ARTIFICIAL INTELLIGENCE (E016350A)
GHENT UNIVERSITY
2024/2025 AY

Professor:
Aleksandra Pizurica
Assistants:
Nicolas Vercheval

Project 1: Image Classification

1 Introduction

In this project, you will take the first steps toward becoming a data scientist. You will apply the deep learning theory learned in the lessons and engage in its practical aspects, including the model implementation, documentation, and validation. You will get hands-on experience with image classification using the Keras library and a standard benchmark dataset (Fashion MNIST).

First, you will deal with a standard classification task. You will then develop an autoencoder model in the second task and use it in the third task to pinpoint the limitations of your classifier and the data.

Each task will ask you to perform some steps indicated in the "Instruction" section. In the following section, we will ask you about your experience with the various steps.

This project is to be completed in groups as they are on Ufora. Each group has to submit a Python package with their code, the documentation and a concise report, see Section 6 for instructions.

Make sure to read very carefully the following:

- **Required dependencies 1.2** and check that they match the ones in your environment.
- **Assignment's guidelines 1.3** to save time and maximize your grade.
- **Considerations about the use of generative AI 5.1** to be aware of our current policy.
- **Submission instructions and remarks 6** to not lose grading points unnecessarily.

1.1 Data

The following tasks will use the Fashion Mnist dataset, which can be easily downloaded using the Keras API.

1.2 Dependencies

This project must be compatible with Python 3.12. You may freely use the following Python libraries:

1. numpy~=1.26.0
2. tensorflow>=2.16.0

3. scikit-learn~=1.5.0
4. pandas~=2.2.0
5. matplotlib~=3.9.0
6. seaborn~=0.13.0
7. imbalanced-learn~=0.12.0
8. pyaml~=24.9.0

If you wish to use a different library, you must send an email to the course mailing list explaining why you need to use it and obtain approval. Failure to run the code, due to version compatibility issues or other bugs, will result in a score of 0 for the code evaluation.

1.3 Guidelines

The success of this assignment relies not just on the performance of your model but mainly on the correct methodology for its development, evaluation, and reproducibility.

- You can find most of the necessary code in lab "Deep Learning", along with best practices and tips to help you prepare for this project.
- Organize your code logically. Use Python files (modules) to define functions and classes, and use Jupyter notebooks for prototyping, visualization, and testing. This approach ensures consistency in your model architecture across different experiments.
- Removing a validation dataset from a training dataset involves a compromise. If the remaining training dataset is too small, your model may overfit easily, which could lead you to apply excessive regularization. Conversely, if the validation dataset is too small, it won't provide reliable performance metrics. You might notice this effect in your initial model. If the validation curve in the history plot appears too "jumpy" for reliable assessment while the training curve is smooth, it likely indicates that your validation dataset is too small. In this case, consider adjusting its size and starting over.
- You can use the architectures in the lab "Deep Learning" as templates for your initial models. However, note that the dimensionality of the data used in the lab is different, so you will need to adapt these architectures to suit your data.
- Avoid spending excessive time tuning your model. Focus on implementing a robust optimization strategy rather than finding the absolute optimal values.
- The quality of your plots can significantly impact your final score. Ensure your plots are clear and include appropriate legends or labels. When plotting images, do so before applying normalization, and remember to revert any normalization applied to the input images (denormalization) when plotting reconstructed images.
- Provide clear and concise answers using the correct terminology presented in the theoretical materials and respect the number of sentences allowed for the question.

2 Task 1: Building the classifier

Your first task will involve using deep learning to address image classification. You will implement all the necessary steps to build and document a neural network classifier for the FashionMNIST dataset.

2.1 Instructions

Create a Jupyter notebook named `classification.ipynb` implementing the following steps (feel free to distribute the logic across different modules and import them into the notebook):

1. **Data exploration**:
   - Print the size of the dataset and the shape of the samples.
   - Show the distribution of the classes.
   - Plot a few samples from each class.
   - Select a classification metric, such as accuracy or another metric you find more suitable for the problem.
2. **Pre-processing**:
   - Set a random seed for NumPy and Keras.
   - Use the default splits for testing and training, and extract a validation dataset from the training set.
   - Normalize the data to ensure compatibility with the Keras neural networks interface.
3. **Define your initial model**:
   - Build a Keras classifier model and print its initial architecture by calling the model's `summary` method.
   - Choose a loss function for training the model and reasonable starting hyperparameter values.
4. **Train your model**:
   - Train your model for ten epochs using the training split of the FashionMnist dataset.
   - Ensure that the model runs without errors and that the loss decreases more or less smoothly.
   - Plot the model history containing its performance at each epoch.
5. **Hyperparameter tuning**:
   - Implement the Keras `EarlyStopping` callback.
   - Select two hyperparameters and conduct a rough parameter search to fine-tune your model.
   - Try adding a regularization scheme and see if it helps at all.

- Create a Pandas DataFrame file to report each run setup and performance metric and display its content.
- Save the DataFram as a CSV file and include it in your package.

6. **Test your final model**:
   - Retrain your model using the complete training dataset (including the validation dataset).
   - Test the model on the test dataset and save its weights to disk.
   - Create a YAML file containing the final values of the hyperparameters, the chosen loss and optimizer, and all information to reproduce your training strategy and include it in the package.
7. **Load configuration and rebuild model**:
   - Load the model's weights and verify that you can replicate the results.
   - Plot ten random samples, indicating the correct and predicted labels.

2.2 Questions

In your report, please answer the following questions precisely and concisely (one to three sentences for each). Optionally, include images from the notebook.

Question 1 *What challenging aspects do you anticipate in this dataset after exploring it?*

Question 2 *Which metric did you select for this task, and why?*

Question 3 *Based on the training and validation curves from the history plot of your initial model, what conclusions can you draw?*

Question 4 *Which hyperparameters did you optimize? Did you attempt all possible combinations of parameter values (grid search) or optimize them one at a time? If you tuned the hyperparameters sequentially, did you select the values to try upfront, or did you base them on the performance of previous values? Please explain and motivate your strategy.*

Question 5 *Did the regularization scheme you tried help? Provide a possible explanation for it.*

Question 6 *For how many epochs have you trained the final model? How did you determine the stopping criterion?*

3 Task 2: Encoding the dataset

In the second task, you will train an autoencoder model using the Fashion MNIST dataset.

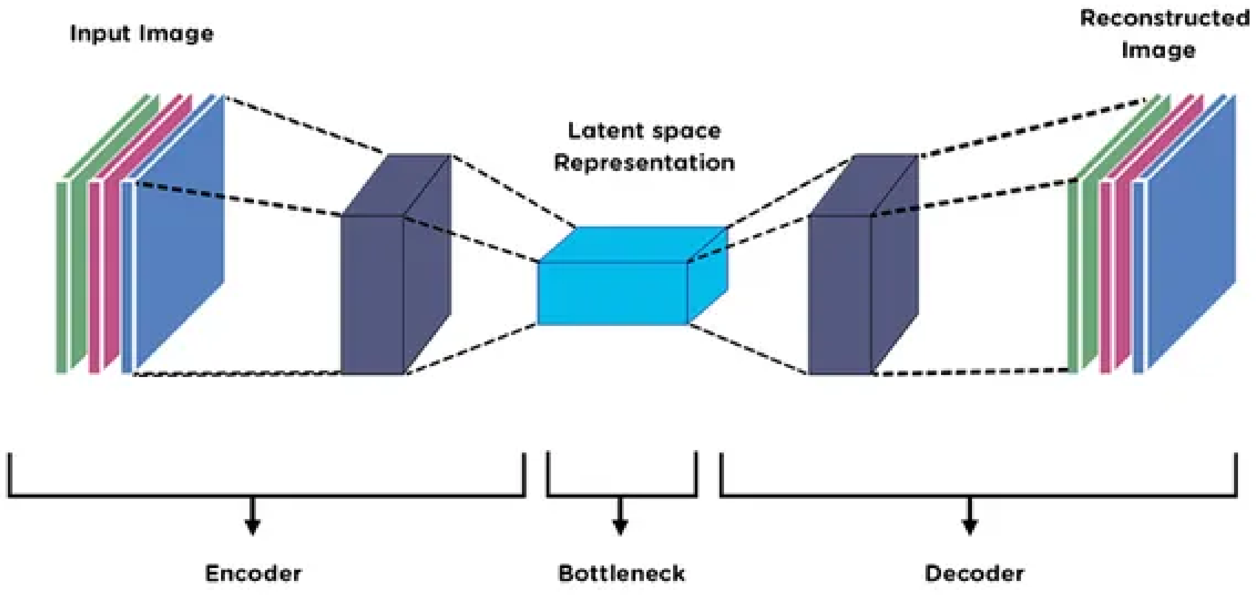

Figure 1: The characteristic shape of the autoencoder.

3.1 What is an autoencoder?

An autoencoder is an unsupervised learning model where the optimization strategy (*self-supervised learning*) mimics supervised learning. However, the model is trained without any labelled data. Instead of using input-output examples $(\mathbf{x}^{(1)}, y^{(1)}), \ldots (\mathbf{x}^{(N)}, y^{(N)})$ for training, the model learns to "encode" the inputs $\mathbf{x}^{(1)}, \ldots \mathbf{x}^{(N)}$ into their latent (typically lower-dimensional) representations $\mathbf{h}^{(1)}, \ldots \mathbf{h}^{(N)}$ which are "decoded" to yield the reconstructed versions of the inputs $\hat{\mathbf{x}}^{(1)}, \ldots \hat{\mathbf{x}}^{(N)}$. In this process, the autoencoder learns to compress the essential information about each input $\mathbf{x}^{(i)}$ into the latent representation $\mathbf{h}^{(i)}$, i.e., it learns itself to extract the important features from the inputs, hence the name self-supervision.

In other words, the primary goal of autoencoders is to compress the dataset into a lower-dimensional space referred to as latent space, which encapsulates the semantics underlying the dataset. Its main parts are:

- **Encoder**: The part of the model that takes the input and generates the encodings.
- **Decoder**: The part of the model that reconstructs the input from the encodings.
- **Bottleneck**: The compression of the feature space that forces it to learn the dataset.

3.2 Instructions

Create a Jupyter notebook file named `encoding.ipynb` that implements the following steps:

1. **Pre-processing and selecting the metric**:
   - Pre-process the dataset similarly as in the previous task.

- Use the same random seed as in the previous task for Keras and NumPy.
- Choose a reconstruction metric, such as Mean Squared Error (MSE), which will also serve as your loss function.

2. **Train your initial model**:
   - Build an autoencoder model and print its initial architecture.
   - Start with a latent space of dimensions $4 \times 4 \times 32$, where the first two dimensions are spatial and the last one is the feature dimension
   - Train your initial model for ten epochs and visualize the evolution of the training and validation performance.

3. **Optimize your model**:
   - Tune one or two hyperparameters until you can only make a marginal improvement.
   - Train the model for ten epochs and record the validation performance.
   - Select a few samples from the validation dataset and display their reconstructions.

4. **Decrease the latent space**:
   - Reduce the encoder's output feature dimension by 4 and repeat step 3, tuning the hyperparameters only when necessary.
   - Continue reducing the latent space and repeating step 3 until the validation reconstruction metric shows a significant degradation.
   - Plot the recorded metric values to determine when reducing the latent space resulted in a significant performance drop.

5. **Final training**:
   - Select the dimension of the latent space before the performance drop.
   - Train your model for thirty epochs on the entire training dataset
   - Save the model weights in the parent directory of your package.

6. **Testing the autoencoder**:
   - Test the final model on the test dataset.
   - Randomly select some test samples and display their reconstructions.

3.3 Questions

In your report, please answer the following questions precisely and concisely (two to four sentences for each). Include the necessary images from the notebook.

Question 7 *Select a validation sample and show its reconstructions when changing the latent space sizes. Describe whether and how the reconstruction quality (as you perceive it) decreases with more compression.*

Question 8 *Show a few representative examples of test inputs and their reconstructed images. Comment on the visual quality of the reconstruction of the final autoencoder.*

4 Task 3: Explaining the classifier

In this final task, you will use the models developed earlier to explain the behaviour of the classifier.

4.1 Local examples and counterexamples

It is difficult to determine if a model classifies an image for the same reasons as a human based solely on classifier evaluation. One way to provide more context for its decision is to present additional samples and observe whether the model evaluation changes. To stress the main reason for the change, these additional samples should be related to the initial one in a meaningful way, that is, they are semantically close. A possible way of establishing a semantic similarity distance between samples is to leverage the Euclidean distance in the autoencoder's latent space. More concretely, two samples are considered semantically close if their encodings are close in the latent space. This semantic similarity can explain any misclassified samples from a classifier.

For a sample from the test dataset, we define its:

- local example as the training sample with the same label that is semantically closest.
- local counterexample as the training sample with a different label that is semantically closest.

4.2 Causes of misclassification

For this task, we consider six possible reasons for a misclassification:

- **Noise** if not even you can classify the sample.
- **Mislabeling** if you suspect the label is incorrect.
- **Underrepresentation** if the sample appears different from its local example.
- **Overfitting** if the sample resembles its local counterexample.
- **Underfitting** if the sample resembles its local example.
- **Overlapping labels** if the sample resembles both its local example and counterexample.

4.3 Instructions

Create a notebook named `model_explainability.ipynb` implementing the following steps:

1. **Pre-processing and compressing the dataset**:
   - Pre-process the dataset similarly to the previous tasks.
   - Use the same random seed as in the previous tasks for Keras and NumPy.
   - Load the trained encoder from the final autoencoder model.
   - Encode the entire training dataset and store it in a variable.
2. **Find the nearest neighbors**:

- Select and encode a random sample from the test dataset.
- Build a k-nearest neighbors model using the `NearestNeighbors` class from the `sklearn.neighbors` module with k set to ten.
- Find the ten training samples semantically closest to the test sample according to the criterion explained in Section 4.1.
- Plot the test and the training samples and display their corresponding labels.

3. **Diagnose the classifier**:
   - Load the final version of your classifier and evaluate the test dataset.
   - Extract 25 misclassified samples from the test dataset.
   - For each of these samples, identify the local example and the local counterexample.
   - Write a CSV file in which, for each misclassified sample, you note one or more likely causes of misclassification as specified in Section 4.2.
   - Load the CSV into a pandas DataFrame and display the distribution of the most likely causes of misclassification.

4.4 Questions

In your report, please answer the following question precisely and concisely (four to seven sentences). Include the necessary images from the notebook.

Question 9 *Considering the sources of misclassification, along with the performance of your model on the training test dataset, explain the limitations of your model regarding data quality, overfitting, and underfitting. Select four examples of misclassification that highlight these underlying issues and plot them. Use local examples and counterexamples to illustrate your reasoning.*

5 Use of generative AI and collaboration within the group

Your report must include two separate sections at the end devoted to the use of generative AI and the collaboration within the group. These should be organised as follows:

- **Use of generative AI**: Explain whether you used generative AI tools and if so in what way. Be as precise as possible without separately naming each instance of the same type. For example, you can write "I used generative AI to identify and fix bugs in my code for all the tasks" or where needed to make a differentiation refer to the specific task(s). Similar for plots and the text.

- **Collaboration and task distribution within the group**: Describe here how you collaborated on the project and specify for each group member what his or her concrete contributions were. Also, describe how you ensured that the project was efficiently executed by splitting the tasks while guaranteeing that every group member understands thoroughly and approves all the code in the project and all the answers in the report. **This is very important because each group member bears the responsibility for all the parts of the submitted code and the report, thus also for possible misconduct in the parts assigned to the other partners in the group.**

5.1 Use of Generative AI

While generative AI is allowed when completing this project, you should be able to easily succeed without relying on it. Before using generative AI, consider that:

- The "Deep Learning" lab already provides most of the resources you need to write the code for the tasks.
- The libraries for the projects are standard and already have curated tutorials.
- Your answers should follow your personal experience while performing these tasks.

Be mindful of excessive reliance on generative AI:

- **For coding**: it may result in overly complex solutions, possibly unsuitable for the data provided.
- **When answering questions**: it could lead to excessively detailed responses and overlook critical insights from your task outcomes.

We recommend using generative AI thoughtfully and sparingly and stress that we will heavily penalize:

- Code outside the project scope.
- Long answers that are not based on your own observations.

6 Submission

6.1 Submission files

You need to submit two separate files:

- A zipped package containing your code and the documentation (CSV and YAML files). Do not include the dataset or the model weights.
- A report in PDF format answering our questions. Each question must be answered separately, clearly indicating the question number. At the end of the report, do not forget to include the sections on the use of generative AI and collaboration within the group.

Please name your PDF report using the following format:

`ReportP2_group#Number_FamilyName1_FamilyName2.pdf`
(e.g. `ReportP2_group21_Smith_Wilson.pdf`).

6.2 Submission instructions

- **How to submit**: Upload the submission files to the Ufora course page via Ufora-tools → Assignments → Project 1: Image Classification.
- **Deadline**: 25/11/2024, 23:59 CET.

6.3 Important remarks

- The two files must be submitted separately (do not submit a zip file).
- If the code does not run in the environment described in Section 1.2, it will receive a score of 0 points.
- If your code is hard to read and understand, it will cost you some points. Consider adding comments in any passage that is not obvious or standard (not everywhere).
- For any technical questions, please post them on the Ufora forum (Ufora → Ufora-tools → Discussions → Project 1: Image Classification). Anonymous questions are permitted.
- We will look for possible plagiarism. If we detect any, you will have severe repercussions. You can share advice, not the code or the answers.