

Alexnet

这个CNN模型在ImageNet数据集上取得了突破性的成绩

- 首次使用ReLU激活函数
- 首次使用Dropout技术防止过拟合
- 证明了GPU加速训练的有效性

GoogleNet

这个CNN模型在ImageNet数据集上取得了优异的成绩

- 首次使用Inception v1模块
- 首次使用辅助分类头
- 首次使用全局平均池化

GoogLeNet模型在ImageNet数据集上取得了优异的成绩，其核心模块是Inception v1

Inception v1

 image-20250804105749792

- Inception v1模块由四个分支组成：1个1x1卷积，3个3x3卷积，5个5x5卷积，以及3个1x1卷积
- 每个分支的输出经过ReLU激活函数

 image-20250804105823459

这个模块的设计旨在减少计算量，同时保持模型的精度

 image-20250804110823817

- GoogLeNet模型由3个Inception v1模块组成
- 每个Inception v1模块由2个Inception v1模块组成
 - 第一个Inception v1模块由1个1x1卷积，3个3x3卷积，5个5x5卷积，以及3个1x1卷积组成
 - 第二个Inception v1模块由1个1x1卷积，3个3x3卷积，5个5x5卷积，以及3个1x1卷积组成

Resnet


2015年，ResNet模型在ImageNet数据集上取得了优异的成绩，其核心模块是ResNet Block。ResNet Block的定义为：
$$H(x) = x + F(x)$$
，其中 x 是输入， $F(x)$ 是残差函数。

 image-20250804145938060

ResNet Block

BasicBlock 这个block由两个layer组成

BottleNeck 这个layer由两个block组成，每个block由4个layer组成

这个模型在ImageNet数据集上取得了优异的成绩  image-20250804154119715

• BasicBlock

- **Layer2** 即 **layer** 即 **Block** 即 **Block** 即 **Block**
- 3×3 卷积核 64
- **block**
- 即
 1. **1** **layer**
 2. **2** **layer** **1** **block** **block** **layer**

• BottleNeck

- **Layer** 即 **Block** 即 **Block** 即 **Block**
- 3×3 卷积核 1×1 3×3 1×1
- 1×1 卷积核
- 3×3 卷积核
- 1×1 卷积核 4
- 即
 1. **layer** **1** **block** **block** **block** **layer**
 2. 1×1 卷积核 4
 3. 1×1 卷积核

-
- **layer** **block** **stride=2** **block** **stride=1**

ResNet

image-20250805173724557

VGG

3×3 卷积核

3×3 卷积核 5×5 , 7×7

- 3×3 卷积核
- 1×1 **padding=1**
- 3×3 卷积核

image-20250805182841345

MobileNet

□□□□□□□□□□□□□□□□

□**□□□95%**□□□\$1\times 1\$□□□

□□□□□□

- □□□□□□□□
 - □□□□□□□□□□□□□□□□
 - □□□□□□□□□□□□□□□□□□□□□□□□□□
- □□□□□□□□
 - □□□□□□□□□□□□

444

□□□□\$\alpha\$□□□□□□□□□□□□□□□□

5/5

□□□□\$\\rho\$□□□□□□□□□□□□

□□□□

ReLU6 $\text{ReLU6}(x) = \min(\text{ReLU}(x), 6)$

□□□□□ Inverted Residuals □

ResNet


- ResNet Bottleneck 1×1 $\times 3 \times 3$ 1×1 $\times 1$
- MobileNetv2 1×1 $\times 3 \times 3$ 1×1 $\times 1$
 - 1×1 $\times 3 \times 3$
 - 1×1 $\times 3 \times 3$

1111

□□□□□ReLU□□□□□□□□□□□□□□□□□□ReLU□□□□□

111

- [illegible]

 image-20250806173546405

- MobileNetV1: 0 ReLU 0 0 ReLU

Vit Vision Transformer

Transformer

□□□□□

1. patches
2. patches D 768 [CLS] Token Bert
3. transformer encoder ** [CLS] Token patch patch **
4. encoder [CLS] Token

111

1. 模型在生成过程中是否会出现幻觉
2. 模型在生成过程中是否会出现Instructive Bias
3. 模型在生成过程中是否会出现patch



 image-20250804103117355




Deit

Vit

□□□□distillation□

- `torch.nn.Loss` `torch.nn.Loss` `torch.nn.Loss`
- Deit `torch.nn.Loss` `torch.nn.Loss` `torch.nn.Loss` [distill] Token `torch.nn.Loss`
 - [distill] `torch.nn.Loss` `torch.nn.Loss` `torch.nn.Loss` KL `torch.nn.Loss`
 - [CLS] `torch.nn.Loss` `torch.nn.Loss`
- `torch.nn.Loss` `torch.nn.Loss` `torch.nn.Loss` `torch.nn.Loss` `torch.nn.Loss`
- `torch.nn.Loss` `torch.nn.Loss` `torch.nn.Loss` `torch.nn.Loss` `torch.nn.Loss` **KL** `torch.nn.Loss`

□□□□□□

- 
- 
- 

□□□□

□□□□ Semantic Segmentation □

-
-

Instance Segmentation

-

□□□□ Panoptic Segmentation□

- □□□



 image-20250806105614387

FPN Feature Pyramid Network

tmkaiming

問題を解決する

- 問題の定義、データの収集、モデルの構築、評価、改善のサイクルを繰り返す


問題の定義とデータの収集

問題の定義

- 問題の定義、データの収集、モデルの構築、評価、改善のサイクルを繰り返す
- 問題の定義、データの収集、モデルの構築、評価、改善のサイクルを繰り返す

データの収集

- 問題の定義、データの収集、モデルの構築、評価、改善のサイクルを繰り返す
- 問題の定義、データの収集、モデルの構築、評価、改善のサイクルを繰り返す
- 問題の定義、データの収集、モデルの構築、評価、改善のサイクルを繰り返す
- $\$1 \times 1$ の畳み込み

 image-20250809155500175

FCN (Fully Convolutional Network)

問題の定義

データの収集


- 問題の定義、データの収集、モデルの構築、評価、改善のサイクルを繰り返す
- 問題の定義、データの収集、モデルの構築、評価、改善のサイクルを繰り返す

問題の定義とデータの収集

- 問題の定義、データの収集、モデルの構築、評価、改善のサイクルを繰り返す
 - $\$1 \times 1$ の畳み込みと softmax 関数
- 問題の定義
- 問題の定義

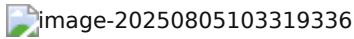
FCN の構築

- 問題の定義、データの収集、モデルの構築、評価、改善のサイクルを繰り返す
- 問題の定義、データの収集、モデルの構築、評価、改善のサイクルを繰り返す
 - FCN-32s: 32 個の畳み込み層
 - FCN-16s: 2 個の畳み込み層 (pool4) と 16 個の畳み込み層
 - FCN-8s: 2 個の畳み込み層 (pool4) と 2 個の畳み込み層 (pool3) と 8 個の畳み込み層

 image-20250809162635744

U-Net

- U-Net の問題の定義、データの収集、モデルの構築、評価、改善のサイクルを繰り返す
- 問題の定義、データの収集、モデルの構築、評価、改善のサイクルを繰り返す
- 問題の定義



1111

- `conv1 = Conv2d($1 \times 572 \times 572, 16, kernel_size=3, stride=1, padding=0)`
- `conv1 = nn.Conv2d(in_channels=1, out_channels=64, kernel_size=3, stride=1, padding=0)`
`conv1 = nn.ReLU(inplace=True)`
- `conv1 = nn.Conv2d(16, 16, kernel_size=3, stride=1, padding=0)`
- `conv1 = nn.Conv2d(16, 16, kernel_size=3, stride=1, padding=0)`
- `conv1 = nn.Conv2d(16, 16, kernel_size=3, stride=1, padding=0)`

000

UnetUnet

GoogleNet

[illegible]

- □□□□□□□□□□□□□□□□
- □□□□□
- □□□□□□□□□□□□□□□□
- □□□□□

IoU□□□□□

$\{A\cup B\}$
 $\$A\cap B\%$
 $\$ \text{IoU} = \frac{A\cap B}{\{A\cup B\}}$

IoU

- [illegible]



Dice Loss

Dice
$$\text{Dice Loss} = 1 - \frac{2 \cdot |\mathcal{A} \cap \mathcal{B}|}{|\mathcal{A}| + |\mathcal{B}|}$$

- $\$A\$$ 0~1
- $\$B$ 0~1
- $\$|A \cap B|$
- $\$|A| + |B|$

□ □

□□□□□□□□□□□□□□□□

BCE Loss

$$\text{DiceLoss} = - \sum_{i=1}^N [y_i \log(p_i) + (1 - y_i) \log(1 - p_i)]$$

- y_i
- p_i
- N

1. 1000 classes
 2. 1000 classes

Deeplab

1000 classes

v2 **v3** **backbone** **ResNet**

1000

1. 1000 classes
2. 1000 classes

LargeFov

1000

Deeplab 1000 classes

- 1000 classes
- 1000 classes

VGG

1. VGG backbone FC6 7×7
2. 3×3 $r=12$ 7×7
3. VGG backbone 1×1

ASPP model V3

1000 classes

1000 classes

 image-20250807104036319

backbone

1000

- 1×1
- $12 \times 3 \times 3$
- $24 \times 3 \times 3$
- $36 \times 3 \times 3$
- 1000

channel 1×1

 image-20250807104102110

□□□□□□

- 0000000000**16**00000000\$3\times 3\$000000000000**6****12****18**
- 0000000000**8**00000000\$3\times 3\$000000000000**12****24****36**000000000000000000

poly

V2 `poly = lr \times (1 - \frac{iter}{max_iter})^{\text{power}}`

111

- `$power$` 0.9-2.0
- `$iter$`
- `max_iter`

RCNN

□□□□□□□□

Backbone VGG16

4

1. 1k~2k
- 2.
3. SVM
- 4.

□□□□□□**NMS**□□□□□□□□

1. Support Vector Machine (SVM)
2. Support Vector Machine (SVM)
3. Support Vector Machine (SVM)

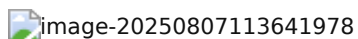
Fast R-CNN

Backbone VGG16

□□□□

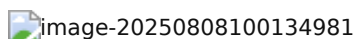
- [illegible]

SSIoU0.5



□ □ □ □ □ □ □ □ □ □

$4(N + 1) \times 4$



$$\hat{G}_x = P_{wd_x}(P) + P_x \setminus \hat{G}_y = P_{hd_y}(P) + P_y \setminus \hat{G}_w = P_{wexp(d_x(P))} \setminus \hat{G}_h = P_{hexp(d_x(P))}$$

- P_x, P_y, P_w, P_h 都是 x, y 的函数
- $\hat{G}_x, \hat{G}_y, \hat{G}_w, \hat{G}_h$ 都是 x, y 的函数

Multi-task loss

$$L(p, \mu, t^{\mu}, v) = \underbrace{L_{cls}(p, \mu)}_{\text{classification}} + \underbrace{\lambda L_{loc}(t^{\mu}, v)}_{\text{localization}}$$

- p 是 softmax 的输出 $p = (p_0, \dots, p_k)$
- μ 是 p 的期望
- t^{μ} 是 p 的期望 u 的函数
- $t_x^{\mu}, t_y^{\mu}, t_w^{\mu}, t_h^{\mu}$ 是 x, y, w, h 的函数
- v 是 (v_x, v_y, v_w, v_h) 的函数
- $\lambda \geq 1$ 是超参数
 - $\lambda = 1$ 时， L_{loc} 和 L_{cls} 权重相等
 - $\lambda = 0$ 时， L_{loc} 权重为 0

$$L_{cls}(p, \mu) = -\log\{p_{\mu}\}$$

$$L_{loc}(t^{\mu}, v) = \sum_{i \in \{x, y, w, h\}} \text{smooth}_{L_1}(t_i^{\mu} - v_i) \setminus \text{smooth}_{L_1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise} \end{cases}$$

$$v_x = \frac{G_x - P_x}{P_w} \setminus v_y = \frac{G_y - P_y}{P_h} \setminus v_w = \ln\{\frac{G_w}{P_w}\} \setminus v_h = \ln\{\frac{G_h}{P_h}\}$$

- G_x, G_y, G_w, G_h 都是 x, y 的函数
- P_x, P_y, P_w, P_h 都是 x, y 的函数

ROI pooling

1. 输入特征图
2. **RoI** 池化操作
3. 池化操作
4. **max pooling**



image-20250811180354848

Faster R-CNN

tm kaiming

RPN + Fast R-CNN

1. 输入特征图 **1k~2k**
2. **RPN** 池化操作
3. **ROI pooling** 操作 7×7

RPN Region Proposal Network

1. **RPN Backbone**
2. **RPN**

- 卷积核大小 k 卷积核大小 $k=9$
 - 卷积核大小 [128, 256, 512] 卷积核大小 [1:1, 1:2, 2:1]
- 卷积核大小 3×3 卷积核大小 3×3 卷积核大小 3×3 **Backbone** 卷积核大小 256-d 卷积核大小
- 卷积核大小 卷积核大小 卷积核大小 卷积核大小 卷积核大小

3. 卷积核大小 RPN 卷积核大小 1×1 卷积核大小

1. 卷积核大小
 - 卷积核大小 $2k$ 卷积核大小 sigmoid 卷积核大小 softmax
2. 卷积核大小
 - 卷积核大小 $4k$ 卷积核大小 卷积核大小 卷积核大小 卷积核大小

卷积核大小

RPN 卷积核大小 **anchor** 卷积核大小 250 卷积核大小 1:1

- 卷积核大小 卷积核大小 0.7 卷积核大小 0.7 卷积核大小 anchor
- 卷积核大小 卷积核大小 0.3

RPN 卷积核大小 anchor 卷积核大小 $2w$ 卷积核大小 $2k$ 卷积核大小 **Fast-RCNN** 卷积核大小

RPN Multi-task loss

$$L(p_i, t_i) = \underbrace{\frac{1}{N_{cls}} \sum_{l \in cls} (p_i, p_i^*)}_{\text{cls loss}} + \underbrace{\lambda \frac{1}{N_{reg}} \sum_{l \in reg} (t_i, t_i^*)}_{\text{reg loss}}$$

卷积核大小

- p_i 卷积核大小 anchor 卷积核大小
- p_i^* 卷积核大小 1 卷积核大小 0
- t_i 卷积核大小 anchor 卷积核大小
- t_i^* 卷积核大小 anchor 卷积核大小
- N_{cls} 卷积核大小 mini-batch 卷积核大小 卷积核大小
- N_{reg} 卷积核大小 anchor 卷积核大小 卷积核大小
- L_{reg} 卷积核大小 smooth 卷积核大小
- L_{cls} 卷积核大小

Mask R-CNN

卷积核大小 Faster R-CNN 卷积核大小 **mask** 卷积核大小 **ROI Pooling** 卷积核大小 **ROI Align**

Mask 卷积核大小 **Faster R-CNN** 卷积核大小 **ROI Align**

卷积核大小 L_{mask} 卷积核大小 BCE-Loss
$$L_{mask_rcnn} = L_{RPN} + L_{Faster_rcnn} + L_{mask}$$

ROI Align

ROI Pooling

ROI Pooling

- ROI pooling 是用于从特征图中提取感兴趣区域 (ROI) 的池化操作。
 - 它接收一个特征图和一个 ROI 坐标。
 - 它返回 ROI 区域内的特征图。

ROI Align

- ROI pooling 是用于从特征图中提取感兴趣区域 (ROI) 的池化操作。
- ROI align 是用于从特征图中提取感兴趣区域 (ROI) 的池化操作。
- ROI pooling 和 ROI align 的主要区别在于 ROI align 能够处理非整数 ROI 坐标，而 ROI pooling 只能处理整数 ROI 坐标。
- ROI align 在精度上优于 ROI pooling，因为它能够更精确地提取 ROI 区域内的特征。

Mask

FPN Mask



image-20250811173140965

Mask

- Mask R-CNN 使用 ROI Pooling 提取 ROI 区域内的特征，然后使用 RPN (Region Proposal Network) 生成 ROI 坐标。
- Mask R-CNN 使用 sigmoid 函数对 ROI 坐标进行归一化，使其在 0 到 1 之间。
- Mask R-CNN 使用 BCE (Binary Cross Entropy) 损失函数对 ROI 坐标进行训练。



image-20250811175302080

- Mask R-CNN 使用 ROI Pooling 提取 ROI 区域内的特征，然后使用 Faster R-CNN 生成 ROI 坐标。
- Mask R-CNN 使用 mask 损失函数对 ROI 坐标进行训练。



image-20250811175327478

Mask R-CNN 使用 ROI Pooling 提取 ROI 区域内的特征，然后使用 ROI Pooling 生成 ROI 坐标。

Mask R-CNN 使用 ROI Pooling 提取 ROI 区域内的特征，然后使用 ROI Pooling 生成 ROI 坐标。

- Mask R-CNN 使用 softmax 函数对 ROI 坐标进行归一化，使其在 0 到 1 之间。
- Mask R-CNN 使用 mask 损失函数对 ROI 坐标进行训练。
- Mask R-CNN 使用 sigmoid 函数对 ROI 坐标进行归一化，使其在 0 到 1 之间。

YoloV5

Backbone 是 DarkNet

YoloV7

YoloV8

Clip

Blip

数据集

- 数据集
- 数据集

COCO数据集

COCO数据集使用 `pycocotools.coco` 与 COCO 数据集的 `annotations` 文件

数据集

方法	描述	代码
<code>.getImgIds()</code>	获取所有ID	<code>train_coco.getImgIds()</code>
<code>.loadImgs(ids)</code>	加载ID对应的数据	<code>train_coco.loadImgs([1,2])</code>
<code>.getAnnIds(imgIds=[])</code>	获取所有ID	<code>train_coco.getAnnIds(imgIds=[1])</code>
<code>.loadAnns(ids)</code>	加载ID对应的数据	<code>train_coco.loadAnns([1,2])</code>
<code>.annToMask(annotation)</code>	将标注转换为掩码	
<code>.showAnns(annotations)</code>	使用matplotlib显示	

Focal Loss

来自kaiming

公式

公式

公式

Focal Loss

公式

- α
- p_t
- γ
-