Week 10 Discussion 1A

Joe Lin and Krystof Latka
Learning Assistants

December 6, 2024

Final Review

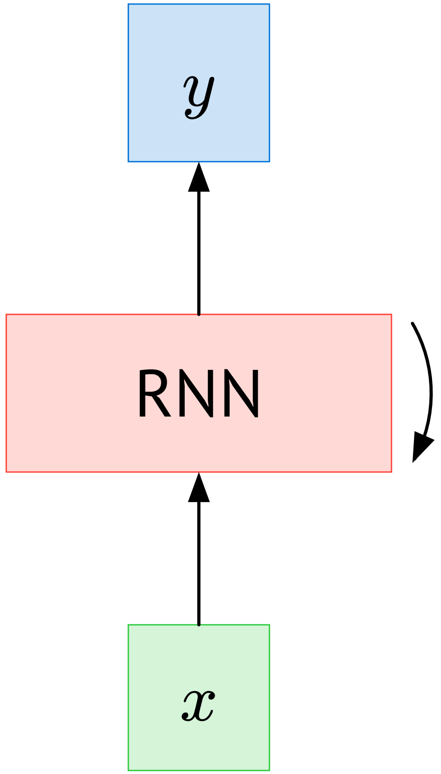
Recurrent Neural Networks

A **Recurrent Neural Network** processes a sequence of input vectors by applying a recurrence formula at every time step.

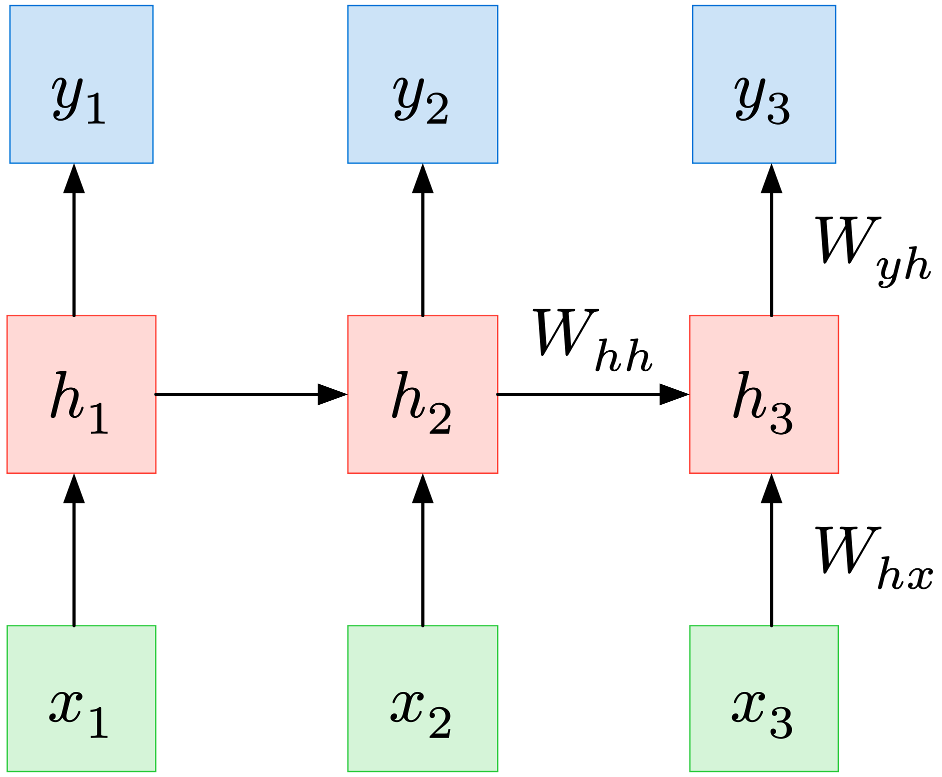
Contains a:

- Hidden State h

$$h_t = f_W(h_{t-1}, x_t)$$



Recurrent Neural Networks



Attention

In basic **attention**, we have the following inputs:

- Input vectors $X \in \mathbb{R}^{N_X \times D_X}$
- Key matrix $W_K \in \mathbb{R}^{D_X \times D_Q}$
- Query vectors $Q \in \mathbb{R}^{N_Q \times D_Q}$
- Value matrix $W_V \in \mathbb{R}^{D_X \times D_V}$

The goal is to use the query vectors and learn to attend to the input vectors and return a relevant combination of those vectors as output.

How do we do this?

Attention

In basic **attention**, we have the following inputs:

- Input vectors $X \in \mathbb{R}^{N_X \times D_X}$
- Key matrix $W_K \in \mathbb{R}^{D_X \times D_Q}$
- Query vectors $Q \in \mathbb{R}^{N_Q \times D_Q}$
- Value matrix $W_V \in \mathbb{R}^{D_X \times D_V}$

The goal is to use the query vectors and learn to attend to the input vectors and return a relevant combination of those vectors as output.

How do we do this? Utilize Key and Value vectors.

Attention

$$K = XW_K \in \mathbb{R}^{N_X \times D_Q}$$

What did we do here?

Attention

$$K = XW_K \in \mathbb{R}^{N_X \times D_Q}$$

What did we do here? Learned a linear transformation W_K to obtain a set of keys from the input vectors.

Attention

$$K = XW_K \in \mathbb{R}^{N_X \times D_Q}$$

What did we do here? Learned a linear transformation W_K to obtain a set of keys from the input vectors.

$$V = XW_V \in \mathbb{R}^{N_X \times D_V}$$

What did we do here?

Attention

$$K = XW_K \in \mathbb{R}^{N_X \times D_Q}$$

What did we do here? Learned a linear transformation W_K to obtain a set of keys from the input vectors.

$$V = XW_V \in \mathbb{R}^{N_X \times D_V}$$

What did we do here? Learned a linear transformation W_V to obtain a set of values from the input vectors.

Attention

$$K = XW_K \in \mathbb{R}^{N_X \times D_Q}$$

What did we do here? Learned a linear transformation W_K to obtain a set of keys from the input vectors.

$$V = XW_V \in \mathbb{R}^{N_X \times D_V}$$

What did we do here? Learned a linear transformation W_V to obtain a set of values from the input vectors.

Essentially, we learn the keys and values associated with X.

Attention

$$E = \frac{QK^T}{\sqrt{D_Q}} \in \mathbb{R}^{N_Q \times N_X} \qquad E_{i,j} = \frac{Q_i \cdot K_j}{\sqrt{D_Q}}$$

What did we do here?

Attention

$$E = \frac{QK^T}{\sqrt{D_Q}} \in \mathbb{R}^{N_Q \times N_X} \qquad E_{i,j} = \frac{Q_i \cdot K_j}{\sqrt{D_Q}}$$

What did we do here? Performed a scaled dot product, where $E_{i,j}$ tells us how similar query Q_i and key K_j are.

Attention

$$E = \frac{QK^T}{\sqrt{D_Q}} \in \mathbb{R}^{N_Q \times N_X} \qquad E_{i,j} = \frac{Q_i \cdot K_j}{\sqrt{D_Q}}$$

What did we do here? Performed a scaled dot product, where $E_{i,j}$ tells us how similar query Q_i and key K_j are.

```
A = softmax(E, dim=1)
```

What did we do here?

Attention

$$E = \frac{QK^T}{\sqrt{D_Q}} \in \mathbb{R}^{N_Q \times N_X} \qquad E_{i,j} = \frac{Q_i \cdot K_j}{\sqrt{D_Q}}$$

What did we do here? Performed a scaled dot product, where $E_{i,j}$ tells us how similar query Q_i and key K_j are.

```
A = softmax(E, dim=1)
```

What did we do here? Normalize scores such that $A_{i,j}$ now tells us the similarity of query Q_i and key K_j in *proportion* to all key vectors.

Attention

$$Y = AV \in \mathbb{R}^{N_Q \times N_V}$$

What did we do here?

$$Y = \mathrm{softmax}\left(\frac{QK^T}{\sqrt{D_Q}}\right)V$$

Attention

$$Y = AV \in \mathbb{R}^{N_Q \times N_V}$$

What did we do here? Based on attention scores, output a linear combination of value vectors.

$$Y = \mathrm{softmax}\left(\frac{QK^T}{\sqrt{D_Q}}\right)V$$

Self-Attention

In **self-attention**, we have the following inputs:

- Input vectors $X \in \mathbb{R}^{N_X \times D_X}$
- Key matrix $W_K \in \mathbb{R}^{D_X \times D_Q}$
- Query matrix $W_Q \in \mathbb{R}^{D_X \times D_Q}$
- Value matrix $W_V \in \mathbb{R}^{D_X \times D_V}$

Primary change is to learn to obtain a set of query vectors (one for every input).

Transformer

Consider a transformer block (with one self-attention layer and one MLP layer) that processes N tokens of dimension D. How many parameters (excluding biases and layer norm) are in this block?

Transformer

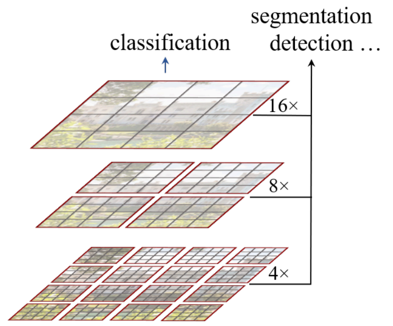
Consider a transformer block (with one self-attention layer and one MLP layer) that processes N tokens of dimension D. How many parameters (excluding biases and layer norm) are in this block?

Self-attention layer has $3D^2$ parameters and MLP layer has D^2 parameters, so there are $4D^2$ total parameters.

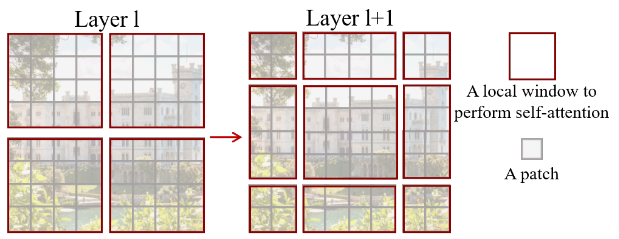
Vision Transformer

1. Treat image patches as tokens.
2. Apply linear projection to obtain D-dimensional patch embeddings.
3. Add positional encoding to embeddings.
4. Input into regular transformer architecture.

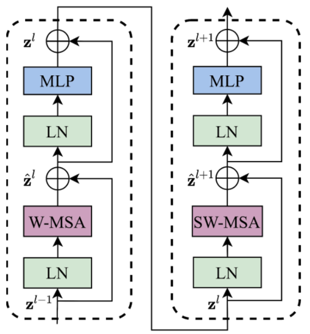
Hierarchical ViT



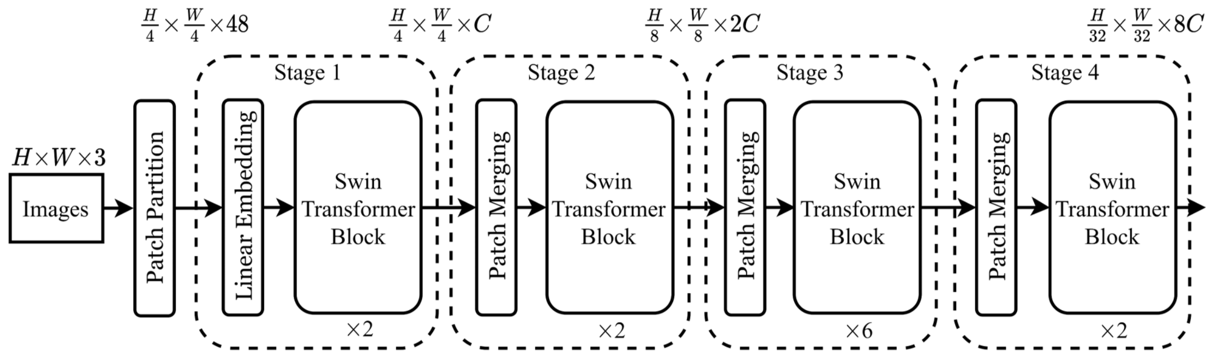
(a) Swin Transformer



(b) Shifted Window



(c) Two Successive Swin Transformer Blocks



(d) Architecture

Hierarchical ViT

What’s the purpose of pipeline displayed in part (d) and how does it accomplish this?

Hierarchical ViT

What’s the purpose of pipeline displayed in part (d) and how does it accomplish this? Enable transformer to learn visual features at multiple scales. Uses patch merging to reduce resolution.

Hierarchical ViT

What’s the purpose of pipeline displayed in part (d) and how does it accomplish this? Enable transformer to learn visual features at multiple scales. Uses patch merging to reduce resolution.

How does Swin Transformer avoid costly global attention computations?

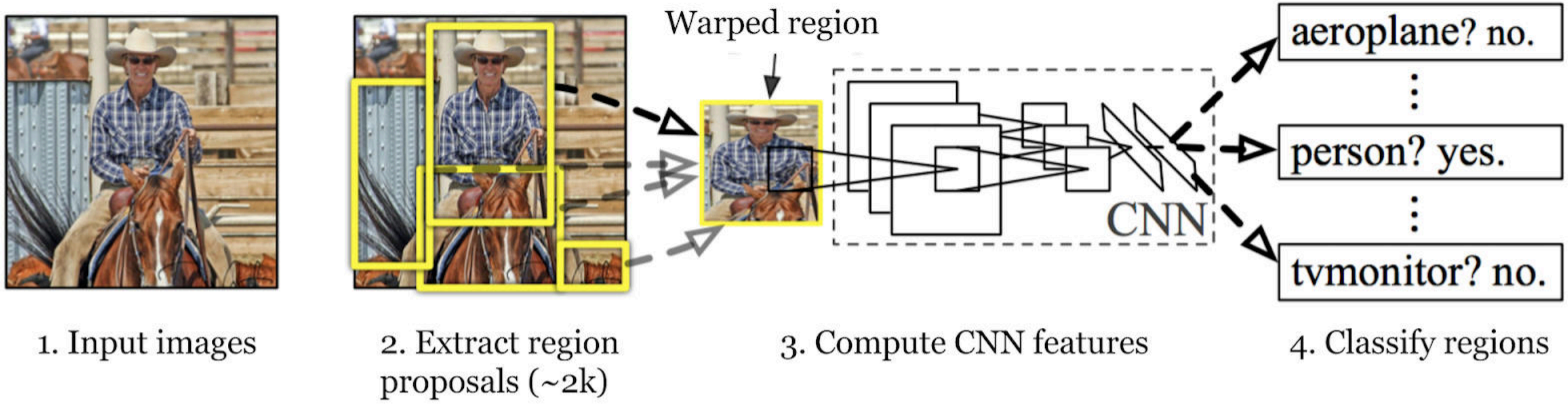
Hierarchical ViT

What's the purpose of pipeline displayed in part (d) and how does it accomplish this? Enable transformer to learn visual features at multiple scales. Uses patch merging to reduce resolution.

How does Swin Transformer avoid costly global attention computations? Uses windowed attention instead. Size of attention matrices can be reduced:

$$H^2W^2 \rightarrow M^2HW$$

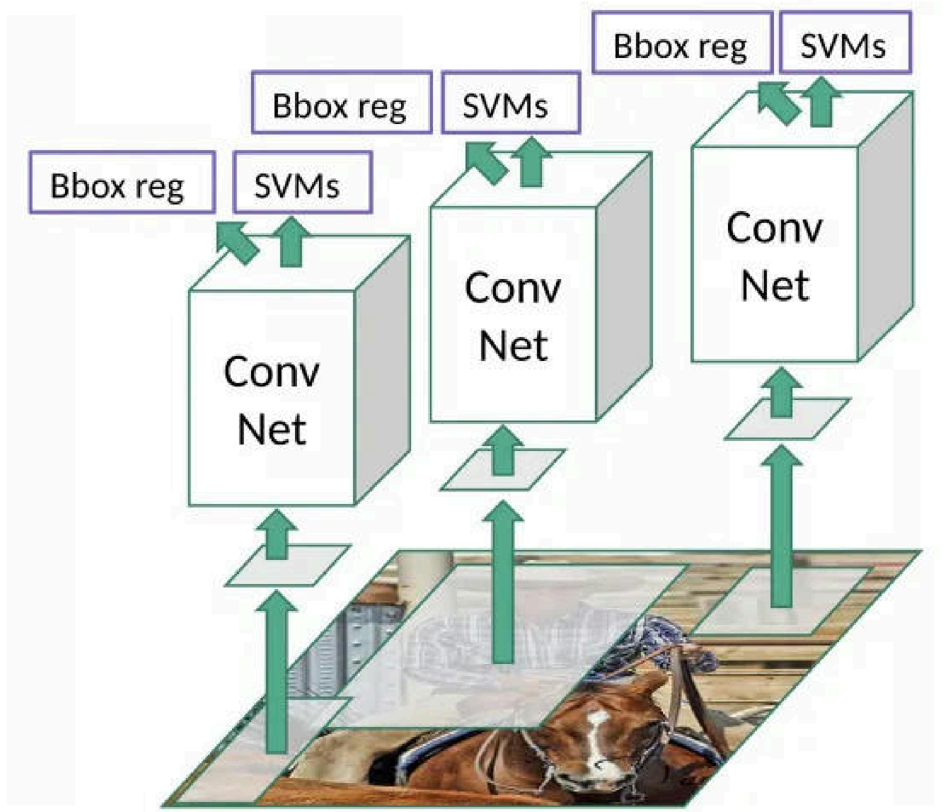
Region-Based CNN



Use **selective search** algorithm to propose region of interests (RoIs).

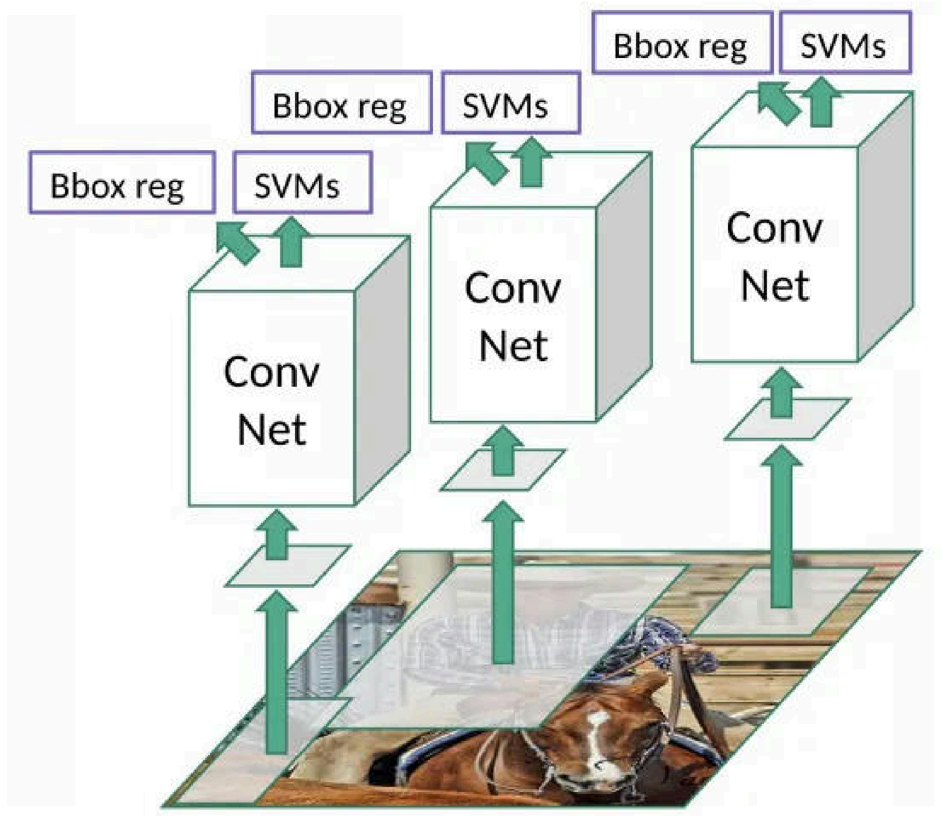
Resize image for CNN compatability and classify.

Region-Based CNN



Problem: How to handle incorrectly sized **region proposals**?

Region-Based CNN



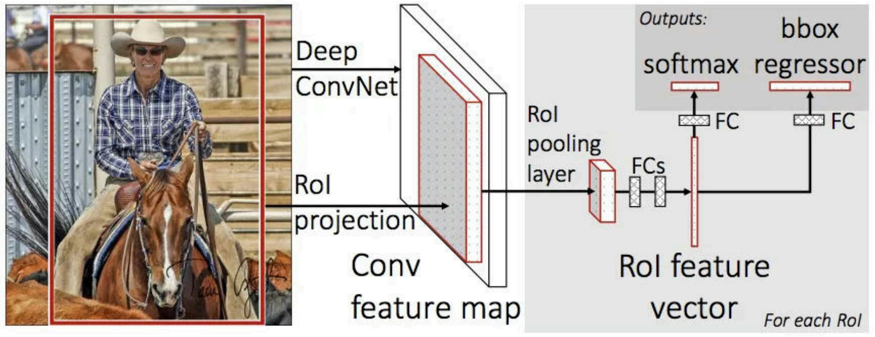
Problem: How to handle incorrectly sized **region proposals**?

Solution: Learn to predict **bounding box transforms**.

$$b_x = p_x + p_w t_x \quad b_y = p_y + p_h t_y$$

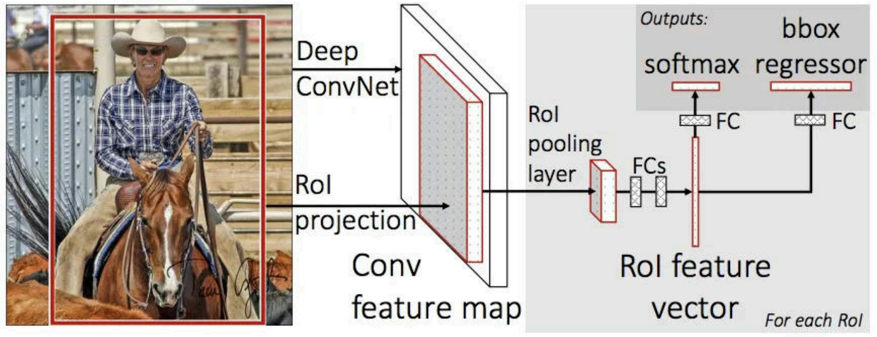
$$b_w = p_w \exp(t_w) \quad b_h = p_h \exp(t_h)$$

Fast Region-Based CNN



Problem: Very slow to feed every proposed region into entire CNN.

Fast Region-Based CNN

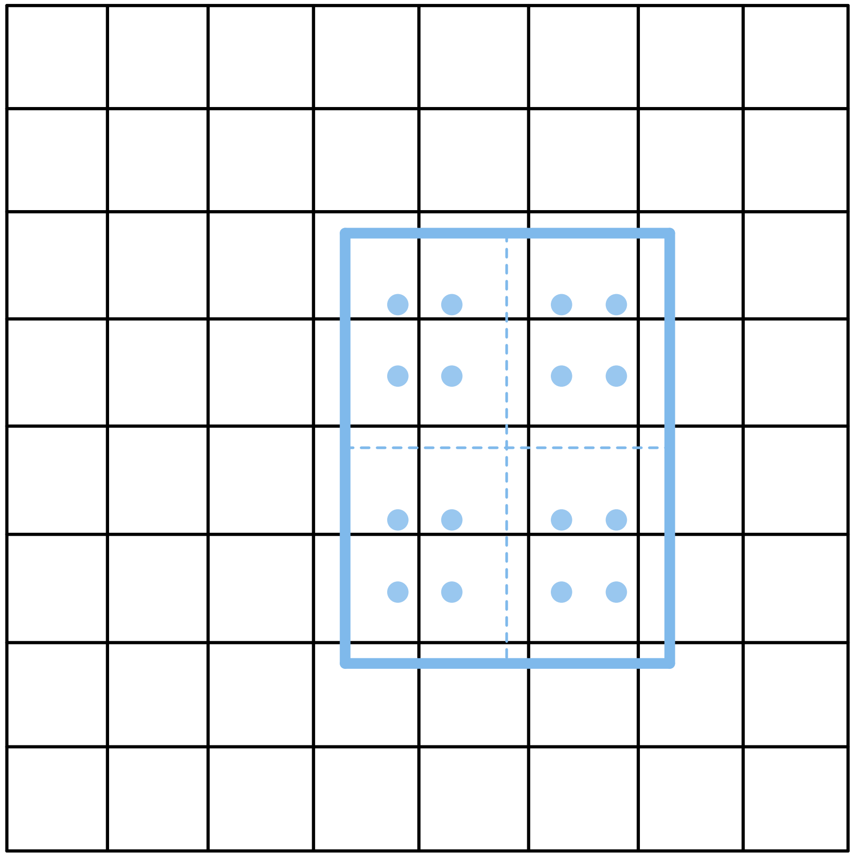


Problem: Very slow to feed every proposed region into entire CNN.

Solution: Pass entire image into CNN, project RoI, and apply **differentiable cropping**.

After projecting regions of interest (RoIs) to feature map, how do we select features to predict class and bounding box transforms on?

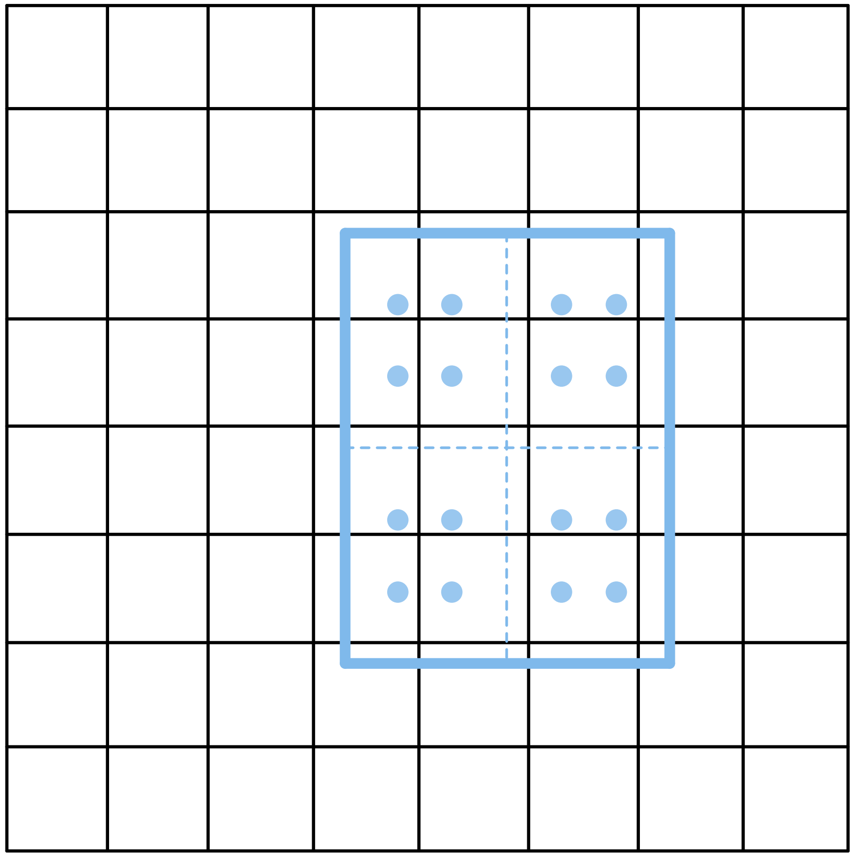
Region of Interest Align



How do we obtain features at sampling points in a differentiable manner?

Note the blue rectangle is a projected region of interest on feature map.

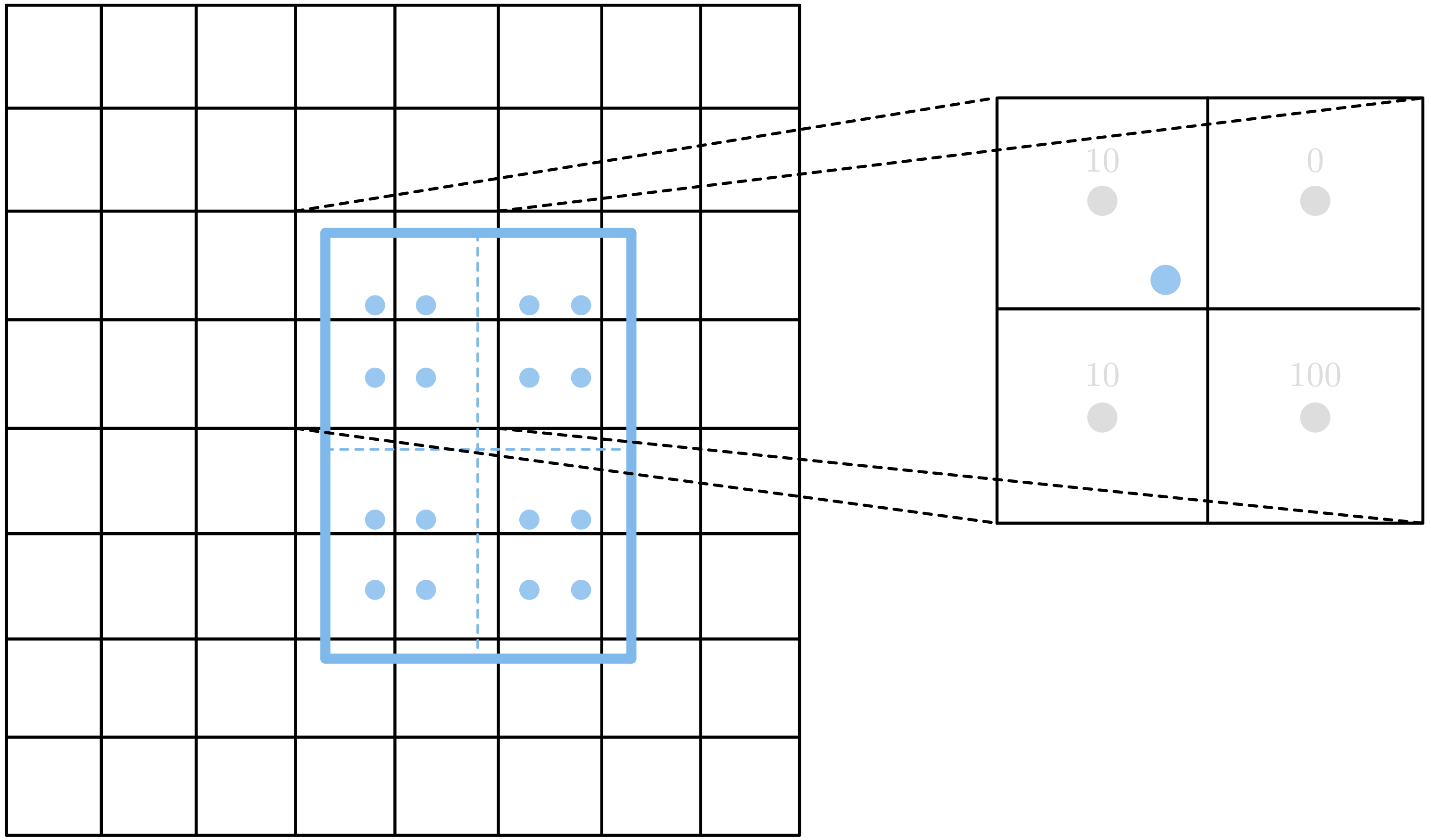
Region of Interest Align



How do we obtain features at sampling points in a differentiable manner? Bilinear interpolation.

Note the blue rectangle is a projected region of interest on feature map.

Region of Interest Align



Region of Interest Align

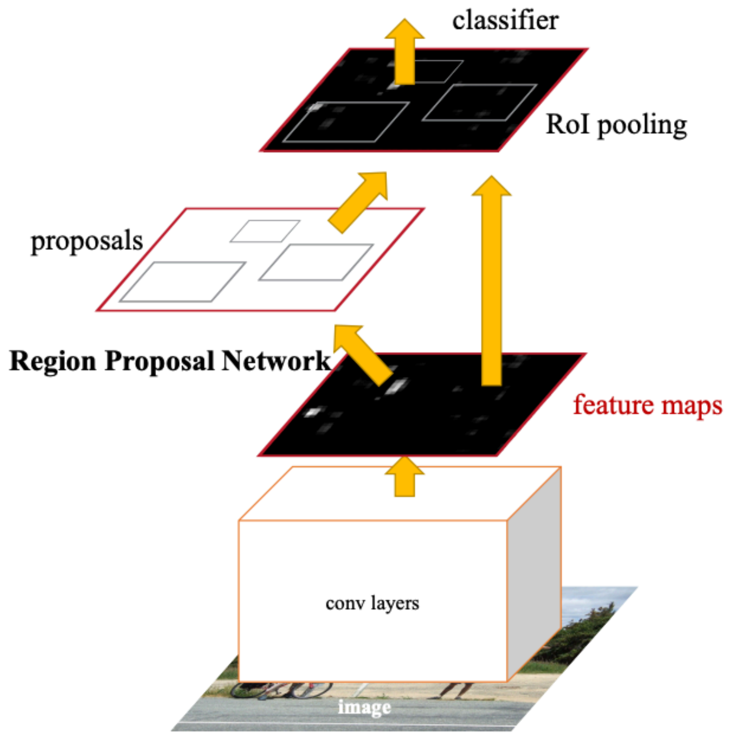
Suppose gray dots are at coordinates $(3, 2), (4, 2), (3, 3), (4, 3)$ (top left, top right, bottom left, bottom right) and blue dot is at $(3.3, 2.4)$. Compute the interpolated feature.

Region of Interest Align

Suppose gray dots are at coordinates $(3, 2), (4, 2), (3, 3), (4, 3)$ (top left, top right, bottom left, bottom right) and blue dot is at $(3.3, 2.4)$. Compute the interpolated feature.

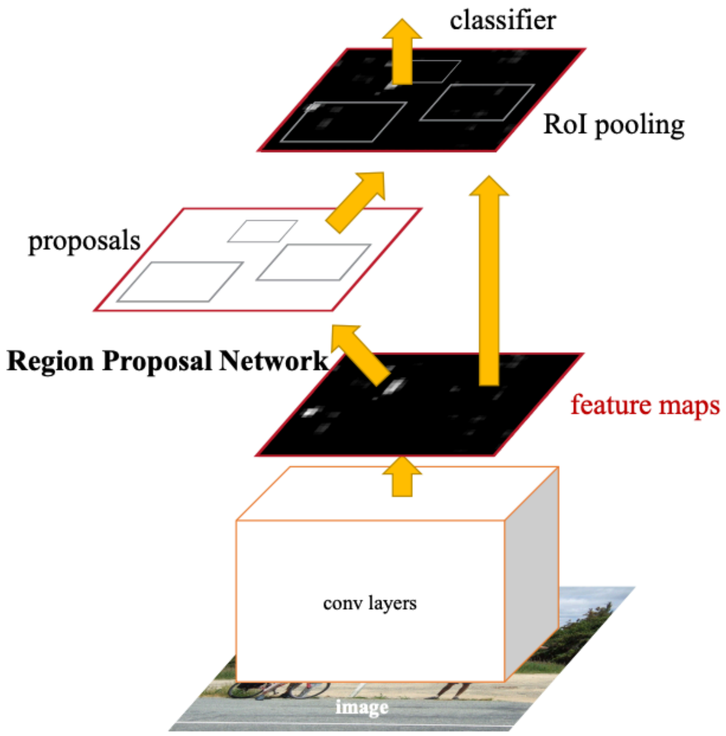
$$\begin{gathered} 10 \cdot 0.7 \cdot 0.6 + 0 \cdot 0.3 \cdot 0.6 + 10 \cdot 0.4 \cdot 0.7 + 100 \cdot 0.3 + 0.4 \\ = 4.2 + 2.8 + 12 \\ = 19 \end{gathered}$$

Faster Region-Based CNN



Problem: Runtime dominated by region proposals (e.g. selective search).

Faster Region-Based CNN



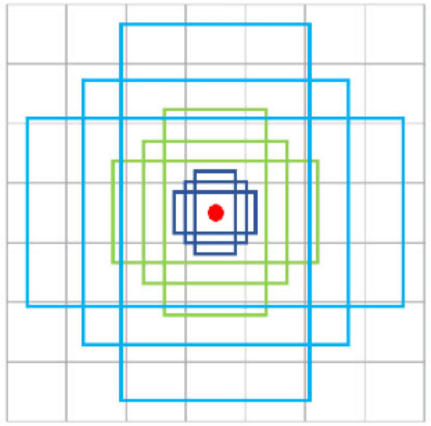
Problem: Runtime dominated by region proposals (e.g. selective search).

Solution: Add a **region proposal network** to predict region proposals.

Region Proposal Networks

Start with K anchors per position (remember that the input is a feature map f).

What's the purpose of a region proposal network?



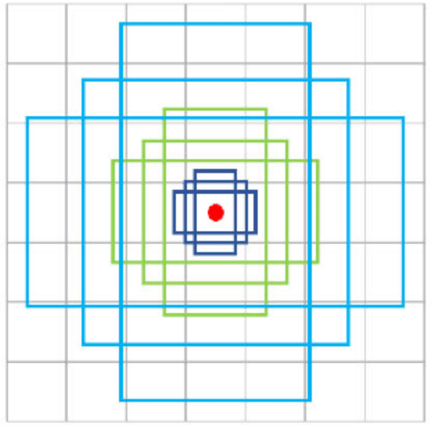
Region Proposal Networks

Start with K anchors per position (remember that the input is a feature map f).

What's the purpose of a region proposal network?

The goal of a region proposal network is to:

1. Classify each anchor as region of interest (RoI) or background.
2. Regress anchor box offset transforms.



Non-Maximum Suppression

Problem: What do we do with overlapping detections?

Non-Maximum Suppression

Problem: What do we do with overlapping detections?

Solution: Use **non-maximum suppression (NMS)** to filter predictions.

1. Sort predictions by confidence.
2. For each remaining prediction, loop through and discard predictions with IoU above some threshold.
   - What does this step do? Removes predictions that have high overlap.

Detection Metrics

Consider the following ground truth and prediction maps. Suppose there are 3 classes. Compute the mean IoU and DICE scores.

2	2	0	0
0	0	0	0
1	1	1	1
2	1	1	2

Ground Truth

1	2	0	1
0	2	1	0
1	1	1	2
2	2	2	2

Predictions

Detection Metrics

To compute scores, we first calculate for each class, then take the mean.

IoU:

- Class 0 – $\frac{3}{6} = \frac{1}{2}$
- Class 1 – $\frac{3}{9} = \frac{1}{3}$
- Class 2 – $\frac{3}{8}$
- Mean – $\frac{29}{72}$

DICE:

- Class 0 – $\frac{6}{9} = \frac{2}{3}$
- Class 1 – $\frac{6}{12} = \frac{1}{2}$
- Class 2 – $\frac{6}{11}$
- Mean – $\frac{113}{198}$

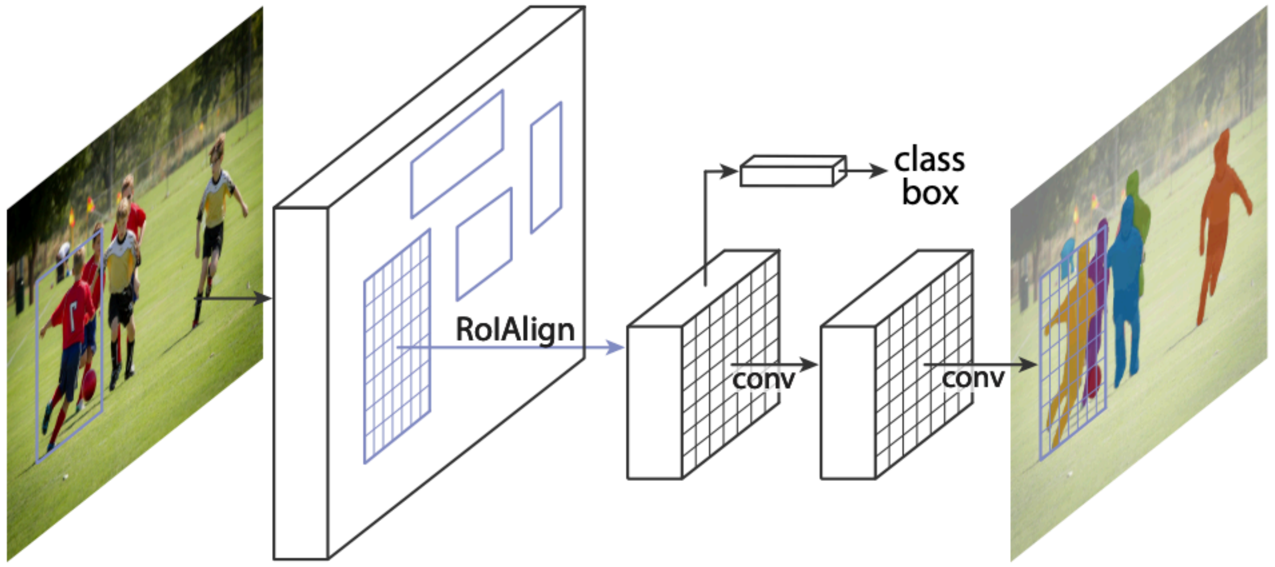
Semantic Segmentation

Typically consists of a encoder-decoder architecture.

- Encoder uses convolutions to **downsample**.
- Decoder **upsamples** with **transposed convolutions** to obtain a semantic map at the original resolution.

Mask Region-Based CNN

Add a mask prediction head to typical R-CNN framework.



Generative Models

What are we trying to learn with a:

- **Discriminative Model** –
  - For MiniPlaces, you trained a predictor to learn a probability distribution over the classes given an image.
- **Generative Model** –
  - Generate an image.
- **Conditional Generative Model** –
  - Generate an image given a certain text.

Generative Models

What are we trying to learn with a:

- **Discriminative Model** – $p(y \mid x)$
  - For MiniPlaces, you trained a predictor to learn a probability distribution over the classes given an image.
- **Generative Model** –
  - Generate an image.
- **Conditional Generative Model** –
  - Generate an image given a certain text.

Generative Models

What are we trying to learn with a:

- **Discriminative Model** – $p(y \mid x)$
  - For MiniPlaces, you trained a predictor to learn a probability distribution over the classes given an image.
- **Generative Model** – $p(x)$
  - Generate an image.
- **Conditional Generative Model** –
  - Generate an image given a certain text.

Generative Models

What are we trying to learn with a:

- **Discriminative Model** – $p(y \mid x)$
  - For MiniPlaces, you trained a predictor to learn a probability distribution over the classes given an image.
- **Generative Model** – $p(x)$
  - Generate an image.
- **Conditional Generative Model** – $p(x \mid y)$
  - Generate an image given a certain text.

Generative Models

Match the following generative models with their properties.

a. Autoregressive Models	explicit density, approximate density, variational
b. Variational Autoencoders	implicit density, direct
c. Diffusion Models	explicit density, approximate density, markov chain
d. Generative Adversarial Networks	explicit density, tractable density

Generative Models

Match the following generative models with their properties.

a. Autoregressive Models	explicit density, approximate density, variational
b. Variational Autoencoders	implicit density, direct
c. Diffusion Models	explicit density, approximate density, markov chain
d. Generative Adversarial Networks	explicit density, tractable density

Variational Autoencoders

Trained by maximizing the **variational lower bound** (often referred to as **ELBO**).

$$\mathbb{E}_{z \sim q_\varphi(z \mid x)}[\log p_\theta(x \mid z)] - D_{KL}\big(q_\varphi(z \mid x), p(z)\big)$$

What’s the first term?

What’s the second term?

Variational Autoencoders

Trained by maximizing the **variational lower bound** (often referred to as **ELBO**).

$$\mathbb{E}_{z\sim q_\varphi(z \mid x)}[\log p_\theta(x \mid z)] - D_{KL}\big(q_\varphi(z \mid x), p(z)\big)$$

What's the first term? $\mathbb{E}_{z\sim q_\varphi(z \mid x)}[\log p_\theta(x \mid z)]$ ensures that original input data is probable in decoder output distribution.

What's the second term?

Variational Autoencoders

Trained by maximizing the **variational lower bound** (often referred to as **ELBO**).

$$\mathbb{E}_{z \sim q_\varphi(z \mid x)}[\log p_\theta(x \mid z)] - D_{KL}\left(q_\varphi(z \mid x), p(z)\right)$$

What's the first term? $\mathbb{E}_{z \sim q_\varphi(z \mid x)}[\log p_\theta(x \mid z)]$ ensures that original input data is probable in decoder output distribution.

What's the second term?

Variational Autoencoders

Trained by maximizing the **variational lower bound** (often referred to as **ELBO**).

$$\mathbb{E}_{z \sim q_\varphi(z \mid x)}[\log p_\theta(x \mid z)] - D_{KL}\big(q_\varphi(z \mid x), p(z)\big)$$

What's the first term? $\mathbb{E}_{z \sim q_\varphi(z \mid x)}[\log p_\theta(x \mid z)]$ ensures that original input data is probable in decoder output distribution.

What's the second term? $D_{KL}\big(q_\varphi(z \mid x), p(z)\big)$ ensures that the encoder output distribution is close to the prior distribution $p(z)$.

Variational Autoencoders

Give one similarity and one difference between the standard auto-encoder and the variational auto-encoder.

Variational Autoencoders

Give one similarity and one difference between the standard auto-encoder and the variational auto-encoder.

Similarities:

- Encoder-decoder paradigm
- Latent bottleneck
- Model x as a function of latents z

Differences:

- Sampling
- Minimize variational lower bound instead of just reconstruction loss
- Variational autoencoders are generative

Generative Adversarial Networks

Consider the loss objective used when training GANs.

$$\min_G \max_D \Big(\mathbb{E}_{x\sim p_{\text{data}}}[\log(D(x))] + \mathbb{E}_{z\sim p(z)}[\log(1 - D(G(z)))] \Big)$$

Traditionally, both D and G are randomly initialized. Assume instead that G is randomly initialized, but D is a perfect discriminator.

How will this affect training?

Generative Adversarial Networks

Consider the loss objective used when training GANs.

$$\min_G \max_D \Big(\mathbb{E}_{x\sim p_{\text{data}}}[\log(D(x))] + \mathbb{E}_{z\sim p(z)}[\log(1 - D(G(z)))]\Big)$$

Traditionally, both D and G are randomly initialized. Assume instead that G is randomly initialized, but D is a perfect discriminator.

How will this affect training?

Vanishing gradient for G because the second term is close to 0.

Generative Adversarial Networks

Consider the loss objective used when training GANs.

$$\min_G \max_D \Big(\mathbb{E}_{x \sim p_{\text{data}}}[\log(D(x))] + \mathbb{E}_{z \sim p(z)}[\log(1 - D(G(z)))] \Big)$$

Traditionally, both D and G are randomly initialized. Assume instead that G is randomly initialized, but D is a perfect discriminator.

Propose one modification to the loss function to improve the training of the GAN under this assumption.

Generative Adversarial Networks

Consider the loss objective used when training GANs.

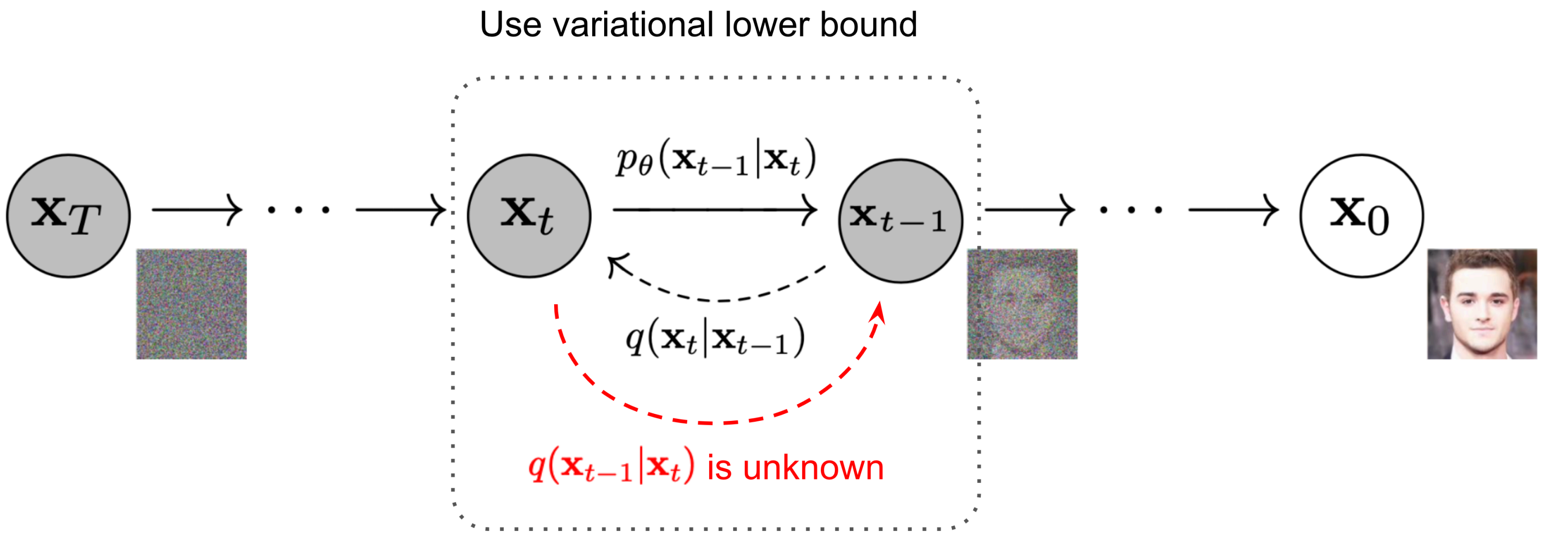
$$\min_G \max_D \Big(\mathbb{E}_{x\sim p_{\text{data}}}[\log(D(x))] + \mathbb{E}_{z\sim p(z)}[\log(1 - D(G(z)))]\Big)$$

Traditionally, both D and G are randomly initialized. Assume instead that G is randomly initialized, but D is a perfect discriminator.

Any modification that increases gradient of the second term near 0, for instance using $-\log D(G)$ instead, while preserving gradient direction.

Diffusion Models

1. Start with a ground-truth input image.
2. Iteratively add random noise for t steps.
3. Given noised image and t, learn to iteratively denoise the image.


Use variational lower bound
$\mathbf{x}_T$
$\mathbf{x}_t$
$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$
$\mathbf{x}_{t-1}$
$\mathbf{x}_0$
$q(\mathbf{x}_t|\mathbf{x}_{t-1})$
$q(\mathbf{x}_{t-1}|\mathbf{x}_t)$ is unknown

Diffusion Models

Forward diffusion process – adds noise to the data in a controlled manner over a series of time steps, effectively transforming the data distribution into a normal distribution

- Note that this is *defined* by us beforehand as follows:

$$x_t = \sqrt{1-\beta_t} \cdot x_{t-1} + \sqrt{\beta_t} \cdot \varepsilon$$

Reverse diffusion process – iteratively denoise a sample starting from pure Gaussian noise, resulting in a reconstruction of original data

- Note that this is learned (typically with a U-Net or Transformer architecture)

Latent Diffusion Models

Describe how **latent diffusion models** differ from standard diffusion models in terms of input space and computational efficiency.

Input Space

- Standard – operate directly on high-dimensional data
- Latent – operate in a lower-dimensional latent space obtained via an encoder from a pre-trained autoencoder

Efficiency

- Standard – computationally expensive, especially for large image resolutions
- Latent – lower computational costs while preserving semantic features of the data