DD2412 - Project report - *Resubmission*

Masked Autoencoders Are Scalable Vision Learners

Maxellende Julienne
maxjul@kth.se
20000829-T304

Victor Manach
manach@kth.se
19990227-T437

David Nordmark
dnordm@kth.se
19980602-7778

Abstract

This project aims at reimplementing the masked autoencoder (MAE) presented in the paper [8]. The MAE is trained in a supervised manner to reconstruct patch-masked images: images are first split into patches of equal sizes and a subset of those patches are then randomly selected. Only the unmasked patches are fed into the encoder part, while the decoder receives the latent representation of the unmasked patches and the binary masks of all the images. The encoder and the decoder of the MAE are based on the Vision Transformer architecture presented in [6].

The new implementation of the MAE is done using the JAX library in order to speed up both the training and the inference of the model. Our experiments were conducted on low resolution images using a small MAE architecture and a few number of epochs compared to the original paper. The experiments show that the model is capable of reconstructing the original images, even with a small architecture and a small training time. In addition, increasing the size of the architecture is much more beneficial than training for a longer time to improve the performance on the reconstruction task. Finally, the model performs well on the classification task, but does not reach the state-of-the-art performances (especially the accuracy obtained with the ViT in [6]). This is mainly due to the size of the architecture we chose, which is much smaller than the best performing models.

Changes made after the first submission of the report:

- We have found and fixed the bug in our first implementation in the function that recreates the image from a list of patches containing the pixels' color for each pixel and for each patch of the image;
- we have implemented the *grid-wise* sampling strategy for the masking of the input image and compared the performance of the MAE with either *random* or *grid-wise* sampling;
- we have implemented the fine-tuning of a pre-trained MAE to perform the classification task of images on the CIFAR-10 dataset;
- we have compared the performances of two MAE architectures on both the reconstruction and the classification of images tasks (all the results presented in the section Experiments and findings are those obtained after we had fixed the bug in our code).

1 Introduction

The paper we chose for the project, *Masked Autoencoders Are Scalable Vision Learners* [8], presents a masked autoencoder. It aims at reconstructing partially visible images by learning useful representations. This is done by dividing the images into patches and randomly selecting a subset of those patches that will be fed into the autoencoder. The architecture of the autoencoder is chosen to be asymmetric with a small decoder compared to the encoder. In addition, the encoder is fed with only the unmasked patches, while the decoder is given the latent representation and the binary masks of the images. The architecture used in the original paper is able to reconstruct the images with great accuracy and show meaningful results with a high masking percentage of 75% (the best accuracy obtained is 87.7% with a vanilla ViT-Huge model). Moreover, transfer learning with this architecture shows better results than using a simple supervised pre-training.

Contrary to RNNs, the attention mechanism in transformer reads the whole sequence at once. Furthermore, its attention mechanism is thus parallelizable as the input is composed of the sequence (here, the patches of the images) and the information about their position in the sequence (the position tokens of the masked patches).

To sum up, this autoencoder architecture is able to generate pixels thanks to transformer blocks and supervised training. This kind of architecture can be used in image enhancement or data completion, as well as for NLP tasks.

During this project, we reimplemented the masked autoencoder with the three main points that appear to us as the novelty of this paper: the vision transformer block used both in the encoder and the decoder, the embeddings and a positional embedding for the patches and finally the asymmetric structure of the MAE. The originality of this reimplementation task is that it has been carried out using the JAX [2] library in Python, compared to prior implementations that used PyTorch or TensorFlow.

The results obtained show a successful reimplementation of the MAE, with reconstructed images having similar colors and patterns as the original ones. In addition, the fine-tuning of the MAE for the classification task yields satisfying classification performance given the model size we used.

Code repository `https://github.com/Victor-Manach/DD2412-deep-learning-project`

2 Related work

To better understand the issues at stakes, we read articles on the main notions used and discussed in the paper. The BERT article [4], presents Bidirectional Encoder Representations from Transformers which is a milestone for NLP tasks, and for sequence processing more largely: Bidirectional Encoder Representations from Transformers have a deeper understanding of natural language and present state-of-the-art results. This article helped us better understand the architecture of transformers, its interest and the fine-tuning. The BEiT article [1], links the use of transformer for NLP tasks with their use on images: images, once flattened, can be seen as sequence of pixels and thus used in transformers. Finally, the paper [6] presents an architecture of visual transformer (ViT) using masked images modelling (MIM). This might have served as a basis for the MAE paper [8] and helps to better understand the topic and the specific structure of the vision transformer block using in the MAE.

3 Methods

The MAE presented in the paper has the following structure: input is fed to the encoder which creates a latent representation which the decoder then uses to recreate the original input. In addition, binary masks are needed to keep track of the visible and masked patches since the encoder uses partial inputs. This means that the architecture of the MAE is asymmetric, see figure 1. This also results in faster training for the encoder since it operates on only a portion of the full set.

Patches & random masking The input images were divided evenly into patches. All patches have the same size and start from the top left corner of the image and go from left to right and from top to bottom. The random masking is done by randomly removing a portion of the patches specified by the masking ratio, leaving the rest of the patches visible. Only the visible patches are then fed into the encoder.

Patch embeddings Each patch of an image is converted to a vector using a 2D-convolution layer with a kernel of size `(patch_size, patch_size)` and strides of size `(patch_size, patch_size)`. This embedding changes the representation of the patches from a a 2-dimensional array to a 1-dimensional array.

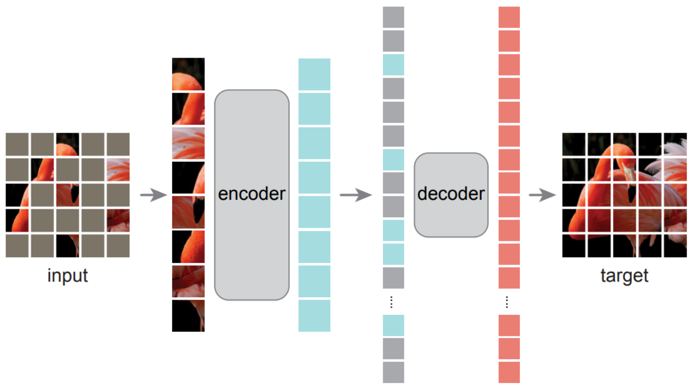


Figure 1: MAE architecture. The input of the encoder are just the visible patches, but the input of the decoder is the full set of patches with the mask tokens.

Positional embeddings The positional encoding method for the patches of an image used here is the *sine-cos* embedding method, as first presented in [10]:

$$\begin{aligned}\mathrm{PE}_{\mathrm{pos},2i} &= \sin\left(\frac{\mathrm{pos}}{10000^{2i/d_{\mathrm{embedding}}}}\right)\\ \mathrm{PE}_{\mathrm{pos},2i+1} &= \cos\left(\frac{\mathrm{pos}}{10000^{2i/d_{\mathrm{embedding}}}}\right)\end{aligned}$$

where

- *pos* is the position of the patch in the list of patches (flattened array containing the list of patches, from left to right and top to bottom);
- $d_{\text{embedding}}$ is the dimension of the embedding.

The positional embedding is computed once as all images have the same number and size of patches. It is then always added to each input during the forward path of the MAE.

Transformer block The transformer blocks used in the MAE are those of the paper [6].

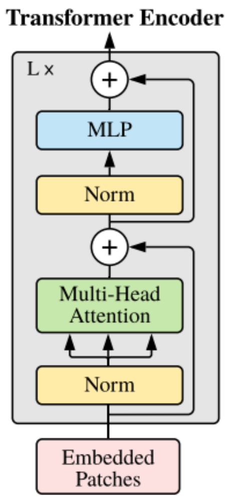


Figure 2: Transformer architecture from the original paper [6].

A Transformer block is composed of 2 sub-blocks. The first sub-block is composed of the following layers:

- **1 normalizing layer** (the LayerNorm normalizes the activations of the layer for each given example in a batch independently, rather than across a batch);
- **1 Attention block**, composed of dense and dropout layers with a softmax activation;
- **1 scaling layer** (the LayerScale multiplies the inputs by an ϵ to ensure that the values do not explode).

The output of the first sub-block is added to the input, i.e., $x = x + \text{sub\_block}(x)$, and it is used as the input of the second sub-block of the transformer block. The second sub-block is composed of the following layers:

- **1 path dropout layer** (the DropoutPath layer randomly sets some of the inputs to 0 during training);
- **1 LayerNorm**;
- **1 MLP block** (Multi-Layer Perceptron with one hidden layer, 2 dropout layers and a GELU activation layer);
- **1 LayerScale**.

MAE encoder The MAE encoder first consists in the creation of the patch embeddings and then the random masking of the patches. Then, the positional embedding is added to each input. Finally, a series of transformer blocks is applied and the output is normalized. The transformer blocks are used in order to make the encoder understand the context and create meaningful representations even though it is given only partial representation of the images.

MAE decoder The MAE decoder first adds the mask tokens to the inputs (latent representation from the encoder) and adds the positional embeddings again. Then a series of shallower and narrower transformer blocks are applied (compared to the transformer blocks of the encoder). This lowers its computational cost and makes the whole MAE able to train faster. The decoder operates on the latent representation of visible patches given by the encoder and the binary masks so that the decoder has information about masked and unmasked patches. Its task is to reconstruct the original image.

Loss function The loss function calculates the mean square error (MSE) between the reconstructed image (output of the decoder) and the original image. As we are mainly interested by the reconstructed patches that were initially masked, the loss is computed only on those masked patches.

4 Data

For the experiments, the dataset CIFAR-10 [9] was used for training and testing the model. In the original paper, the dataset ImageNet [3] was used, but due to limited resources, we have decided to train on a smaller dataset, with images having a lower resolution. That lead us to the CIFAR-10 dataset, with images having a size of 32×32, with 50 000 train images and 10 000 test images. The dataset from the *tensorflow-datasets* [7] library. The preprocessing that has been done is to normalize the values of the pixels between 0-1.

The CIFAR-10 dataset is widely used for image classification tasks. An example of new research using this dataset would be the enhancement into classification using CNNs [5], that yields very promising results. The CIFAR-10 has also been used to create models that defend against adversarial attacks which is a current problem for deep learning and neural networks.

5 Experiments and findings

5.1 Pre-training the model on the CIFAR-10 dataset

The first step is to pre-train an MAE model in a self-supervised way on the CIFAR-10 dataset. We have run our experiments with two different model architectures: a small and a medium one. The architectures are described in Table 1, with the following parameters:

- the depth of the encoder/decoder which is the number of vision transformer blocks;
- the number of heads for the encoder/decoder which is the number of heads in each of the multi-head attention block inside the vision transformer block;
- the MLP ratio which is used to define the hidden dimension inside each MLP used in the vision transformer (*hidden_dimension* = *embedding_dimension* × *MLP_ratio*).

Table 1: Architectures used in our experiments.

	Small architecture	Medium architecture
Embedding dimension of the encoder	128	256
Depth of the encoder	3	4
Number of heads in the encoder	4	4
Embedding dimension of the decoder	64	128
Depth of the decoder	1	2
Number of heads in the decoder	4	4
MLP ratio	2	2

For all our experiments, we used a masking ratio of 75% as done in the original paper.

During the pre-training phase, the loss is computed on the train images and plot with respect to the number of epochs. The figure 3 shows the loss for the small and medium MAE architectures trained for 100 epochs. The loss is the MSE between the masked image and the reconstructed image (the loss is only computed on the masked patches). We can already see that the loss obtained with the medium architecture decreases faster than with the small architecture and reaches a lower value at the end of the pre-training phase.

5.2 Reconstruction performance

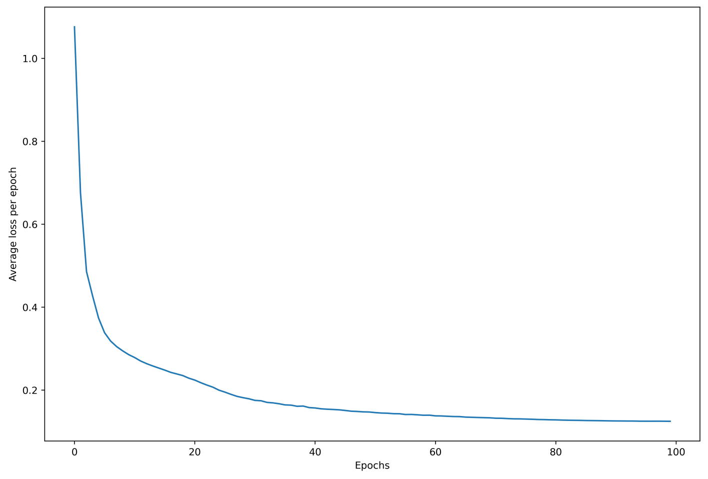
After the pre-training of the model, we check the performance of the model at the reconstruction of masked images. To evaluate the model performance, we visually compare the output image of the model with the original unmasked image. The comparison between the reconstructed and original images are displayed in figures 4, 5 and 6, where the pre-trained small and medium architectures were used on images of the test dataset, with different sampling strategies.

5.3 Fine-tuning for the classification of images

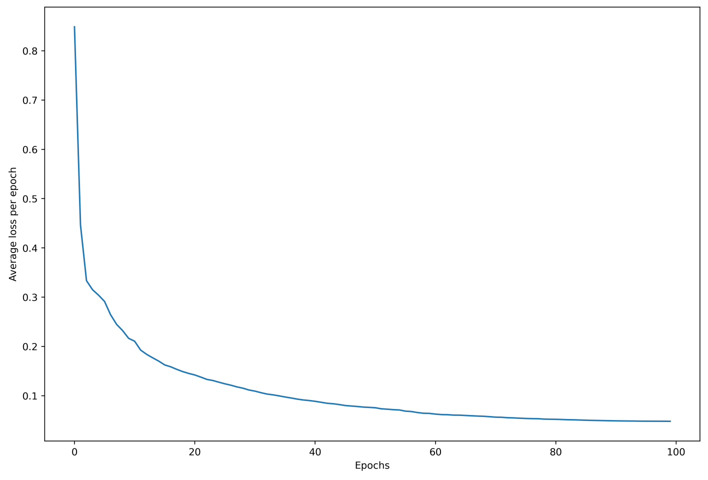
After the pre-training phase, the goal is to evaluate the performance of the model on the classification of images. In order to do that, we kept the encoder of the MAE (with the pre-trained parameters) and we added on top of that either a Dense layer or an MLP block with the output dimension equal to the number of classes in the CIFAR-10 dataset (i.e., 10 different classes). During the fine-tuning, the parameters of the encoder are frozen, and only the parameters of the Dense layer/MLP block are updated. The goal is to measure the accuracy of the model, how the accuracy changes with respect to some parameters of the MAE (masking ratio, mask sampling strategy and architecture of the MAE) and compare the conclusions with those presented in the original paper (see Section 5.4). The classification accuracy is computed as follows:

$$\text{accuracy} = \frac{\text{number of correct classifications}}{\text{number of samples}}$$

The figure 9 shows the evolution of the accuracy with respect to the number of epochs during fine-tuning for the small and medium architectures with the random sampling strategy. The accuracy obtained at the end of the fine-tuning is not improving when using a model pre-trained for a longer time: 0.906 vs. 0.901 with a pre-trained model for 100 epochs vs. 1 000 epochs, with the small architecture and random sampling (see figure 9 in Appendix A for the evolution of the accuracy with respect to the number of epochs during fine-tuning). Thus, in the following experiments, the pre-training of the models is always for 100 epochs and the fine-tuning for 50 epochs.



(a) Small architecture



(b) Medium architecture

Figure 3: Evolution of the train loss with respect to the number of epochs during the pre-training of the MAE with random sampling.

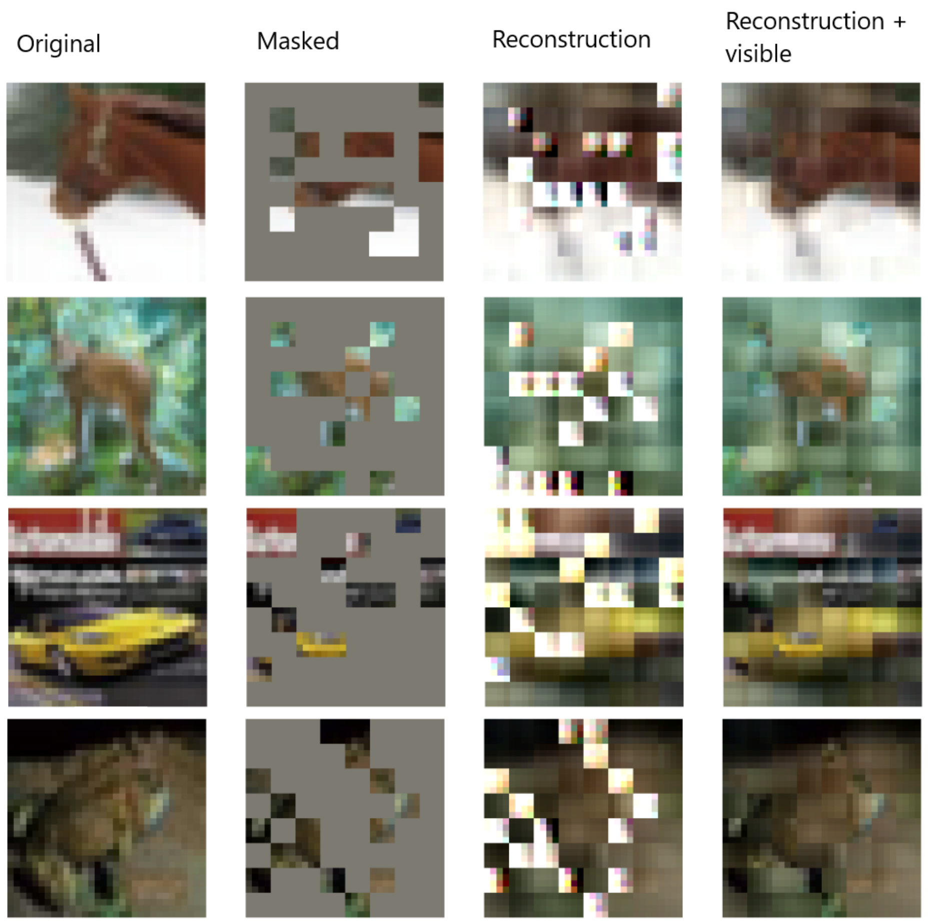


Figure 4: Reconstruction of 4 images from the test set of the CIFAR-10 dataset. The model used is the **small MAE architecture with random sampling**, pre-trained for 100 epochs. Each quadruplet shows the original image, the masked image, the reconstructed image and the reconstructed image with the visible patches.

5.4 Influence of parameters on the classification accuracy

The following experiments are also presented in the original paper, and the goal of this section is to compare if we get the same conclusions on the influence of some of the most important parameters of the MAE on its classification accuracy.

5.4.1 Masking ratio

The figure 8 shows the evolution of the classification accuracy with respect to the masking ratio of the images for both the small and medium architectures, with the random sampling strategy. Both architectures are fine-tuned for 50 epochs on the CIFAR-10 dataset, using the pre-trained MAE parameters for 100 epochs.

The results obtained are in line with those presented in the original paper, showing that the best accuracy is obtained for a masking ratio around 75%, with an accuracy increasing with the masking ratio until a certain point, and then decreasing after that point (around 75%).

5.4.2 Mask sampling strategy and architecture size

The Table 2 displays the classification accuracy for different sets of parameters at the end of the fine-tuning of 50 epochs. The figures of the evolution of the classification accuracy during the fine-tuning phase are displayed in Appendix A (see figures 10 and 11).

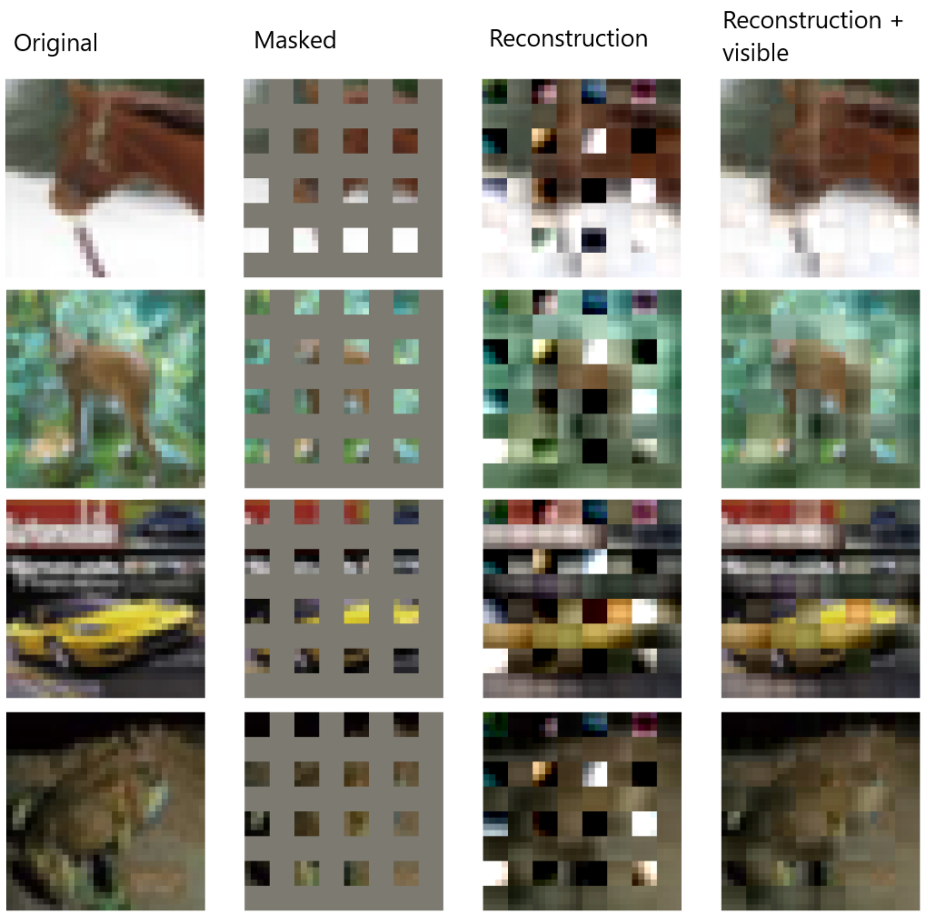


Figure 5: Reconstruction of 4 images from the test set of the CIFAR-10 dataset. The model used is the **small MAE architecture with grid-wise sampling**, pre-trained for 100 epochs. Each quadruplet shows the original image, the masked image, the reconstructed image and the reconstructed image with the visible patches.

For the mask sampling strategy, the results obtained are also in line with the original paper where they describe the random sampling strategy as the best working strategy for the MAE, both for the computation time and for the classification accuracy. In our case, we do not notice a significant change in the computation time between the two strategies, but the classification accuracy is better with the random sampling strategy.

For the architecture size, the results obtained show that the accuracy is significantly higher for the medium architecture, which is in line with the fact that a very large architecture is needed to get a high accuracy, especially for bigger and more complex images (for instance, for the ImageNet dataset used in the original paper).

Table 2: Classification accuracy on the train dataset after 50 epochs of fine-tuning for different parameters.

	Random sampling	Grid-wise sampling
Small architecture	0.906	0.862
Medium architecture	0.933	—

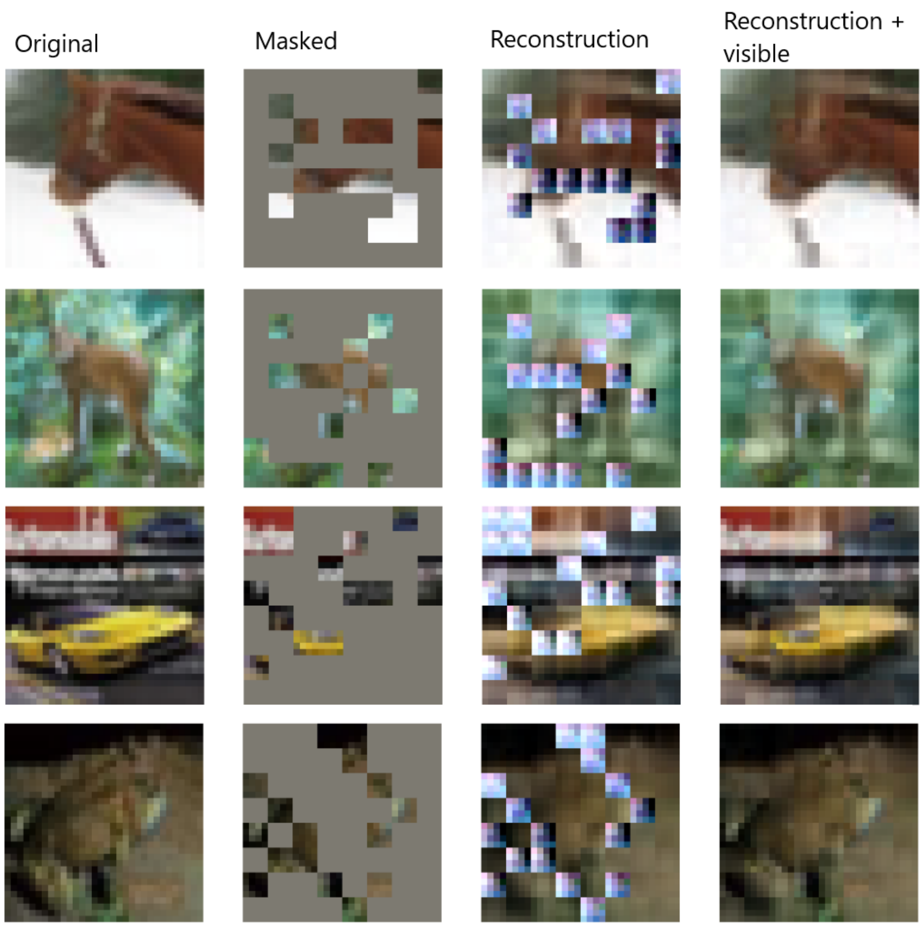


Figure 6: Reconstruction of 4 images from the test set of the CIFAR-10 dataset. The model used is the **medium MAE architecture with random sampling**, pre-trained for 100 epochs. Each quadruplet shows the original image, the masked image, the reconstructed image and the reconstructed image with the visible patches.

5.5 Findings

The figures 4, 5 and 6 clearly show that the model is able to reconstruct the original images well, even with rather small model architectures (both the small and medium architectures are small compared to the ones used in the original paper).

All the experiments described above show that the bigger the model architecture the better, with the figure 7 showing the decrease of the loss between the original and the reconstructed image for several model architectures and different training times. With both architectures, we get the reconstructed masked patches with coherent colors (similar to the original ones) and some pattern reconstructions, yet quite blurry. The improvement we get with the medium architecture is that there are more details in the reconstructed images and the shapes in the image are less blurry than what we get with the small architecture.

Finally, the fine-tuning of the model for the classification task gives decent results, but still far from the best performing algorithms on the CIFAR-10 dataset. This can be explained by the fact that the model sizes we used are really small compared to the best performing algorithms, and we have shown that increasing the model size significantly increases both the reconstruction performance and the classification accuracy. In addition, the experiments we conducted to test the classification performance with respect to some of the MAE parameters gave us the same results as the original paper.

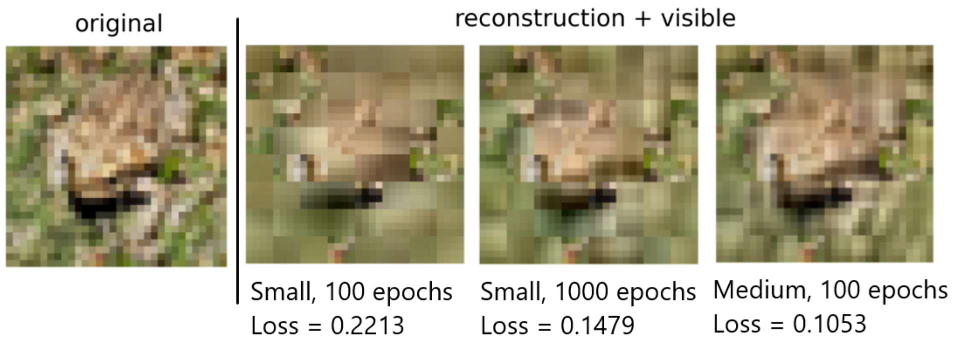


Figure 7: Image reconstruction comparison between the small and medium architectures, pre-trained for different number of epochs.

6 Challenges

6.1 Reimplementation using JAX library

The first challenge faced during the reimplementation of the paper was to write the code using the JAX library. The main motivations for that choice was that the paper had already been implemented using Tensorflow and PyTorch and that a successful reimplementation of the paper using JAX could speed up the training and inference phases.

Although most details needed on the architecture of the MAE were given in the paper (especially all the hyperparameters to define the architecture of the MAE and how to compute the embeddings), we looked at the code for the vision transformer block as well as the authors' code to help us on the structure to follow for our code. Indeed, neither of us had a clear vision of where to start to write the code just by reading the paper, so we used their code structure and adapted it to match the JAX library requirements and functions.

In addition, it was really challenging to be able to find the bug in our first implementation, as the code was running without error. It was quite difficult to test only certain parts of the code and a lot of efforts and time were needed to find the exact function that had the bug.

6.2 Deploying code to Google Cloud

The second challenge we faced was to deploy the model on a Google Cloud Virtual Machine (VM) instance. We have tried several methods to do so:

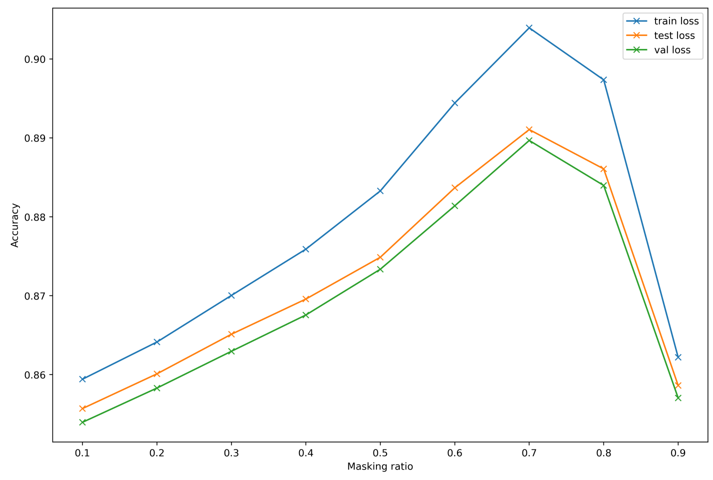
1. Have a CPU-only VM instance with all the required libraries installed.
2. Have a GPU VM instance with all the required libraries installed and compatible with CUDA.
3. Create a Docker container that contains the Python code and the requirements for the Python libraries and deploy the container to a GPU VM instance.

The second and third methods were not successful because we have not been able to create a working environment with CUDA-enabled Python libraries to make use of the GPU resource. Thus, all the experiments were conducted on a CPU-only VM instance, which slowed down the training time. The VM instance had 2 vCPUs and 13 GB of RAM.

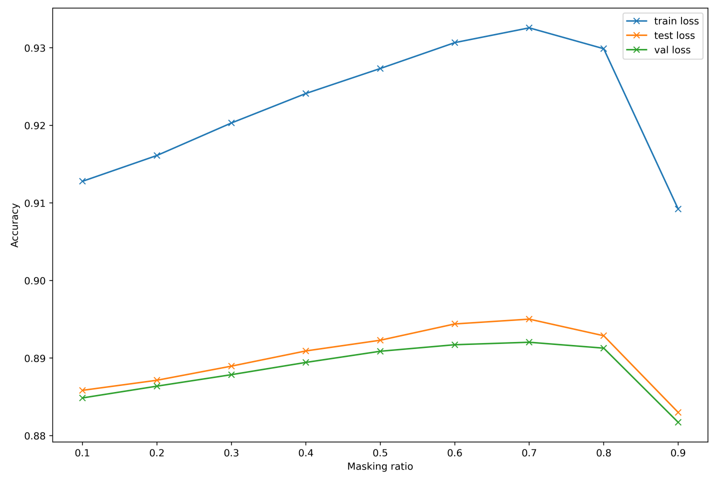
6.3 Computational need and resources

The last challenge faced was that the model presented in the paper has a very large architecture with a lot of parameters, and is trained on a large dataset with high resolution images (ImageNet with 224×224 images).

Training time As we had less computational resources, we downgraded the resolution of the images (we used CIFAR-10 dataset instead of ImageNet), we used a smaller architecture for the



(a) Small architecture



(b) Medium architecture

Figure 8: Evolution of the classification accuracy with respect to the masking ratio: accuracy on the train, test, validation datasets.

MAE and we trained the model for a low number of epochs (100 epochs maximum). With those parameters, the pre-training time was significantly reduced, and the fine-tuning was quite fast as we only use the encoder of the MAE. It allowed us to train the model on a CPU and still get satisfying results with good reconstruction quality and high classification accuracy.

7 Conclusions

The main takeaway from this project for us is the ability to structure the code and define the needed functions for a given task. In addition to that, we have a better in depth understanding of how the JAX library works, especially for random numbers and function vectorization.

Another takeaway, related to our work after the first submission of the project, is that it is very important to test each function to check if it does what it is supposed to do, in order to avoid unexpected results when training and using the model. We had underestimated how difficult and time consuming it can be to spot a bug in the code, and that lead us to submit a first version of our code with a bug.

During the project, we managed to build a working model based on the work of the original paper in a new Python library and then trained and fine-tuned that model. The experiments we have conducted with our model shows that we get coherent results compared to those of the original paper, even with a small model architecture. The discrepancies observed between our results and those presented in the original paper are highly due to the fact that we have worked with a small model, with a lower training time.

Another key element of the project that we had underestimated was the complexity of deploying and training a model on a machine with high computational performance. That is one of the main challenge that prevented us from training our model on images with higher pixel quality like the original papers image data.

8 Ethical consideration

The powerful interest of the MAE is its ability to generate pixels to reconstruct images. This is important since it can help to complete missing or incomplete data. Let's reflect upon the ethical implications of this technology.

This technology could be useful in fields like medical research, for example diseases prediction using AI or Machine Learning, where some records or vitals data are incomplete.

Yet, it raises concerns when a part of a picture is "masked" for safety or privacy reasons, for example to hide a persons identity (soldier, victim of crime, whistle-blower, etc.). This MAE and other similar technologies could, if ambitious enough, disregard visual anonymity. The right to privacy is already a big issue for many people around the world, it is easy to imagine how the "surveillance society" and totalitarian regimes could benefit from and have questionable use cases with these AI progresses.

In the same way of thinking, we could argue for the societal benefits of this research, such as restoring damaged images and records, criminal investigations, etc. Indeed, one could simply argue that there is no such thing as bad or unethical research, only bad practices.

9 Self Assessment

In this project, we managed to reimplement the model described in the original paper with some fine-tuning using the JAX library. Some experiments were conducted to confirm the conclusions given in the original paper, mainly about the influence of some of the most important hyperparameters on the classification performance of the model. The main barrier to get closer results to the ones presented in the original paper was the computational resources, but the conclusions of the experiments that we have conducted match the conclusions of the original paper.

This report follows a clear structure, with a short description of the original paper and a strong focus on the work done during our project. The main results are explained with figures to support our conclusions. In addition, we paid attention to having a clear and commented code that can be easily understood by others.

Since the first submission of our work, we have fixed a bug in our implementation and added a fine-tuning to test the classification performance of the model. For those reasons, we believe that our work during this project corresponds to a *satisfactory project*, for which we expect the grade E.

References

[1] Hangbo Bao, Li Dong, and Furu Wei. Beit: BERT pre-training of image transformers. *CoRR*, abs/2106.08254, 2021.

[2] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018.

[3] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *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)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[5] Siripuri Divya, Bhaskar Adepu, and P. Kamakshi. Image enhancement and classification of cifar-10 using convolutional neural networks. In *2022 4th International Conference on Smart Systems and Inventive Technology (ICSSIT)*, pages 1–7, 2022.

[6] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *CoRR*, abs/2010.11929, 2020.

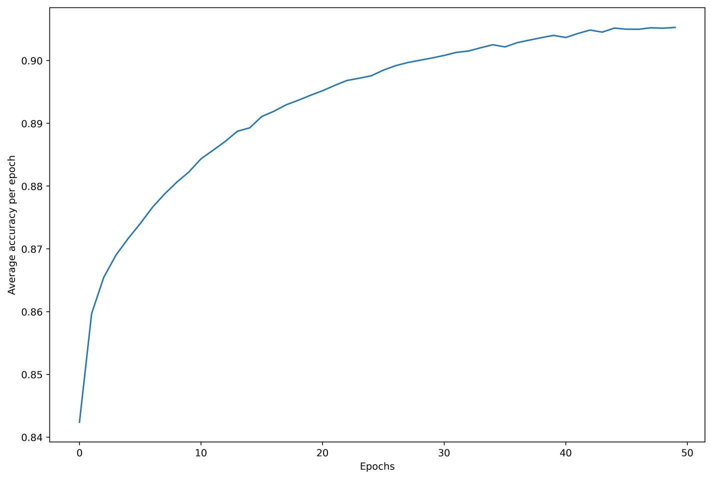
[7] TensorFlow Datasets, a collection of ready-to-use datasets. `https://www.tensorflow.org/datasets`.

[8] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners, 2021.

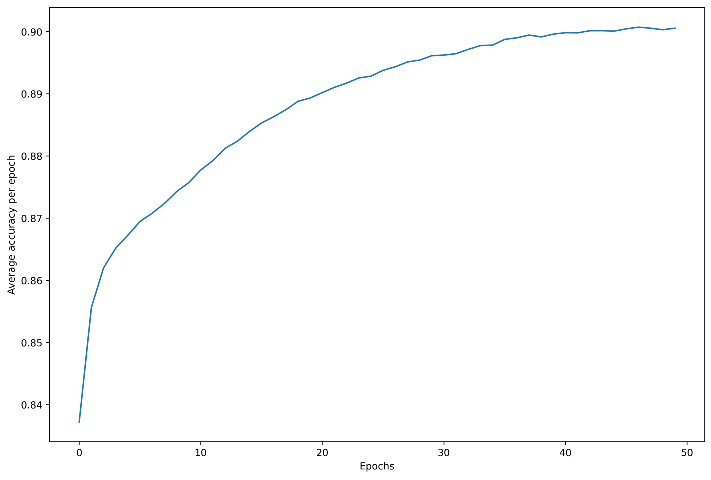
[9] Alex Krizhevsky. Learning multiple layers of features from tiny images. *Master's thesis, Department of Computer Science, University of Toronto*, pages 32–33, 2009.

[10] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

A Fine-tuning for different set of parameters

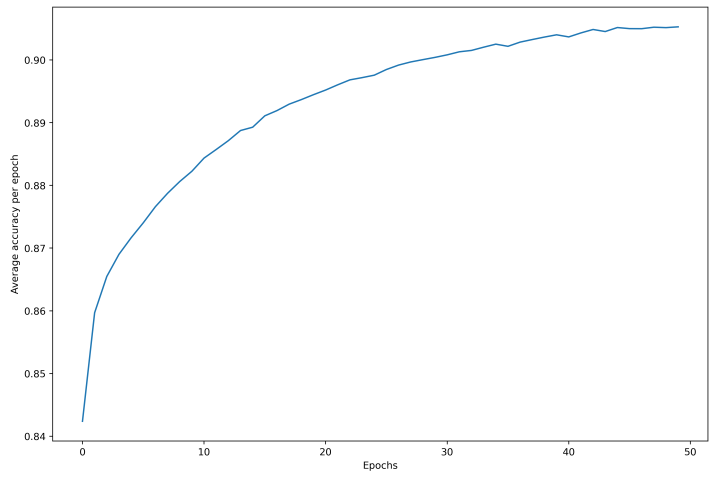


(a) Pre-training of the MAE for 100 epochs

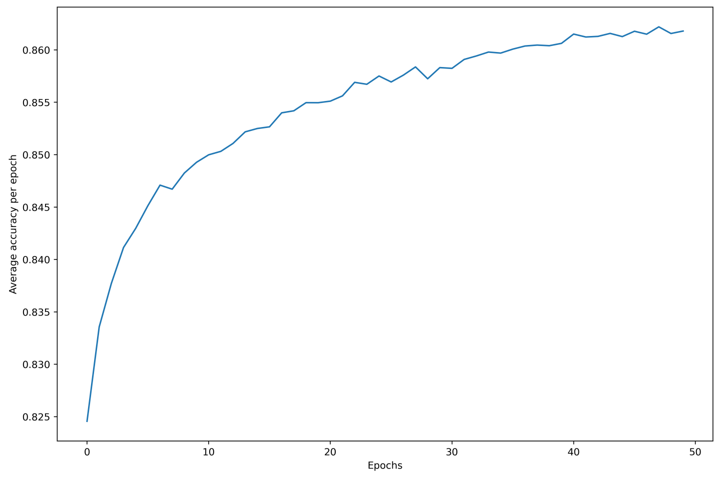


(b) Pre-training of the MAE for 1 000 epochs

Figure 9: Evolution of the classification accuracy with respect to the number of epochs during fine-tuning with the small MAE architecture and random sampling.

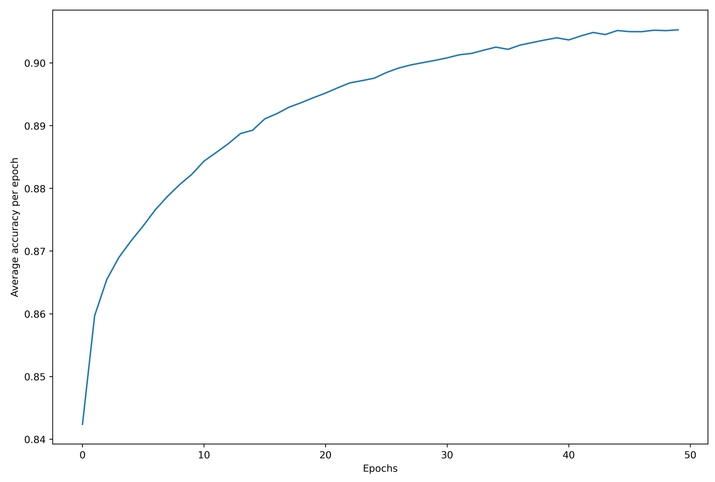


(a) Random sampling

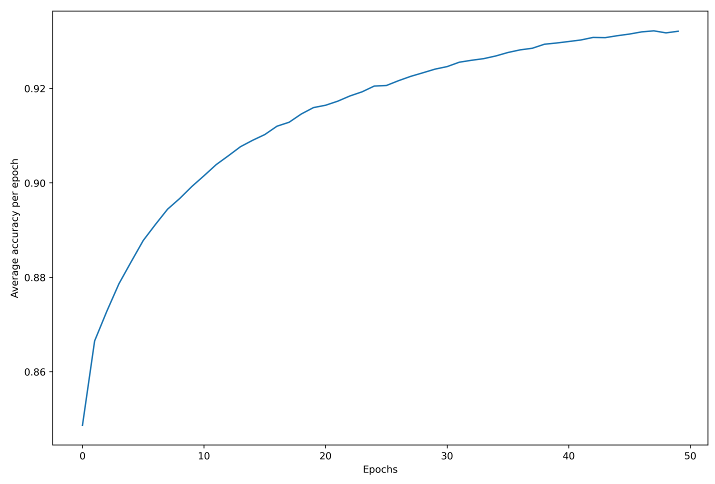


(b) Grid-wise sampling

Figure 10: Evolution of the classification accuracy with respect to the number of epochs during fine-tuning with the small architecture.



(a) Small architecture



(b) Medium architecture

Figure 11: Evolution of the classification accuracy with respect to the number of epochs during fine-tuning with random sampling.