CSE669 Deep Learning Assignment 3: Dog vs Cat vs Bird Classifier

Ariba Khan (17270)
Dr. Sawera Hanif (29413)

December 31, 2024

Abstract

This report explores the development and experimentation of a deep learning classifier designed to distinguish between images of cats, dogs, and birds. Various neural network architectures, data augmentation techniques, and optimization strategies were explored to improve classification accuracy. The final model achieved a validation accuracy of 87.92% and a test accuracy of 85.46%. The findings highlight key insight into effective model selection and training methodologies.

1 Introduction

Image classification is a critical task in computer vision with roots in deep learning, with the objective of assigning the correct label to an image. This project aims to classify images into one of three categories: cat, dog, or bird. Given the challenge of distinguishing between similar-looking animals, we experiment with various deep learning architectures and techniques to achieve optimal accuracy and robustness.

2 Dataset

The dataset used in this experiment comprises of 12,000 labeled images of dogs, cats, and birds, provided through a dedicated Kaggle competition. The test set includes 3,000 unlabeled images.

3 Methodology

3.1 Data Preprocessing

Each image was resized to 32x32 pixels. The training dataset was split into training and validation sets with an 80/20 ratio, and a random state of 42 was set for reproducibility. Data augmentation techniques, including random rotations, flips, color adjustments, and normalization, were applied to enhance the model's generalization.

For model preparation, a 32x32 resolution was used for the CNN and VGG16 models, as mentioned earlier. However, for ResNet18 and ResNet50, the resolution was increased to 224x224, as these models require this specific resolution for their parameters.

3.2 Label Mapping

We utilized the `train.csv` file, which contains the class labels, to map them to our training dataset through label mapping. Specifically, we assigned the labels 0, 1, and 2 to represent cat, dog, and bird, respectively.

3.3 Data Augmentation

We applied random rotations of up to 45 degrees in either direction, along with horizontal flips, and color jitter with adjustments to brightness (0.2), contrast (0.2), saturation (0.2), and hue (0.2). The training dataset was then transformed into tensors and normalized with a mean of [0.5, 0.5, 0.5] and a standard

deviation of [0.5, 0.5, 0.5]. The validation dataset did not undergo the same transformation to maintain the robustness of the model.

The training dataset consisted of 9,600 images, while the validation dataset contained 2,400 images.

3.4 Model Architectures

3.4.1 Baseline Convolutional Neural Network (CNN)

We started with a basic CNN architecture consisting of two convolutional layers and applied ReLU activation in between them. We followed them with a pooling layer of kernel size = 3, stride = 1, and the other of kernel size = 2, stride = 2 respectively. After flattening their output, we added two fully connected layers with ReLU activation between them. The output nodes were equal to the number of classes, three. Here, we used CrossEntropyLoss as our loss function and Stochastic Gradient Descent (SGD) as our optimizer using a Learning Rate (lr) of 0.001.

The baseline architecture was progressively modified through several adjustments, such as incorporating dropout, weight decay, and batch normalization. The convolutional layers were arranged sequentially, each followed by a ReLU activation, with a single max pooling layer placed after the consecutive convolutional layers.

Initially, the dropout layer was added after the first fully connected layer, and then later another one was added to follow the convolutional block.

3.4.2 Transfer Learning

We used the pre-trained models, VGG16, ResNet18 and ResNet50. We fine-tuned them to our dataset to improve performance. The idea was to leverage the knowledge these models had learned from large datasets. Considering the GPU poor situation and not wanting to 're-inventing the wheel' when better pre-trained options are available, we used transfer learning to work on our dataset.

3.5 Hyperparameter Tuning

3.5.1 Learning Rate

We tested different learning rates to determine the optimal one for faster convergence and to avoid overshooting the optimal solution.

3.5.2 Batch Size

The batch size was varied to observe its impact on training stability and generalization performance.

3.5.3 Epochs

We trained models for 50 epochs with the exception of Learning Rate Finder for our Baseline CNN model, and used early stopping to prevent overfitting.

3.6 Optimization Strategies

3.6.1 Optimizer Selection

We experimented with different optimizers like SGD and Adam to determine the most effective one for this task. We also used early stopping for some models to observe where the model started overfitting. All these changes were made to observe which yielded the best results.

3.6.2 Regularization

We incorporated regularization techniques such as dropout, batch normalization, and L2 regularization to reduce overfitting. We experimented with two dropout options, 0.3 and 0.5, for our models with different combinations.

4 Results

The experimental results are summarized below:

4.1 Losses

The training and validation losses for each model is displayed in the following table:

Model	Training Loss	Validation Loss
Baseline CNN	0.8643	0.9033
CNN - Optimizer (SGD → ADAM)	0.5680	0.7729
CNN - Learning Rate Scheduler (ADAM + Cosine Annealing with Warm Up)	1.0993	1.0982
CNN (Learning Rate Finder)	0.7504	0.8107
CNN (ADAM + Dropout)	0.5719	0.7381
CNN (ADAM + Dropout + Weight Decay)	0.6824	0.7353
CNN (ADAM + Dropout + Weight Decay + Batch Normalization)	0.5021	0.6557
VGG16	0.8779	0.7488
VGG16 (Cosine Annealing)	0.8624	0.7896
VGG16 (ADAM)	0.4318	0.8126
VGG16 (ADAM + 50% Dropouts)	0.8733	0.7350
VGG16 (ADAM + 30% Dropouts)	0.7096	0.7671
VGG16 (ADAM + 50% Dropouts + 0.01 Weight Decay)	0.8516	0.7273
VGG16 (ADAM + 50% Dropouts + 0.1 Weight Decay)	0.8287	0.7693
VGG16 (ADAM + 50% Dropouts + 0.01 Weight Decay + Batch Normalization)	0.7059	0.6957
VGG16 (ADAM + 50% Dropouts + 0.001 Weight Decay + Batch Normalization)	0.8479	0.8563
VGG16 (ADAM + 30% Dropouts + 0.01 Weight Decay + Batch Normalization)	0.6769	0.6836
VGG16 (ADAM + 30% Dropouts + 0.1 Weight Decay + Batch Normalization)	0.8240	0.8241
VGG16 (ADAM + 30% Dropouts + 0.1 Weight Decay + 0.1 Learning Rate + Batch Normalization)	1.408	1.645
VGG16 (ADAM + 30% Dropouts + 0.001 Weight Decay + 0.01 Learning Rate + Batch Normalization)	0.7332	0.7469
VGG16 (Last FCNN Replaced with CNN)	0.7573	0.8572
ResNet18	0.8656	0.8916
ResNet18 (Cosine Annealing)	0.8581	0.8773
ResNet18 (ADAM + 30% Dropouts + 0.01 Weight Decay + Batch Normalization)	0.8935	0.8982
ResNet18 Architecture (No Transfer Learning)	1.1296	1.1357
ResNet18 Pre-Trained (Transfer Learning)	0.3944	0.3794
ResNet50 (ADAM + 30% Dropouts + 0.01 Weight Decay + Batch Normalization)	0.8825	0.8901
ResNet50	0.3281	0.3219

Table 1: Loss Progression Across Experiments

4.2 Confusion Matrix

The confusion matrix for the best-performing model, ResNet50, is shown below:

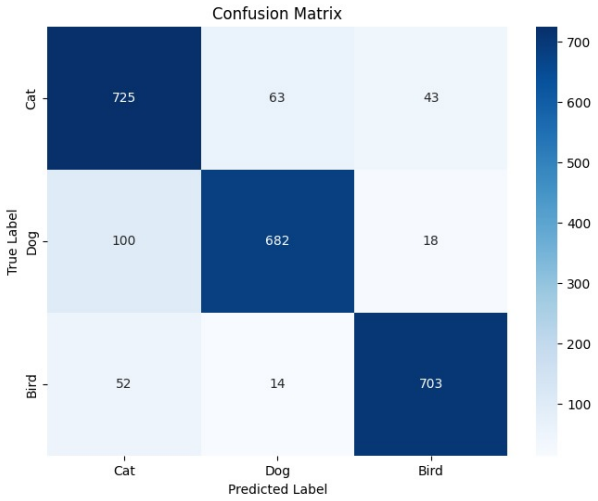

Figure 1: Confusion Matrix for ResNet50

5 Discussion

The results indicate that the ResNet18 Pre-Trained (Transfer Learning) model achieved the lowest losses, outperforming other models like the CNN and VGG16. This is likely due to its deeper architecture and pre-trained weights. The CNN, despite being a simpler model, performed reasonably well, indicating that simpler architectures can still provide satisfactory results when trained correctly. We believed that

VGG16 being a smaller model as ResNet18 would be adequate for the dataset we were working on, however, we would like to explore larger models for this purpose.

We took measures to counter overfitting, including early stopping, dropout, etc. We found early stopping to be particularly effective with our time as we did not have to go for 50 epochs (as we had decided for each model initially). Data augmentation and regularization techniques such as dropout and L2 regularization were useful in mitigating this problem.

Our experimentation with different dropout rates indicated that a higher dropout of 0.5 led to relatively higher losses which were about 12-13% higher compared to the losses we observed when we dropped 30% of the neurons. A higher learning rate of 0.1 led to poorer accuracy scores for the train and validation sets, both.

We made an observation regarding the weight decay, the smaller the decay, the more epochs the model took in training. For instance, when using a weight decay of 0.001 with a dropout of 0.3 and learning rate of 0.001, the model took 44 epochs to train and approximately 30 minutes on Kaggle's GPU T4 x 2.

We used a few optimization techniques including early stopping. The interesting thing about early stopping was that with most of our hyperparameters, it stopped at 7-11 epochs, which is intriguing to us, and went up to 25 epochs when the learning rate or weight decay were set at higher values.

In the Baseline CNN model with the Learning Rate Finder, we observed an interesting finding where the loss decreased steadily for the first 50 epochs. When we ran it for the next 50 epochs, we encountered the losses starting to increase steadily till they reached a value approximately around the loss we started with.

This highlights the importance of selecting the number of epochs wisely and underlines the role early stopping can potentially play in preventing such.

We used Losses to report here because it hit us harder when our models were not performing well compared to the accuracy scores. Based on this, we believe we still have ways to go as we mainly used smaller pre-trained models for our experiments. We will be using larger models in our future work to further improve on the losses and accuracies emboldened by the ResNet50 score. The experimentation with ResNet50 with the dropout and batch normalization did not work out well, however, that is something we will further experiment with to determine the reason behind the vast difference in the performance of the two ResNets50s.

6 Conclusion

In conclusion, the deep learning classifier developed in this project demonstrated strong performance in distinguishing between cats, dogs, and birds. The best-performing model was ResNet50, achieving a validation accuracy of 87.92% and a test accuracy of 85.46%. Future work could focus on exploring even deeper architectures or augmenting the dataset to improve generalization.

7 References

[1] Training Neural Networks with Validation using PyTorch. GeeksforGeeks.

[2] Piyush Kashyap: Transfer Learning in PyTorch: Fine-Tuning Pretrained Models for Custom Datasets. Medium.

[3] Tanaya: Transfer Learning using VGG16 in Pytorch. AnalyticsVidhya.

[4] Kiel Dang: Deep learning — Computer vision (CV) using Transfer Learning (ResNet-18) in Pytorch — Skin cancer classification. Medium.