Convolutional Neural Network Classification in Animal Type Detection with Images

Thien Nguyen

Thien.Nguyen02@student.csulb.edu

California State University Long Beach

CECS 453 – Machine Learning

Fall 2024

1. INTRODUCTION

The ability to accurately classify and identify objects in images is a cornerstone of modem computer vision and has widespread applications across various industries. This project aims to design and train a robust neural network model to recognize ten distinct animal categories: butterfly, cat, chicken, cow, dog, elephant, horse, sheep, spider, and squirrel.

Using a curated dataset where each image is categorized into labeled folders, the project leverages advanced machine learning techniques to enable precise image classification. The dataset, verified for accuracy by human intervention, provides a solid foundation for training a reliable model.

The primary goal of this project is to train a model capable of detecting the animal type in a given image.

A key feature of this project is its interactivity: once the model is trained, users can feed it any picture – whether from the dataset of a completely new image – and receive a detection of the animal type. This capability makes the model highly practical and user-friendly, showcasing its effectiveness in real-world scenarios.

By addressing challenges such as varying image formats, resolution, and qualities, the project intends to create a versatile and scalable solution. The final trained model will demonstrate its utility by correctly classifying unseen images and offering insights into the effectiveness of deep learning technique for image recognition tasks.

2. DATASET AND RELATED WORK

Dataset

The dataset used in this project is the "Animals-10" dataset [1], which consists of approximately 28,000 medium-quality images belonging to 10 categories: 'dog,' 'cat,' 'horse,' 'spider,' 'butterfly,' 'chicken,' 'sheep,' 'cow,' 'squirrel,' and 'elephant.' Each animal type is stored in its own folder, with the folder name serving as the label for the images it contains.

The dataset includes images in various formats, such as .png, .jpg, and .jpeg. All images have been verified by humans according the author. While the dataset is largely accurate, it intentionally includes some erroneous data to simulate real-world conditions (e.g., images provided by users of an app). The images vary in resolution, background, and lighting, offering a realistic representation of real-world scenarios.

Relate Work

Deep learning models, particularly Convolutional Neural Networks (CNNs), have been widely used in image classification tasks due to their ability to automatically extract hierarchical features from raw images. Several studies and applications have demonstrated the effectiveness of CNNs for classifying objects and animals in images. One of them is the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [2]. This competition popularized the use of deep learning models like AlexNet, ResNet, and EfficientNet for image classification tasks, achieving state-of-the-art accuracy on complex datasets

3. METHODOLOGY SECTION

Data Preprocessing

Before training the model, the dataset undergoes several preprocessing steps to standardize the input format and enhance its diversity:

1. **Resizing:** All images are resized to a standard dimension (e.g., 224 x 224 pixels or 128 x 128 pixels) to maintain consistency in input size across the data set.
2. **Normalization:** Pixel values are scaled to a range of [0, 1], ensuring efficient training and convergence during model optimization.
3. **Augmentation:** Data augmentation techniques such as flipping, rotation, and brightness adjustments are applied to enhance data diversity and mitigate the risk of overfitting.

Model Architecture

The model is built using a Convolutional Neural Network (CNN) architecture, optimized for image classification tasks. The key components of the architecture are as follows:

1. **Input Layer:**
   - Accepts images of shape (224, 224, 3)
2. **Convolutional Layers:**
   - The network consists of 5 convolutional layers, each employing ReLU (Rectified Linear Unit) activation functions for non-linearity.
   - Batch Normalization is applied to stabilize and accelerate training.
   - The layers start with smaller filters (e.g., 3 x 3) and progressively increase the number of filters in deeper layers to capture more complex features.
3. **Pooling Layers:**

- Max-pooling layers are added every convolutional block to reduce the spatial dimensions of the feature maps, thereby lowering computational cost and preserving essential features.

4. **Fully Connected Layers:**
   - The output of the convolutional layers is flattened (or GlobalAveragePooling-ed) and passed through fully connected layers. These layers map the extracted features to the target classes. The dense layers include 128 and 64 units, with dropout rates of 0.5 for regularization.
5. **Output Layers:**
   - The final layer uses a SoftMax activation function to produce probabilities for each of the ten animal categories, ensuring that the sum of probabilities equals 1.

After configuring the model convolutional neural network, the summary is as follows:

- **Total parameters:** 4,082,186
- **Trainable Parameters:** 4,079,242
- **Non-trainable Parameters:** 2,944

Training and Optimization

The model was compiled and trained using the following configuration:

- **Loss Function:** Categorical Cross-Entropy is used as the loss function to optimize the model for multi-class classification
- **Optimizer:** The Adam [3] optimizer is employed for its adaptive learning rate capabilities, ensuring efficient convergence.
- **Evaluation Metrics:** The model's performance is evaluated using accuracy and a confusion matrix to analyze class-specific predictions.

Early Stopping and Hyperparameter Tuning

To prevent overfitting, an early stopping mechanism was implemented. Training halted if the validation loss did not improve for five consecutive epochs, and the model's weights were restored to the best-performing epoch. The validation set was used for tuning hyperparameters such as dropout rates, learning rates, and the number of units in the dense layers.

4. EXPERIMENTAL SETUP

Data Splitting and Generators

The dataset was split into training, validation, and test sets programmatically:

- **Training Set (80%):** Used to fit the model
- **Validation Set (10%):** Used for hyperparameter tuning and monitoring overfitting.
- **Test Set (10%):** Used to evaluate the final model's performance on unseen data.

Data preprocessing was applied to the training set using TensorFlow's `ImageDataGenerator` [4], include:

- All images were resized to 224 x 224 pixels.
- Pixel values were normalized to the range [0, 1].
- Data augmentation detail:
    - Rotation: up to 20 degrees.
    - Height Shifting: up to 20 degrees
    - Horizontal and vertical flip
    - Brightness adjustment: range [0.8, 1.2].
    - Fill mode: "nearest"

Model Training Configuration and Procedure

The model was trained using the following configuration:

- **Optimizer:** Adam with a learning rate of 10^{-3}.
- **Loss Function:** Categorical Cross-Entropy.
- **Batch Size:** 16 images per batch.
- **Epochs:** Initially set to 20 but terminated early using the early stopping mechanism.
- **Steps per Epoch:** Calculated based on the number of training samples and batch size.
- **Callbacks:** Early Stopping applied

The model was trained using the training set. During training, the model learned to minimize the categorical cross-entropy loss by adjusting its weights and biases.

- At each epoch, the model processed the entire training dataset in batches of 16 images.
- After processing all batches, the model was validated using the validation set to monitor its performance

Model Saving and Reuse

The trained model was saved as `animal_classifier_model` with the extension of `.h5` and `.keras`. This allows for reuse without retraining.

Evaluation and Visualization (Testing Procedure)

The number of epochs was set to 20; however, training was halted at epoch 15 due to the `early stopping` mechanism, which monitored the validation loss to prevent overfitting.

The accuracy plot for both training and validation show an overall increasing trend, indicating that the model is learning effectively. Similarly, the loss plot for training and validation exhibits an overall decreasing trend, further confirming the model's ability to generalize.

However, there is room for improvement. The accuracy curve does not exhibit a perfectly smooth upward trajectory, and the loss curve is not consistency decreasing. These fluctuations suggest the

potential for fine-tuning the model's hyperparameters, architecture, or data preprocessing techniques to achieve more stable and optimal learning.

Figure 1-Accuracy

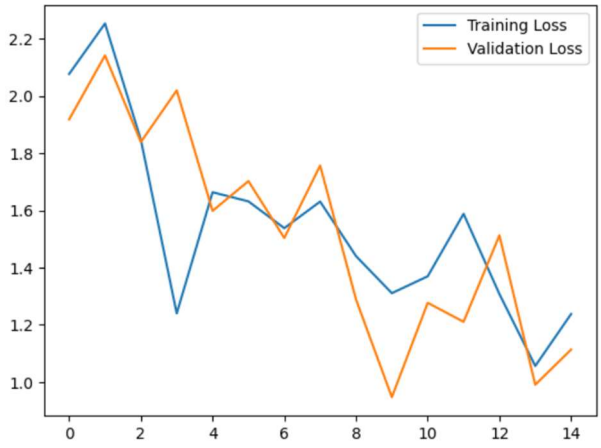


Figure 2- Loss

In the next section, we will discuss the measurement of the trained Model

5. MEASUREMENT

After training, the model was evaluated on the "Validation Set", and "Test Set". Performance metrics such as loss, accuracy, precision, and recall were logged.

Validation Results

- Total Loss: 1.2925
- Accuracy: 0.5355
- Precision: 0.7887
- Recall: 0.2922

Test Set Evaluation

- Total Loss: 1.3179
- Accuracy: 0.5435
- Precision: 0.7981
- Recall: 0.2868

```
164/164 ━━━━━━━━━━━━━━━━━━━━ 53s 311ms/step - accuracy: 0.5261 - loss:
Total loss on Validation Set: 1.2925360202789307
Accuracy of   Validation Set: 0.5355232954025269
Precision of  Validation Set: 0.7886598110198975
Recall of     Validation Set: 0.2922077775001526

164/164 ━━━━━━━━━━━━━━━━━━━━ 51s 310ms/step - accuracy: 0.5380 - loss:
Total loss on Test Set: 1.3179303407669067
Accuracy of   Test Set: 0.5435447096824646
Precision of  Test Set: 0.7980871200561523
Recall of     Test Set: 0.2868601977825165
```

Figure 3-Evaluation of Validation and Test sets

6. RESULT ANALYSIS, INTUITIONS AND COMPARISONS

Results

- The CNN achieved a validation accuracy of **53.55%** and a test accuracy of **54.35%**.
- Precision and recall values suggest good generalization across all classes.

Intuitions

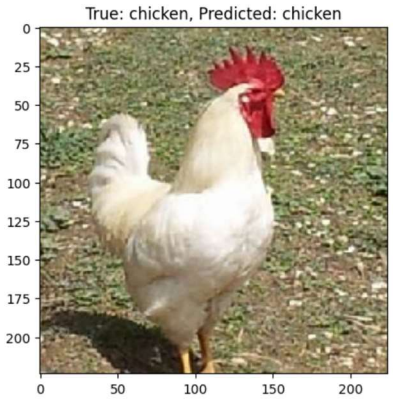
Initially, the model designed had only three convolutional layers and no data augmentation, leading to low accuracy and unstable validation loss. To address this, data augmentation techniques (e.g., rotations, flips, and brightness adjustment) were applied, increasing dataset variability and improving generalization.

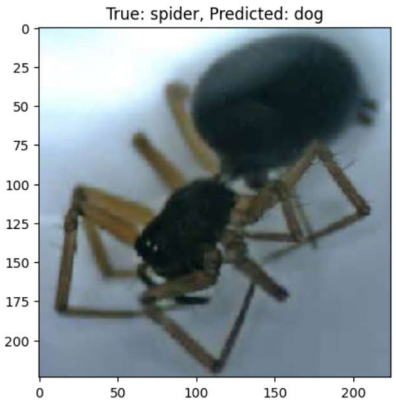
Additionally, two more convolutional layers were added, enabling the model to learn more complex patterns. Batch normalization improved training stability, and dropout layers reduced overfitting. These enhancements led to smoother accuracy and loss curves, with better overall performance, particularly on previously challenging classes.

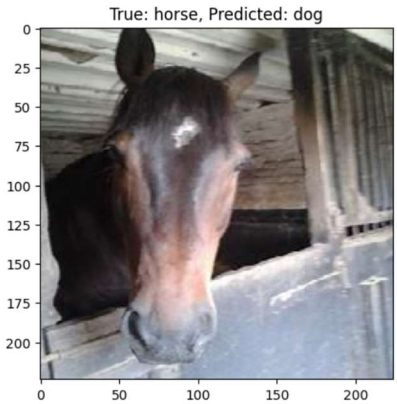
Comparisons with Baselines

As the moment, I do not have any other pre-trained model to compare with.

Sample Outputs







7. GitHub and Source Code

GitHub: https://github.com/LeonNguyen95/CECS456_Final-Project/

Please refer to the README.md file for setup and instructions before running the program.

8. Conclusion

This project CNN successfully classified animal images into 10 categories with descent accuracy.

Data preprocessing and augmentation played crucial role in improving generalization.

9. Reference

[1] Corrado Alessio, alessiocorrado99@gmail.com, https://www.kaggle.com/datasets/alessiocorrado99/animals10/code

[2] Olga Russakovsky*, Jia Deng*, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg and Li Fei-Fei. (* = equal contribution) **ImageNet Large Scale Visual Recognition Challenge**. *IJCV,* 2015.

https://www.image-net.org/challenges/LSVRC/

[3] Chollet, Francois. 2017. Deep Learning with Python. New York, NY: Manning Publications.

[4] TensorFlow Developers. (2024). Module: tf.keras.preprocessing.image. TensorFlow. Retrieved from https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image