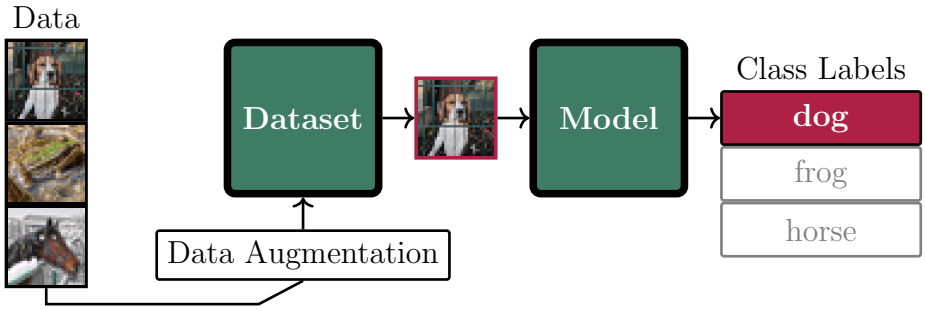
Animal Image Classification

Joint Assignment for
702303_25S and 702403_25S

Christian Walter

June 3, 2025



Introduction

This is the joint assignment of the courses "Image-Processing Systems; Capture and Analysis" (702303_25S) and "Modelling, System/Process Analysis, Forecasting Models and Decision Support" (702403_25S). The goal of the assignment is to get familiar with the task of image classification through machine learning models. Specifically, animal images are classified from the CIFAR-10 dataset [4]. The classes {`dog`, `frog`, `horse`} are extracted from the dataset and used to train the machine learning models. The model zoo includes: Convolutional Neural Networks (CNNs) [5], Support Vector Machines (SVMs) [1], and Random Forests (RFs) [3]. Either the images are used directly in the case of the CNN model, or the Histogram of Oriented Gradients (HOG) [2] features are extracted. Moreover, the task is to create a very small test image dataset on which the trained models are evaluated.

The assignment project repository is provided on Github: `https://github.com/gartenschlaefer/pixel_animal_ml`, where the tasks are located in specific python files (`task[nr]_[name].py`). Most parts of the python files are already implemented but some intentionally blank code snippets need to be filled. Basic guidelines and installation instructions are given in the repository. In the following task descriptions, we refer to the functions and classes in this project repository.

In order to complete the assignment, a short documentation of the results and plots from your experiments have to be sent to christian.walter@vetmeduni.ac.at. The format of the documentation is a `.pdf` file with your student number in the

filename. Use the above mail also for questions regarding the assignment, however, note that the question might be broadcasted to the other students to avoid duplicates of the same question.

Task 1: Data Creation

In this task, a small test image dataset of the target classes {`dog`, `frog`, `horse`} has to be created. You are asked to provide the individual test images for the dataset by yourself. The project contains a folder called:

```
/datasets/pixel_animals/data
```

where the data should be placed. Use the following naming convention for the folder and file structure:

```
.../data/[classname]/[filename].png
```

e.g. save an image file of a dog as: `.../data/dog/mydog.png`. The images have to be saved as `.png` files with a required size of **32 × 32 pixels**. At least one image for each class has to be created!

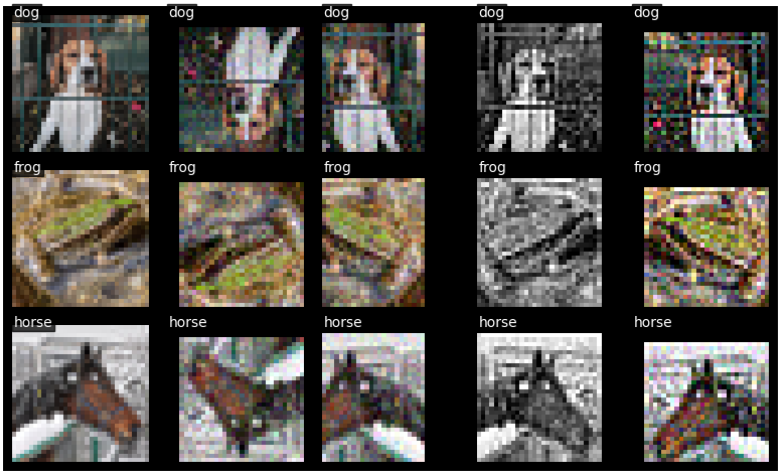
> **Tip:** For the image creation, either photograph the images yourself or search for images on the internet (also AI generated images are allowed). The difficult part will be the downsampling of the images to 32 × 32 pixels, however, the project provides a simple downsampling algorithm with the function `downsample_files`.

e.g. 450 × 450 px 32 × 32 px

Task 2: Datamodule and Data Augmentation

Continuing from the previously created test dataset, a datamodule and data augmentation function has to be implemented. The datamodule for training is already provided by the class `CIFAR10AnimalDatamodule`, as well as, most of the code for the test dataset (class `PixelAnimalDatamodule`), only the code snipped for storing the data into the data containers is missing.

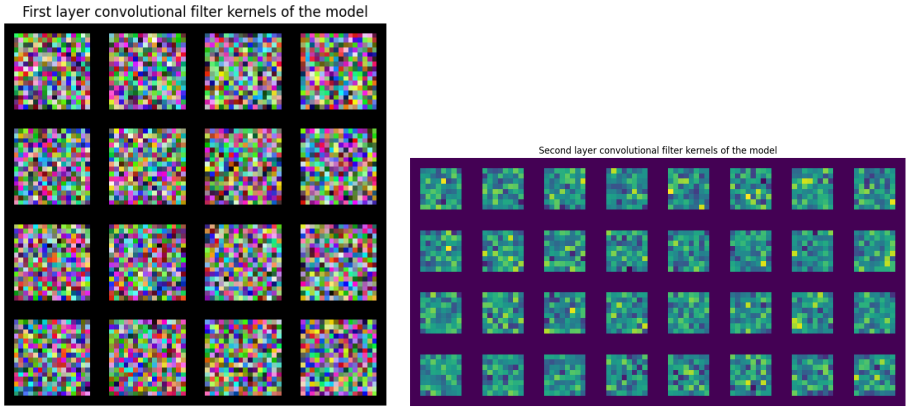
The second task is to implement the data augmentation function (`augment_data`) to create variants of the already existing data. The hereby augmented data is then added to the training of the machine learning models. You should uncomment and select some of the data augmentation transforms.



Task 3: Model

The next step is to create a machine learning model for the classification task. A CNN has to be implemented with the help of the `pytorch` framework. Again most of the work is already done in the class `AnimalCNNModel` where only a second convolutional layer has to be defined in its `__init__` function. Name the second layer "`self.layer2`", make it a sequential module (`torch.nn.Sequential`), include a convolutional layer of input channels 16, out channels 32, kernel size (8, 8), and stride (1, 1). Further, add a dropout layer (`torch.nn.Dropout2d`) with a probability of 0.25, a max-pool layer of kernels size (3, 3) and stride (1, 1), and a Rectified Linear Unit (ReLU) activation function.

The other code snipped is located in the class member function `forward`, which is used to forward propagate the input data through the network. The layers have to be called and the output has to be passed through.

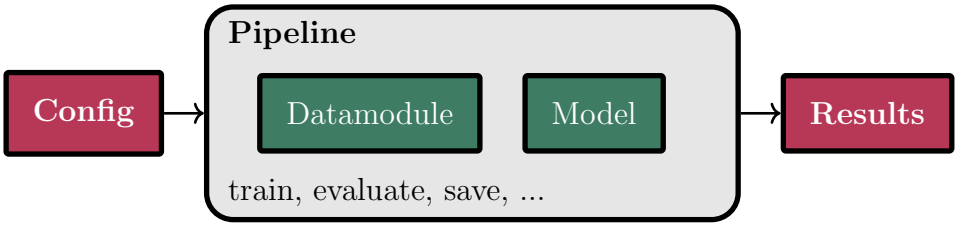


Note: All other models, namely the, Random, SVM, and RF model, are provided in the file `model_collection.py`. The Random model outputs random class labels and is merely used for testing.

Task 4: Pipeline

To evaluate different types of models with the same processing steps, a machine learning pipeline has to be implemented. Add the training steps of the model by calling the required functions in the function `pipeline`. In detail, the augmentation function, the feature extraction function (`extract_hog_features` from "feature_extraction.py"), the model training step, and the score update should be called in that order. Further, remove or comment the debug plot after the verification of the pipeline, so that the pipeline does not stop after each epoch.

Note: The pipeline uses the training data from CIFAR-10 and applies a cross-validation to compute evaluation scores. The parameters for the pipeline are provided via a config dictionary, where each component, such as the datamodule or model is defined and further initialized.



Task 5: Experiments

Now the pipeline can be used to run different experiments and store their results. This time no code has to be added but experiment config files have to be written. In the folder `./experiments/` the experiment `.yaml` files should be placed. Use the example experiment file of the Random model to create following experiments:

- Animal CNN model without data augmentation
- Animal CNN model with data augmentation
- Animal CNN model with HOG features enabled
- SVM model
- RF model.

The parameters for training may be chosen arbitrary but consider a high number of augmentations increases the training time. Also the use of HOG features requires more time, however, all the models should be able to run on the CPU of a consumer PC. It is important to use the same structure as shown in Listing 1. The `module` refers to a python file and and the `attr` is the function or class in this file, which further will be loaded in the pipeline.

Listing 1: Example experiment file of the Random model in `.yaml` file format.

```yaml
training:
  num_epochs: 1
  num_augmentations: 0
  use_extracting_hog: False
  show_plots: False
  dataloader_kwargs:
    batch_size: 64

datamodule:
  module: 'task2_datamodule'
  attr: 'CIFAR10AnimalDatamodule'
  args: []
  kwargs:
    root_path: './datasets/cifar10'

model:
  module: 'model_collection'
  attr: 'RandomModel'
  args:
    -
      model_file_name: 'random_model'
      save_path: './'
      num_classes: 3
  kwargs: {}
```

The validation accuracies of the models (except the Random model) should range from around 0.7 to 0.8. If better accuracies are achieved, you can consider yourself already a successful deep learning engineer, congratulations. The results of the models are saved to `./output/experiment_results`. One training result of a CNN model is shown in Figure 1.

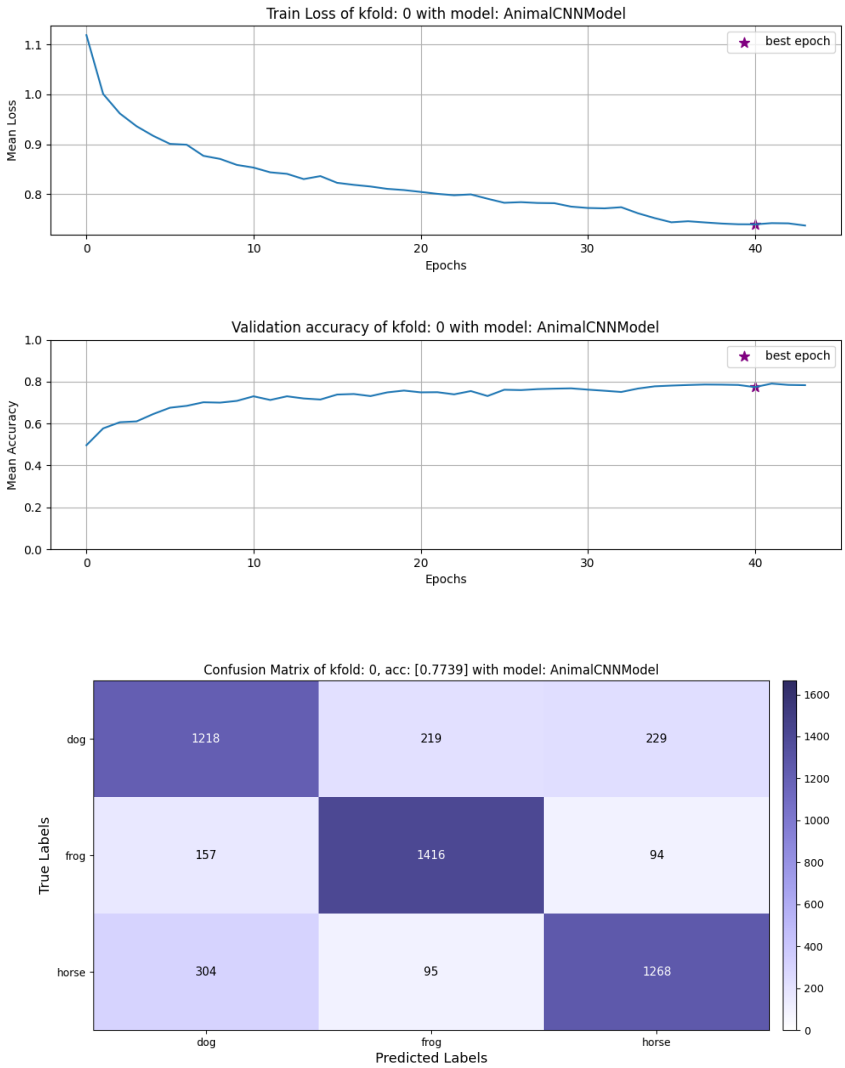


Figure 1: Training results of a CNN model.

Note: The SVM and RF models always use HOG features and have only one epoch as they are not designed to be incremental learning models. The number of epochs are important for the CNN model (an early stopping mechanism was implemented for the CNN). Also make sure to get the `args` and `kwargs` right.

Task 6: Final Results

In the final task, the trained models should be loaded and tested on the created test set (`PixelAnimalDatamodule`) with data augmentations. The model files are retrieved from the result folder `output/experiment_results/` created by the previous task. Implement the code snippet in the function `test_models` by correctly

loading the model and datamodule, as well as, the desired data augmentation, feature extraction, and model prediction function. If everything is correct, the results of each model can be found in the folder `output/final/` where the classifications and performances can be observed and compared. Note that for each experiment there will be 3 model files (for each cross-validation split one). The overall performance is the mean accuracy of each of the 3 models performing on the corresponding validation or test data.

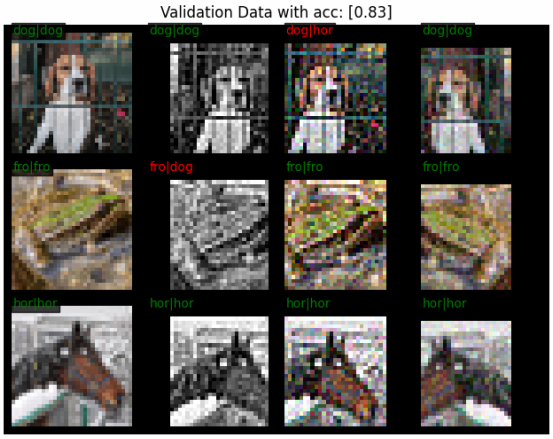


Figure 2: Example of a CNN model classification on the test set with augmentations.

Important: The results have to include the created data from task 1!

References

[1] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[2] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages 886–893 vol. 1, 2005.

[3] T. K. Ho. Random decision forests. In *Proceedings of 3rd International Conference on Document Analysis and Recognition*, volume 1, pages 278–282, 1995.

[4] A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, Canadian Institute for Advanced Research, 2009.

[5] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.