[700.851] Machine Learning - Binary Image Classifier - Final Report

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

February 8, 2026

1 Introduction (Bozhidar Bozhikov)

Image classification is a fundamental problem in the field of computer vision with widespread applications where an assignment of a label or a class to an entire image is necessary, such as autonomous systems and medical image processing. In this project we develop a binary image classifier to discriminate between images of cats and dogs - originally a Microsoft hosted competition which has since become a popular benchmark.

While contemprorary models achieve near-perfect accuracy on this task through the use of transfer learning and large pre-trained networks, our goal is to build a discriminator from scratch and to cover all aspects of machine learning development, them being data preparation, model architecture, training, evaluation and deployment. Workload was distributed in such a way so that each team member is in charge of a single task, while still having roundtable discussions to cover questions and offer help.

To evaluate the effectiveness of regularization and optimization techniques, we developed two variants of the model for comparative study:

- Baseline model (Version 1): A 'vanilla' CNN implementation without additional features. This version only includes the core convolutional and fully-connected layers. Training uses an Adam optimizer with a fixed learning rate and runs for a predetermined number of epochs before stopping. Version 1 is our control group model.
- Regularized model (Version 2): An improved model featuring dropout layers, L2 weight decay, learning rate scheduling and early stopping to avoid overfitting.

Both models otherwise share identical architectural foundations (layer configurations, filter sizes, channel dimensions) to ensure a controlled comparison. We discuss the specifics of Version 2 in Sections 3, 4 and 5, and we compare the results in Section 7.

2 Dataset & Data Preparation (Nikita Smolianinov)

We downloaded and organized the kaggle cats/dogs dataset.

Data was segmented in a 70/15/15 training/validation/testing split by counting files and separating them virtually inside the code and copying them into new directories afterwards accordingly.

For augmentation two functions were created - one for training (adding random rotation in -15 to 15 degree range, color jitter, random movement/scaling and resizing/normalization) and one with just resizing and normalization for testing the model and utilizing it.

Dataloaders were used to transfer data from directories to the training and testing algorithms.

After all the data loading steps, data_prep.py prints out batch sizes and batch/image shapes.

3 Model Architecture (Bozhidar Bozhikov)

We designed a convolutional neural network from scratch specifically for binary image classification, following practical Session 5 from the course. The architecure follows a classical CNN design progressively increasing feature complexity through convolutional blocks, followed by fully connected layers for final classification. The network accepts 224×224 RGB images as input and outputs a single probability value between 0 and 1, representing the likelihood of the image containing a dog (with cat being the complement).

The model consists of a feature extraction component [ref] and a classifier. The feature map is composed of three convolutional blocks each following a Conv-ReLU-Maxpool-(Dropout, Version 2 only) layout. This hierarchical structure allows the network to extract and learn increasingly abstract features - from edges and contrast to fur texture and animal facial characteristics. After feature extraction, the spatial feature maps are flattened then passed to the classifier - a series of fully-connected layers, before arriving to the binary output.

We used several common CNN design decisions to balance efficiency and computational complexity. The number of convolutional filters double each layer (32 →64 →128) to capture finer and finer details. We use a 3×3 kernel with $padding = 1$, which is a good compromise between step size, parameter efficiency and image border fail-safe. MaxPooling reducing the spatial dimension by 2×each convolutional block (224×224→28×28) to reduce computational cost. The ReLU activation functions are used for their simplicity and robustness against the vanishing gradient problem. The final layer uses a sigmoid function to discriminate the result with $threshold = 0.5$. Version 2 additionally follows with spatial dropout layers. This forces the neural network to randomly set to zero a fraction of the neurons during training, making the network learn more generalizable fea-

tures to help prevent overfitting. We use an aggressive dropout value ($p = 0.5$) because of our very large parameter count of >25 million parameters compared to our training dataset of 17.5 thousand images, which present a risk of overfitting.

4 Training Methodology & Evaluation (Vedran Andric and Egor Chebotarev)

[Vedran Andric and Egor Chebotarev] jointly worked on the model's training and evaluation. We access the datasets and loaders provided by [Nikita Smolialinov] and set up the binary image classifier. We use an Adam optimizer, with version 2's optimizer utilizing weight decay.

We performed several training and evaluation runs with different hyperparameters: $batchsize, epochscount$ and $learningrate$. For the enhanced model we also added tweakable $L2weightdecay, learningratepatience, learningratefactor andearlystoppingpatience$. [Bozhidar Bozhikov] contributed to logging the ouput: each run writes its status to a catalogued log file along with the chosen hyperparameters and generates two graphs - loss and accuracy curves.

In total we performed 15 runs of model Version 1 and an additional 7 runs of Version 2. The logged run evaluations and generated graphs can be found in folders $/logs$ and $/plots$, respectively.

5 Deployment (Islam Elikhanov)

I created *run_model.py* and had the inference logic implemented in it to be used by my colleagues [Vedran Andric] and [Egor Chebotarev] during their work on training models and testing their performance. The script accepts either a single image path or a folder path, uses the same preprocessing as training (resize 224×224, ToTensor), loads the model and weights, runs inference, and prints for each image path, predicted class (cat/dog), and confidence.

For the said script I also added convenience config variables at the top (WEIGHTS_PATH, IMAGE_OR_FOLDER) so I can paste paths and run without command-line arguments; the script still supports two optional CLI arguments (weights path, then image or folder).

I also was responsible for the deployment of both of our models in a web app with a user friendly and easy to use interface. For that purpose I chose Gradio. The UI is defined in Python, and the only extra dependency is gradio.

The web app is implemented in *app.py*. The app loads both models when it starts,

uses the same preprocessing, and exposes a Gradio interface (image upload →text output). When you run *pythonapp.py*, *main*() builds both models, loads their weights, puts them on the same device, and stores them in a dictionary attached to the predict function. The dictionary keys are the two dropdown options "v1 (original)" and "v2 (updated)". So both models stay in memory the whole time; nothing is loaded or unloaded when the user switches. Preprocessing is shared, one transform (resize to 224x224 and $ToTensor$) and one device are used for both models.

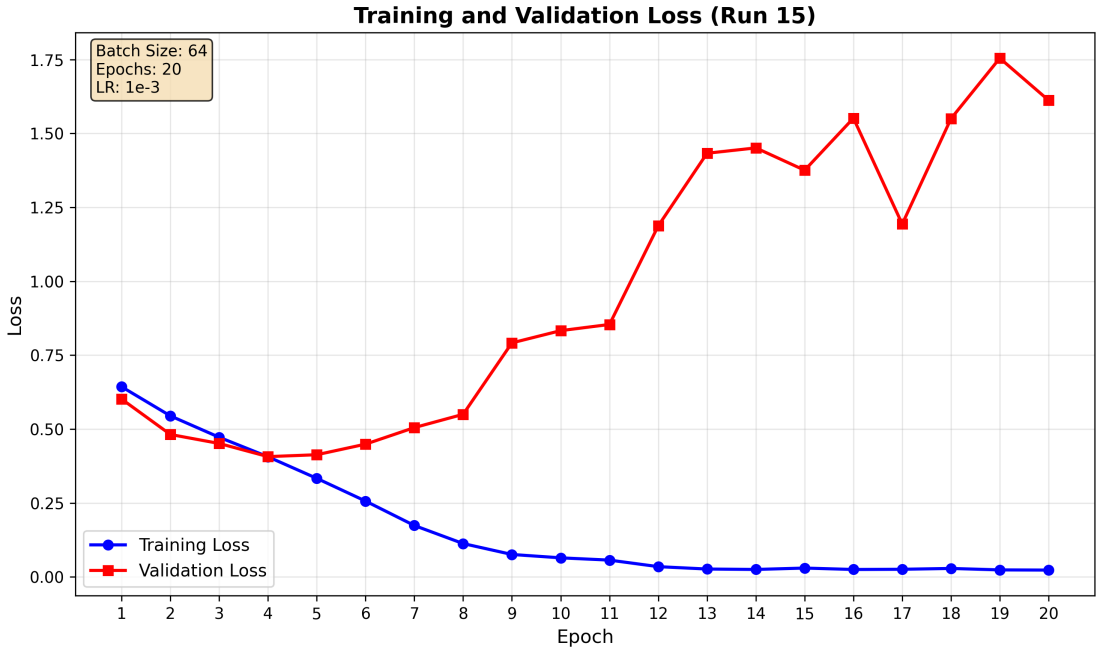
Switching is done with a dropdown at the top of the Gradio interface. The user picks either "v1 (original)" or "v2 (updated)" and uploads an image. When they submit, Gradio calls predict with two arguments, the chosen dropdown value (a string) and the path to the uploaded image. predict looks up that string in the models dictionary and gets the corresponding model (v1 or v2). The rest of the function is the same no matter which model was chosen. It loads the image, applies the transform, runs the selected model on the tensor, gets a single probability, and turns it into a class (cat or dog) and a confidence string. So switching is just choosing which of the two preloaded models is used for that one prediction.

6 Discussion & Analysis (Bozhidar Bozhikov)

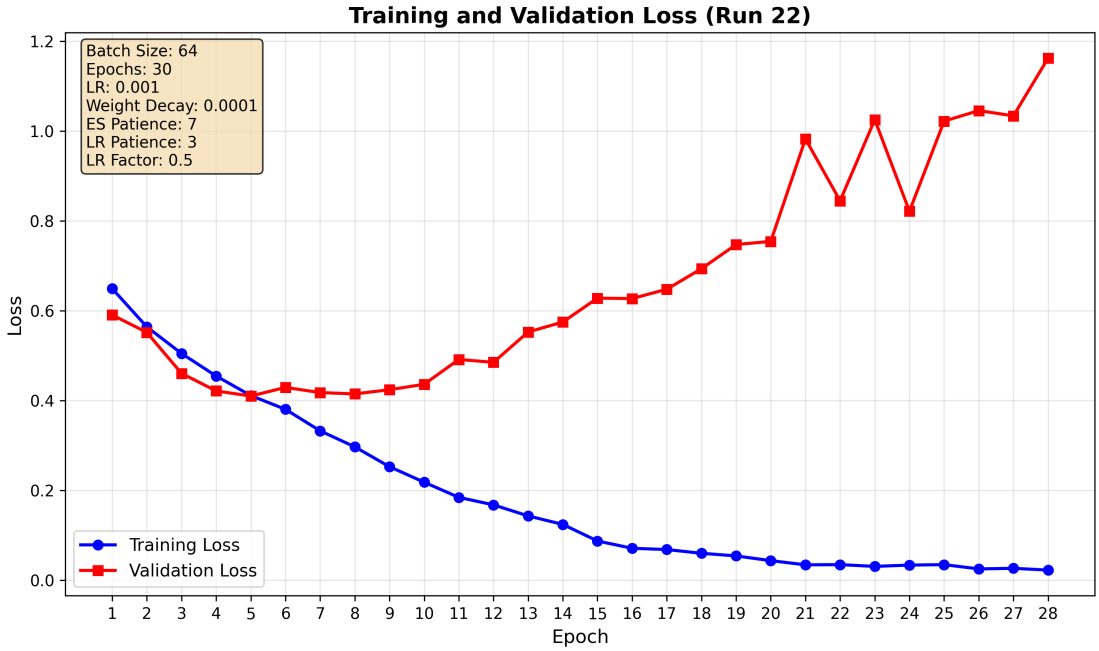
This study developed and compared two versions of a CNN architecture for binary image classification. Our baseline model (v1) achieved a maximum validation accuracy of 82%, while our enhanced model (v2) achieved 83.49% validation accuracy. Beyond this 1.49% of raw accuracy improvement and marginal train-validation gap improvement (17.6% →16.0%), analysis shows additional improvements in training stability, overfitting resistance and validation loss rates.

The baseline model, trained without regularization techniques, demonstrated inital rapid learning before succumbing to overfitting in later epochs. We consistantly observed peak training and validation accuracy around epoch 6-7, with later iterations' accuracies either stagnating or slightly declining and validation loss tripling in some test runs. This behavior can be blamed on overfitting due to the very high parameter-to-data ratio of over a thousand. This can be seen most clearly when tracking validation loss, with it slightly improving in epochs 1-4 (0.60 →0.41), before slightly degrading in epochs 4-8 (0.41 →0.55), until nearly tripling in the last epochs 8-20 due to overfitting (0.55 → 1.61). By epoch 19 validation loss reached 1.75, more than quadruple its best value of 0.4 early on in epoch 4. This increase of validation loss suggests the model would make incorrect predictions with extreme confidence.

The enhanced model benefits from many regularization strategies: spatial dropout, L2 weight decay, learning rate scheduling and early stopping. Spatial dropout is discussed in Section 3. L2 weight decay penalizes large weights by adding a term proportional to the sum of squared weights to the loss function, promoting simpler models that generlize better thus preventing overfitting. Learning rate scheduling and early stopping



(a) Baseline Model (v1)



(b) Enhanced Model (v2)

Figure 1: Training and validation loss curves comparison

increment a counter for each epoch where validation loss does not improve. After reaching a certain threshold, the learning rate is decreased by a specified factor, with training completely stopping after passing a greater threshold. Afterwards the model reverts to the weights from the epoch with the lowest validation loss, ensuring optimal performance without overfitting. During the discussed v2 training rate the scheduler triggered thrice, halving the learning rate each time. With these regularization techniques we observe the following model validation loss trajectory: loss improves during the early epochs 1-5 (0.59 →0.41), then gradually increasing during epochs 5-21 (0.41 →0.98) and 21-28 (0.98 →1.16), before early stopping after epoch 28. We can observe that from peak accuracy to final epoch the model experiences only an 18% increase (0.98 →1.16) versus baseline's 193% increase (0.55 →1.61). This results in the final model being more reasonable in placing its confidence scores.

7 Limitations & Future Work (Bozhidar Bozhikov)

The regularization improvements discussed prior increased the model's stability by extending the productive time window from around 8 epochs in the baseline to 28. They also prevented some validation loss, with baseline's maximum achieved loss reaching 1.75, while v2 only reaching 1.16 - a 34% difference. However, the train-validation gap barely improved, from 17.6% to 16%. Late-stage overfitting still occured after epoch 21; regularization delayed the onset yb 13 epochs but didnt prevent it. The final learning rate reduction was ineffective and produced no validation improvement. Additionally, the validation accuracy improvement was small - only 1.49%. We hypothetize that the 1470:1 parameter-to-train is still too large even with these improvements.

This would suggest the issue is architectural rather than algorithmic. Due to hardware constraint and storage resources available, we made a decision to not go above our original dataset and model specifications to ensure flexibility in the testing stages. The two models that we ended up with are rather modest, and for future work we suggest either decreasing the parameter count by replacing the fully-connected layer with global average pooling, reducing the 128×28×28 to 128×1×1, or increasing the data set size, adding performance-altering changes such as including random object occlusion to the data augmentation pipeline, setting the training cutoffs higher and more. [Vedran Andric and Egor Chebotarev] additionally attempted to increase the count of the convolution layers by 1-2, thus reducing the parameter count from ∼25M to ∼10M. This was done before model version 2 was finalized, therefore we could not produce data to validate or disprove this hypothesis.

8 Conclusion (Nikita Smolianinov)

Considering the ∼400MB size of the training dataset, average 30 minutes to train a model on a commercial laptop and ∼80% accuracy, it is safe to say that the latest

model performs well. We suggest that most issues are caused by the high parameter-to-data ratio. We found that learning rate scheduling and the dropout layers in particular showed demonstrable improvements compared to the baseline.

References

1. IBM Think. "What are Convolutional Neural Networks?" *IBM Topics*. https://www.ibm.com/think/topics/convolutional-neural-networks

2. CodeGenes. "Image Binary Classification with PyTorch." *CodeGenes Blog*. https://www.codegenes.net/blog/image-binary-classigication-with-pytorch/

3. Kashyap, P. (2024). "Understanding Dropout in Deep Learning: A Guide to Reducing Overfitting." *Medium*. https://medium.com/@piyushkashyap045/understanding-dropout-in-deep-learning-a-guide

4. "Weight Decay: L2 Regularization in Neural Networks." *Towards Data Science*. https://towardsdatascience.com/weight-decay-l2-regularization-90a9e17713cd/

5. Brownlee, J. "A Gentle Introduction to Learning Rate Schedulers." *Machine Learning Mastery*. https://machinelearningmastery.com/a-gentle-introduction-to-learning-rate-scheduler

6. "Early Stopping Techniques for Deep Learning Models." *ACM Digital Library*. https://dl.acm.org/doi/full/10.1145/3731806.3731844

7. "Complete Guide to Data Augmentation." *DataCamp Tutorial*. https://www.datacamp.com/tutorial/complete-guide-data-augmentation

8. "Understanding Occlusion in Computer Vision." *Roboflow Blog*. https://blog.roboflow.com/occlusion-computer-vision/