CSC2042S 2025
Assignment 2
Perceptron
Image Classification

Total marks: 30

In this assignment you will extend the perceptron from binary to multi-class classification, using one-vs-rest classification, as discussed in class. You will apply your multi-class perceptron to the Simpsons-MNIST dataset. It is available on Amathuba under Resources > Datasets > A2-dataset or you could download it from here: https://github.com/alvarobartt/simpsons-mnist, where you will also find a detailed description of the dataset.

The dataset is structured similarly to the original MNIST digit dataset, but the images depict 10 different Simpsons characters rather than handwritten digits. Each class corresponds to a single character: *Bart Simpson, Charles Montgomery Burns, Homer Simpson, Krusty the Clown, Lisa Simpson, Marge Simpson, Milhouse Van Houten, Moe Szyslak, Ned Flanders, and Principal Skinner*.

Images are provided in both grayscale and RGB formats. Each format comes pre-divided into train and test folders, but for model development you will also need to split the training data into a separate training and validation subset. This allows you to evaluate hyperparameters and prevent overfitting. Since performance can differ substantially depending on whether RGB or grayscale images are used, you will train and evaluate two models: one operating on grayscale inputs and one on RGB inputs.

Tasks

1. Data processing (3 marks)

Start by implementing your code to load the Simpsons-MNIST data in both grayscale and RGB formats. The dataset is stored as JPEG image files and you have to load it into numpy arrays using the Pillow PIL Image package. Consult the package documentation to determine how to do this: https://pillow.readthedocs.io/en/stable/reference/Image.html.

Next, create a training/validation/test split by dividing the provided training set into a smaller training set and a held-out validation set, while leaving the test set untouched. Prepare the data in a form suitable for the perceptron: each image should be flattened into a vector, with optional normalisation or scaling applied (more about this below).

2. Multi-class perceptron implementation (5 marks)

You are required to use an object-oriented design. Implement at least the following two classes:

- BinaryPerceptron, with a method called **`predict`** that outputs either 0 or 1.
- MultiClassPerceptron, which uses a one-vs-rest approach by using 10 binary perceptrons to extend the perceptron to 10-class classification (as discussed in class). It should have a method called **`predict`** that outputs either {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}, where each number corresponds to a Simpsons character.

The multi-class perceptron must be implemented from scratch using a one-vs-rest approach. The binary version should implement the perceptron learning rule and a predict function returning a binary label. The multi-class version should maintain one binary perceptron per class and predict by selecting the class with the highest score. **This must be implemented using only NumPy (no machine learning libraries such as scikit-learn or PyTorch).**

3. Training (4 marks)

Implement a perceptron training loop. At each epoch, cycle through the training data and update the weights using the perceptron update rule, one image at a time. Shuffle the order of examples between epochs.

The perceptron does not always converge, especially when data are not linearly separable. Investigate at least two stopping criteria discussed in class (such as a fixed number of epochs, an error threshold, or early stopping on validation accuracy). Apply them to your models and briefly compare their effectiveness.

4. Hyperparameter tuning (4 marks)

Train and tune two models: one using RGB inputs and one using grayscale inputs. For each, search for a combination of hyperparameters that yields the best performance. You should consider the following factors:

- The learning rate.
- The initialisation strategy for weights (zero / constant / uniform / Gaussian).
- The normalisation technique applied to the input images (no normalisation / normalised between 0 and 1 / z-score normalisation).

These settings affect performance and training stability/convergence. Report which settings you found most effective for each modality (RGB and grayscale), and how they influenced training.

NOTE: You don't have to tune hyperparameters separately for each binary perceptron in a one-vs-many perceptron. For one multiclass perceptron, always apply the same hyperparameter settings to all its binary perceptrons. But vary these settings systematically (e.g. using a grid search) and see how it affects performance.

6. Evaluation (4 marks)

During training, track the accuracy on the validation set to monitor progress. After you have selected your best hyperparameters and modelling decisions, for RGB and grayscale respectively, evaluate the final models on both the validation and the test sets. When comparing the two final models, also report the following metrics, overall and on a per-class basis:

- Accuracy
- Precision
- Recall
- F1

Use a confusion matrix to visualise and analyse how performance differs across characters – which characters are classified most reliably and which characters are most often misclassified. **You are allowed to use the methods in the scikit-learn Python package to compute these metrics. You can also use the sklearn.metrics.confusion_matrix method to visualise your confusion matrix.**

Note: Test set results should only be reported at the end, once hyperparameters have been tuned on the validation set.

7. RGB vs grayscale analysis (4 marks)

Compare the performance of your RGB and grayscale models, both in terms of overall accuracy and per-class behavior. Discuss which characters benefit from color cues and which remain challenging even with RGB input. Comment on whether grayscale suffices for most characters, or whether RGB provides a clear advantage.

8. Code quality and reproducibility (3 marks)

Your code should be well-structured and reproducible. Use fixed random seeds where appropriate, so we can rerun your code to replicate your results, and organise your notebook with clear functions, classes, and documentation.

9. Final report (3 marks)

Alongside your notebook, submit a short PDF report (maximum 5 pages). This should describe your OOP design (the classes you used), describe your hyperparameter tuning process, and present your key results and insights.

Submission Requirements

Submit a single compressed archive named as your student number, e.g. GWRBRA001./tar.xz. The code should be submitted as a Jupyter notebook with clear documentation. Version control with git is recommended but not required. The code should be able to run based with only the

provided data and standard libraries such as Pillow, numpy, scikit-learn and matplotlib (check with a tutor if you are unsure about using other libraries). A README file should include setup instructions and a description of each submitted file. Do not submit any data.

Also include a report of at most 6 pages (submitted as a PDF document) that describes your design decisions, reports and discusses your results, including visualizations. Marks will be given primarily based on showing understanding of the concepts being applied and demonstrating ability to develop appropriate models and to analyse the results. Tasks 1–7 will be marked based on both your report and the code in your notebook.

Please ensure that your tarball works and is not corrupt (you can check this by trying to downloading your submission and extracting the contents of your tarball - make this a habit!). **Corrupt or non-working tarballs will not be marked - no exceptions. A 10% penalty will be incurred for a submission that is one day late (handed in within 24 hours after the deadline). Any submissions later than 1 day will be given a 0% mark.**

Academic integrity

All code and analysis must be your own work. You may use standard libraries (in line with the specification of what is permitted for the different components of the assignment) and reference documentation, but must cite all sources. You may discuss your work with other students, but code sharing is prohibited. Use of AI tools must be declared and limited to debugging assistance only. AI assistance is not permitted for writing the report.

Marking rubric

Category	**Marks**
Data processing	3
Multi-class perceptron implementation	5
Training	4
Hyperparameter tuning	4
Evaluation	4
RGB vs grayscale analysis	4
Code quality	3
Report quality	3
Total	**30**