ARTIFICIAL INTELLIGENCE (E016350B)
GHENT UNIVERSITY
2024/2025 AY

Prof. Aleksandra Pizurica
Assistants: Nicolas Vercheval
Ide Van den Borre
Roel Huysentruyt

Project: Medical Image Classification

1 Overview

In this project, you will take the first steps toward becoming a data scientist. Not only will you apply the deep learning theory learned in the lessons, but you will also practice the implementation, documentation, and validation of your model.

The project deals with medical image classification. It consists of four parts, each corresponding to a different task: pre-processing and data exploration, developing a baseline model, fine-tuning a pre-trained model, and evaluating and describing the models' performance and behavior. Under each task, some concrete steps and questions are specified to guide your development, analysis, and report organization.

The report should be written in the form of a scientific article, with sections corresponding to the four specified tasks, and completed with an introductory section and conclusions. You may include additional sections and references. The report should not exceed 15 pages (double column, font size 12), including figures and references. The recommended LaTeX template is provided (see Section 5).

Pay attention to the dependencies that your code should meet (see Section 2) and to the guidelines for a successful project (Section 3).

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 the report, see Section 8 for instructions.

2 Dependencies

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

1. numpy~=2.0.2
2. tensorflow>=2.18.0
3. scikit-learn~=1.6.1
4. pandas~=2.2.3
5. matplotlib~=3.10.0
6. seaborn~=0.13.2

7. imbalanced-learn~=0.13.0
8. pyaml~=25.1.0

We recommend importing Keras as a submodule of TensorFlow. If you wish to use a different library from the one listed, you must email 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.

3 Guidelines

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

- Familiarize yourself with the Keras library and with the lab "Lab 3", containing best practices and tips for developing a deep learning model. You can use the model architectures from the notebooks as templates for your (initial) model. 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.
- 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.
- Ensure your plots are clear and include appropriate legends and units. Images should be large enough to be easily observable and accompanied by the original label and, during the evaluation phase, their prediction/evaluation.
- Avoid spending excessive time tuning the hyperparameters. Focus on implementing a robust search strategy rather than finding the optimal values.
- If the best value for a hyperparameter is at a boundary of your search range, extend the search range accordingly until you are confident that it contains the best value.
- Ensure your model is re-initialized between runs during hyperparameter tuning. The easiest way is to instantiate a new model for each run.
- When tuning the hyperparameters, do not optimize the number of training epochs using a search. Instead, adopt early stopping to find the best value according to the validation dataset. When training the final model, use as many training epochs as when holding out the validation dataset. Optionally, train for a few more epochs. Do not use early stopping for the final model.
- The test dataset should be evaluated once for the baseline and once for the fine-tuned model. If the results differ significantly from the validation dataset, there is probably a bug in your pipeline. If no bugs are present, the test performance is final.
- In your report, use a succinct technical style, with clear answers to the key questions that are posed under each task. Use correct terminology and careful notation as presented in the theory materials.

4 COVID-19 assisted diagnosis

4.1 Introduction

The goal of the project is to build and evaluate a deep learning-based classifier to automatically detect COVID-19-positive cases from a **medical image dataset** of real chest X-ray images.

4.2 Data

The data for this project comes from Kaggle's open-source COVID-19 Radiography Database. The accompanying `covid_xray.zip` file contains the training, validation, and test sets already organized in separate folders and categorizes the images into normal and COVID-positive cases. COVID-19 primarily affects the respiratory system, leading to pneumonia and acute lung injury, which can be identified in chest X-rays through characteristic abnormalities. These radiographic patterns help differentiate COVID-19 pneumonia from other respiratory infections, although additional clinical context and confirmatory tests are often required for an accurate diagnosis. Figure 1 shows typical radiographic signs of COVID lesions visible on X-ray imaging.

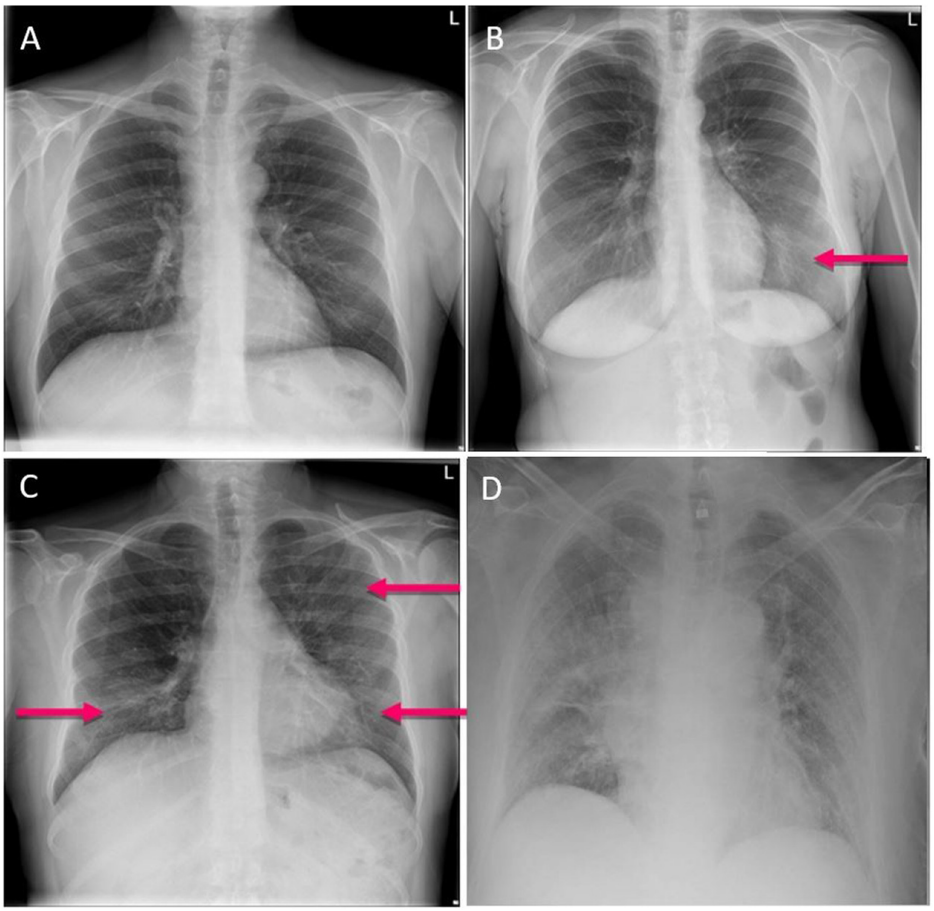

Figure 1: A) Normal. B) Mild, showing alveolar–interstitial opacities in a single lung field. C) Moderate, showing patchy bilateral alveolar–interstitial opacities. D) Serious, showing extensive bilateral alveolar consolidations.

4.3 Task 1: Data Exploration, Preprocessing and Augmentation

1. **Loading the data**
    - Set up your Image Generators for the training, validation, and test data.
    - Load the datasets as arrays.
2. **Data Exploration**
    - Print the training dataset size.
    - Check whether the images always have the same size.
    - Show the distribution of two classes present in the dataset.
    - Plot a few samples from each class with the corresponding label (COVID/normal).
    - Display some statistics on the pixel intensities (e.g., pixel average, global average, pixel standard deviation, global standard deviation, ...)
    - Repeat the previous steps for the validation and test set:
3. **Pre-processing**
    - Optionally, downsample the images by rescaling their size to 128×128 pixels.
    - Implement a normalization strategy, using fixed values, the sample statistics, or the statistics of the training datasets, and test it on a sample from the validation dataset.
4. **Augmentation**
    - Implement three different relevant types of data augmentation.
    - Visualize the transformations on a sample from the training dataset individually and combined.
5. **Pipeline**
    - Implement the pipelines performing pre-processing and augmentation (only for the training dataset) for your next tasks.

4.3.1 Questions

Question 1 *In what aspects is this dataset challenging for automatic classification in the designated categories?*

Question 2 *According to the computed pixel statistics and label distribution, do the training, validation, and test datasets appear uniformly divided?*

Question 3 *Are there any noticeable differences in image quality between the COVID and normal X-ray images? Could variations in intensity distribution or artifacts affect model performance?*

Question 4 *If not accounted for, the original resolution may lead to unnecessarily extended training time and memory usage when developing the classifiers, without any performance benefit. How do you intend to deal with it? Is downsampling a good solution, or would it lead to significant information loss? Do you have a different approach?*

Question 5 *Did you normalize the images using fixed values, dataset statistics, or sample statistics? Did you normalize along the spatial dimensions or normalize each pixel differently? Assuming a standard convolutional neural network model, which strategy would give the best discriminative features?*

Question 6 *Do the augmented images still represent realistic X-ray variations, or could excessive transformations introduce unrealistic patterns? How can we ensure our model generalizes well to unseen data?*

4.4 Task 2: Building the baseline model

In this task, you will build and train a classification model from scratch. This will serve as a baseline to compare with in the following tasks.

Create a Jupyter notebook named `baseline.ipynb`, which implements the following steps (you may distribute the logic across different modules and import them into the notebook):

4.4.1 Instructions

1. **Set up the classification**:
    - Choose one or metrics to evaluate the success of your classifier.
    - Set a random seed for NumPy and Keras.
2. **Build your initial model**:
    - Build a Keras classifier model with a convolutional part followed by a fully connected part. The convolutional part must have convolutional layers, but you can also include pooling or batch normalization layers. The fully connected part must have fully connected layers and one or more Dropout layers.
    - Print the initial architecture by calling the model's `summary` method.
3. **Train your model**:
    - Choose a loss function for training the model and reasonable starting hyperparameter values.
    - Train your initial model on the COVID dataset for thirty epochs while holding out the validation set to monitor its performance.
    - 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.
4. **Hyperparameter tuning**:
    - Implement the Keras `EarlyStopping` callback.
    - Select four hyperparameters, one of which must be the dropout rate(s), and conduct a rough parameter search to tune your model. Optionally, instead of one hyperparameter, try other losses or optimizers.

- Create a Pandas DataFrame file to report each run setup and performance metric(s), save it as a CSV file, and include it in your package.

5. **Your final baseline**:

   - Retrain your model using the complete training dataset (including the validation dataset).
   - Plot the training curve.
   - Save the model to disk and load it back.
   - Test the model on the test dataset and print the performance metric(s) you chose for this problem.
   - Display the confusion matrix.
   - Plot a few samples from the test dataset (without normalization) with their evaluations (after normalization).

4.5 Questions

Question 7 *Which metrics did you find appropriate for this task, and why?*

Question 8 *What conclusions can you draw from the training and validation curves on the history plot of your initial model? Which hyperparameters would need to be optimized first?*

Question 9 *Did you observe signs of overfitting in your initial model? If so, which countermeasures against overfitting did you take?*

Question 10 *Which hyperparameter search strategy did you employ? 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?*

Question 11 *Which optimizer and which loss function did you select for this problem? Did you try others? How did you compare them?*

Question 12 *What conclusions can you draw from the training curves on the history plot of your final model?*

4.6 Task 3: Transfer Learning

In this task, you will apply **transfer learning** using a pre-trained ResNet50V2 model trained on ImageNet.

Transfer learning is a powerful machine learning technique for leveraging knowledge learned from one task and applying it to a new task. For instance, if a model has been trained to recognize raccoons, the features it has learned (such as shapes, textures, or patterns) can be useful for identifying tanukis, animals with similar visual characteristics. This approach is particularly valuable when the available dataset for the new task is limited in size, making it difficult to train a full-scale model from scratch. The typical workflow for transfer learning in deep learning involves several key steps:

1. **Starting with a pretrained model:** The architecture of a model trained on a large dataset is (partially) replicated, and the weights of some of its layers are imported.
2. **Freezing the pretrained layers:** These layers are frozen (i.e., excluded from the back-propagation update) to prevent the information learned from the initial task from being overwritten during the new training process.
3. **Adding new trainable layers:** These layers are introduced on top of the frozen ones and are specifically designed to adapt the previous knowledge to the new task.
4. **Training on the new dataset:** The model with the new layers is then trained using data from the current task.
5. **Fine-tuning (optional):** In some cases, the entire model (or parts of it) is unfrozen and re-trained with a low learning rate. This fine-tuning allows the model to adapt more precisely to the new dataset, often resulting in improved performance.

You will use pre-learned features from models trained on a large-scale image recognition dataset, study the effect of transfer learning on model performance and training time, and compare it to the baseline CNN model you trained from scratch in the previous section. For more information on transfer learning using Keras, we refer you to the Keras guidelines.

4.6.1 Instructions

1. **Set up the classification**:
   - Choose the same metric(s) for this task as you did in the previous one.
   - Set a random seed for NumPy and Keras.
2. **Setup the base model**:
   - Instantiate the model with the "imagenet" pre-trained weights using the dedicated class. Do not include the final fully connected layers. Note: in the documentation, the "top" layers of the network are the last layers evaluated in the forward pass.
   - Freeze all layers in the base model by setting `trainable = False`.
   - Add one or more fully connected layers from the base model to fit the classification task.
   - Add one Dropout layer before the last layers. Optionally, add more.
   - Print the final architecture.
3. **Train your model**:
   - For the initial model, start with the best hyperparameters found while optimizing the baseline (when applicable).
   - Train your initial model on the COVID dataset for thirty epochs while holding out the validation set to monitor its performance.
   - Ensure that the model runs without errors and that the loss decreases more or less smoothly.

- Plot the training and validation curves.

4. **Hyperparameter tuning**:
   - Implement the `EarlyStopping` callback.
   - Tune the following hyperparameters: batch size, learning rate, and dropout rate, and conduct a rough parameter search.
   - Create a Pandas DataFrame file to report each run setup and performance metric, save it as a CSV file, and include it in your package.
5. **Train your pre-trained model**:
   - Retrain your model using the complete training dataset (including the validation dataset).
   - Plot the training curve.
   - Save and reload your model.
6. **Fine-tuning of the entire model**:
   - Unfreeze all or part of the base model and retrain the whole model end-to-end (including validation data).
   - If you notice a sudden drop in performance on the **training** dataset, reload the model and try again with a smaller learning rate. If there is no sign of improvement, skip this step and use the saved model instead. Do not use the test dataset in this step.
   - Save your model in a different file and reload it.
   - Test the model on the test dataset and print the performance metric(s) you chose for this problem.
   - Display the confusion matrix.
   - Plot a few samples from the test dataset (without pre-processing) with their evaluations (after pre-processing).

4.7 Questions

Question 13 *How did you choose the architecture for the final layers you added to the pretrained model?*

Question 14 *How did the pre-trained model's training curve compare to the final baseline model's from Task 4.4?*

Question 15 *Did fine-tuning the entire model improve model performance? How small did you have to make your learning rate to avoid a drop in performance?*

Question 16 *How did your fine-tuned model's performance compare to the final baseline model's from Task 4.4?*

Question 17 *Based on this experience, what are the advantages and disadvantages of transfer learning?*

Question 18 *What could be done differently from the proposed transfer learning approach to strengthen its effectiveness?*

4.8 Task 4: Explainability through Grad-CAM

To help understand the decisions made by our COVID-19 classifier, you will incorporate Gradient-weighted Class Activation Mapping (Grad-CAM) into the analysis. Grad-CAM is a visualization technique highlighting the regions in an input image that contribute to the model's decision. It works by computing the gradients of the predicted class score concerning the final convolutional layer, using these gradients to weight the feature maps, and generating a heatmap that is then overlaid on the original image.

Applying Grad-CAM to our chest X-ray classifier is particularly useful for several reasons. First, it enables explainability by revealing which lung regions the model relies on for classification. This is essential in medical imaging, where transparency is crucial for trust and clinical validation. Additionally, it ensures that the model focuses on medically relevant areas, such as ground-glass opacities and consolidations, rather than irrelevant features like image artifacts or text labels. Grad-CAM can also help identify potential biases in the model and serve as a debugging tool, ensuring that the learned representations align with established clinical knowledge.

To integrate Grad-CAM into your analysis, you will generate heatmaps for normal and COVID-positive chest X-rays and overlay them on the original images. By analyzing these visualizations, you can verify whether the highlighted areas correspond to known COVID-19 manifestations, compare the activation patterns between normal and affected lungs.

In this task, you are going to implement Grad-CAM from scratch. As this requires more advanced TensorFlow features, we recommend you follow the Keras tutorial. Note that this tutorial assumes a model that outputs a value for each class. If your model outputs only a scalar, as is usually the case with binary classification, you must account for it in the implementation.

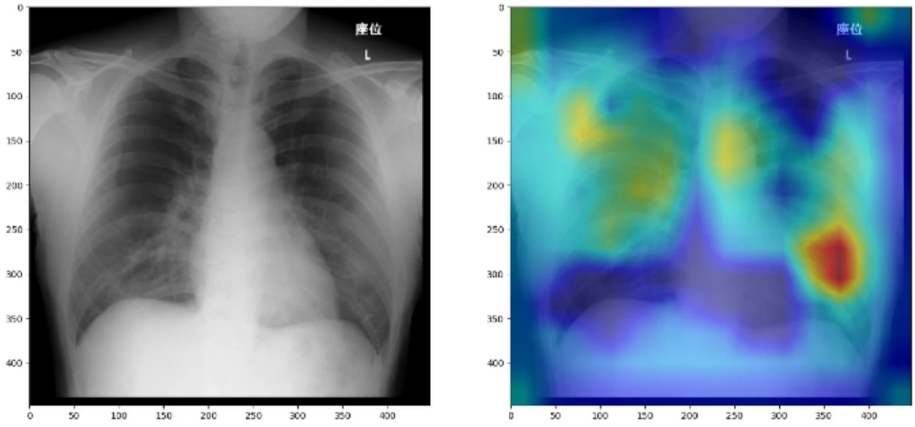

Figure 2: Example of Grad-CAM applied to a COVID-positive X-ray image, highlighting important regions for the model's decision.

4.8.1 Instructions

1. **Grad-CAM implementation**:
    - Load your best-performing model according to test performance.
    - Remove the final activation if present.
    - Select a test sample and pre-process it.

- Evaluate it and obtain the last convolutional layer's output.
- Compute the gradients of the predicted class with respect to this feature map (adjust for binary classification).
- Average the gradients along the spatial dimension to get a proxy for the channel importance.
- Obtain a heatmap by averaging the activations along the channel dimension, weighting each channel according to its importance (i.e., use the averaged gradients from the previous step as weights).
- Normalize the heatmap between 0 and 1.
- Map the heatmap onto the original sample space, accounting for the image size and value range.
- Overlay the heatmap on the original image, by adding to its values rescaled by 0.4, and plot it.

2. **Explaining the decision with Grad-CAM**:

   - Divide the test dataset into misclassified samples and break it further down by label, getting four distinct groups.
   - Extract a few samples from each group and visualize their Grad-CAM explanations.

4.8.2 Questions

Question 19 *Did you have to modify the algorithm to account for a scalar output? If so, how did you modify it?*

Question 20 *Do you think that having only two class labels affects the explanation? If you do, how so?*

Question 21 *Do the Grad-CAM visualizations highlight medically relevant areas in the X-ray, such as lung opacities or other features indicative of COVID-19?*

Question 22 *Did Grad-CAM visualizations help you identify potential biases in the model?*

Question 23 *Was Grad-CAM useful for understanding the model's misclassifications?*

Question 24 *Consider a scenario in which you can collect additional data and retrain the model. Would you change anything based on the Grad-CAM explanations?*

5 Report

Your report should be written in the style of a two-column article with a font size of 12, not exceeding 15 pages, including figures and references. Please use the LaTeX and Word templates available on the Ufora Content section dedicated to the project.

The report must include:

- Introduction
- Sections corresponding to each of the specified tasks
- Conclusions
- Author contributions and collaboration (see description further on)
- Use of generative AI (see description further on)

Additional sections such as Discussion and Background may be included, provided the total page limit is respected. Ensure that the report answers all questions provided in the tasks. Use a technical writing style (concise, precise, and with correct terminology/notation) and follow a professional layout. Avoid verbose and vague descriptions—write to the point with careful attention to detail.

6 Author contributions and collaboration

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 sections 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.**

7 Use of Generative AI

Explain whether you used generative AI tools and if so, in what way. Specify which task(s) or section(s) were affected. For example, you can write "We used generative AI to identify and fix bugs in our code for all the tasks" or, "We used generative AI to improve the text of Section 5 and Section 6.". Similarly, indicate if generative AI was used to generate code for images and plots, to generate templates for (parts of) the code or for drafting sections in the report.

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 "Lab 3" lab already provides good templates for deep learning code.
- The libraries for the projects are standard and already have curated tutorials.
- Your report should include your personal experience with the 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 drafting your report**: it could lead to generic sentences that overlook critical insights from your task outcomes.

We recommend using generative AI thoughtfully and sparingly. We will heavily penalize:

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

Furthermore, while generating code to plot real data is permitted, **generating synthetic data**, such as images or plots, even for demonstrative purposes, **is not allowed**.

8 Submission

8.1 Submission Files

Submit two separate files:

- A zipped package containing your code and documentation (the CSV files). Exclude the dataset and model weights.
- A PDF report, 10 to 15 pages, formatted as an article using the provided template.

Name your PDF report `Report_group{#Number}.pdf` (e.g., `Report_group21.pdf`).

8.2 Submission Instructions

- **Submission Method**: Upload files to Ufora via Ufora-tools → Assignments → Project: COVID-19 Assisted Diagnosis.
- Submit the two files separately (do not submit a single zip file).
- **Deadline**: 25/04/2025, 23:59 CET.

8.3 Evaluation Criteria

The final grade is based on (in order of importance):

1. Methodology.
2. Report quality.
3. Code quality.
4. Model performance.

Important Remarks:

- Data leakage (directly or indirectly using test data information in training) results in a zero for methodology.
- Report quality includes cohesion, accurate internal references (e.g., to sections or plots), and overall consistency.
- Poor English, grammatical errors, and unclear sentences lower the report quality score.
- The presence and quality of plots and images significantly impact the report score.
- Code that fails to run in the environment defined in Section 2 receives a zero.
- References to external sources are optional. Use them sparingly, primarily for context. While demonstrating familiarity with related academic works may improve your report's quality, we discourage devoting too much time to it. In particular, we do not expect you to acquire deep medical knowledge or reproduce a state-of-the-art model.
- Code readability affects the score. Add comments for non-obvious or non-standard sections (not for every line).
- Generative AI is allowed, but its misuse will lower your score (see Section 7).
- Plagiarism will have repercussions beyond this project and course. Share advice with other groups, not code or the report.

For technical queries, use the Ufora forum (Ufora → Ufora-tools → Discussions → Project 1: Image Classification). Anonymous questions are allowed.