GSPS
Graduate Student Paper Seminar

CVPR 2022

Proposed by Kaiming He et al.
Facebook AI Research

Self-Supervised Learning

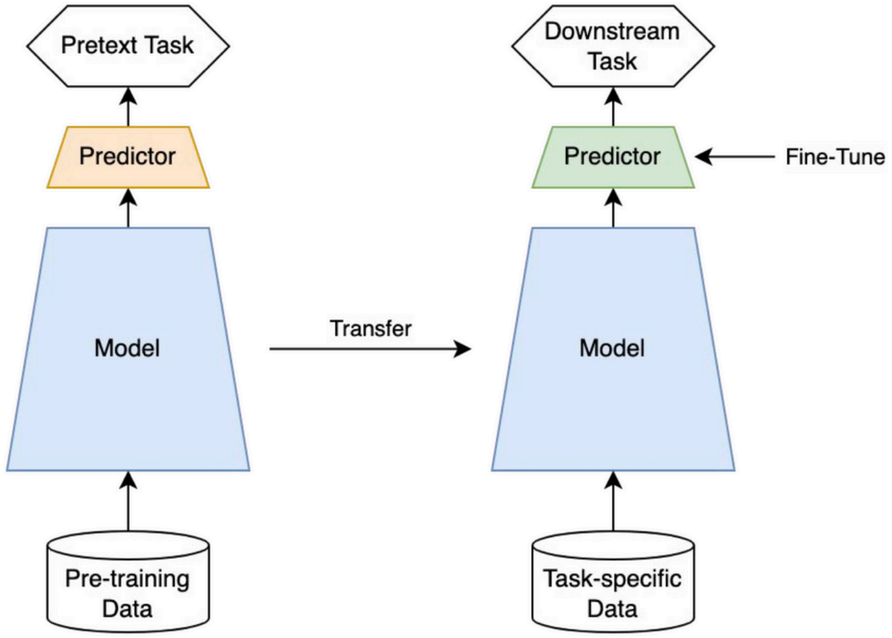
Masked Autoencoders Are Scalable Vision Learners

JinYong Kim

Introduction

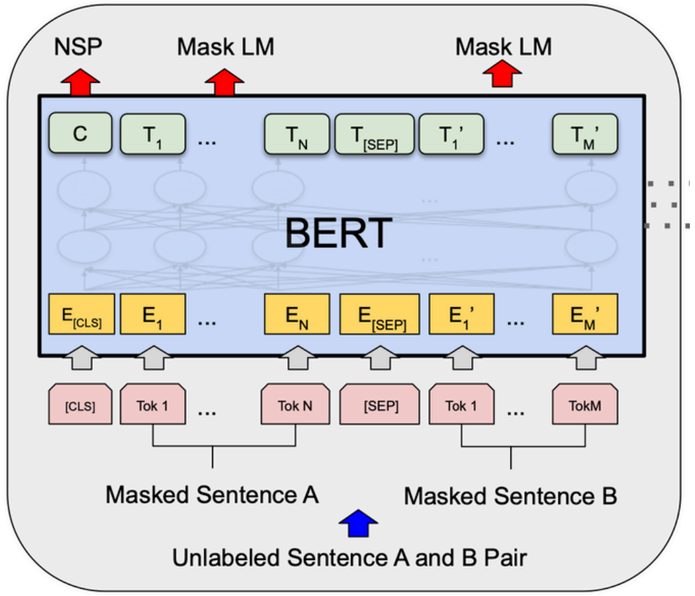
Self-Supervised Learning

- During pretraining, the model learns to perform a specific pre-task
- However, the ultimate goal is not to perform this pre-task itself but rather to transfer the learned representations to downstream tasks
- Self-supervised learning is particularly useful when labeled data is scarce or unavailable
- The model learns meaningful representations of the data, which can then be transferred and fine-tuned for downstream tasks
- This approach is especially beneficial when dealing with large-scale datasets with little or no labeled data



Language - Vision

- In the field of NLP, Self-Supervised Pre-training using Masked AutoEncoders, which involves masking specific parts of the data and predicting them, has been widely used (BERT, GPT)
- NLP has been able to effectively train models with hundreds of billions of parameters using a pretraining-to-finetuning paradigm
- However, while there have been attempts to apply Masked AutoEncoders in the vision domain, they have not achieved performance levels comparable to those in NLP



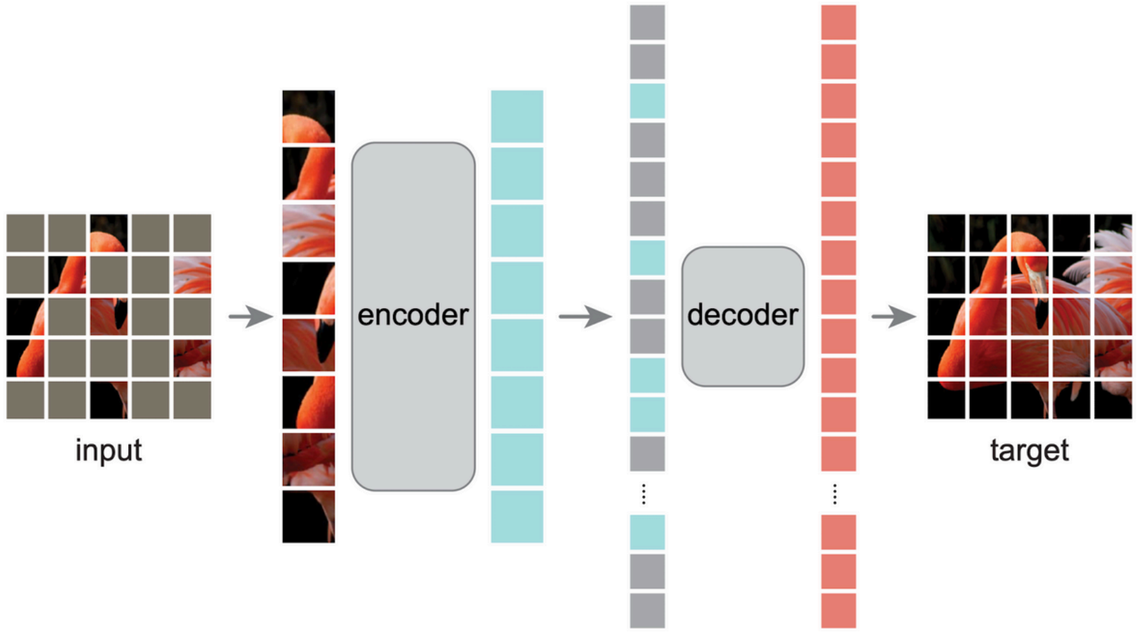
Masked Language Model

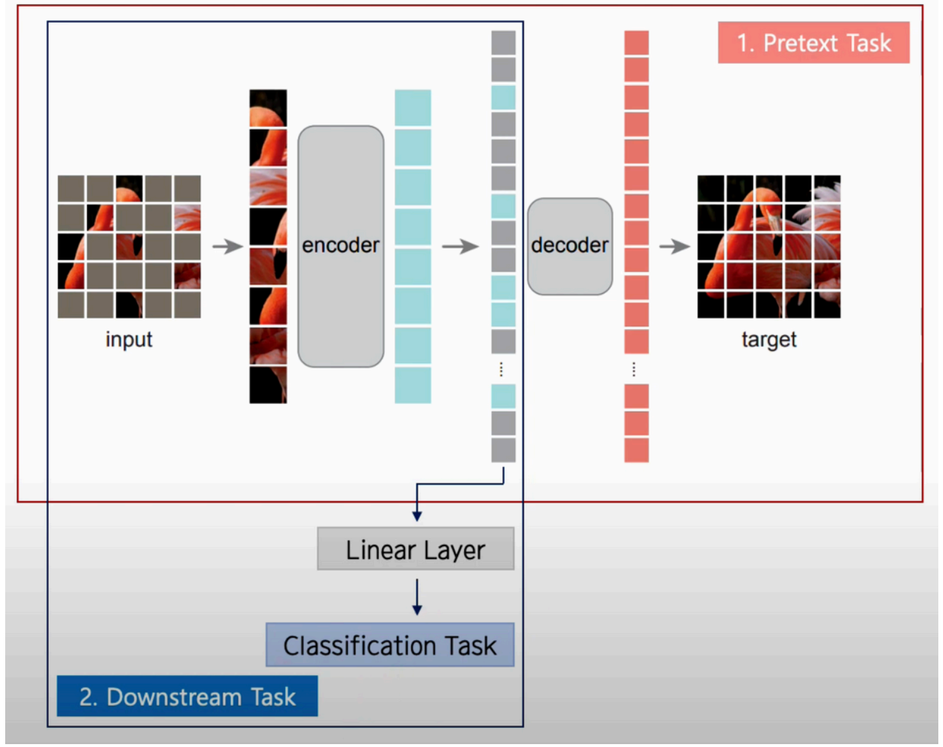
Differences

- Model
  - Previously, the vision domain was primarily based on CNNs, while the NLP domain was built on Transformers
  - Recently, with the adoption of Transformers (ViT) in the vision domain, the gap between vision and NLP has been reduced
- Information Density
  - In images, a single patch typically carries little meaning on its own, whereas in NLP, a single word can hold significant meaning
  - Therefore, in NLP, masking and predicting a few words requires understanding the entire sentence, which helps in learning the language structure itself
  - However, in the vision domain, removing and predicting a few image patches is not as beneficial because the model can infer missing parts using only the surrounding patches without truly understanding the whole image
  - To address this issue, a large portion (75%) of the image patches is masked
- AutoEncoder Decoder in Text and Vision
  - In the vision domain, the decoder reconstructs images at the pixel level. Since each pixel contains only a small amount of information, this process operates at a lower semantic level compared to cognition tasks
  - In contrast, in NLP, each word carries a significant amount of information. As a result, filling in missing words, as seen in models like BERT, is a relatively simple task. This makes the nature of decoding in NLP fundamentally different from that in the vision domain

MAE Architecture

- Based on this analysis, the Masked AutoEncoder (MAE) framework was proposed
- By applying a high masking ratio to patches, MAE significantly reduces computational cost while achieving outstanding performance
- During pre-training, both the encoder and decoder are used
- During downstream task training, only the encoder is used

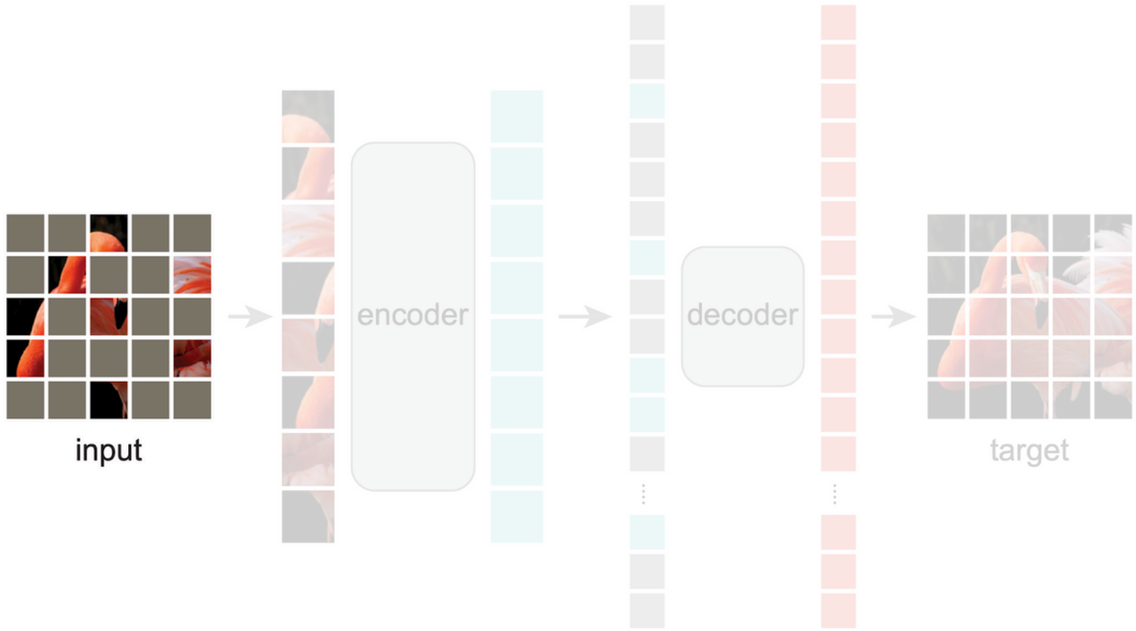




Approach & Implementation

Masking

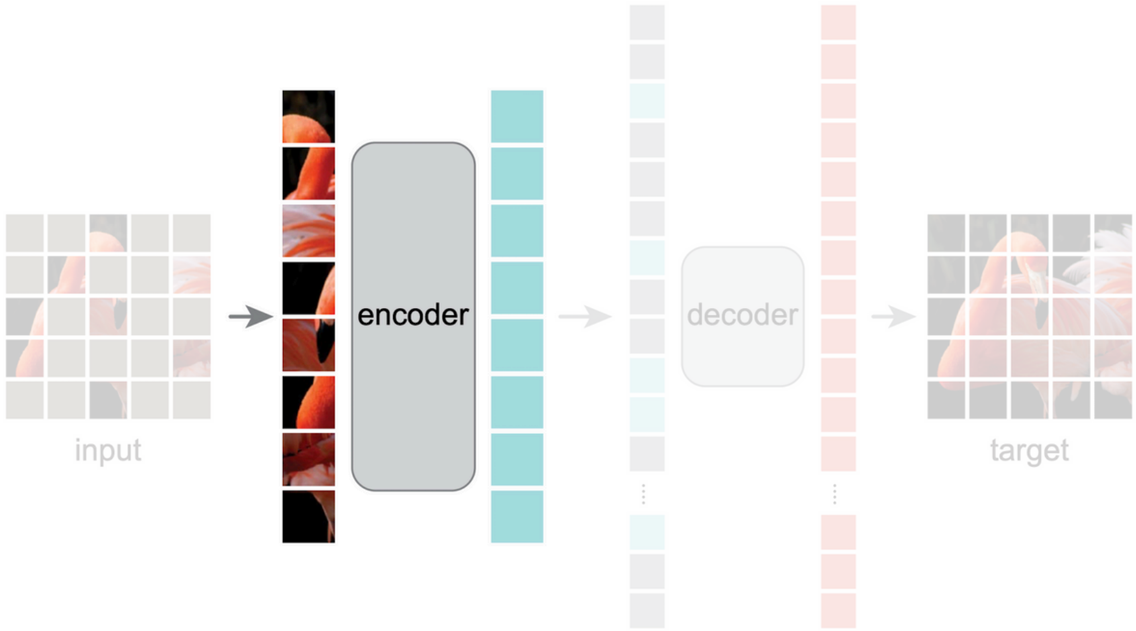
- A high masking ratio (75%) is applied to image patches
- If fewer patches are masked, the model can easily predict the missing parts using surrounding patches, making the task trivial and less effective for learning meaningful representations




```python
class Random_masking(nn.Module):
    def __init__(self,
                 masking_ratio: float):
        super().__init__()

        self.Ratio = masking_ratio

    def forward(self, x):

        batch, lenght, dim = x.shape

        num_keep = int(lenght * (1 - self.Ratio))

        noise = torch.rand(batch, lenght, device=x.device)

        ids_shuffle = torch.argsort(noise, dim=1)
        ids_restore = torch.argsort(ids_shuffle, dim=1)

        ids_keep = ids_shuffle[:, :num_keep]
        x_encoder = torch.gather(x, dim=1, index=ids_keep.unsqueeze(-1).repeat(1, 1, dim))

        mask = torch.ones([batch, lenght], device=x.device)
        mask[:, :num_keep] = 0

        mask = torch.gather(mask, dim=1, index=ids_restore)

        return x_encoder, mask, ids_restore
```

MAE Encoder

- ViT (Vision Transformer) is used, and only the visible patches are fed into encoder
- This approach enables training on a large amount of data at a lower computational cost

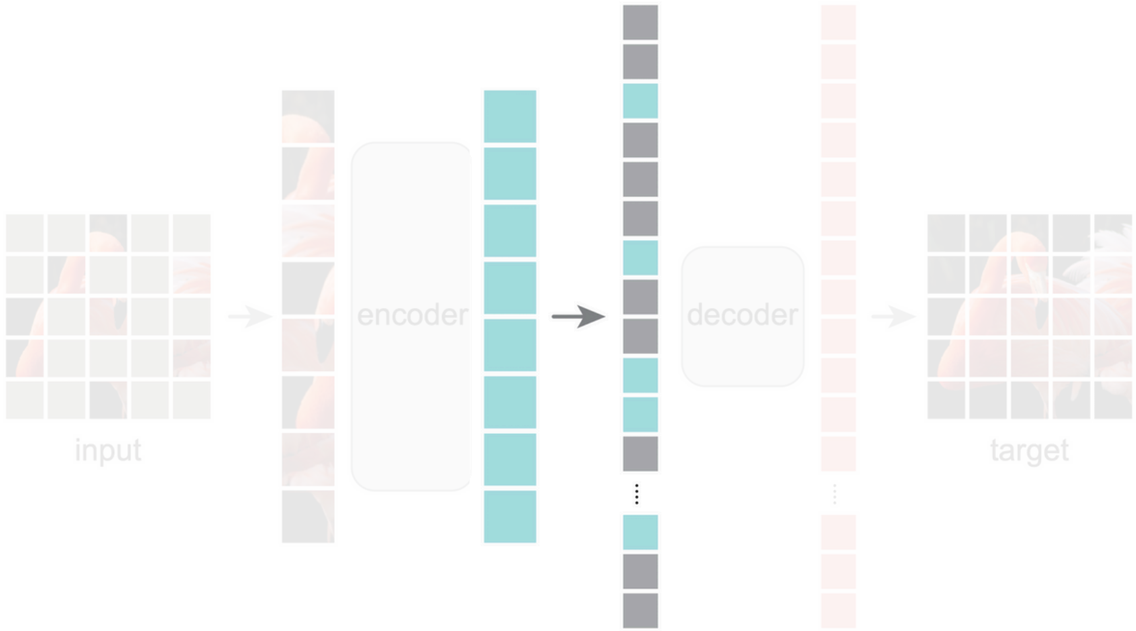



```
Encoder_emb_dim: int = 1024,
Encoder_n_heads: int = 16,
Encoder_depth: int = 24,
```

```
        self.Encoder = nn.Sequential(
            TransformerEncoder(depth = Encoder_depth,
                               emb_dim = Encoder_emb_dim,
                               n_heads = Encoder_n_heads,
                               dropout = dropout,
                               MLP_Expansion = MLP_Expansion,
                               MLP_dropout = MLP_dropout),
            nn.LayerNorm(Encoder_emb_dim)
        )
```

Encoder - Decoder

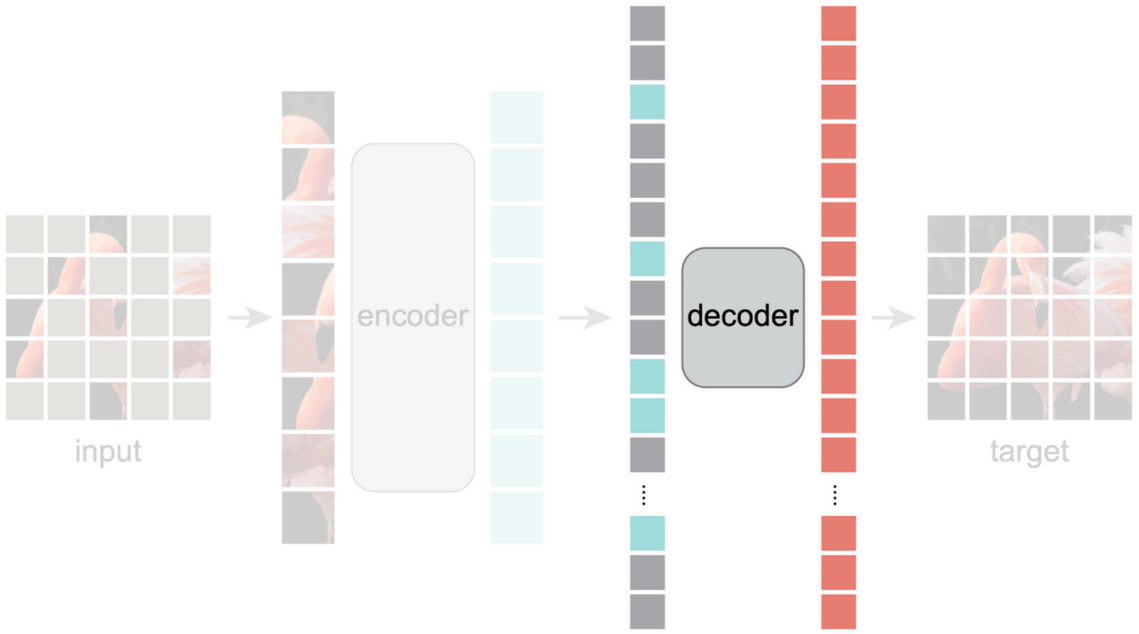
- The encoder output dimension is 1024, while the decoder operates with a smaller dimension of 512
- Therefore, dimension reduction is applied before passing the features to the decoder
- The Encoded Visible Tokens and Masked Tokens are concatenated before being fed into the decoder




```python
class Encoder_to_Decoder(nn.Module):
    def __init__(self,
                 Encoder_emb_dim,
                 Decoder_emb_dim):
        super().__init__()

        self.to_decoder_embedding = nn.Linear(Encoder_emb_dim, Decoder_emb_dim)
        self.mask_token = nn.Parameter(torch.zeros(1, 1, Decoder_emb_dim))

    def forward(self, x, ids_restore):
        x = self.to_decoder_embedding(x)

        mask_tokens = self.mask_token.repeat(x.shape[0], ids_restore.shape[1] + 1 - x.shape[1], 1)
        x_ = torch.cat([x[:, 1:, :], mask_tokens], dim=1)
        x_ = torch.gather(x_, dim=1, index=ids_restore.unsqueeze(-1).repeat(1, 1, x.shape[2]))
        x = torch.cat([x[:, :1, :], x_], dim=1) # [2, 197, 512] cls

        return x
```

MAE Decoder

- Position embeddings are added to all tokens
- masked tokens provide only positional information without additional content
- The decoder also utilizes the Transformer blocks of ViT, but its dimension and depth are smaller than that of the encoder to reduce computational cost

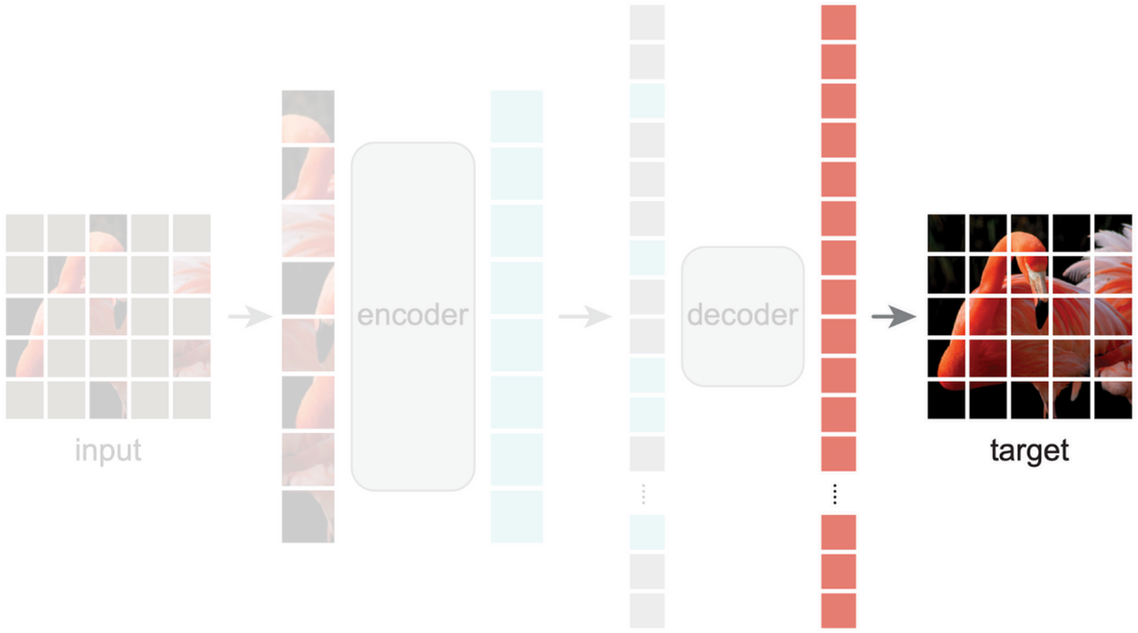



```
Decoder_emb_dim: int = 512,
Decoder_depth: int = 8,
Decoder_n_heads: int = 16,
```

```
self.Decoder_Positions = nn.Parameter(torch.randn((img_size // patch_size) **2 +1, Decoder_emb_dim), requires_grad=False)
self.Decoder = nn.Sequential(
    TransformerEncoder(depth = Decoder_depth,
                       emb_dim = Decoder_emb_dim,
                       n_heads = Decoder_n_heads,
                       dropout = dropout,
                       MLP_Expansion = MLP_Expansion,
                       MLP_dropout = MLP_dropout),
    nn.LayerNorm(Decoder_emb_dim)
)
```

Linear Layer

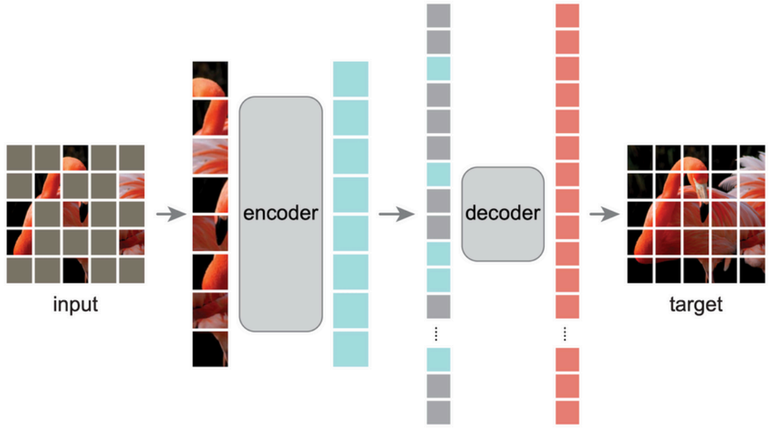
- The final linear layer of the decoder has the same number of channels as the number of pixels in an input patch
- This allows the model to reconstruct the masked regions at the pixel level




```
self.Last_Linear = nn.Linear(Decoder_emb_dim, patch_size**2 * in_channels)
```

Summary

- Overall Summary of the MAE Architecture

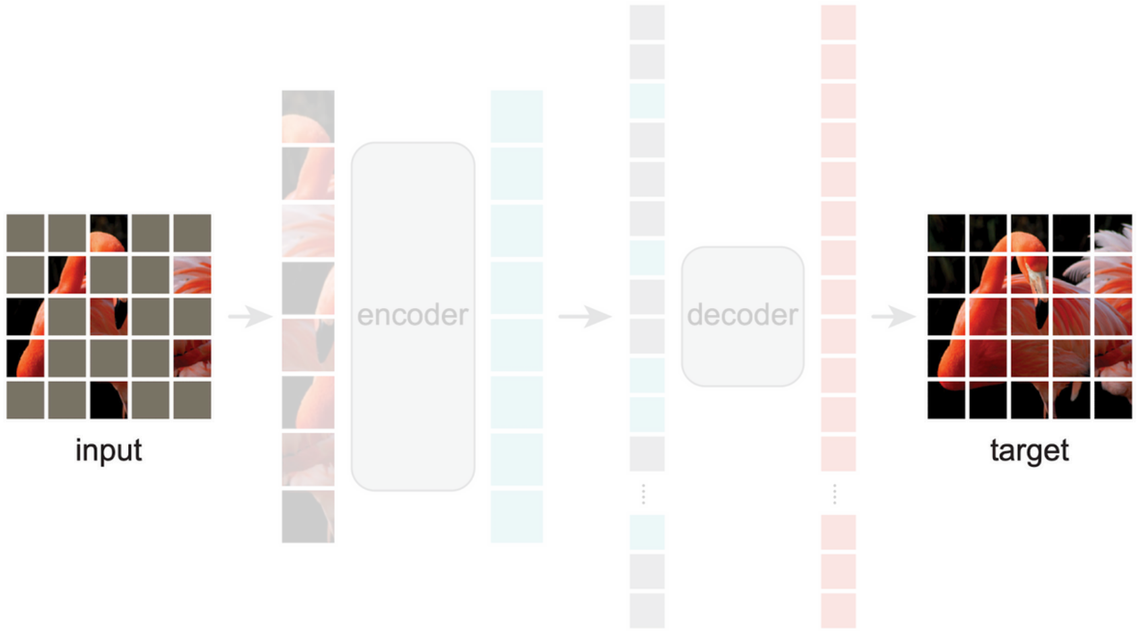



```
==========================================================================
Total params: 329,541,888
Trainable params: 329,239,296
Non-trainable params: 302,592
Total mult-adds (M): 747.25
==========================================================================
Input size (MB): 1.20
Forward/backward pass size (MB): 236.47
Params size (MB): 880.28
Estimated Total Size (MB): 1117.96
==========================================================================
```

```python
def forward(self, x):
    # [2, 3, 224, 224]
    x = self.PatchEmbedding(x) # [2, 196, 1024]
    x = x + self.Encoder_Positions[1:, :]

    x, mask, ids_restore = self.random_masking(x) # [2, 49, 1024] 75%

    cls_token = self.cls_token + self.Encoder_Positions[:1, :]
    cls_token = cls_token.expand(x.shape[0], -1, -1)
    x = torch.cat((cls_token, x), dim=1) # [2, 50, 1024] cls

    x = self.Encoder(x)

    x = self.encoder_to_decoder(x, ids_restore) # [2, 197, 512]

    x = x + self.Decoder_Positions
    x = self.Decoder(x) # [2, 197, 512]

    x = self.Last_Linear(x) # [2, 197, 768], patch_size^2 * 3
    x = x[:, 1:, :] # [2, 196, 768], remove cls

    return x, mask
```

Loss

- The loss function is the Mean Squared Error (MSE) between the predicted patches and the original input patches
- Similar to BERT, the loss is computed only over the masked patches, excluding the visible ones
- Additionally, using normalized pixels as targets has been shown to improve the quality of the learned representations



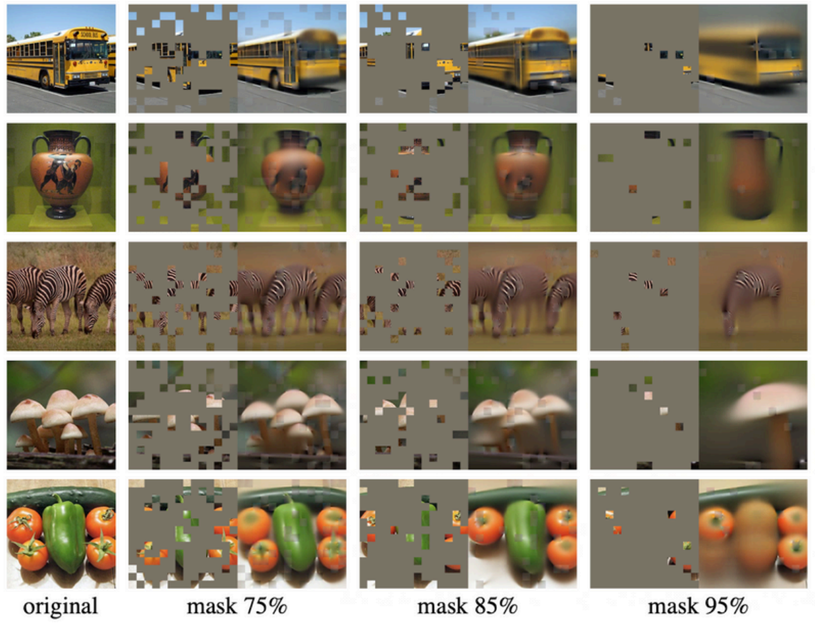
$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

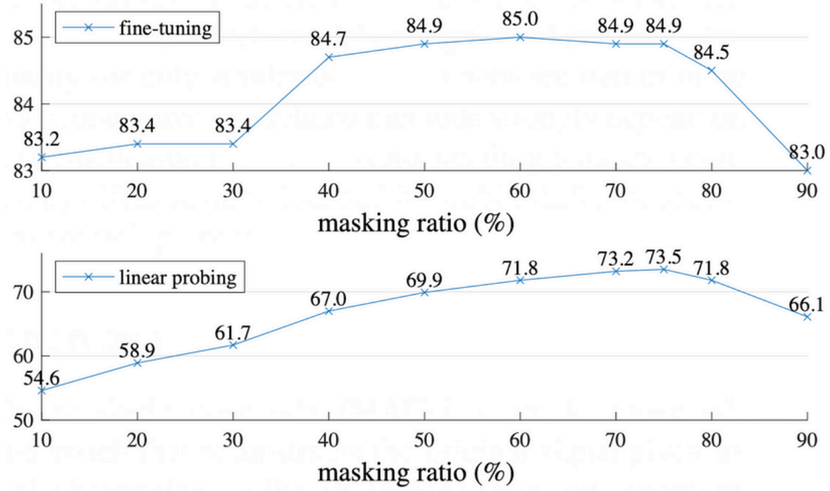
MSE = mean squared error
n = number of data points
Y_i = observed values
$\hat{Y}_i$ = predicted values

Experiment

Masking Ratio

- If the masking ratio is too high, very little information from the original image remains, leading to poor reconstruction and unrealistic outputs
- If the masking ratio is too low, the task becomes too easy, limiting the model's ability to learn meaningful representations





Decoder Design

- The performance of the model during pre-training is influenced by the depth and dimension of the decoder
- In fact, during fine-tuning, using a single-block decoder is sufficient to achieve strong performance

blocks	ft	lin
1	84.8	65.5
2	**84.9**	70.0
4	**84.9**	71.9
8	**84.9**	**73.5**
12	84.4	73.3

(a) **Decoder depth**. A deep decoder can improve linear probing accuracy.

dim	ft	lin
128	**84.9**	69.1
256	84.8	71.3
512	**84.9**	**73.5**
768	84.4	73.1
1024	84.3	73.1

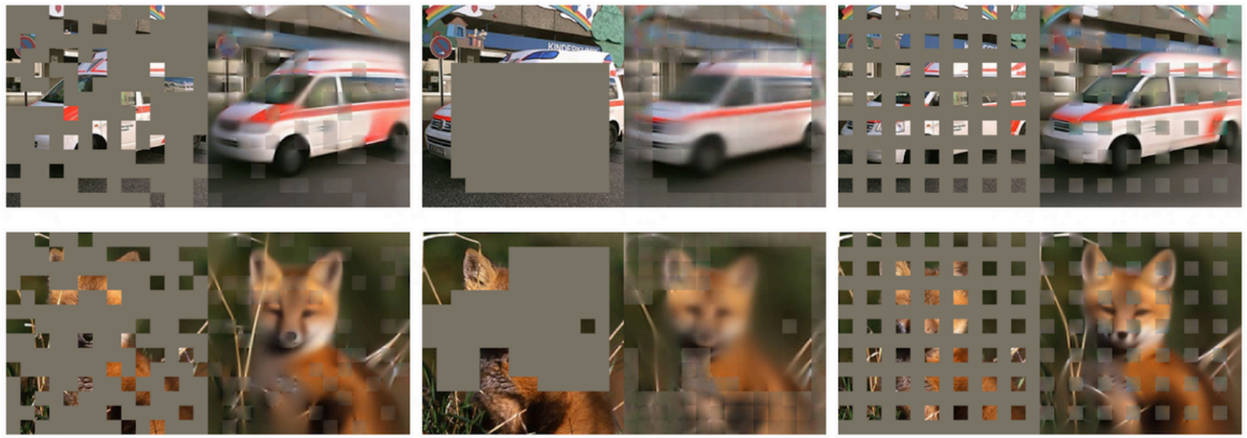
(b) **Decoder width**. The decoder can be narrower than the encoder (1024-d).

Mask Token & Sampling

- Excluding mask tokens from the encoder leads to improved performance in both fine-tuning and linear probing
- Among masking strategies, random masking yields the best results
- Block masking is more challenging than random masking; performance remains stable up to a 50% masking ratio, but degrades at 75% due to excessive information loss
- Grid masking, on the other hand, is too simple to be effective

case	ft	lin	FLOPs
encoder w/ `[M]`	84.2	59.6	3.3×
encoder w/o `[M]`	**84.9**	**73.5**	**1**×

(c) **Mask token**. An encoder without mask tokens is more accurate and faster (Table 2).



random 75% block 50% grid 75%

case	ratio	ft	lin
random	75	**84.9**	**73.5**
block	50	83.9	72.3
block	75	82.8	63.9
grid	75	84.0	66.0

(f) **Mask sampling**. Random sampling works the best. See Figure 6 for visualizations.

Target & Augmentation

- Using normalized target pixels leads to a slight improvement in performance
- Even with simple cropping alone, the model achieves strong results, with random-sized cropping yielding the best performance
- Unlike contrastive learning, MAE performs well even without heavy data augmentation, likely due to the randomness introduced by patch selection

case	ft	lin
pixel (w/o norm)	84.9	73.5
pixel (w/ norm)	**85.4**	**73.9**
PCA	84.6	72.3
dVAE token	85.3	71.6

(d) **Reconstruction target**. Pixels as reconstruction targets are effective.

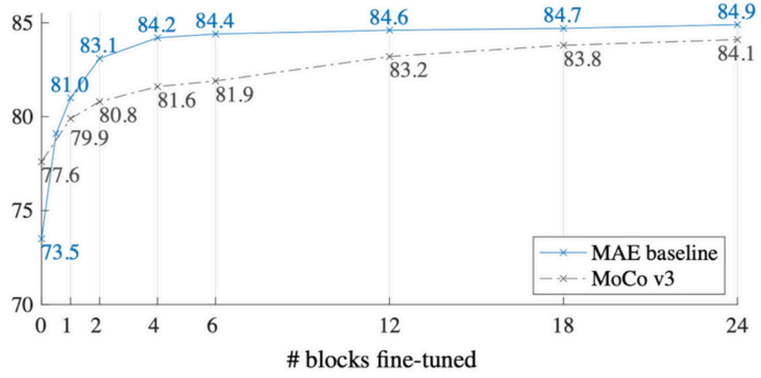
case	ft	lin
none	84.0	65.7
crop, fixed size	84.7	73.1
crop, rand size	**84.9**	**73.5**
crop + color jit	84.3	71.9

(e) **Data augmentation**. Our MAE works with minimal or no augmentation.

Benchmark

- When the pre-trained encoder is fine-tuned end-to-end, it consistently achieves strong performance across various tasks
- The number of fine-tuned blocks influences performance, with the most significant gain observed when increasing from 0 to 1 fine-tuned blocks
- In transfer learning settings, the pre-trained encoder also demonstrates exceptional performance

method	pre-train data	ViT-B	ViT-L	ViT-H	ViT-H$_{448}$
scratch, our impl.	-	82.3	82.6	83.1	-
DINO [5]	IN1K	82.8	-	-	-
MoCo v3 [9]	IN1K	83.2	84.1	-	-
BEiT [2]	IN1K+DALLE	83.2	85.2	-	-
MAE	IN1K	83.6	85.9	86.9	**87.8**



		AP^{box}		AP^{mask}	
method	pre-train data	ViT-B	ViT-L	ViT-B	ViT-L
supervised	IN1K w/ labels	47.9	49.3	42.9	43.9
MoCo v3	IN1K	47.9	49.3	42.7	44.0
BEiT	IN1K+DALLE	49.8	**53.3**	44.4	47.1
MAE	IN1K	**50.3**	**53.3**	**44.9**	**47.2**

dataset	ViT-B	ViT-L	ViT-H	ViT-H$_{448}$	prev best
iNat 2017	70.5	75.7	79.3	**83.4**	75.4 [55]
iNat 2018	75.4	80.1	83.0	**86.8**	81.2 [54]
iNat 2019	80.5	83.4	85.7	**88.3**	84.1 [54]
Places205	63.9	65.8	65.9	**66.8**	66.0 [19]†
Places365	57.9	59.4	59.8	**60.3**	58.0 [40]‡

Discussion

Discussion

- Task-specific optimal masking rate
- Pixel-level loss is sensitive to high-frequency noise.
- Self-Supervised on Radar Tasks

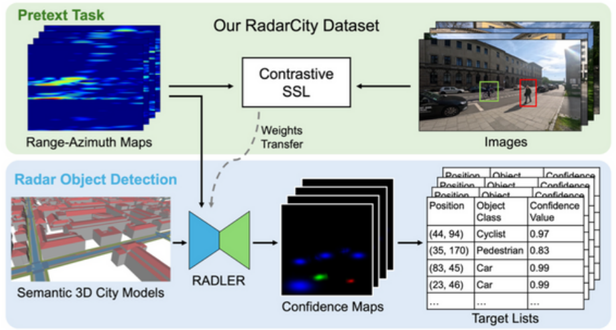
CVPR Nashville JUNE 11-15, 2025

RADLER: Radar Object Detection Leveraging Semantic 3D City Models and Self-Supervised Radar-Image Learning

Yuan Luo[1,3], Rudolf Hoffmann[3], Yan Xia[*1,2], Olaf Wysocki[1], Benedikt Schwab[1], Thomas H. Kolbe[1], Daniel Cremers[1,2]

[1] Technical University of Munich, [2] MCML, [3] GPP Communication GmbH

yan.xia@tum.de* corresponding author



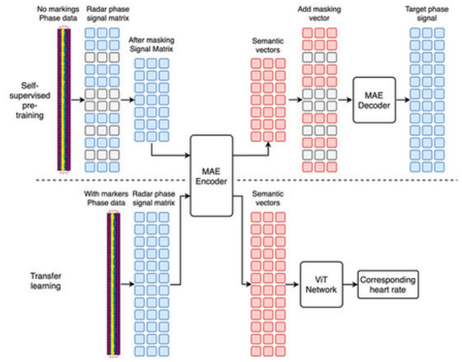
sensors

MDPI

Article

MAE-Based Self-Supervised Pretraining Algorithm for Heart Rate Estimation of Radar Signals

Yashan Xiang [1,2], Jian Guo [1,2,*], Ming Chen [1,2], Zheyu Wang [1,2] and Chong Han [1,2]



Reference

References

Paper

- He, Kaiming, et al. "Masked autoencoders are scalable vision learners." Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2022.
- Devlin, Jacob, et al. "Bert: Pre-training of deep bidirectional transformers for language understanding." Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers). 2019.
- Cao, Shuhao, Peng Xu, and David A. Clifton. "How to understand masked autoencoders." arXiv preprint arXiv:2202.03670 (2022).
- Xiang, Yashan, et al. "Mae-based self-supervised pretraining algorithm for heart rate estimation of radar signals." Sensors23.18 (2023): 7869.
- Luo, Yuan, et al. "RADLER: Radar Object Detection Leveraging Semantic 3D City Models and Self-Supervised Radar-Image Learning." Proceedings of the Computer Vision and Pattern Recognition Conference. 2025.

Media

- MSE Loss Function
  - https://emilia-orellana44.medium.com/not-nice-square-error-2d18c248391c
- MAE github
  - https://github.com/facebookresearch/mae/tree/main
- NeuraIPS
  - https://neurips.cc/media/neurips-2021/Slides/21895.pdf

Thank you

JinYong Kim

E-mail: valere2709@naver.com
Phone: (+82) 10-7317-2709