CNN on the Natural Images Dataset

Christopher Chan
College Of Engineering
California State University, Long Beach
Long Beach, USA
Christopher.Chan01@student.csulb.edu

Abstract

This report presents the design, implementation, and evaluation of a lightweight convolutional neural network for multi-class image classification across eight categories for the Natural Images dataset. Design wise, the model implements progressive feature extraction, batch normalization, dropout regularization, and adaptive learning rate and average pooling to balance model size and capacity. Additionally it includes a comprehensive test driven workflow which includes unit, gradient, transformation, and integration tests to ensure accuracy and reproducibility across the training portions of the model. Data was split into 80/20 for training and validation sets respectively, with data augmentation and an adaptive learning-rate scheduling strategy to improve generalization. The model achieves and plateaus at 92.02% validation accuracy, with analysis of training dynamics, diagnostic tests, and baseline comparisons demonstrating stable optimization, minimal overfitting, and ideal architectural choices. The results validate that compact CNN architectures can perform competitively on real-world image dataset and provide a strong foundation for future work in deployment focused optimization.

I. Introduction

A. Motivation and Problem Statement

Image classification remains one of the fundamentals of computer vision in both challenge and application, spanning automated content tagging, environmental monitoring, robotics, and medical imaging. While modern pre-trained models achieve excellent results on benchmark datasets, understanding the principles of CNN design and training from scratch provides essential learning for individuals working with domain specific datasets or resource constrained environments.

This project tackles multi-class natural image classification on a dataset containing eight categories: airplane, car, cat, dog, flower, fruit, motorbike, and person. The task requires distinguishing between objects with significant intra-class variation (different dog breeds, flower species) and inter-class visual similarities (cats vs dogs, cars vs motorbikes).

B. Project Goals and Scope

The primary objectives of this project are to:

1. Design and implement a CNN architecture optimized for 128x128 RGB images
2. Achieve greater than 80% validation accuracy on an 80/20 train and validation respective split
3. Implement robust data augmentation and training procedures
4. Develop a comprehensive test suite validating model correctness
5. Analyze model behavior, failure modes, and performance characteristics

This work demonstrates key CNN design principles including progressive feature extraction, regularization strategies, and learning rate scheduling.

II. Background & Related Work

A. Dataset Description

The Natural Images dataset contains 6,899 images across 8 numerically balanced classes. Images show real world variance in regards to lighting, object scaling, backgrounds, and perspective. Key characteristics include but are not limited to:

- **Resolution:** Resized to 128x128 during preprocessing
- **Split:** 80% training (5,519 images), 20% validation (1,380 images)
- **Classes:** Airplane, car, cat, dog, flower, fruit, motorbike, person
- **Format:** RGB (3 channels)
- **Normalization:** ImageNet statistics (mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])

The 80/20 split was implemented using deterministic random seeding (seed=42) for reproducibility. Analysis of class distribution confirms the mentioned balanced representation across categories, which eliminates bias concerns.

B. Related Work

Traditional Approaches: Before deep learning became popular, image classification usually relied on manually designed features, utilizing methods like SIFT or HOG that involved detected edges, corners, or textures in an image using fixed algorithms. These would then be fed into SVMs, and while some degree of success was achieved, these methods struggled with more complex images or data.

Early CNNs: In 2012, AlexNet demonstrated that including many layers, or deep architecture, could enable the model to learn features or patterns automatically, achieving breakthrough results on the model. VGNet in 2014 also showed that utilizing depth with simple convolutions like 3x3s could improve performance further.

Modern Architectures: ResNet introduced skipping connections to enable training very deep networks (100+ layers). EfficientNet achieved peak modern accuracy through compound scaling of depth, width, and resolution. However, these architectures contain millions of parameters that are unsuitable for educational purposes or deployment on resource constrained devices like microcontrollers.

Compact CNNs: To respond to the need for smaller models fit for resource constraints, MobileNet and SqueezeNet were developed to prove that carefully designed lightweight architectures could achieve comparable or satisfactory accuracy with up to 10-50x less parameters. The architecture designed for this report follows similar practices, with progressive channel expansion, aggressive and global average pooling utilized to reduce model size while maintaining sufficient capacity.

III. Methodology

A. Architecture Design

The CNN created is structured in layers, with four convolutional blocks followed by a small classifier section at the end.

Feature Extraction Pipeline:

1. Input (3x128x128)
2. Conv1(32) + BN + ReLU + MaxPool -> (32x64x64)
3. Conv2(64) + BN + ReLU + MaxPool -> (64x32x32)
4. Conv3(128) + BN + ReLU + MaxPool -> (128x16x16)
5. Conv4(256) + BN + ReLU + MaxPool -> (256x8x8)
6. AdaptiveAvgPool(1x1) -> (256x1x1)

Classification Head:
Flatten -> Dropout(0.5) -> FC(256->128) -> ReLU -> Dropout(0.5) -> FC(128->8)

Design Rationale:
A multitude of design choices were made when creating the model architecture, which includes but is not limited to:

1. **Progressive Channel Expansion (32->64->128->256)**, where earlier layers capture more simpler features like edges and textures while deeper layers capture more complex ones.
2. **Batch Normalization**, occurring after each convolution and before the activation function, normalizing the values and making the training more stable, allowing the use of higher learning rates.
3. **MaxPooling (2x2)**, which downsamples and reduces the size while keeping the important features and filtering noise. It also provides translation invariance, which reduces the criticality of a feature's exact position
4. **Adaptive Average Pooling**, which collapses 8x8 feature maps into a 1x1, providing flexibility to different resolutions while also reducing parameters needed compared to fully connected layers on the previously mentioned 8x8 maps.
5. **Dropout (p=0.5)**, which prevents overfitting by make sure the model does not rely on any one feature, basically forcing an ensemble method for the network to find multiple ways of identifying an object or pattern

B. Data Augmentation Strategy

Training and validation datasets utilized separate transformation pipelines. In regards to training datasets, the images were augmented with modifications listed below to make the model's recognition capabilities more robust to variations in real world images.

Training Augmentations:

- Random horizontal flips (p=0.5): Increases dataset diversity
- Random rotation (±10°): Handles viewpoint variations
- Color jitter (brightness/contrast/saturation ±20%): Improves robustness to lighting
- Normalization with ImageNet statistics

Meanwhile images in the validation dataset were only given the transformations listed below to emulate performance with realistic images given.

Validation Transforms:

- Resize only (no augmentation)
- Same normalization as training

This approach to both the training and validation datasets helped prevent data leakage while maximizing the variability applied in training to make the model more robust. Normalizing the images with methods used in big datasets like ImageNet help facilitate transfer learning if pre-trained weights are included later on.

C. Training Protocol

The choice of optimization algorithms, loss functions, and learning rate scheduling used during model training is shown below, with additional notes if not discussed prevalently in class:

Optimizer: AdamW (weight_decay=1e-4)

- Adam's adaptive learning rates allow handling of different parameter scales
- Weight decay provides L2 regularization independent of gradients

Loss Function: CrossEntropyLoss
Learning Rate Schedule: ReduceLROnPlateau (mode='max', factor=0.5, patience=2)

- Monitors validation accuracy
- Reduces LR by 50% if no improvement for 2 epochs

Hyperparameters:

- Initial learning rate: 1e-3
- Batch size: 64
- Epochs: 15
- Seed: 42 (for reproducibility)

IV. Experimental Setup & measurements

A. Implementation Details

The implementation uses PyTorch 2.10 with CUDA 12.8 support. All experiments ran on a single GPU. Key implementation decisions:

- Deterministic seeding: torch.backends.cudnn.deterministic=True ensures reproducible results across runs
- Pin memory: Enabled for GPU training to accelerate data transfer
- Num workers: 4 parallel data loading processes

B. Test-Driven Development Approach

A variety of test files implemented utilizing pytest helped validate accuracy and consistently at multiple different points across the model:

1. **Unit Tests:**
   - *test_model_shapes.py:* Feeding sample inputs into the model produces the correct output shape

- *test_data_basic.py:* Validation of correct image and label loading
- *test_config.py:* Ensures that setting random seeds outputs deterministic results, very important for the reproducibility portion of the project grade

2. **Gradient Tests:**
   - *test_loss_and_gradients.py:* Confirms that for a forward pass and backpropagation, the gradients are not zero and loss decreases as expected
3. **Overfitting Tests:**
   - *test_tiny_overfit.py:* Ensures the model has the capacity by training it on a very small subset of the dataset and evaluating to where bottlenecks would not be the issue if problems arise
4. **Integration Tests:**
   - *test_training_loop.py:* Confirmation of weight change after a few training iterations
   - *test_smoke_train.py:* Checking of expected behavior by training for a few epochs on a sample of data and evaluating it to reach a respective minimal accuracy
5. **Transform Validation:**
   - *test_transformers.py:* Verifies that training data augmentation is not being applied to validation data, checking for data leakage

This testing practice follows QA development principles, catching bugs early and providing regression detection.

C. Baseline Comparison

Random guessing baseline for an 8 class balanced dataset would be around 12.5% accuracy while a well initialized untrained CNN typically achieves 8-15% due to random weight initialization biases.

D. Primary Metrics

Validation Accuracy:

- Direct measure of classification correctness
- Reported as fraction of correctly predicted samples
- Threshold: above 80% for project success

Training/Validation Loss:

- CrossEntropyLoss values monitor optimization progress
- Divergence between train and validation loss indicates overfitting

E. Metric Rationale

Since the classes were balanced, the use of validation accuracy and training/validation loss was sufficient enough, but if the dataset was imbalanced, more comprehensive metrics would be needed such as recall, precision, and F1 score per class.

F. Limitations

- Accuracy treats all errors equally: Misclassifying "cat" as "dog" may be more forgivable than "cat" as "airplane"
- No confidence calibration: Model may be overconfident on certain classes
- Single number summary: Doesn't reveal which classes struggle

Further work on the model could incorporate confusion matrices to resolve the above issues as well as the per class metrics mentioned in section E for more thorough analysis.

V. Results & Analysis

A. Quantitative Results

Epoch	Train Loss	Val Loss	Val Accuracy	Learning Rate
1	1.4325	0.8404	69.76%	1e-3
5	0.5448	0.5494	77.66%	1e-3
10	0.3881	0.3683	84.77%	1e-3
15	0.2762	0.2142	92.02%	5e-4
20	0.2420	0.2440	89.99%	5e-4
24	0.2239	0.1920	92.60%	2.5e-4

Key Observations:

1. *Rapid Initial Learning*, where accuracy jumped from random (12.5%) to 69.76% in one epoch, indicating effective architecture and initialization
2. *Steady Improvement:* Model consistently improved to epoch 15 before plateauing, indicating minimal effectiveness beyond mentioned epoch
3. *Learning Rate Schedule Effectiveness*, where reductions occurring around epochs 10-15 after stagnation which enabled continued optimization, improving from 88.47% to 92.60%
4. *Minimal Overfitting:* Validation loss tracked training loss closely, with only a minor divergence. Dropout and weight decay effectively regularized the model
5. *Target Achievement:* Final 92.60% accuracy substantially exceeds the 80% threshold

B. Error Analysis

While confusion matrices were not implemented, debug_training.py provides a deeper analysis, which helped debug the initial problem of not providing the correct file path initially and utilizing the script to see that only two classes were being read, indicating that file path had to move one directory deeper. However, we can assume or expect challenging pairs such as:

- **Cat vs Dog:** Both are furry quadrupeds with similar poses
- **Car vs Motorbike:** Shape similarities, especially from certain angles
- **Airplane vs Person:** If the background is dominant (sky for planes, varied for people)

The model's high accuracy (92%) suggests it learned discriminative features beyond simple texture or color, likely capturing shape characteristics.

C. Diagnostic Insights from Test Suite

1. **test_tiny_overfit.py**
   - **Success:** Model achieved >99% accuracy on 32 samples, confirming sufficient capacity and ruling out architectural bottlenecks.
2. **test_transformers.py**
   - **Validation:** Confirmed separate transform objects for train/val, preventing data leakage from augmentations
3. **test_loss_and_gradients.py:**

- Verified loss reduction over 10 steps on a single batch, confirming proper gradient flow

These tests validate that performance stems from correct implementation, not accidental bugs that produce inflated metrics.

D. Comparison to Baseline and Expectations

- **vs Random (12.5%):** 7.4x improvement
- **vs Untrained CNN (~10%):** 9x improvement

For a from-scratch CNN on 128x128 images with ~5,500 training samples, 92% is a strong performance. Models like ResNet and EfficientNet would likely achieve 95-98%, but at the cost of millions of parameters versus this model.

E. Limitations and Future Work

Current Limitations:

1. **Dataset Size:** 6,899 images is modest. Current performance has plateaued at ~90% and could improve or make the model more robust if additional data is implemented
2. **Single-Scale Training:** Currenting training was performed at a fixed image size of 128x128, adding multi-scale training could improve handling of different resolution images
3. **No Test Set:** Evaluation was based on validation set only. A more robust approach would be to include a test set to better estimate generalization
4. **Architecture Search:** This architecture was utilized due to the constraints of the project requirements and current best practices, but a more optimal method would be to utilize neural architecture search (NAS)
5. **Class Imbalance Not Explored:** It is assumed the classes were balanced, which in this case was true enough to where validation accuracy was sufficient to utilize as a model metric, however, like mentioned before it would be ideal to include other model accuracy metrics as well as adjusting the model and training strategy to accommodate for those cases.

Future Directions:

- **Transfer Learning:** Initialize with ImageNet pre-trained weights, fine-tune top layers
- **Advanced Augmentation:** Implement AutoAugment, RandAugment, or CutMix
- **Ensemble Methods:** Train multiple models with different seeds, average predictions
- **Attention Mechanisms:** Add channel/spatial attention to focus on discriminative regions
- **Deployment Optimization:** Quantize to INT8, prune redundant filters for mobile deployment like on microcontrollers for current research study

VI. Conclusion

This project successfully designed, implemented, and validated a CNN for natural image classification, achieving 92.60% validation accuracy. The architecture implements modern best practices which include batch normalization, adaptive pooling, and dropout regularization, while maintaining computational efficiency. Key contributions include:

1. **Clean Implementation:** Modular design with separate config, model, data, and training components
2. **Comprehensive Testing:** 9 test files covering unit, integration, and regression scenarios
3. **Effective Training Protocol:** Learning rate scheduling and augmentation strategies that prevent overfitting
4. **Reproducibility:** Deterministic seeding and documented hyperparameters enable result reproduction

The high accuracy demonstrates that relatively simple CNNs can achieve strong performance on structured visual tasks when properly designed and trained. This work provides a foundation for more advanced projects including multi-task learning, few shot classification, or domain adaptation.

The test driven development approach proved invaluable for catching bugs early (e.g., shared transform objects between train/val) and building confidence in results. Adoption of similar validation strategies is ideal for reliable research outcomes, but may not be plausible due to individual time and resource constraints.

Practical Impact: This model could serve as a lightweight baseline for applications requiring on-device image classification, such as wildlife monitoring cameras, agricultural produce sorting systems, or mobile apps with privacy constraints requiring local inference. Application towards fall detection was also considered, but withdrawn as RNNs like LSTMs and GRUs are better suited to the task.

References

[1] P. Roy, “Natural Images Dataset,” Kaggle, 2018. [Online]. Available: https://www.kaggle.com/datasets/prasunroy/natural-images.

[2] S. R. Richter, A. Shai, and Z. Zhou, “Image Classification Using Convolutional Neural Networks,” Stanford CS231n Project Report, Stanford University, 2017. Available: https://cs231n.stanford.edu/reports/2017/pdfs/426.pdf.