid changes in flow (e.g., edges of moving objects) because the smoothness constraint may over-smooth the flow. As an instance in a video where a ball is moving quickly against a static background, a high α might cause the flow vectors around the ball's edges to be smoothed excessively, blending the ball's motion with the background.

On the other hand flow field adheres more closely to the image gradients. So it captures detailed variations in flow, including sharp changes at object boundaries and this leads to noisier flow estimates because small variations and noise in the image gradients are not smoothed out effectively. In the same video of a moving ball, a low α would better capture the sharp motion boundaries of the ball but might also introduce noise in the flow field due to image noise or small artifacts.

An example in Optical Flow Calculation Using Horn & Schunck

Let's use a simple numerical example to illustrate the effect of the regularization parameter α on the optical flow computed by the Horn & Schunck method. We'll use two small 3x3 matrices representing frames from a video.

Frame 1: $\begin{pmatrix} 10 & 20 & 30 \\ 15 & 25 & 35 \\ 5 & 10 & 15 \end{pmatrix}$

Frame 1: $\begin{pmatrix} 12 & 22 & 32 \\ 17 & 27 & 37 \\ 7 & 12 & 17 \end{pmatrix}$

Here pixels in Frame2 are slightly shifted and increased in intensity compared to Frame1, meaning a small motion to the right and down.

We will calculate the optical flow for two different values of α: high alpha α=10 and low alpha α=0.1.

α=0.1 → Calculating gradient using masks:

I_x: $\begin{pmatrix} 2 & 2 & 2 \\ 2 & 2 & 2 \\ 2 & 2 & 2 \end{pmatrix}$

I_y: $\begin{pmatrix} 5 & 5 & 5 \\ 5 & 5 & 5 \\ 5 & 5 & 5 \end{pmatrix}$

I_t: $\begin{pmatrix} 2 & 2 & 2 \\ 2 & 2 & 2 \\ 2 & 2 & 2 \end{pmatrix}$

Optical flow estimation might yield:

$u \simeq$: $\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$

$$v \simeq: \begin{pmatrix} 0.4 & 0.4 & 0.4 \\ 0.4 & 0.4 & 0.4 \\ 0.4 & 0.4 & 0.4 \end{pmatrix}$$

α=10 → the same gradient as above. Optical flow estimation might yield:

$$u \simeq: \begin{pmatrix} 0.6 & 0.6 & 0.6 \\ 0.6 & 0.6 & 0.6 \\ 0.6 & 0.6 & 0.6 \end{pmatrix}$$

$$v \simeq: \begin{pmatrix} 0.1 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0.1 \\ 0.1 & 0.1 & 0.1 \end{pmatrix}$$

So a small alpha leads to more precise but potentially noisier flow estimates, but a large alpha results in smoother but less detailed flow estimates.

c) The Aperture Problem in Optical Flow and Its Impact on Motion Estimation:

The aperture problem is a fundamental issue in motion estimation and optical flow analysis. It arises due to the inherent limitations of observing motion through a small, finite-sized aperture (or window). When we observe motion through a small window (aperture), only the component of the motion perpendicular to the edge of the object is visible. So we can not determine motion parallel to the edge from this limited viewpoint.

Consequently, the true motion vector would be ambiguous within the confines of the aperture.

Let's imagine a straight line moving across a circular aperture. We will see the line move but we can not determine whether it’s moving purely horizontally, vertically, or diagonally. Only the component of motion perpendicular to the line's orientation is perceptible. Also multiple motion vectors could produce the same observed change within the aperture.

Impact on Motion Estimation

The aperture problem significantly impacts the accuracy of optical flow estimation in several ways. With this problem the motion of points along an edge cannot be fully determined and only motion perpendicular to the edge is accurately detected. Methods like Horn & Schunck use global smoothness assumptions to resolve ambiguities. These assumptions may not always hold, leading to inaccurate flow estimation, especially near object boundaries. Local methods like Lucas & Kanade rely on small windows for motion estimation. These windows suffer from the aperture problem, resulting in less reliable estimates for complex motions.

Addressing the Aperture Problem

Several strategies are used to mitigate the aperture problem's effects on motion estimation:

1. **Global Methods**:
    - **Horn & Schunck**: Incorporates a global smoothness constraint that encourages a coherent flow field, reducing ambiguity by leveraging information from the entire image.
2. **Multiple Scales**:
    - Using multi-scale approaches (pyramidal methods) to capture motion at different resolutions helps overcome the limitations of small apertures.
3. **Feature-Based Methods**:
    - Tracking distinct features (e.g., corners, textures) that provide more reliable motion information than uniform edges.
    - Methods like Shi-Tomasi Good Features to Track can identify points less susceptible to the aperture problem.#

3. a) **Comparing FlowNet Variants: FlowNetS and FlowNetC**

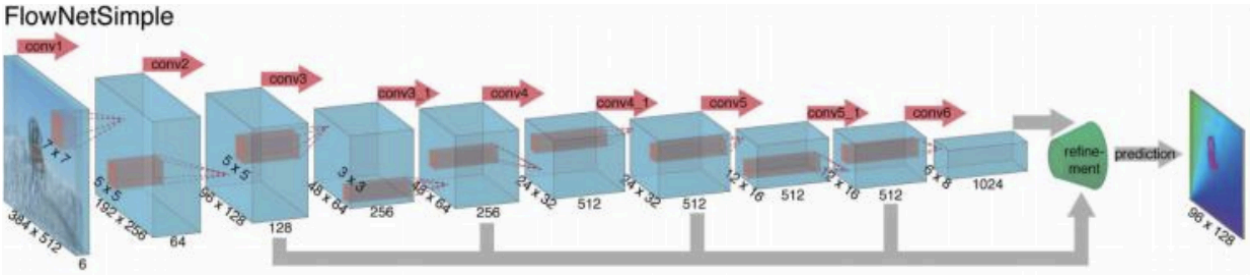
FlowNet is a deep learning architecture specifically created for estimating optical flow. It comes in two primary variants: FlowNetS and FlowNetC, each differing in their architectural design and the unique advantages they offer for this task.

To estimate optical flow, the model processes two sequential video frames through a convolutional network to detect displacement changes (velocity). The training involves using a set of image pairs along with ground truth data, optimizing the output (x-y flow fields) using the Euclidean distance error.

Pooling is essential in this process to make network training computationally feasible and to aggregate information over large areas of the input images. However, since pooling reduces resolution, a refinement process is necessary to ensure dense per-pixel predictions. Both FlowNet models incorporate a CNN-based autoencoder structure, consisting of an encoder to capture feature maps and a decoder to enhance resolution, effectively refining the optical flow to a high resolution.

FlowNetS Architecture:

- FlowNetS has a straightforward architecture inspired by traditional CNNs. It takes a pair of consecutive frames as input, concatenates them along the channel dimension, and processes them through a series of convolutional layers to estimate optical flow.
- This network is easy to implement and train and it is computationally efficient and faster in inference time. It also performs well for general optical flow estimation tasks.



FlowNetC Architecture:

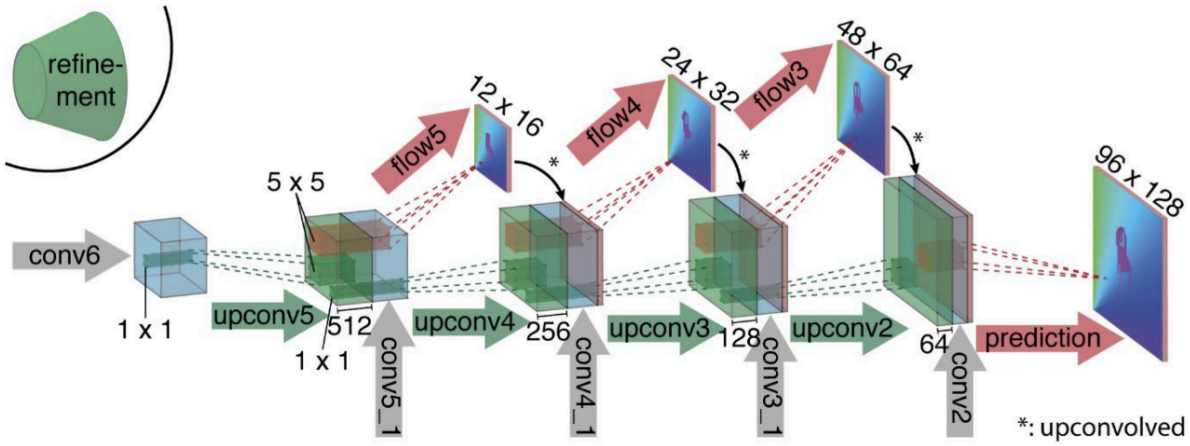
- FlowNetC introduces a correlation layer to measure similarity between feature maps from the two input frames. It uses a dual-stream architecture where each stream processes one of the input frames separately through its own network. The features from these streams are correlated using the correlation layer to compute a cost volume encoding displacement information.

  The correlation layer performs multiplicative patch comparisons between two feature maps (f1 and f2). Given two multi-channel feature maps, the correlation operation involves each pixel from f2 performing a convolution operation with each pixel from f1, producing a single-channel feature map. This process repeats for all pixels in f2, resulting in a feature map with matching information passed to subsequent layers for motion prediction.

$$C(x_1, x_2) = \sum_{o \in [-k,k] \times [-k,k]} < f_1(x_1 + o), f_2(x_2 + o) >$$

- The network enhances the network's ability to learn displacement between corresponding pixels, improving optical flow estimation.it also captures large displacements between frames more effectively. This will provide better robustness against noise and varying illumination conditions.

Both FlowNetC and FlowNetS have The expanding part of the network consists of ‘upconvolutional’ layers (unpooling and convolution). The refinement involves applying ‘upconvolution’ to feature maps, concatenating them with corresponding feature maps from the contracting part (skip connections), and an upsampled coarser flow prediction. This preserves both high-level information from coarser feature maps and fine local information from lower layer feature maps. Each step increases the resolution twice, repeating four times, resulting in a predicted flow with resolution still four times smaller than the input.



FlowNetS is a good choice for applications where computational efficiency is crucial, and the scene does not involve very large displacements or highly dynamic motion. FlowNetC is more suitable for scenarios with significant motion and where higher accuracy in optical flow estimation is required, albeit with a higher computational cost

b) **Components of FlowNetC:**

- Encoder: The encoder is responsible for extracting high-level features from the input frames. It consists of multiple convolutional layers that progressively downsample the input, capturing increasingly abstract and complex features. The encoder processes the two input frames separately but with shared weights to ensure that the extracted features are comparable.
- Correlation Layer: It computes a cost volume that measures the similarity between feature maps from the two encoded frames. This layer helps in explicitly capturing the displacement between corresponding pixels by comparing features at different spatial locations. The output of the correlation layer is a feature map that encodes the potential matches between the frames, aiding in accurate flow estimation.
- Warping Operation: The warping operation uses the estimated flow to align (or warp) one frame towards the other. This operation is essential for refining the

flow estimates iteratively by ensuring that the features from the two frames are aligned according to the current flow prediction. Warping helps in minimizing the difference between the frames, leading to more accurate flow estimation in subsequent stages.

- Decoder: The decoder takes the output from the correlation layer and progressively upsamples it to produce the dense flow field. It uses a series of deconvolutional (transposed convolutional) layers to upsample the low-resolution correlation features to the original input resolution. The decoder refines the flow estimates at each stage, incorporating information from the correlation layer and previous warping operations.

Challenges in FlowNetC

1. Large displacements pose a significant challenge for optical flow models because they require capturing substantial motion between frames. The correlation layer helps by explicitly encoding the potential matches between distant pixels, but accurately estimating large displacements still remains difficult. FlowNetC may struggle with very large motions because the spatial resolution of the features decreases as they are downsampled in the encoder, potentially losing fine details necessary for precise large displacement estimation.
2. Occlusions occur when parts of the scene in one frame are not visible in the other frame, making it challenging to find corresponding points. When an object moves in the scene, it can cover (occlude) other objects or parts of the background that were visible in the previous frame. For example, if a car moves to the right, parts of the scene behind the car that were visible in the previous frame may become hidden (occluded) in the current frame. Also If the camera moves, it changes the viewpoint, which can lead to occlusions. For instance, if the camera pans to the left, objects on the right edge of the frame may come into view while objects on the left edge move out of view, causing occlusions. Optical flow estimation relies on finding corresponding points between frames. When occlusions occur, there are no corresponding points in the occluded regions, making it impossible to estimate the motion accurately in these areas. Occlusions create discontinuities in the flow field where the motion abruptly changes, which can be challenging for smoothness assumptions used in many optical flow algorithms. In regions near the occlusion boundaries, the motion can be ambiguous, as part of the region might belong to the moving object and part to the background. FlowNetC, like other optical flow models, has difficulty accurately estimating flow in occluded regions because there is no direct correspondence available. Techniques such as regularization and smoothness constraints help mitigate this issue to some extent, but accurately predicting flow in occluded areas remains a significant challenge.

4. Vision Transformers (ViTs) have revolutionized the field of computer vision by leveraging the concept of self-attention, which was originally introduced in the context of natural language processing (NLP) transformers. Self-attention allows the model to weigh the importance of different parts of the input when making predictions, leading to more robust and context-aware representations. Here, we'll delve into the concept of self-attention and its application in Vision Transformers.

   What is Self-Attention?

   Self-attention is a mechanism that allows the model to focus on different parts of the input data to understand their relationships. In the context of transformers, self-attention computes a weighted sum of the input features, where the weights are determined by the relevance of each feature to every other feature.

   - Key Components of Self-Attention:

Query, Key, and Value Vectors (Q, K, V): For each input element (e.g., a pixel or patch in an image), the model computes three vectors: the query (Q), the key (K), and the value (V). These vectors are obtained through linear transformations of the input embeddings.

   - Attention Scores:

Attention scores are computed by taking the dot product of the query vector of one element with the key vectors of all elements. These scores indicate the relevance of one element to others.

   - Softmax Normalization:

The attention scores are passed through a softmax function to obtain attention weights. This ensures the weights sum to 1, allowing them to be interpreted as probabilities.

   - Weighted Sum:

The final output for each element is a weighted sum of the value vectors, where the weights are the attention scores.

   Self-Attention in Vision Transformers:

Vision Transformers apply the self-attention mechanism to image patches rather than words or tokens. Here's how ViTs typically work:

- Image Patching: An image is divided into a grid of fixed-size patches. Each patch is flattened into a vector and linearly embedded into a fixed-size embedding space.

- Position Embedding:

Since transformers lack an inherent sense of order, position embeddings are added to the patch embeddings to retain spatial information about the patches' positions within the image.

- Transformer Encoder:

The transformer encoder, composed of multiple layers of self-attention and feed-forward neural networks, processes the sequence of patch embeddings. In each layer, self-attention allows the model to focus on different patches to understand the global context.

- Classification Token:

A special classification token is added to the sequence of patches. This token aggregates information from all patches and is used for the final classification task.

Specifications in the example:

- Input image size: 224×224 pixels
- Patch size: 16×16 pixels
- Number of attention heads: 8
- Embedding dimension: 512
- Number of layers: 12

Number of patches per row : $\frac{224}{16} = 14$

Number of patches per column : $\frac{224}{16} = 14$

Total number of patches: $14 \times 14 = 196$

- Each 16×16 patch is flattened into a vector and then linearly embedded into a 512-dimensional space.

  Size of each patch : $16 \times 16 = 256$ pixels

  The linear transformation matrix used for embedding each patch will be of size $256 \times 512 + 512 = 131584$. The additional 512 parameters are for the bias term in the linear layer.

- For each self-attention layer, we need to compute the parameters for the query, key, and value matrices and the output projection. Since we have 8 heads and an embedding dimension of 512:

  Embedding dimension per head : $512/8 = 64$

  For each attention head, the query, key, and value matrices are of size 512×64.

  Parameters for Q, K, and V matrices per head :
  $3 \times (512 \times 64 + 64) = 98304$. The additional 64 parameters are for the bias term in each matrix.

  Since there are 8 heads, the total number of parameters for all heads:

  Total parameters for Q, K, V matrices: $98304 \times 8 = 786432$

  The output of the multi-head attention is also projected back to the embedding dimension (512):

  Parameters for output projection : $512 \times 512 + 512 = 262656$

  So, the total parameters for one multi-head self-attention block :
  $786432 + 262656 = 1049088$

- The feed-forward network typically consists of two linear layers with a ReLU activation in between. If the intermediate dimension is also 512:

  Parameters for the first linear: $512 \times 2048 + 2048 = 1050624$

Parameters for the second linear : $2048 \times 512 + 512 = 1049088$

So, the total parameters for the FFN block:
$1050624 + 10049088 = 2099712$

- Each transformer layer consists of one MHA block and one FFN block:

  Total parameters for one transformer layer:
  $1049088 + 2099712 = 3148800$

- Given there are 12 transformer layers:

  Total parameters for 12 layers : $12 \times 3148800 = 37785600$

  Adding the patch embedding parameters.

  Total trainable parameters : $37785600 + 131584 = 37917184$

- Also we need to calculate positional embedding parameters too:

- To add positional embedding parameters we'll follow these steps:

  Positional embeddings are used to maintain the order of the patches. If the embedding dimension is 512, the number of positional embedding parameters is the product of the number of patches and the embedding dimension.

  As the input image size is 224 *224 the parameters would be:

  Positional embedding parameters} = Number of patches * Embedding dimension = 196×512 = 100352

  Now we add positional embedding parameters to the total parameters:

Total parameters including positional embeddings = 37917184 + 100352 = 38017536

Advantages of ViTs over CNNs:

Self-attention mechanisms in ViTs can capture long-range dependencies and relationships between distant parts of an image. This global context understanding is beneficial for tasks requiring holistic image analysis, such as object detection and segmentation, unlike CNNs that primarily rely on local receptive fields and progressively larger convolutional layers to capture broader contexts, which can be less effective in capturing long-range dependencies.

ViTs are more flexible in handling variable input sizes and shapes due to their patch-based processing. They can be more adaptable to different resolutions and aspect ratios without requiring significant architectural changes. However, CNNs typically require fixed input sizes, and changes in input dimensions often necessitate architectural adjustments.

ViTs can be easily scaled by increasing the number of layers, heads, or embedding dimensions, similar to how transformers are scaled in NLP tasks. This scalability allows for creating larger and more powerful models, but scaling CNNs involves increasing the depth or width of the network, which can be more complex and less straightforward compared to the scaling of transformers.

The self-attention mechanism in ViTs enables the model to focus on the most relevant parts of the image for a given task, potentially leading to better performance in complex scenes. While attention mechanisms can be incorporated into CNNs, they are not inherently part of the architecture and usually require additional components and complexity.

Disadvantages of ViTs over CNNs:

ViTs generally require a large amount of labeled data for training to achieve competitive performance. They are less data-efficient compared to CNNs, which are more data-efficient and can achieve good performance with smaller datasets due to their inductive biases, such as locality and translation invariance.

The self-attention mechanism has a quadratic complexity with respect to the number of patches, leading to high computational costs and memory usage, especially for high-resolution images. On the other hand, convolutions are more computationally efficient, especially with optimizations like strided convolutions and pooling layers that reduce the spatial dimensions progressively.

Training ViTs are harder and require careful tuning of hyperparameters. They often benefit from pre-training on large datasets and then fine-tuning on specific tasks. However, CNNs are generally easier to train with well-established training procedures and hyperparameters. They converge more reliably and quickly in many vision tasks.

Lack the inherent inductive biases present in CNNs, such as the assumption of local connectivity and translation invariance. While this allows more flexibility, it also means ViTs do not leverage these beneficial biases naturally. These biases help CNNs to generalize better from limited data and learn meaningful features more efficiently.

Sources:

[Review: Fast R-CNN (Object Detection) | by Sik-Ho Tsang | Coinmonks | Medium]
[Review: Faster R-CNN (Object Detection) | by Sik-Ho Tsang | Towards Data Science]
[FlowNet | Lecture 32 (Part 3) | Applied Deep Learning (Supplementary) (youtube.com)]
[Transformer Neural Networks, ChatGPT's foundation, Clearly Explained!!! - YouTube]
[An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale (Paper Explained) (youtube.com)]
ChatGPT
Course Slides