Binary image classifier

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

Main idea

- Binary image classifier to discriminate between images of cats and dogs
- Originally a Microsoft hosted competition which has since become a popular benchmark.

Versions:

- V1: A 'vanilla' CNN implementation without additional features. 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.
- V2: An improved model featuring dropout layers, L2 weight decay, learning rate scheduling and early stopping to avoid overfitting.

Dataset & Data Preparation

- Kaggle cats/dogs dataset.
- 70/15/15 training/validation/testing split
- Two augmentation functions
- Dataloaders to transfer data from directories to the training and testing

Model Architecture

- Convolutional neural network made from scratch for binary image classification
- Classical CNN architecture design
- Progressively increasing feature complexity through convolutional blocks
- Fully connected layers for final classification
- 224×224 RGB images as input - a single probability value between 0 and 1 as outputs
- Feature extraction component and a classifier.
- The feature map is composed of three convolutional blocks each following a Conv-ReLU-Maxpool-(Dropout, Version 2 only) layout.
- After feature extraction, the spatial feature maps are flattened then passed to the classifier - a series of fully-connected layers

The number of convolutional filters double each layer (32 -> 64 -> 128)

We use a 3x3 kernel with padding=1 - a good compromise between step size, parameter efficiency and image border fail-safe

MaxPooling is reducing the spatial dimension by 2 times each convolutional block (224x224 -> 28x28) to reduce computational cost.

The ReLU activation functions are used against the vanishing gradient problem.

The final layer uses a sigmoid function to discriminate the result with threshold=0.5

Version 2 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 features to help prevent overfitting

We use an aggressive dropout value because of our large parameter count of 25 million parameters (compared to our training dataset of 17.5 thousand images – risk of overfitting)

Training Methodology & Evaluation

- We use an Adam optimizer, with version 2's optimizer utilizing weight decay.
- We performed several training and evaluation runs with different hyperparameters: batch size, epochs count and learning rate.
- For the enhanced model we added tweakable L2 weight decay, learning rate patience, learning rate factor and early stopping patience.
- 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: 15 runs of model Version 1; 7 runs of Version 2

Deployment

- run_model.py had the inference logic implemented in it to be used by our team 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
- Deployment of both of our models in a web app with a user friendly and easy to use interface running on Gradio. The UI is defined in Python, and the only extra dependency is Gradio

When you run python app.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.

Conclusion

- 400MB size of the training dataset
- Average 30 minutes to train a model on a laptop
- 80% accuracy

Thus, 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