Animal Classification Model Documentation: Problem Analysis and Solutions

Technical Report

April 17, 2025

Contents

1 Initial Problem Analysis

1.1 Original Issue

The initial CNN model for animal image classification demonstrated poor performance with an accuracy rate of only 4 correctly identified animals out of 19 test samples (∼21% accuracy). The model particularly struggled with recognizing cats, sheep, and elephants.

1.2 Root Causes Identified

After analyzing the code and performance, several issues were identified:

1. **Limited Image Resolution**: The model processed images at only 32×32 pixels, which is insufficient for capturing the detailed features necessary to distinguish between various animal species.
2. **Simple CNN Architecture**: The original architecture consisted of only 3 convolutional layers with relatively few filters, limiting the model's ability to learn complex features.
3. **Insufficient Training**: The model was trained for only 20 epochs without any learning rate adjustments, which may not have been enough to reach optimal performance.
4. **Basic Data Augmentation**: While some augmentation was implemented, it wasn't comprehensive enough to generate sufficient training variety.
5. **Potential Class Imbalance**: The training dataset likely contained an uneven distribution of examples across different animal classes, causing bias toward over-represented classes.
6. **Lack of Validation During Training**: The original training process did not include validation steps to monitor performance on unseen data.

2 Implemented Solutions

2.1 Image Processing Improvements

- **Increased Resolution**: Image size was increased from 32×32 to 64×64 pixels to capture more detailed features.
- **Enhanced Data Augmentation**:

```
transforms.Compose([
    transforms.Resize((64, 64)),  #     from 32 32
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(20),  #     from 10
    transforms.RandomCrop(64, padding=8),  # Adjusted for new size
    transforms.ColorJitter(brightness=0.3, contrast=0.3, saturation=0.2, hue
        =0.1),  # Added more parameters
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])
```

2.2 CNN Architecture Enhancements

- **Added Fourth Convolutional Block**: Increased depth for better feature extraction
- **Increased Filter Counts**: Expanded from (32→64→128) to (64→128→256→512)
- **Improved Fully Connected Layers**: Added an additional layer for better classification

```
class ImprovedCNN(nn.Module):
    def __init__(self, num_classes):
        super(ImprovedCNN, self).__init__()
        # First block
        self.conv1 = nn.Conv2d(3, 64, 3, padding=1)
        self.bn1 = nn.BatchNorm2d(64)

        # Second block
        self.conv2 = nn.Conv2d(64, 128, 3, padding=1)
        self.bn2 = nn.BatchNorm2d(128)

        # Third block
        self.conv3 = nn.Conv2d(128, 256, 3, padding=1)
        self.bn3 = nn.BatchNorm2d(256)

        # Fourth block (added)
        self.conv4 = nn.Conv2d(256, 512, 3, padding=1)
        self.bn4 = nn.BatchNorm2d(512)

        self.pool = nn.MaxPool2d(2, 2)
        self.dropout = nn.Dropout(0.4)  # Adjusted dropout rate

        # Expanded fully connected layers
        self.fc1 = nn.Linear(512 * 4 * 4, 512)
        self.fc2 = nn.Linear(512, 256)
        self.fc3 = nn.Linear(256, num_classes)
```

2.3 Training Process Improvements

- **Learning Rate Scheduler**: Added ReduceLROnPlateau to dynamically adjust learning rate

```
scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, 'min', patience=3,
    factor=0.5)
```

- **Weight Decay**: Added L2 regularization to prevent overfitting

```
optimizer = optim.Adam(model.parameters(), lr=0.001, weight_decay=1e-5)
```

- **Early Stopping**: Implemented to prevent overfitting by monitoring validation loss
- **Extended Training Duration**: Increased from 20 to 50 epochs with early stopping criteria

2.4 Class Imbalance Handling

- **WeightedRandomSampler**: Added to balance class representation during training

```
# Calculate class weights for balanced sampling
class_counts = [0] * len(classes)
for _, label in trainset:
    class_counts[label] += 1

weights = [1.0 / class_counts[label] for _, label in trainset]
sampler = WeightedRandomSampler(weights, len(weights))

# Use sampler in dataloader
trainloader = torch.utils.data.DataLoader(
    trainset,
    batch_size=32,
    sampler=sampler,
    num_workers=2
)
```

2.5 Monitoring and Evaluation Improvements

- **Validation During Training**: Added a separate validation phase during training
- **Class-wise Performance Tracking**: Implemented per-class accuracy metrics
- **Confusion Matrix**: Added for detailed error analysis
- **Visualization**: Added training/validation curves and sample predictions

3 Enhanced Prediction Interface

3.1 GUI-Based Prediction Tool

Developed an interactive GUI application with the following features:

- File browser for selecting custom images
- Visual display of prediction results with confidence bars
- User feedback system for marking predictions as correct/incorrect

3.2 Statistical Reporting

- Real-time tracking of prediction accuracy
- Final report with:
  - Total number of predictions made
  - Correct vs. incorrect predictions
  - Overall accuracy percentage
  - Per-class prediction statistics

3.3 User Experience Improvements

- Continuous prediction loop with option to continue or exit
- Error handling for image loading issues
- Visual confidence indicators for predictions

4 Results and Impact

After implementing these improvements, the model was expected to show significant increases in accuracy across all animal classes, especially for previously problematic categories like cats, sheep, and elephants.

The enhanced prediction interface provided a user-friendly way to test the model and collect accurate performance statistics, enabling ongoing assessment and further refinement of the classification system.