E2: Unimodal Visual Representations

Prof. Dr. Anna Rohrbach,
Prof. Dr. Marcus Rohrbach,
Hector Garcia Rodriguez, M. Sc.
Jonas Henry Grebe, M. Sc.

TECHNISCHE UNIVERSITÄT DARMSTADT

Deadline Wednesday 28th May, 2025

General Information

In this exercise, you will gain hands-on experience with vision transformers (ViTs) and convolutional neural networks (ConvNets). We will explore the popular ImageNet Large Scale Visual Recognition Challenge 2012 (ILSVRC2012, also known simply as ImageNet).

To hand in your solutions to this exercise, you will have to submit the following:

- Your report as a PDF named `E2_GROUP_<YOUR_GROUP_NUMBER>.pdf` containing your answers to the question part and any additional explanations, descriptions, or discussions required for the coding part. **You must clearly indicate which answers belong to which tasks in your submitted report**. Parts of your solution that are not clearly linked to a task won't be graded.

  Please use LaTeX and the minimal TUDaReport template to compile a PDF for the submission in the official TU Darmstadt design. The university provides ShareLaTeX to students that you can use to collaborate.[1] We do not provide customized templates for every exercise sheet.

- Additionally, a Google Colab link (with edit access to everyone) to your copy of the provided notebook has to be submitted to Moodle. Ensure that the results mentioned in your report are reproducible from the submitted checkpoint of your notebook.

  Important: Before you share it, you have to save the checkpoint of your notebook, which gives it a time stamp and shows it in the revision history (`File` → `Save and checkpoint`), where you can rename it to `Final Submission (Group <YOUR_GROUP_NUMBER>)`. Every change after this submission checkpoint won't be graded. If there is no pinned revision that is marked as your submission, we will just grade the last existing checkpoint from before the deadline. *If you used Kaggle, please upload your notebook to Google Colab again to provide this link.*

In case of questions regarding the submission process, please use the **Admin Q&A** forum on Moodle.

Exercises

This exercise can achieve a maximum of 25 Points (Pts.) with up to 8 Pts. for the text and 17 Pts. for the coding part.

Question Part (8 Pts.)

Before and during your work on the coding part of this exercise, you are supposed to solve the following questions and tasks:

[1]You can adapt the `accentcolor` to our orange by replacing `9c` with `8b` in the `documentclass` command

1. ✍ **ConvNet vs. ViT** (3 Pts.) Compare the ConvNet and ViT architectures, highlighting their differences. What are the drawbacks of each of them? Note: here and in all questions, you can refer to the ResNet paper and the ViT paper, as the standard ConvNet and ViT definitions, respectively.

2. ✍ **Invariance and equivariance** (3 Pts.) $f(x)$ is equivariant under another function $g(x)$ iff $f(g(x)) = g(f(x))$ (i.e. output of f changes like its input), or invariant under $h(x)$ iff $f(h(x)) = f(x)$ (i.e. output of f is independent to changes that g causes on it's input). In the lectures, you were taught how ConvNets learn to be approximately invariant to translation. However, following the formal definition and without any network optimization, do Conv layer (i.e., `torch.nn.Conv2d`) or ViT blocks satisfy invariance and/or equivariance under spatial translation, when considered in isolation? What about the full ConvNet or ViT networks? Explain why. You can ignore image boundaries and patchification (e.g., by setting patch size 1). Discuss the role of absolute positional encodings on ViT network invariance and equivariance. If necessary, provide examples.

3. ✍ **Model components** (2 Pts.) Describe each component in a ConvNet and ViT layers and their contribution to the overall architecture. Component refers to any function of one or more embeddings that outputs another embedding. You can list some abstractions without needing to describe their subcomponents in detail, like normalization layers (e.g., LayerNorm, BatchNorm), parametrized modules (e.g., Multi-head self-attention, 2d-Conv layer), or composite activations (e.g., GELU).

Coding Part (17 Pts.)

We provide the code to run all the experiments in this Starter Kit. We recommend you use Kaggle or another provider with longer timeouts; some training runs here might last longer.

1. **ConvNet**:
   a) 💻 **ConvNet Implementation** (2 Pts.): Implement a multi-layer Convolutional Neural Network (ConvNet). Specifically, we will work with a small `ResNet` model. The class is mostly already provided except for the following parts that you have to implement:
      - **Constructor**: Some code is missing in the `__init__()` method of the provided `BasicBlock` class.
      - **Forward Pass**: You need to implement the logic in the `forward()` method on how the layers of this block are connected during a forward pass.

   b) **ConvNet Training** (2 Pts.):
      i. 💻 **Train and evaluate**: Training and evaluation code is also provided. You should run the training, which will output loss and accuracy curves on the training and validation sets. Train and validate your ResNet.
      ii. **Fix the evaluation**: You might observe that the training accuracy and loss are worse than or very close to those for validation. Generally, in a narrow task with limited data, you'd expect the performance on the training set to be larger than on the validation set. There is a mistake in the current way we evaluate our model on the training set, that is responsible for this result.
         A. ✍ Identify and explain the cause for the lower training accuracy in our code.
         B. 💻 Correct the mistake of the current evaluation that you observed in the previous subsection.

   c) **Experiment with order of operations in the ConvNet block** (2 Pts.): We instantiate two new blocks, `BasicBlockv2` and `BasicBlockv3`:
      i. 💻 Complete the `__init__()` and `forward()` methods of each of the two classes. In each case, the implementation should be meaningfully distinct and reasonable, obtained by modifying the order of operations in the original `BasicBlock`. The final accuracy should be similar. By operation here we mean the use of a function that outputs a modified embedding (e.g. addition of two embeddings or applying a `torch.nn` module). The constructor part of your module should be exactly the same as for your BasicBlock. If you need to modify it, you won't be granted full grade.
      ii. ✍ Write about your observations, drawing on your intuition and hypothesizing to explain the results.

2. **ViT**:

    a) 💻 **ViT Implementation** (2 Pts.): We have provided the skeleton for implementing a *Vision Transformer* (ViT). As mentioned before, you should follow the specific instantiation shown in the original ViT paper. The class is missing the following parts, which you must implement:

        i. **MLP**: Some code is missing in the `__init__()` and `forward()` methods of the provided `MLP` class.

        ii. **Self-Attention Block**: Some code is missing in the `__init__()` and `forward()` methods of the provided `Attention` class. Use multi-head attention. The only learnable module you must use is `nn.Linear`.

        iii. **ViT Block**: Some code is missing in the `forward()` method of the provided `Block` class. Follow the order of operations in the original ViT paper.

        iv. **ViT Network**: Some code is missing in the `forward()` method of the provided `ViT` class.

    b) **ViT Training** (2 Pts.):

        i. 💻 **Train and evaluate**: Use the same training and evaluation functions. Train and validate your ViT.

        ii. ✍ **Analyse the results**: Discuss and explain how the metrics for ViT compare to those you obtained for ConvNet. Are they better, worse, or similar? Why?

    c) **Experiment with order of operations in the ViT block** (3 Pts.): We instantiate two new blocks, `Blockv2` and `Blockv3`:

        i. 💻 Complete the `__init__()` and `forward()` methods of each of the two classes. In each case, the implementation should be meaningfully distinct and reasonable, by modifying the order of operations in the original `Block`. Operation here means the same as in Sec. 3.c)i. The validation accuracy of these two alternatives should be at most 10% lower (in absolute terms) than the original Block. Note you cannot add additional components to the constructor of `Blockv2` and `Blockv3` that wasn't used by `Block`. If any of your modifications obtains a better validation accuracy, you'll get a 1 Pts bonus.

        ii. ✍ Write about your observations, drawing on your intuition and hypothesizing to explain the results.

3. **ViT Enhancements** (4 Pts.):

    a) ✍ List all the hyperparameters of the ViT training setting that you expect would improve the validation accuracy. Explain why you believe each of these changes would help classification performance.

    b) 💻 Implement any number of these changes in order to improve the accuracy. The model must be trained on the same number of samples.

    c) ✍ Briefly write about the hyperparameter change(s) that you implemented, and how they changed accuracy.