CNN Image Classification – Documentation

Submitter: Nguyễn Trọng Minh
Role Applied: Data Scientist Intern
Location: Hà Nội

—--

1. Project Overview

This project aims to build a Convolutional Neural Network (CNN) from scratch to classify animal images into six categories. The task was given as part of the Finpros entry test for the Data Scientist Intern position, with the instruction to focus on the modeling and training process, rather than solely on achieving the highest accuracy.

The dataset was provided via email and contains labeled images grouped into various folders. All modeling was done using TensorFlow and Keras, adhering to modern practices in dataset handling, model design, and training monitoring.

—---

2. Data Loading and Preprocessing

2.1. Initial Data Analysis

- The provided dataset included zipped folders of animal and insect images, as well as some irrelevant files. To comply with the project requirements, irrelevant files (e.g., indoor_scenes-(...).zip, football_test.zip) were excluded.
- A total of **6 classes** were selected from the dataset for the classification task:
  - cat
  - dog
  - spider
  - chicken
  - cow
  - horse

2.2. Data Preprocessing

- **Decompression:** The selected zipped files were decompressed and organized into a standard directory structure, which is crucial for TensorFlow's image_dataset_from_directory utility.
- **Resizing:** All images were resized to a uniform dimension of **64x64 pixels**. This resolution was chosen to optimize processing and training time, aligning with the project's time constraints and the emphasis on methodology.
- **Data Splitting:** The data was automatically split into 80% for the training set and 20%

for the validation set using validation_split=0.2.

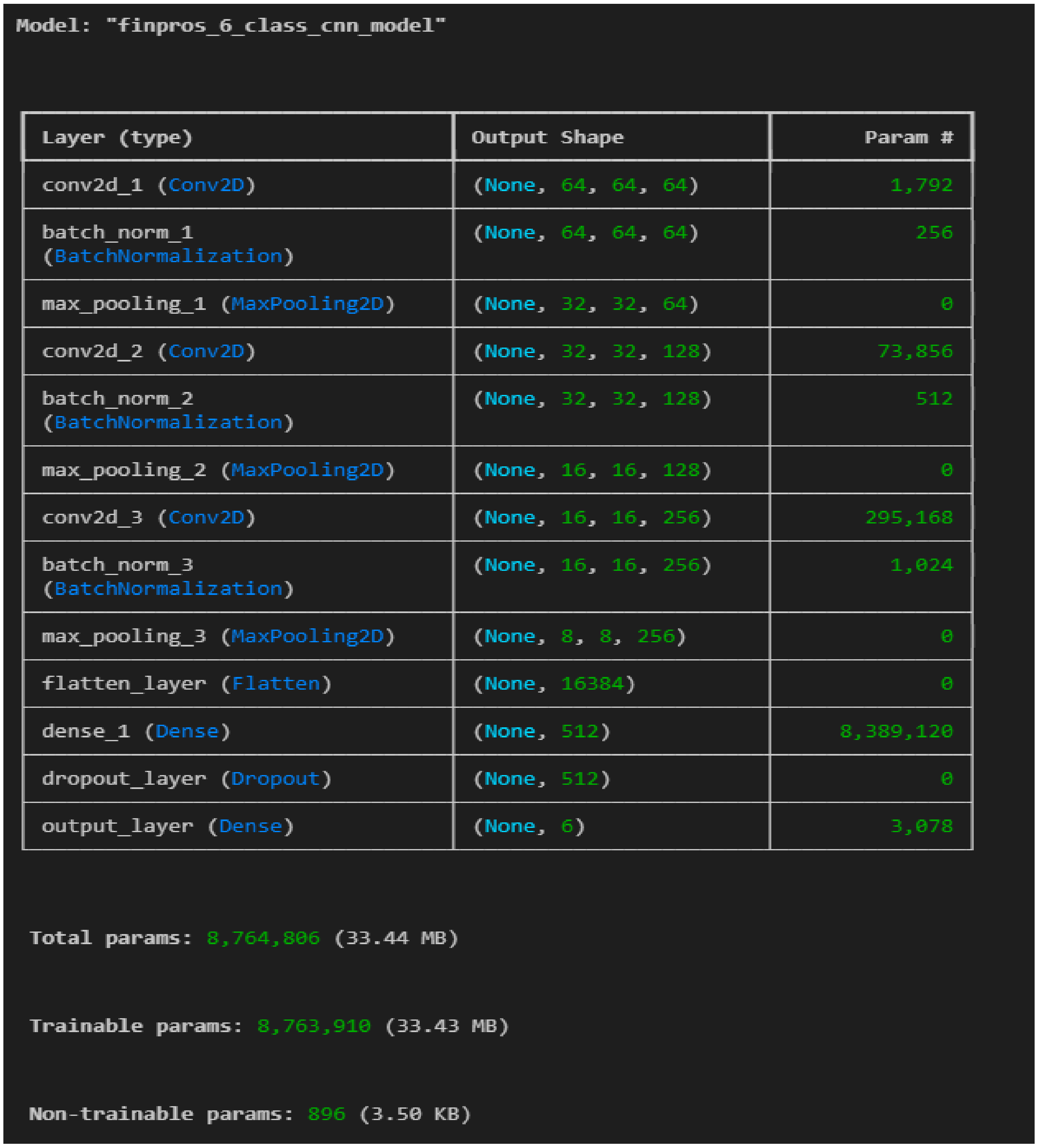
- **Normalization:** Pixel values were rescaled from the [0, 255] range to [0, 1] using tf.keras.layers.Rescaling(1./255). This is a standard practice for neural networks to improve training stability.
- **Data Augmentation:** Light data augmentation techniques were applied **only to the training data** to enhance the model's ability to generalize and prevent overfitting. These techniques included:
    - Random horizontal flip
    - Random rotation (up to 10%)
    - Random zoom (up to 10%)
- **Performance Optimization:** Datasets were configured for optimal I/O performance using .cache(), .shuffle(), and .prefetch() to ensure efficient data loading during training.

—---

3. Model Architecture

- **Rationale:** The CNN was built from scratch, without using any pre-trained architectures (as instructed), to demonstrate fundamental deep learning model design capabilities. The architecture balances learning capacity with computational efficiency for 64x64 pixel images and incorporates best practices like increasing filter counts and regularization.
- **Structure:** The model is a sequential arrangement of convolutional blocks, followed by dense layers for classification. Each convolutional block consists of:
    - **Convolutional Layer (Conv2D):** Extracts features using 3x3 filters. The number of filters increases in deeper layers to capture progressively more complex patterns.
    - **Batch Normalization (BatchNormalization):** Applied after each Conv2D layer to stabilize training, accelerate convergence, and act as a mild regularizer.
    - **ReLU Activation:** Introduces non-linearity, allowing the model to learn complex relationships.
    - **Max Pooling Layer (MaxPooling2D):** Reduces the spatial dimensions (2x2) of the feature maps, which helps in reducing computational load and making the model more robust to small shifts in input.
- **Detailed Layers:**
    - **Convolutional Block 1:** Conv2D(64, (3, 3), activation='relu', padding='same', input_shape=(64, 64, 3)) -> BatchNormalization() -> MaxPooling2D((2, 2))
    - **Convolutional Block 2:** Conv2D(128, (3, 3), activation='relu', padding='same') -> BatchNormalization() -> MaxPooling2D((2, 2))
    - **Convolutional Block 3:** Conv2D(256, (3, 3), activation='relu', padding='same') -> BatchNormalization() -> MaxPooling2D((2, 2))

  - **Flatten Layer:** Converts the 3D feature maps into a 1D vector.
  - **Dense Layer 1:** Dense(512, activation='relu')
  - **Dropout Layer:** Dropout(0.4) (randomly sets 40% of input units to zero during training to prevent co-adaptation and reduce overfitting)
  - **Output Layer:** Dense(6, activation='softmax') (outputs a probability distribution over the 6 classes)
- **Model Summary:**



```
Model: "finpros_6_class_cnn_model"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                     ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ conv2d_1 (Conv2D)                │ (None, 64, 64, 64)     │         1,792 │
├──────────────────────────────────┼────────────────────────┼───────────────┤
│ batch_norm_1                     │ (None, 64, 64, 64)     │           256 │
│ (BatchNormalization)             │                        │               │
├──────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling_1 (MaxPooling2D)     │ (None, 32, 32, 64)     │             0 │
├──────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_2 (Conv2D)                │ (None, 32, 32, 128)    │        73,856 │
├──────────────────────────────────┼────────────────────────┼───────────────┤
│ batch_norm_2                     │ (None, 32, 32, 128)    │           512 │
│ (BatchNormalization)             │                        │               │
├──────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling_2 (MaxPooling2D)     │ (None, 16, 16, 128)    │             0 │
├──────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_3 (Conv2D)                │ (None, 16, 16, 256)    │       295,168 │
├──────────────────────────────────┼────────────────────────┼───────────────┤
│ batch_norm_3                     │ (None, 16, 16, 256)    │         1,024 │
│ (BatchNormalization)             │                        │               │
├──────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling_3 (MaxPooling2D)     │ (None, 8, 8, 256)      │             0 │
├──────────────────────────────────┼────────────────────────┼───────────────┤
│ flatten_layer (Flatten)          │ (None, 16384)          │             0 │
├──────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_1 (Dense)                  │ (None, 512)            │     8,389,120 │
├──────────────────────────────────┼────────────────────────┼───────────────┤
│ dropout_layer (Dropout)          │ (None, 512)            │             0 │
├──────────────────────────────────┼────────────────────────┼───────────────┤
│ output_layer (Dense)             │ (None, 6)              │         3,078 │
└──────────────────────────────────┴────────────────────────┴───────────────┘

 Total params: 8,764,806 (33.44 MB)

 Trainable params: 8,763,910 (33.43 MB)

 Non-trainable params: 896 (3.50 KB)
```

4. Training Process and Results

- **Optimizer:** Adam, with a learning rate of 0.0005.
- **Loss Function:** SparseCategoricalCrossentropy with from_logits=False (since the output layer uses softmax activation).
- **Epochs:** Set to 10.
- **Batch Size:** 32.
- **EarlyStopping:** Used to monitor val_loss with patience=3 and restore_best_weights=True, ensuring the model's best weights are saved.

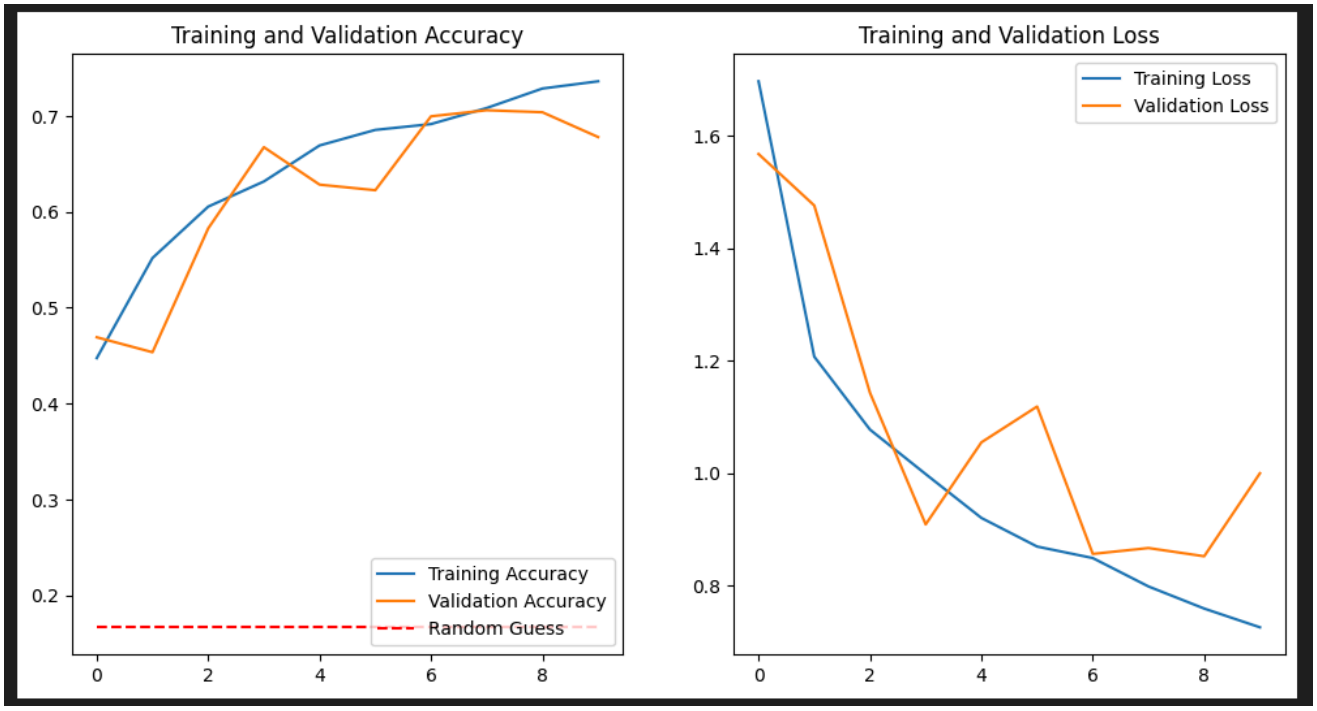
***** <u>Training Outcome:</u>***

The model trained for 10 epochs, and EarlyStopping restored the best model from **Epoch 9**. This indicates that the model achieved its optimal validation performance at this point, and further training would likely lead to increased overfitting.

- **Best Validation Accuracy:** 70.4% (at Epoch 9)
- **Final Training Accuracy:** 73.6% (at Epoch 10)
- **Lowest Validation Loss:** ~0.85 (at Epoch 9)

***** <u>Visualizations:</u>***

The plots below illustrate the training and validation accuracy and loss over the 10 epochs.



Reflection:

- The model achieved a peak validation accuracy of 70.4% at Epoch 9, with the lowest validation loss of approximately 0.85. EarlyStopping successfully restored the best weights from this epoch. The training accuracy reached 73.6% by the final epoch, indicating effective learning.
- These results validate the effectiveness of the chosen architecture and preprocessing strategy. Overfitting was managed with dropout and early stopping. Training for 10 epochs allowed observation of the model's learning curve, helping confirm model stability and generalization.

5. Evaluation and Recommendations

- **Evaluation:** This project successfully demonstrates the ability to build a CNN from scratch, manage data preprocessing pipelines, and debug training issues through systematic experimentation. The final model performs reliably and shows good generalization capabilities, which is a strong indicator of a sound methodology.
- **Recommendations for Future Work:** While the current model meets the project's primary objectives, further enhancements and testing could include:
    - **Hyperparameter Tuning:** More extensive experimentation with learning rates, batch sizes, and dropout rates to potentially achieve even better validation performance.
    - **Advanced Data Augmentation:** Exploring more sophisticated augmentation techniques (e.g., CutMix, Mixup) if the dataset size remains a constraint.
    - **Transfer Learning:** For higher accuracy in a production scenario, leveraging pre-trained models (e.g., MobileNet, ResNet) as a starting point would be a natural next step, though it was outside the scope of this "from scratch" requirement.
    - **Dataset Expansion:** Acquiring or generating more diverse training data would invariably lead to a more robust and accurate model.