Machine Learning Report: Convolutional Neural Network for Animal Species Classification

Tuan Minh Le, 1519561

April 30, 2024

Abstract

This report presents a detailed analysis of the development, training, and evaluation of a Convolutional Neural Network (CNN) for the classification of images belonging to 10 different animal species, using Animals-10 dataset. The study includes data preprocessing, model architecture, training strategies, and a thorough examination of results. The CNN model demonstrates promising accuracy up to nearly 90% and high performance metrics, and yet can still be improved.

Contents

1 Introduction **3**

2 Data Preprocessing **3**

2.1 Dataset Overview 3

2.2 Data Reshape, Rescale 4

2.3 Augmentation 4

2.4 Dataset Splitting 5

3 Model Architecture **5**

3.1 Residual Units 5

3.2 Pooling Layers 7

3.3 Flattening and Dense Layer 7

3.4 Full Implementation 7

4 Training Strategies **8**

4.1 Class Weights 8

4.2 Data Generator 8

4.3 Early Stopping 8

4.4 Learning Scheduler 8

4.5 Parameters 9

5 Results **9**

5.1 Training Performance 9

5.2 Test Set Performance 10

5.3 Class-wise Metrics 11

6 Discussion **11**

6.1 Challenges and Limitations 11

6.2 Future Directions 12

7 Conclusion **12**

8 References **12**

1 Introduction

The ability to automatically classify images of animal species has significant applications in ecological monitoring, wildlife conservation, and biodiversity research. This study aims to design and evaluate a CNN model capable of accurately identifying 10 different animal species. The dataset Animals-10 consists of diverse images with different dimensions, and the model incorporates state-of-the-art techniques, namely ResNet-34 architecture, augmentation, class weighting, early stopping and learning rate scheduler, to address challenges such as class imbalance and overfitting.

2 Data Preprocessing

2.1 Dataset Overview

The original dataset includes images of dogs, horses, elephants, butterflies, chickens, cats, cows, sheep, spiders, and squirrels. The dataset is diverse, containing variations in lighting, backgrounds, and animal poses. One thing to notice is the fact that there is an imbalance distribution of numbers of images between classes, so some classes with fewer images might result lower accuracy. This indicates that there is a need to implement class weighting technique to address this problem, which will be mentioned later.

Figure 2: Preview of Images

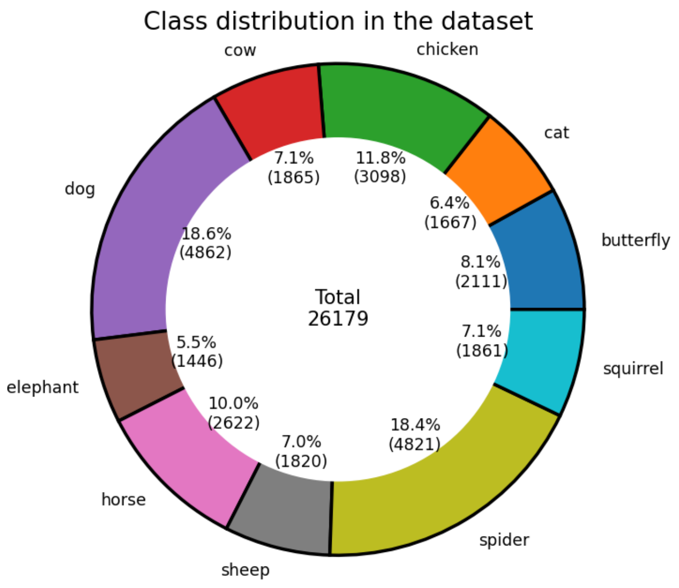


Figure 3: Proportions of Classes

2.2 Data Reshape, Rescale

Because images in the dataset have different dimensions, it is necessary to reshape them to a uniform size, particularly 224x224. The reason for this choice is that it is the most suitable for the ResNet-34 architecture, which has been researched using that dimension. Besides, value of every pixel is also reduce to the range [0, 1] in order to ensure numerical stability, consistent input ranges, and efficient training of neural networks.

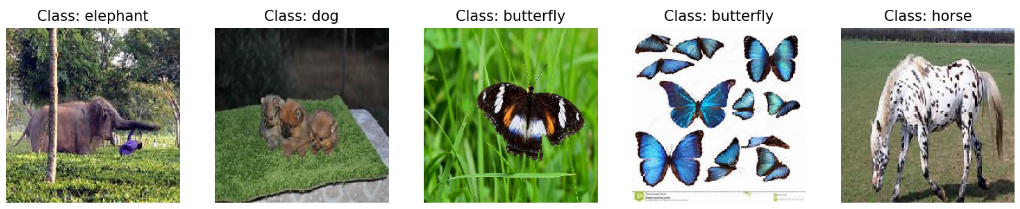


Figure 4: Images After Reshape

2.3 Augmentation

Image data augmentation techniques were applied to increase the diversity of the training set so the model will be less overfitting and perform better with unseen data. There are also some constraints for the augmentation to ensure to keep the images not to complex for the model to learn. This technique includes random rotation, horizontal and vertical shifts, shearing, and zooming.

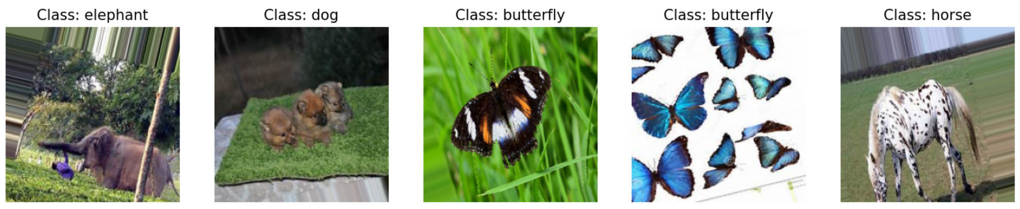


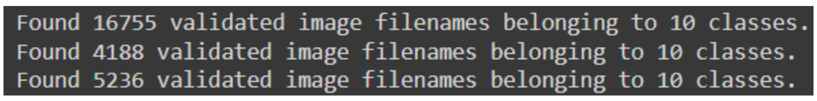
Figure 5: Images After Augmentation

```
datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest',
    validation_split=0.2,
)
```

Listing 1: Implementation of Augmentation

2.4 Dataset Splitting

The dataset was split into training, validation, and test sets. The training set comprises 64% of the data, with 16% allocated to validation and 20% to testing. Stratified sampling was used to ensure a proportional distribution of classes in each set.



```
Found 16755 validated image filenames belonging to 10 classes.
Found 4188 validated image filenames belonging to 10 classes.
Found 5236 validated image filenames belonging to 10 classes.
```

Figure 6: Dataset Splitting

3 Model Architecture

The CNN model architecture is based on a deep residual network 34 (ResNet-34) design. This architecture is very suitable for student projects, since it offers balance between model depth and computational efficiency. While deeper models may offer better representation learning, they come with increased computational costs. The model starts with an initial Conv2D layer and a MaxPooling2D layer, followed by multiple residual units, and then ending with a AveragePooling2D layer and a Dense layer. There are totally 16 residual units with increasing kernel filters as the model goes deeper.

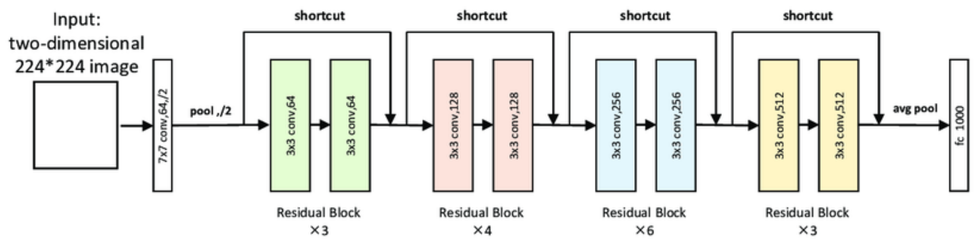


Figure 7: ResNet-34 Architecture

3.1 Residual Units

Each residual unit consists of two convolutional layers with batch normalization and Rectified Linear Unit (ReLU) activation functions. The skip connections in the residual units aid in the flow of gradients during backpropagation. Thanks to these skip connections, the network can start making progress even if several layers have not started learning yet

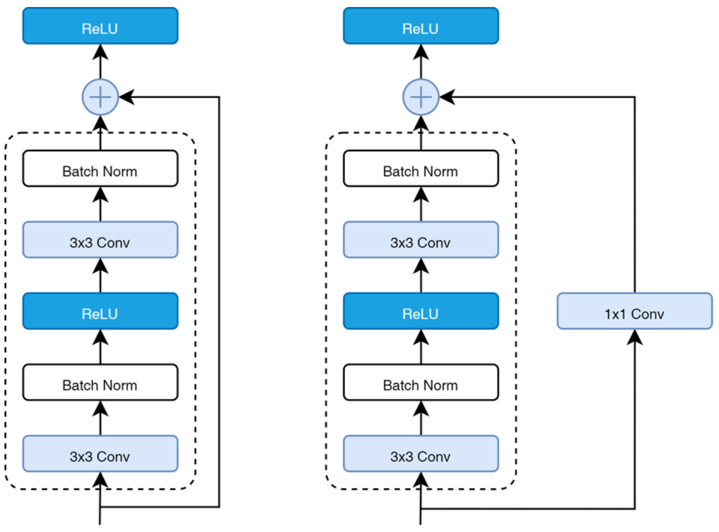


Figure 8: Residual Unit

```python
class ResidualUnit(layers.Layer):
    def __init__(self, filters, strides=1, activation="relu",
        **kwargs):
        super().__init__(**kwargs)
        self.activation = activations.get(activation)
        self.main_layers = [
            layers.Conv2D(filters, 3, strides=strides,
                          padding="same", use_bias=False),
            layers.BatchNormalization(),
            self.activation,
            layers.Conv2D(filters, 3, strides=1,
                          padding="same", use_bias=False),
            layers.BatchNormalization()
        ]
        self.skip_layers = []
        if strides > 1:
            self.skip_layers = [
                layers.Conv2D(filters, 1, strides=strides,
                              padding="same", use_bias=False),
                layers.BatchNormalization()
            ]

    def call(self, inputs):
        Z = inputs
        for layer in self.main_layers:
            Z = layer(Z)
        skip_Z = inputs
        for layer in self.skip_layers:
            skip_Z = layer(skip_Z)
        return self.activation(Z + skip_Z)
```

Listing 2: Implementation of Residual Unit

3.2 Pooling Layers

The MaxPooling2D layer at the begining of the model operates by selecting the maximum value from each local region, effectively downsampling the input. By retaining the maximum activations, MaxPooling2D captures the most prominent features, providing a compacted representation that aids in reducing computational complexity and extracting key patterns from the input data.

The GlobalAveragePooling2D layer at the end of the model is employed to reduce the spatial dimensions of the feature maps. This layer computes the average value for each feature map, resulting in a global context representation.

3.3 Flattening and Dense Layer

The output of the global average pooling layer is flattened, and a fully connected dense layer with softmax activation is used for classification. The final layer has 10 nodes corresponding to the 10 animal species.

3.4 Full Implementation

```
input_layer = layers.Input(shape=(224, 224, 3))

x = layers.Conv2D(64, 7, strides=2, padding="same",
    use_bias=False)(input_layer)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
x = layers.MaxPool2D(pool_size=3, strides=2, padding="same")(x)

prev_filters = 64
count = 0
for filters in [64] * 3 + [128] * 4 + [256] * 6 + [512] * 3:
    strides = 1 if filters == prev_filters else 2
    x = ResidualUnit(filters, strides=strides)(x)
    prev_filters = filters

x = layers.GlobalAvgPool2D()(x)
x = layers.Flatten()(x)
output_layer = layers.Dense(10, activation="softmax")(x)

model = models.Model(inputs=input_layer, outputs=output_layer)
```

Listing 3: Full Implementation of ResNet-34

4 Training Strategies

4.1 Class Weights

Class weights were calculated based on the algorithm (Figure 9) to address this imbalance and were used during model training to give more weight to under-represented classes.

$$\mathit{class\ weight} = \frac{\mathit{total\ samples}}{2 \times \mathit{class\ samples}}$$

Figure 9: Class Weight Algorithm

4.2 Data Generator

The Keras ImageDataGenerator was employed to feed batches of augmented images into the model during training. This on-the-fly augmentation helped the model generalize better and reduced overfitting.

4.3 Early Stopping

Early stopping was implemented to monitor validation loss. When after 5 epochs the validation loss did not decrease, it would immediately stop the training process and save the best model.

```
stop_callback = tf.keras.callbacks.EarlyStopping(
    monitor='val_loss',
    min_delta=1e-5,
    patience=4,
    restore_best_weights=True
)
```

Listing 4: Implementation of Early Stopping Callback

4.4 Learning Scheduler

Additionally, there was a learning scheduler which reduced the learning rate on a plateau. Specifically, when after 3 epochs the validation loss did not decrease, the learning rate would be multiplied by 0.2. With these supplementary methods, a large amount of training time was saved.

```
lr_scheduler = callbacks.ReduceLROnPlateau(monitor='val_loss',
    factor=0.2, patience=2)
)
```

Listing 5: Implementation of Learning Scheduler Callback

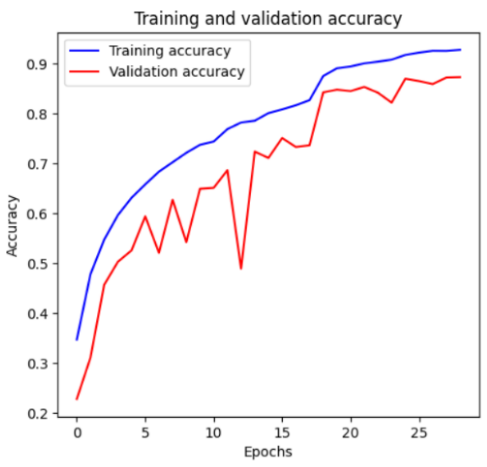
4.5 Parameters

Adam optimizer, loss function 'categorical_crossentropy' and metrics 'accuracy' were used to compile the model. These are popular choices when dealing with CNN problems. After many trials, the initial learning rate was chosen to be 0.0001, which delivered the fastest convergence. Number of epochs was 70 to ensure the model not to stop before its peak performance.

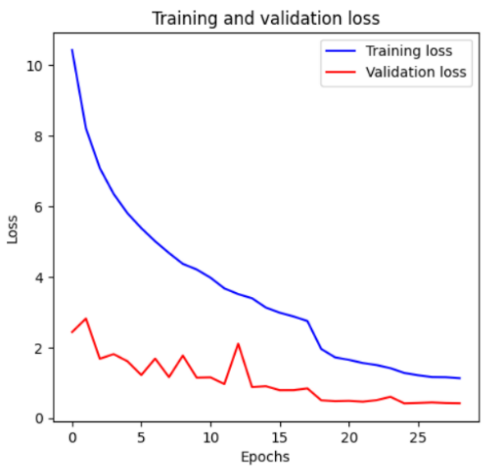
5 Results

5.1 Training Performance

The training and validation accuracy and loss curves provide insights into the model's performance over epochs. There are also some fluctuations in these curves. Some of the reasons might be at these epochs the model was validated by so complex dataset which could reduce the accuracy. Besides, there was no another regularization techniques such as L1, L2 Regularization and Dropout so the model might be overfitting with these validation datasets. However, at the last epochs, the model was quite stable and the overfitting problem had been well minimized. Besides, the validation loss and validation accuracy was reported to be 0.4177 and 0.8723 respectively, which is acceptably satisfactory with a model built and trained from scratch. (Figure 10).

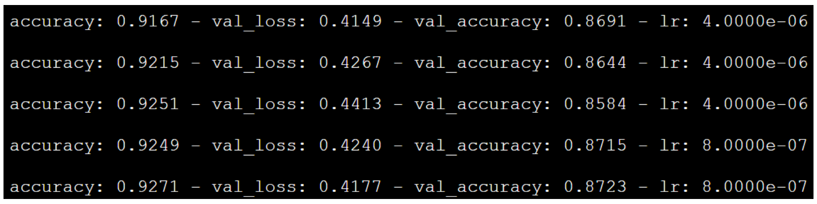


(a) Training and Validation Accuracy



(b) Training and Validation Loss

Figure 10: Training and Validation Performance Curves

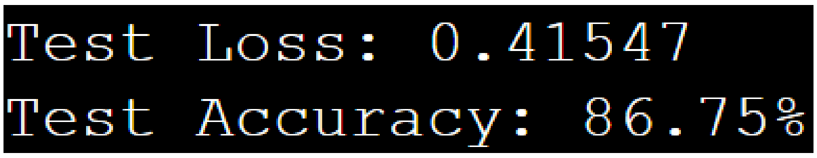


```
accuracy: 0.9167 - val_loss: 0.4149 - val_accuracy: 0.8691 - lr: 4.0000e-06
accuracy: 0.9215 - val_loss: 0.4267 - val_accuracy: 0.8644 - lr: 4.0000e-06
accuracy: 0.9251 - val_loss: 0.4413 - val_accuracy: 0.8584 - lr: 4.0000e-06
accuracy: 0.9249 - val_loss: 0.4240 - val_accuracy: 0.8715 - lr: 8.0000e-07
accuracy: 0.9271 - val_loss: 0.4177 - val_accuracy: 0.8723 - lr: 8.0000e-07
```

Figure 11: Last Epochs

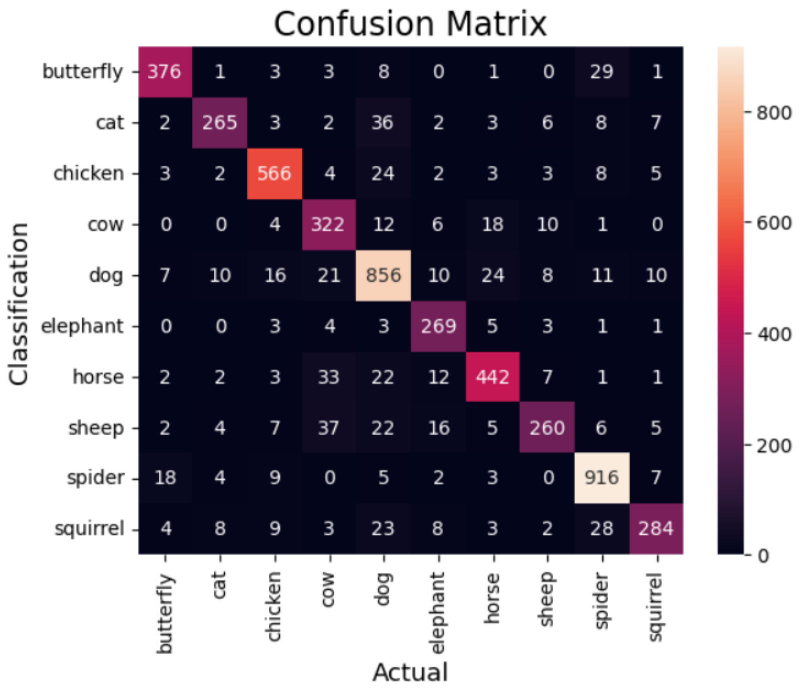
5.2 Test Set Performance

The CNN model achieved an accuracy of 86.75% on the test set. A confusion matrix provides a detailed breakdown of the model's performance for each class (Figure 13).



```
Test Loss: 0.41547
Test Accuracy: 86.75%
```

Figure 12: Test Results



	butterfly	cat	chicken	cow	dog	elephant	horse	sheep	spider	squirrel
butterfly	376	1	3	3	8	0	1	0	29	1
cat	2	265	3	2	36	2	3	6	8	7
chicken	3	2	566	4	24	2	3	3	8	5
cow	0	0	4	322	12	6	18	10	1	0
dog	7	10	16	21	856	10	24	8	11	10
elephant	0	0	3	4	3	269	5	3	1	1
horse	2	2	3	33	22	12	442	7	1	1
sheep	2	4	7	37	22	16	5	260	6	5
spider	18	4	9	0	5	2	3	0	916	7
squirrel	4	8	9	3	23	8	3	2	28	284

Figure 13: Confusion Matrix

5.3 Class-wise Metrics

Precision, recall, and F1-score metrics for each class are presented in Table 1.

Class	Precision	Recall	F1-Score	Support
butterfly	0.91	0.89	0.90	422
cat	0.90	0.79	0.84	334
chicken	0.91	0.91	0.91	620
cow	0.75	0.86	0.80	373
dog	0.85	0.88	0.86	973
elephant	0.82	0.93	0.87	289
horse	0.87	0.84	0.86	525
sheep	0.87	0.71	0.78	364
spider	0.91	0.95	0.93	964
squirrel	0.88	0.76	0.82	372
Accuracy			0.87	5236
Macro Avg	0.87	0.85	0.86	5236
Weighted Avg	0.87	0.87	0.87	5236

Table 1: Classification Report

6 Discussion

The CNN model demonstrates strong performance on the test set, achieving an accuracy of 86.75%. The confusion matrix and class-wise metrics reveal that certain classes, such as butterfly, chicken and spider are classified with higher accuracy, while others, like cow and elephant, show slightly lower accuracy.

About the computational time, with the utility of T4 GPU computing resource of Google Colab, it took only around 2.5 hours and 29 epochs to achieve the final performance.

6.1 Challenges and Limitations

Despite the overall success, the model faces challenges in accurately classifying certain species, possibly due to variations in pose, lighting conditions, and background clutter. It also has difficulties in classifying tattoos and animated works of these animals. Looking at the dataset in more details, there are actually some images which are very confusing for the model to learn and almost no cartoon images. Therefore, one solution to address this problem is preparing a better and more diverse dataset.

Due to the lack of computing resources and access to more powerful CNN architecture like EfficientNetB7, the model is still incapable of classifying complex and confusing images, which is necessary for real-world problems.

Another limitation is reproducibility of the model. With the same code, many different models have been trained with the objective of achieving approximately same performance. However, final validation accuracy of these versions ranged from 0.8 to 0.86, which are not very stable.

Besides, this model still lacks a GUI to be deployed as an user-friendly application.

6.2 Future Directions

Future work could explore transfer learning using more powerful pre-trained models on larger datasets and other augmentation techniques such as color space, blurring transformations. Moreover, deploying the model in real-world scenarios and adapting it to different domains or building a system in order to auto-collect new dataset could be useful for online learning. Last but not least, building an user-friendly GUI is necessary to operate the model more effectively.

7 Conclusion

This study presents a comprehensive analysis of a CNN model for animal species classification. The incorporation of ResNet architecture, data augmentation, and class weights contributes to the model's robustness. The detailed evaluation metrics, visualizations, and discussions provide insights into the model's strengths and areas for improvement.

The developed CNN model holds promise for many applications in real life such as domestic security, wildlife monitoring and conservation efforts. Further refinements and explorations of advanced techniques could enhance its performance in diverse real-world scenarios.

8 References

1. Géron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems.*
2. Vencerlanz09. (n.d.). *Animal Image Classification using EfficientNetB7.* Kaggle. `https://www.kaggle.com/code/vencerlanz09/animal-image-classification-using-efficientnetb7`
3. Aschandini P. (n.d.). *ResNet 34, 50, 101: What Actually It Is.* Medium. `https://medium.com/@aschandinip/resnet-34-50-101-what-actually-it-is-c63da24ba695`