Binary Image Classification: Cats vs. Dogs with Data Augmentation and Early Stopping

Bozhidar Bozhikov, Nikita Smolianinov, Vedran Andric, Egor Chebotarev, Islam Elikhanov

December 23, 2025

1 Dataset

We've opted to use the Cats vs. Dogs dataset on *kaggle*, containing roughly 25.000 labeled images of cats and dogs, divided evenly. The dataset is appropriate because it has images of varying sizes, real world photos with realistic lighting, backgrounds, poses and color.

Images will be resized to a singular resolution (e.g. 224x224 pixels) and normalized. The data split between training, validation and testing will be a standard 70/15/15 split. The data augmentation applied may include random rotation of $\pm 15°$, image flipping, zooming, brightness/sharpness/saturation etc.

2 Model Design

We decided we would use a Convolutional Neural Network (CNN) built from scratch as our architecture. It will have an input layer that accepts RGB images of size 224x224x3, 3 convolutional blocks with a flatten layer (with ReLu activation), a fully-connected layer and a SoftMax function for the binary classification itself.

If time permits we would also like to compare the "traditional" CNN to an alternative model: a Vision Transformer (ViT) adapted for binary classification. The ViT shall split the input images into 16x16 patches and process them using self-attention mechanisms. To achieve this, we could use a pre-trained ViT (e.g. ViT-Base).

3 Input/Output

The input shall be RGB images of PNG or JPG format. The images shall be preprocessed and normalized by the program.

The output shall consist of binary classification - Cat or Dog - with an attached confidence score $[P(cat), P(dog)]$. For example:

Input: image.png (224x224x3 tensor)
Output: Predicted class "Cat" with a confidence of 0.87 given [0.87, 0.13]

4 Workflow Distribution

4.1 Data Preparation and Augmentation (Nikita Smolianinov)

Smolianinov shall download and organize the kaggle cats/dogs dataset, segment the data in the afformentioned 70/15/15 split, create the data preprocessing pipeline (resize and normalize), implement the data augmentation functions and generate and document the dataset statistics.

4.2 Model Design (Bozhidar Bozhikov)

Bozhikov shall design the CNN architecture from scratch, implement the convolutional layers, pooling layers and fully-connected layers in PyTorch, design the appropriate filter size, number and layer depth and calculate and document the parameter count. If time permits: implement the ViT for comparison, document and discuss the comparison.

4.3 Training (Egor Chebotarev)

Chebotarev shall implement the training loop in PyTorch, set up a loss function and optimizer, implement the early stopping functionality using model checkpointing to save the best weights based on validation accuracy to prevent overfitting, track training/validation loss and accuracy per epoch and generate training curves and visualization during testing.

4.4 Evaluation (Vedran Andric)

Andric shall implement comprehensive evaluation metrics (Accuracy, Precision, Specificity and overall F1 score), generate the confusion matrix on the test set, graph ROC curve and AUC score, conduct error analysis on misclassified images and test final model on test set and document results.

4.5 Deployment (Islam Elikhanov)

Elikhanov shall develop an interactive web app using Gradio or similar, display predictions with confidence scores and visualizations, create a user friendly interface with clear instructions, prepare final project documentation and presentation if missing and record a demonstration video.