AIML331: Computer Vision Assignment 3

CNN vs Transformer for Image Classification on Oxford-IIIT Pets Dataset

---

Due Date: 26 May (Week 12)
Total Marks: 100%

Objective: Students will explore and compare Convolutional Neural Networks (CNNs), Vision Transformers (ViTs), and hybrid CNN-Attention architectures by implementing them from scratch using PyTorch. The models will be trained on the Oxford-IIIT Pets dataset for image classification. Students will analyse training behaviour, generalisation, and convergence, and visualise the model development process using TensorBoard.

Dataset: The Oxford-IIIT Pets dataset is a popular benchmark for image classification and object recognition tasks, particularly in fine-grained classification. It consists of 7,149 images spanning 37 distinct pet breeds, including both cats and dogs. Each breed has approximately 200 images captured under varying lighting conditions, poses, and backgrounds, providing a diverse and challenging dataset. The dataset includes labels not only for breed classification but also for pet species (cat or dog), and it offers pixel-level annotations for segmentation tasks, although this assignment will focus solely on the breed classification labels. Mode details of the original Oxford-IIIT Pets Dataset can be found: https://www.robots.ox.ac.uk/~vgg/data/pets/.

Dataset customisation: To make the classification task less complex and more suitable for a wide range of devices with varying computational capabilities, we have implemented a wrapper for you on the original Oxford-IIIT Pet dataset to customise the dataset into four meaningful categories based on animal type and hair length. This wrapper is provided as a Python file named dataset_wrapper.py. The wrapped dataset is the actual dataset you will use for this assignment. Details on dataset preprocessing and usage of the wrapper are provided below:

The wrapper reorganises the original dataset into the following four categories, each represented by an integer label:

- **0**: Long-haired cat
- **1**: Short-haired cat
- **2**: Long-haired dog
- **3**: Short-haired dog

After preprocessing, the dataset contains a total of 7,149 labelled samples with the following distribution:

- Short-haired cats: 1,771 samples
- Short-haired dogs: 2,188 samples
- Long-haired dogs: 2,590 samples
- Long-haired cats: 600 samples

To facilitate model training and evaluation, the dataset is split into training, validation, and test sets randomly (with fixed random seeds) using an 80/10/10 ratio. The resulting splits are:

- **Training set**: 5,719 samples
- **Validation set**: 714 samples
- **Test set**: 716 samples

To load the dataset, the wrapper provides a single function: get_pet_datasets(). This function accepts parameters for image resizing and download location. An example of how to use the wrapper is as follows:

```
import dataset_wrapper
train_dataset, val_dataset, test_dataset = dataset_wrapper.get_pet_datasets(
  img_width=img_width,
  img_height=img_height,
  root_path='path/to/your/dataset/download/location'
)
```

The original dataset includes images with varying resolutions. You can either:

1) resize all data uniformly to **128×128** (set img_width= img_height=128), which is preferred if you have sufficient computational resources available.
2) resize all data uniformly to **64×64** (set img_width= img_height=64).

Both options will be marked equally.

The root_path argument defines the directory where the dataset will be downloaded and stored. The datasets returned by the wrapper (train_dataset, val_dataset, test_dataset) are ready to be wrapped in PyTorch DataLoaders for training and evaluation.

You are required to conduct training and validation during the model development process. **Although the testing dataset is provided, it must be treated as unseen data and should not be used at any stage of model training or validation. It is reserved strictly for final evaluation to assess the model's generalisation performance. You should not use it to help design your model!**

Device: You may use either **CUDA (GPU)** or **CPU** for training, and will not be disadvantaged if you do not have a GPU at home. You have access to both the CPUs on the ECS workstations, as well as the "cuda-small" GPU servers in the school.

Programming Language: Python + Python-based OpenCV.

If you would like to use a different programming language, please email Jennifer Zhao to discuss this. We encourage you to use a Python-based approach, as it has been the primary focus of our tutorials and discussions and is the main language used in AI/ML.

***Note:** While accuracy is an important metric, it will not be the primary criterion for awarding high marks in this assignment. Instead, emphasis will be placed on the correctness of implementation and the quality of discussion in your report. **However, you are expected to attain an accuracy on the testing dataset of at least 45% for the developed CNN model and at least 35% for the developed ViT model.**

Part 1: CNN Model Implementation and Analysis (50%)

The main objective is to achieve the highest possible classification accuracy across four categories: long-haired cat, short-haired cat, long-haired dog, and short-haired dog using CNN models.

Tasks:

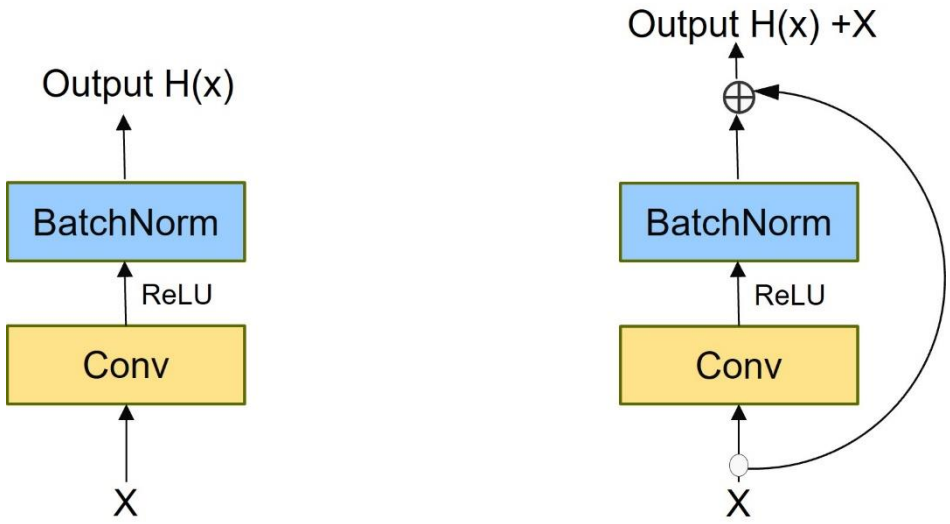
- **(10%) Build a CNN architecture from scratch using PyTorch.** You may utilise basic layers and components provided by PyTorch (e.g., Conv2D, BatchNorm2D, MaxPool2D, Linear, ReLU), but you are required to design and assemble the overall model architecture (including the loss function) independently.
- **(10%) Train the model for a few epochs until the loss value stabilises or flattens, indicating convergence (normally within 20 epochs for this dataset)**. Track the training process over epochs using TensorBoard, including metrics such as training loss and validation accuracy. You should include in your report:
    - Loss curves on the training dataset
    - Accuracy curve on the validation dataset
    - Final test accuracy at the selected stopping epoch
- **(20%) Conduct the following experiments and provide your analysis in the report, including:**
    - With vs. without batch normalisation

- Different activation functions. You should choose three activation functions from options including: ReLU, LeakyReLU, and your choice of more modern ones such as Swish and GELU (Gaussian Error Linear Unit).
- Varying the number of layers. Choose the number of hidden layers from the range [3, 4, 5, 6, 7] (not including the output layer).

- **(10%) Residual Integration.** Modify your CNN architecture to incorporate a residual connection around the CNN layer. Specifically, instead of computing the output as CNN(x), the modified network should compute the output as x + CNN(x), enabling the model to learn residual mappings.

  You may refer to the illustration below as a guide for modifying your standard CNN into its residual version, where the left side depicts the standard CNN layer with BatchNorm, and the right side shows the inclusion of the residual connection.

  Analyse the impact of this residual connection on the model's performance, particularly in terms of training behaviour and final testing accuracy.



Deliverables:

- Submit the code (or code link) for implementing and training/validating/testing the CNN model.
- Include an analysis in your report discussing the CNN model specifications, experimental setup, and results. The analysis should include a comparison of the model's performance with and without BatchNorm, using different activation functions, and varying depths (number of layers), supported by visualisation results (e.g. validation accuracy/loss curves and testing accuracy numbers).

- Include an analysis in the report of the impact of adding residual connections on training stability and speed, and the final testing accuracy.

Part 2: Vision Transformer (ViT) Implementation and Comparison (50%)

The main objective is to achieve the highest possible classification accuracy across four categories: long-haired cat, short-haired cat, long-haired dog, and short-haired dog using a ViT transformer model.

Tasks:

- **(10%) Build a Vision Transformer (ViT) model from scratch using PyTorch.** You may utilise basic layers and components provided by PyTorch (e.g., multi-head attention, nn.Linear, nn.LayerNorm, activation functions, positional embedding), but you are required to design and assemble the overall architecture independently.
- **(10%) Train the ViT model for a few epochs until the loss function stabilises or flattens, indicating convergence(normally within 25 epochs for this dataset).** Track the training process over epochs using TensorBoard, including the visualisation of training loss and validation accuracy curves. You should include in your report:
    - Loss curves on the training dataset
    - Accuracy curve on the validation dataset
    - Final test accuracy at the selected stopping epoch
- **(20%) Conduct the following experiments on ViT and provide a detailed analysis in your report:**
    - Vary the number of attention heads, selecting values from the range: [3, 4, 5, 6].
    - Adjust the number of transformer layers, using values from: [3, 4, 5, 6] (excluding the output layer).
    - Test different patch sizes, choosing from: [4, 8, 16].
    - Compare the model performance with and without positional embeddings.

    Please keep the dimension of input embedding = 32.

- **(10%) Compare your ViT and CNN models according to the following aspects as a discussion in your report**:
    - Compare the final testing accuracy between the ViT and CNN models. Analyse the results and discuss any observed performance gap, including possible reasons for the differences in testing accuracy.
    - Evaluate and compare the number of learnable parameters in each model you designed, along with their convergence speed (i.e., the number of epochs required to reach optimal performance) and inference time (runtime) for each image on your device. Discuss the computational efficiency and practical suitability of each model for the four-category classification task.

Deliverables:

- Submit the code (or code link) for implementing and training/validating/testing the Vision Transformer model.
- Include an analysis in the report discussing the ViT model specifications, experimental setup, and results. The analysis should include a comparison of the model's performance against the CNN in terms of accuracy, model size, convergence speed, and runtime. Support the analysis with visualisation results (accuracy/loss curves, etc.).

Part 3 Advanced Challenge: Hybrid CNN + Attention Model
(Optional for AIML331 students. Compulsory for COMP/AIML 440 DIS students)

For AIML331 students, completing this part will award up to 20% bonus marks, which can be used to offset a loss of marks in Parts 1 and 2. You are encouraged to complete it, as time allows, to enhance your learning and skills gained from AIML331.

For COMP/AIML440 students, this part is worth 20% of your final course grade for COMP/AIML 440.

Design and build a hybrid model that integrates a CNN backbone with an attention mechanism (either self-attention or general attention in spatial/channel-wise). The architecture should include a reasonable integration of the attention layer within the CNN structure.

Task 1 (10%) – CNN + Channel-wise Attention
Design and implement a hybrid model that integrates a **CNN backbone** with a **channel-wise attention mechanism** (either self-attention or general attention). The attention

module should be placed appropriately within the CNN to enhance feature representations by re-weighting channel responses.

Reference paper: Hu, Jie, Li Shen, and Gang Sun. "Squeeze-and-excitation networks." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018. DOI: *https://doi.org/10.48550/arXiv.1709.01507*

Tasks:

- Modify a standard CNN by inserting channel-wise attention modules at suitable locations.
- Train the model and evaluate its performance compared to the baseline CNN.

Task 2 (10%) – CNN + Spatial Attention
Design and implement a hybrid model that integrates a **CNN backbone** with a **spatial attention mechanism** (e.g., spatial attention maps that focus on important regions in the feature maps). The goal is to allow the model to learn *where* to focus in the input image or feature maps.

Tasks:

- Modify a standard CNN by inserting spatial attention modules at suitable locations.
- Train and evaluate the model against the baseline CNN.

Reference paper: Woo, Sanghyun, et al. "CBAM: Convolutional block attention module." *Proceedings of the European conference on computer vision (ECCV)*. 2018. *DOI: https://doi.org/10.48550/arXiv.1807.06521*

Deliverables:

- Submit the code (or code link) for implementing, training, and testing the Hybrid CNN + Attention model.
- Submit a report analysing the hybrid model architecture, training and testing results (with visualisations such as validation accuracy and loss curves). Make a performance comparison with the standalone CNN. Discuss whether the attention mechanism you designed improved or reduced the final performance. What might be the reasons behind these outcomes?

Submission Instructions:

1) **PDF Report:**
   Submit a PDF report that includes the analysis and discussion required for each task outlined above.

   The report should be clear, concise, and well-structured, offering detailed explanations for your implementation and experimental findings.

2) **Code Submission:**
   Either:
   a) Submit your code in a **compressed (zip) folder**, containing the code for each part in a separate directory. For each part, you may either provide the code as separate files or as a single combined file. If the code is lengthy, please provide detailed guidance on how to navigate and understand it.
   b) Alternatively, you may **submit a link to your code repository** (e.g., GitHub, GitLab, Google Drive). Please ensure that the link has appropriate permissions for the teaching team to access it. Links without proper access will be treated as missing submissions.

   Your code must be executable on the ECS workstations! Ensure it is well-documented, with clear comments explaining each key step. Additionally, provide a README file to guide us on how to review and run your code.

3) **Marking:**
   We will utilise **in-person marking** for this assessment. This is both to validate the authenticity of your work, but also is a good opportunity for you to discuss and explain your work in more detail with a tutor so that we can award the grade you deserve.