

Multimodal AI hw1

Shankai Wu, Student ID:r14922146

Environment Screenshot

NVIDIA-SMI 550.144.03			Driver Version: 550.144.03			CUDA Version: 12.4		
GPU	Name	Perf	Persistence-M	Bus-Id	Disp.A	Volatile Uncorr. ECC	GPU-Util	Compute M.
Fan	Temp		Pwr:Usage/Cap		Memory-Usage			MIG M.
0	NVIDIA GeForce RTX 4090		Off	00000000:01:00:0	Off	Off	100%	Default
45%	77C	P2	430W / 450W	20105MiB / 24564MiB				N/A

(a) NVIDIA GPU

```
- Python: 3.10.12  
- PyTorch: 2.8.0
```

(b) Python and PyTorch versions

Figure 1. (a) NVIDIA GPU: RXT-4090, (b) Python and PyTorch versions.

Codebase Structure

`mae_pretrain.py, train_classifier.py` (1)

I've implemented a new feature that automatically writes the training loss for each epoch to a separate CSV file. Additionally, I have modified the TensorBoard logging process to generate a uniquely named directory for each experiment (e.g., for different `decoder_depth` values). These directory names include both the experiment's parameters, which prevents the logs from overwriting one another.

`visualize_predictions.py` (2)

This is a new script I wrote to complete the "classifier prediction visualization" task for the assignment. Its function is to load a fine-tuned classifier model, randomly select several images from the test set, and generate a comparison image that includes the "original image," the "ground truth label," and the "model's predicted label."

1. MAE Pretraining on CIFAR-10

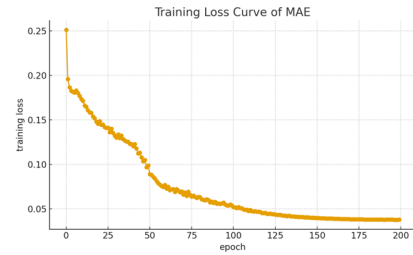


Figure 2. MAE Pretraining:MAE Training Loss Curve

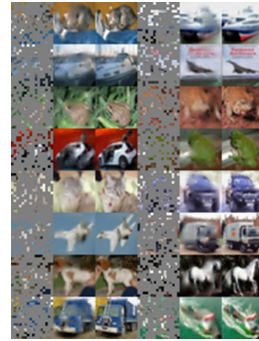


Figure 3. MAE Pretraining: MAE reconstruction visualization

2. Classification with/without MAE Pretraining

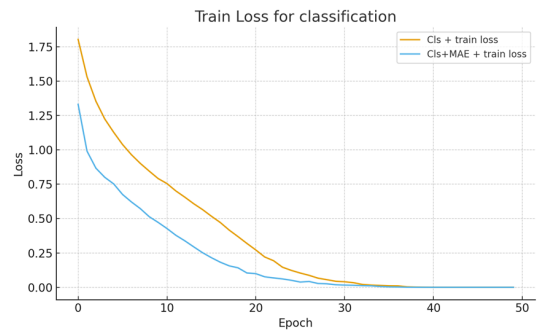


Figure 4. Classification: Training loss curve comparison

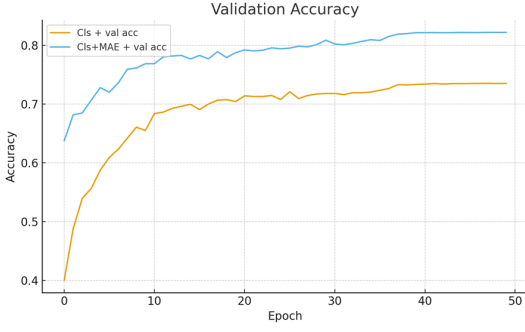


Figure 5. Classification: Validation accuracy curve comparison

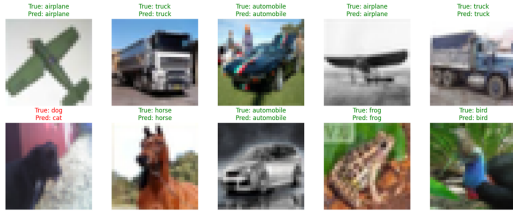


Figure 6. Classification: Visualization of Classifier Predictions with MAE Pretraining

3. Ablation Study

3.1. Decoder depth

3.1.1. Result

Decoder Depth	Final Validation Accuracy (%)
2	79.6084
4	79.7567
8	81.0324

Table 1. (a)Decoder depth: Comparison of final validation accuracy against decoder depth.

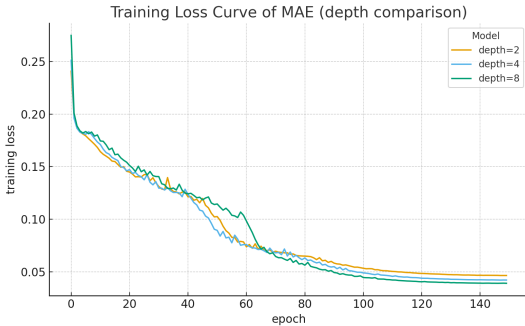


Figure 7. (a)Decoder depth: Training Loss Curve of MAE (depth comparison)

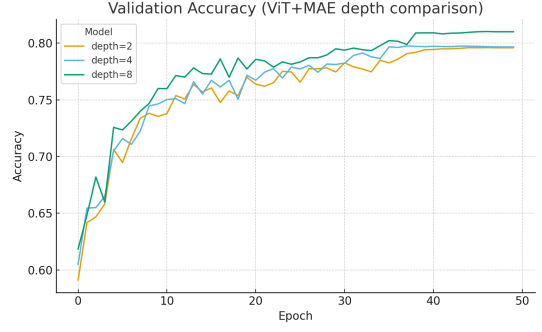


Figure 8. (a)Decoder depth: Validation Accuracy with MAE depth

3.1.2. Implementations

mae_pretrain.py (3)

First, add a command-line argument named "decoder depth" with a default value of 4, which will serve as our baseline. Then, conduct pretraining experiments with the decoder depth set to 2 and 8, respectively. Finally, compare the results among these different configurations.

Pass the received `args.decoder_depth` value to the model constructor, `MAE_ViT`. First, execute the following commands with "`--decoder_depth k`", `k` for 2, 4, 8.

train_classifier.py (4)

"I have updated `train_classifier.py` to implement a new file naming convention. This change affects the log files, the output CSVs, and the saved model checkpoints (.pt files), ensuring that the outputs from different training runs are clearly distinguished from one another."

Listing 1. `mae_pretrain.py`

```

1 if __name__ == '__main__':
2     ...
3     parser.add_argument('--mask_ratio', type=float,
4                           default=0.75)
5     # MODIFY
6     # =====
7     parser.add_argument('--decoder_depth', type=int,
8                           default=4, help='Depth of the MAE decoder
9                           ')
10    # =====
11    parser.add_argument('--total_epoch', type=int,
12                          default=2000)
13    ...
14    # MODIFY
15    # =====
16    model = MAE_ViT(mask_ratio=args.mask_ratio,
17                      decoder_layer=args.decoder_depth).to(device)
18    # =====

```

```

14     optim = torch.optim.AdamW(model.parameters(),
15                               lr=args.base_learning_rate * args.
                                batch_size / 256, betas=(0.9, 0.95),
                                weight_decay=args.weight_decay)
    ...

```

3.2. mask strategy

3.2.1. Result(Table 2., Figure 9., Figure 10., Figure 11.)

Mask Strategy	Final Validation Accuracy (%)
random	0.797567247
block	0.828026108
grid	0.802412975

Table 2. (f)Mask Strategy: Comparison of final validation accuracy against mask strategy.

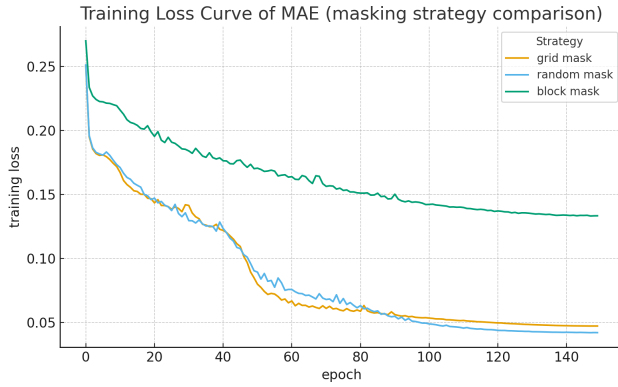


Figure 9. (f)Mask Strategy: Training Loss Curve of MAE mask strategy

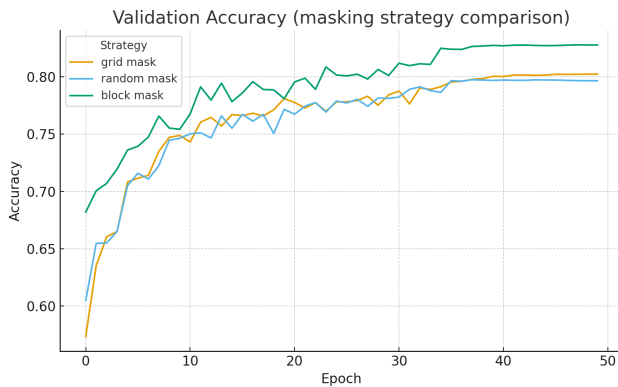


Figure 10. (f)Mask Strategy: Validation Accuracy against mask strategy

3.2.2. Implementations

mae_pretrain.py (5)

The script is modified to accept the command-line argument `args.mask_strategy`, with "random" set as the default value.

model.py - class PatchShuffle (6)

Random Masking : This is the baseline strategy from the original codebase.

Block Masking : To keep a single, contiguous square region of patches visible while masking all others(different from the paper [1]).

Grid Masking : This strategy aims to retain patches in a regular, intermittent pattern, similar to a checkerboard or grid.

Listing 2. model.py

```

1  # --- New PatchShuffle Class ---
2  class PatchShuffle(torch.nn.Module):
3      def __init__(self, ratio, strategy='random') ->
4          None:
5          ...
6          if self.strategy == 'random':
7              # do not changed
8              ...
9          elif self.strategy == 'block':
10             ...
11             # choose a visible block
12             for _ in range(B):
13                 all_indexes = np.arange(T)
14                 top = np.random.randint(0,
15                                         grid_size - remain_grid_size +
16                                         1)
17                 left = np.random.randint(0,
18                                         grid_size - remain_grid_size +
19                                         1)
20                 visible_indexes = []
21                 ...
22             elif self.strategy == 'grid':
23                 ...
24  # --- PatchShuffle Class ---
25  ...

```

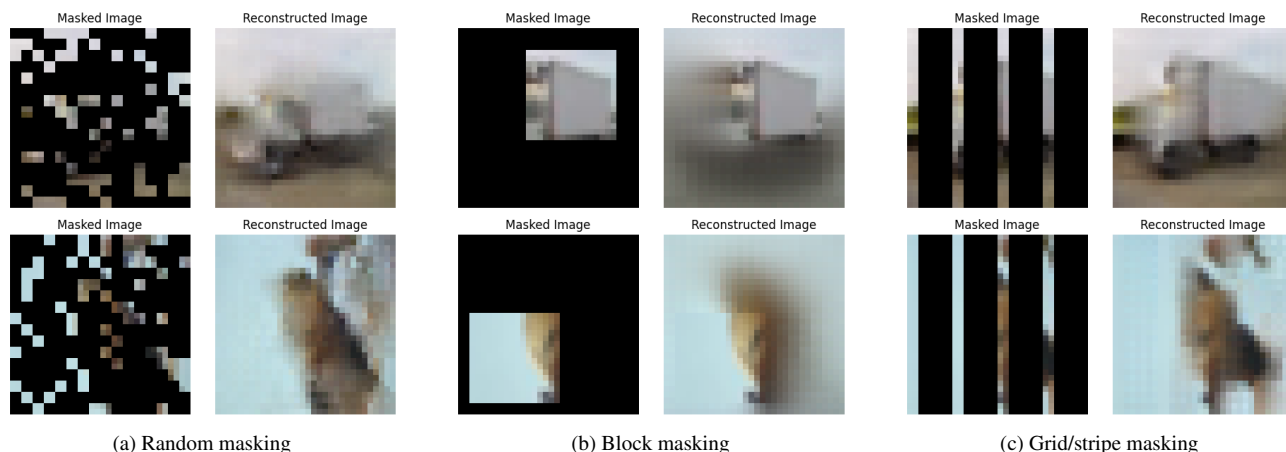


Figure 11. **MAE reconstruction under different masking patterns.** Each panel shows masked inputs (left) and reconstructions (right).

4. Findings

4.1. MAE Pretraining is a Key Driver for Performance Improvement:

The data reveals that the pretrained model not only achieves a higher final validation accuracy but also exhibits faster convergence and lower loss from the early stages of training. This finding confirms the core value of MAE: through the "mask and reconstruct" pretext task, the encoder is compelled to learn deep semantic representations of images, rather than just superficial, pixel-level features. This information-rich starting point allows for a much more effective adaptation to the downstream classification task compared to randomly initialized weights.

4.2. The Decoder Depth of the Decoder Architecture is Crucial:

In the ablation study on "Decoder Depth," we observed a critical phenomenon: a deeper, more powerful decoder actually encourages the encoder to learn better features. The experimental results show that the 8-layer decoder achieved higher final accuracy than the 4-layer decoder, while the 2-layer version performed the worst. The insight here lies in MAE's asymmetric design philosophy: a lightweight encoder processes only the visible patches, while a heavyweight decoder is tasked with the difficult job of reconstruction from sparse information. It is precisely because the reconstruction task is sufficiently challenging that it exerts enough "pressure" on the encoder, forcing it to produce highly abstract and information-dense features to aid the decoder in completing its task.

5. Discussions

5.1. The Implementation of Masking Strategy Directly Impacts Learning Outcomes:

In the "Mask Sampling" experiment, one of the most interesting findings was that our custom implementation of the masking strategies led to a trend in results that differs from the original paper [1]. As seen in the comparison chart, the baseline Block masking strategy achieved the best classification accuracy, followed by Grid masking, with Random masking performing the worst.

The reason behind this is related to the specific implementation of the strategies:

5.1.1. Block Strategy

The implementation of block masking does not mask several random blocks as in the paper [1]. Instead, it preserves only a single, contiguous square of unmasked patches, masking everything else.

5.1.2. Grid Strategy

The grid mask is formed by regularly and intermittently retaining patches to create a visible grid. This approach results in the masked areas forming a striped pattern.

This might be the fundamental reason why the accuracy trend does not fully align with that of the original paper [1]. The experiment inadvertently demonstrates that the specific design of the pretext task has a decisive impact on the final effectiveness of self-supervised learning. Different "rules of the game" guide the model to learn different types of "skills".

References

- [1] Kaiming He. Masked autoencoders are scalable vision learners, 2022. Face and Gesture submission ID 324. Supplied as supplemental material. 3, 4