EEL 6825 Project Report

Group member: Xinrun Li, Hang Shu, Yuyang Zhang

Project Name: Benefits of Model Pruning in Animals-10 Classification

Introduction

1. Project Overview

We aim to analyze the benefits of applying model pruning techniques to convolutional neural networks (CNNs) for animal image classification. Specifically, we will compare the performance, efficiency, and resource requirements of a custom CNN and transfer learning CNN models (EfficientNet B7) after different pruning methods with the unpruned model. By focusing on model pruning, we intend to explore how reducing model complexity affects accuracy and efficiency. This helps us address real-world challenges such as limited computing resources and the need for efficient deployment in practical applications such as wildlife monitoring and ecological research.

2. Motivation

In real-world applications such as wildlife monitoring and ecological research, the deployment of deep learning models is often constrained by limited computing resources, especially in remote or resource-constrained environments. High-performance models, such as transfer learning architectures, are computationally intensive and may not be suitable for deployment on devices with limited processing power or memory. One approach to mitigate this issue is model pruning, which focuses on reducing model complexity without significantly affecting accuracy. We aim to apply pruning to models for animal classification tasks to reduce model size and computational requirements while maintaining high classification accuracy. This project allowed us to delve deeper into model pruning techniques and understand how model complexity affects performance on pattern recognition tasks.

3. Background and Related Work

The use of deep learning models for pattern recognition tasks is well established. On Kaggle (a data modeling and data analysis competition platform), there are several CNN models designed to perform Animals-10 classification. Some CNN models have achieved an accuracy of about 75%. These models are relatively simple and require less computing power, but have limitations in terms of robustness to data noise and have difficulty coping with class imbalance. CNN models using transfer learning have achieved an accuracy of about 95%, demonstrating the potential of deep learning models for image classification tasks. However, they require a lot of computing resources and storage space, which makes them less practical for resource-limited applications or situations that require rapid deployment.

Model pruning techniques were first introduced in the late 1980s to reduce the complexity of neural networks without significantly affecting their performance. Model pruning refers to reducing the size of a neural network by eliminating unnecessary or less important parameters (such as weights, neurons, or filters) without significantly affecting the performance of the model. There are several commonly used pruning techniques: Unstructured Pruning, Structured Pruning, Sensitivity-Based Pruning, and Regularization-Based Pruning.

In the Animals-10 Classification model code section published by Kaggle, we did not find any model using pruning. This may be because the authors did not consider the importance of model size for practical application scenarios.

Methodology

1. Approach

- **Theoretical framework**

We first build two baseline models by developing and training a custom CNN and a CNN transfer learning model. The goal of these models are to achieve similar performance to competing models on Kaggle. By tuning hyperparameters and adding batch normalization and dropout to models , the validation accuracy of the model is improved. Performance and resource consumption are evaluated on the test set for comparison. We then apply model pruning techniques to baseline models. We apply weight pruning, where low magnitude weights are removed to reduce model complexity, and filter pruning, where entire filters or channels that contribute the least to the output are eliminated. Different percentages of sparsity are tested for each pruning method. After pruning, we use the best sparsity to retrain the models and evaluate their performance and resource consumption on the test set again to find the most effective model.

- **Methods**

Weight Pruning: Weight pruning involves eliminating the least important weights in the network, essentially setting these weights to zero. Use the Keras package: tensorflow_model_optimization to complete this task.

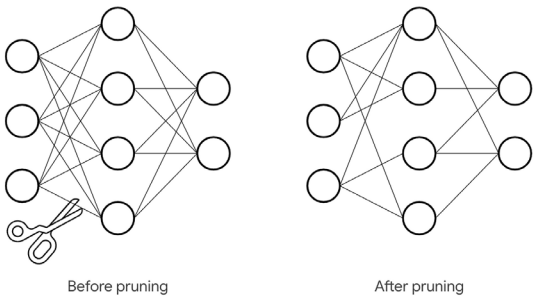

Figure 1. Weight Pruning

Filter pruning: Filter pruning is a compression technique that uses some criterion to identify and remove the least important filters in a network, reducing the overall memory footprint of the network without significant reduction in the network accuracy.

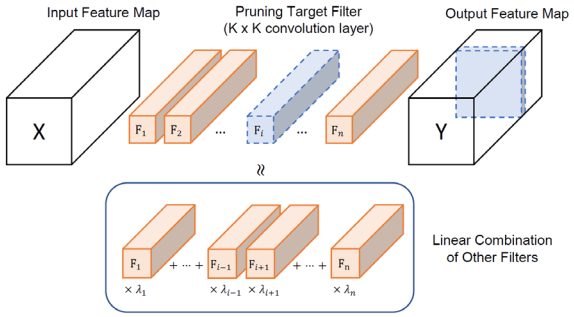

Figure 2. Filter Pruning

Transferred model: EfficientNet B7

Link for EfficientNet B7: https://keras.io/api/applications/efficientnet/

- **Practical steps taken during implementation**
  1. Load data
  2. Preprocessing
  3. Implement transfer learning CNN
  4. Apply weight pruning on transfer learning CNN
  5. Evaluate the performance and size of transfer learning models and the model after weight pruning.
  6. Implement custom CNN
  7. Apply weight pruning on custom CNN
  8. Evaluate the performance and size of custom CNN model after weight pruning
  9. Apply filter pruning on custom CNN
  10. Evaluate the performance and size of custom CNN model after filter pruning

2. Technical Details

Transfer learning CNN (Transferred model: EfficientNet B7)

We do not include the architecture and parameters of EfficientNet B7 here because it has about 800 layers. For details, please see https://keras.io/api/applications/efficientnet/

Transfer learning CNN parameters:

- Activation function: relu
- Dropout rate: 0.45
- Epoch: 100
- Early stopping: monitor=' val_loss' , patience=5
- ReduceLROnPlateau: monitor='val_loss', factor=0.2, patience=3, min_lr=1e-6
- Compile: optimizer=Adam(0.00001), loss='categorical_crossentropy', metrics=['accuracy']

Weight pruned transfer learning CNN parameters:

Same as transfer learning CNN parameters except for epochs, and add a new hyperparameter sparsity level:

- Epochs: 30
- Sparsity_levels: 0.9

Transfer learning CNN model architecture (the dense layers added to EfficientNet B7):

```
dense_3 (Dense)                 (None, 128)          327808      ['max_pool[0][0]']

batch_normalization_2 (BatchNo  (None, 128)          512         ['dense_3[0][0]']
rmalization)

dropout_2 (Dropout)             (None, 128)          0           ['batch_normalization_2[0][0]']

dense_4 (Dense)                 (None, 256)          33024       ['dropout_2[0][0]']

batch_normalization_3 (BatchNo  (None, 256)          1024        ['dense_4[0][0]']
rmalization)

dropout_3 (Dropout)             (None, 256)          0           ['batch_normalization_3[0][0]']

dense_5 (Dense)                 (None, 10)           2570        ['dropout_3[0][0]']
```

Figure 3. Transfer learning CNN dense layers architecture

Custom CNN

Custom CNN parameters:

- Activation function: relu
- Dropout rate: 0.25 for first 3 sets, 0.5 for fully connected layer
- Epochs: 50
- Early Stopping: monitor='val_loss', patience=5, restore_best_weights=True
- ReduceLROnPlateau: monitor='val_loss', factor=0.5, patience=3
- Compile: optimizer=Adam(learning_rate=0.001), loss="sparse_categorical_crossentropy", metrics=["accuracy"]

Weight pruned custom CNN parameters:

Same as custom CNN parameters except for epochs, and add a new hyperparameter sparsity level:

- Epochs: 30
- Sparsity_levels: 0.7

Filter pruned custom CNN parameters:

Same as custom CNN parameters except for epochs, and add a new hyperparameter sparsity level:

- Epochs: 30
- Compile: optimizer=Adam(learning_rate=0.0001), loss="categorical_crossentropy"
- Pruning_ratios: 0.3

Custom CNN model architecture: Find in train_CNN.ipynb

3. Experimental Design

- **Data sources**

  The data loading phase of the project involves organizing and validating the image dataset to ensure it is suitable for model training. We first download the image files from Kaggle. The file paths and corresponding labels are converted into a structured Pandas DataFrame for efficient data manipulation. Each image file is then iterated over to identify and report any corrupted or unreadable images to ensure the reliability of the dataset. Finally, the dis

tribution of image labels is calculated and visualized to assess the class balance within the dataset.

Figure 4. Data label distribution

- **Preprocessing**

The preprocessing pipeline systematically prepares the dataset for effective model training:

1. Dividing the dataset into training, validation, and testing subsets to facilitate unbiased model evaluation.
2. Utilizing ImageDataGenerator to streamline the loading and preprocessing of image data, incorporating essential preprocessing functions tailored to the chosen neural network architecture (EfficientNet).
3. Establishing data flows (train_images, val_images, test_images) that feed data into the model in manageable batches, optimizing memory usage and computational efficiency.
4. Implementing a comprehensive augmentation strategy to artificially expand the training dataset, introducing variability that enhances the model's ability to generalize to new, unseen data.

Through the pipeline we obtained 17983 images for the training set, 4495 images for the validation set and 5620 images for the test set.

- **Hyperparameter tuning and evaluating metrics**

Sparsity tuning on weight pruned transfer learning CNN:

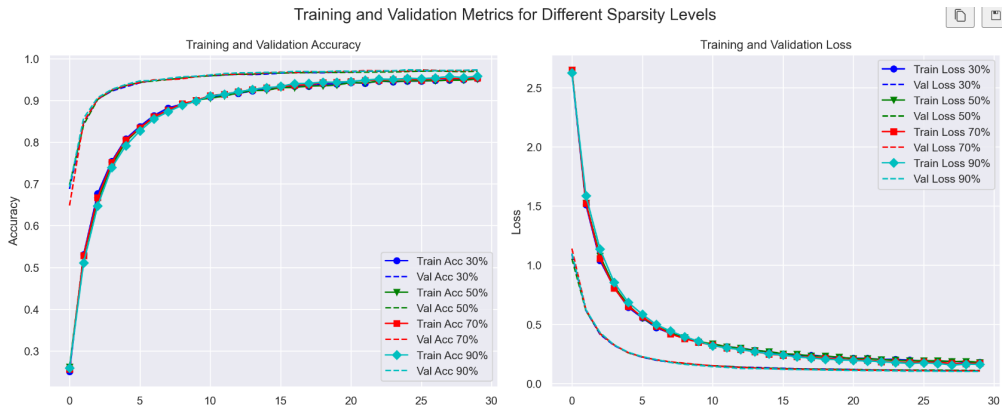

Figure 5. Training and Validation Metrics for Different Sparsity Levels

Metrics\Sparsity	0% (Origianl model)	30%	50%	70%	90%
Val Accuracy	97.44%	97.22%	97.22%	97.22%	97.26%
Size (after zip)	226.25 MB	225.97 MB	225.74 MB	225.48 MB	225.22 MB
Size Reduction	0.00%	0.13%	0.23%	0.34%	0.46%
Model Sparsity	0.00%	0.17%	0.28%	0.40%	0.51%

Table 1. Weight pruned transfer learning CNN metrics table

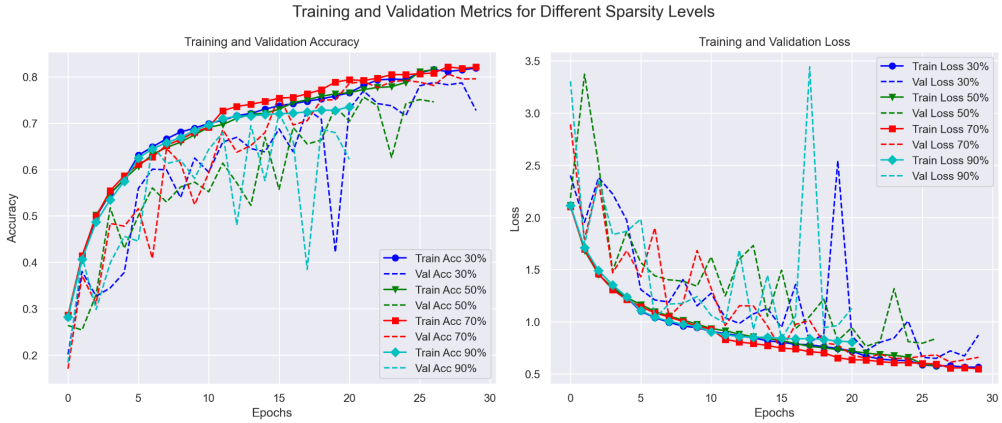

Figure 6. Training and Validation Metrics for Different Sparsity Levels

From the table we choose sparsity 0.9 to implement weight pruned transfer learning CNN. Because it has the highest validation accuracy and biggest size reduction.

Sparsity tuning on weight pruned custom CNN:

Metrics\Sparsity	0% (Origianl model)	30%	50%	70%	90%
Val Accuracy	77.46%	72.73%	74.57%	79.56%	62.18%
Size (after zip)	52.46MB	0.94 MB	0.77MB	0.56MB	0.44MB
Size Reduction	0.00%	98.20%	98.52%	98.94%	99.16%
Model Sparsity	0.00%	26.70%	43.63%	62.31%	71.69%

Table 2. Weight pruned custom CNN metrics table

Sparsity tuning on filter pruned custom CNN:

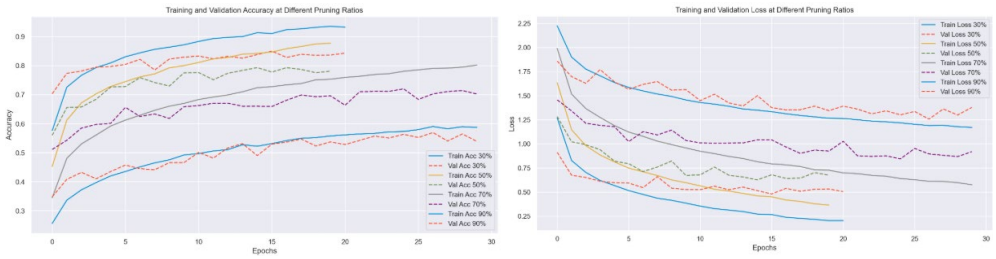

Figure 7. Training and Validation Metrics for Different Sparsity Levels

Metrics\Sparsity	0% (Origianl model)	30%	50%	70%	90%
Val Accuracy	77.46%	84.32%	78.11%	70.23%	53.90%
Size	57.48MB	39.80 MB	28.02 MB	16.96 MB	5.72 MB
Size Reduction	0.00%	30.76%	51.25%	70.49%	90.06%
Parameters	5011242	3465462	2436122	1469515	486607
Parameter Reduction	0.00%	30.85%	51.39%	70.68%	90.29%

Table 3. Filter pruned custom CNN metrics table

Results and Analysis

1. Results

Metrics\Model	Transfer learning CNN	Weight pruned transfer learning CNN	Custom CNN	Weight pruned custom CNN	Filter pruned custom CNN
Test Accuracy	96.98%	95.00%	77.46%	79.18%	83.20%
Size	226.25MB	225.22MB	52.46MB	0.56MB	39.80MB
Size Reduction	0%	0.46%	0%	98.94%	30.76%

Table 4. Evaluation Metrics table for all models

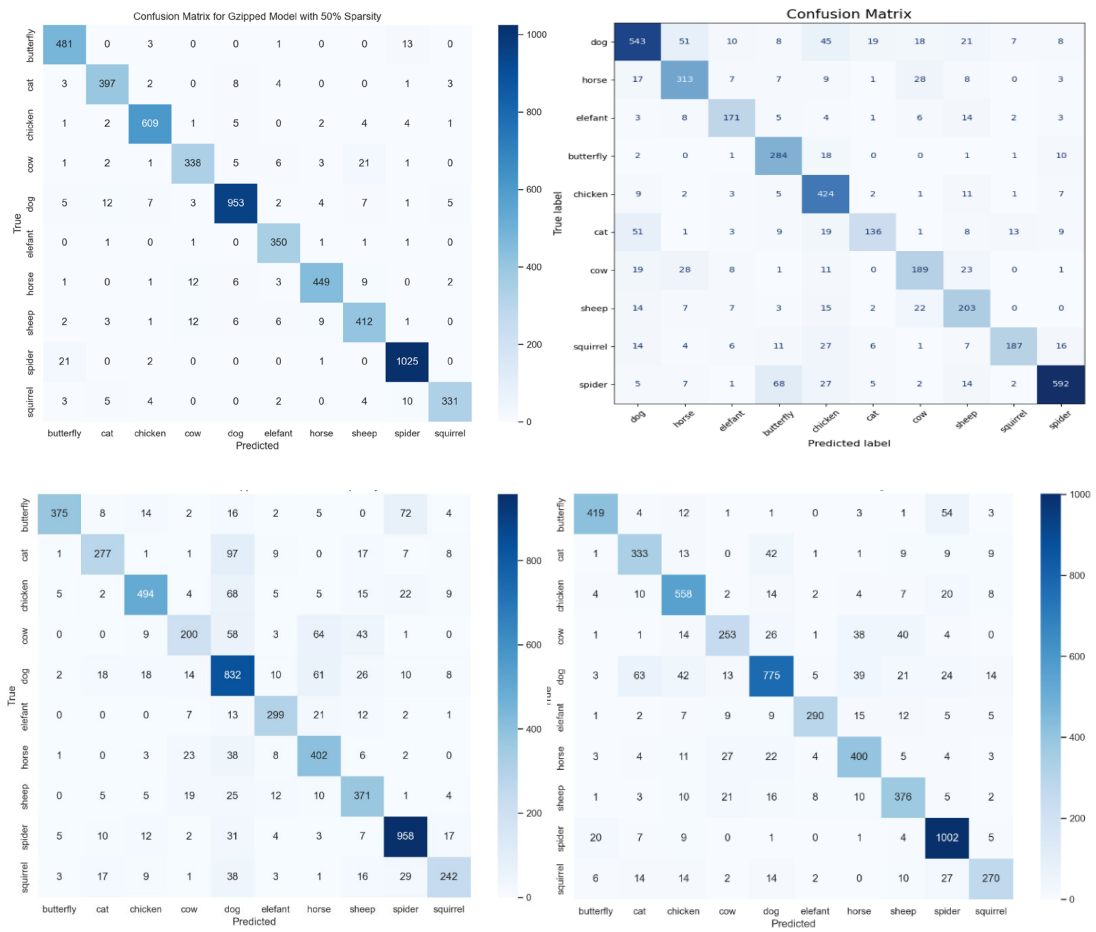

Figure 8. Confusion matrix on testset: 1. CNN base model 2. Transfer model 3. Weight pruning 4. Filter pruning

2. Analysis

From table 1, we can see that no matter which sparsity is chosen, weight pruning does not perform well on the transfer learning model. The reason why it cannot significantly reduce the model size is that our computing resources only support us to prune the two dense layers we added. We tried to apply pruning to the transferred model EfficientNet B7, but its 66M parameters and a depth of about 800 layers crashed our test environment. Even the last convolutional layer still has about 1.6M parameters, which our RAM still cannot support. Therefore, we gave up pruning the transferred model. Instead, we implemented a CNN to verify the impact of weight pruning and filter pruning. Although pruning CNN cannot directly reflect the performance of pruning on the transfer learning model, similar architectures provide comparisons and implications.

The weight pruning results on the custom CNN show significant advantages, especially in terms of model compression, achieving about 98% size reduction, making it very efficient to deploy in resource-constrained environments. This is because the application of gzip compresses all 0 weights, while the original model using .h5 still retains these weights. It is worth noting that the model maintains or even slightly improves the validation accuracy at moderate sparsity levels, achieving a maximum accuracy of 79.56% at 70% sparsity, while the original accuracy is 77.46%.

Filter pruning differs from weight pruning in its impact and effectiveness. Experiments demonstrate that increasing sparsity significantly compresses model size. The compression effect of filter pruning correlates positively with sparsity but its effect on validation accuracy is more complex. At a sparsity rate of 30%, validation accuracy improved from 77.46% to 84.32%, showcasing the potential of pruning techniques to enhance model performance at low sparsity levels. However, as sparsity continues to increase, validation accuracy gradually decreases, reaching only 53.90% at a sparsity rate of 90%, where model performance is significantly impaired. This indicates that excessive pruning leads to the loss of critical feature information, negatively affecting classification performance.

Compared to weight pruning, filter pruning demonstrates a more notable improvement in validation accuracy at low sparsity levels but shows a more pronounced performance decline at high sparsity levels. This is because filter pruning directly removes entire convolutional filters, significantly affecting the model's feature extraction capabilities, whereas weight pruning makes more fine-grained adjustments to the model. Additionally, the compression capability of weight pruning relies on storage technology optimization (e.g., gzip), while filter pruning directly reduces the number of parameters and the complexity of the network structure.

Discussion and Conclusion

We have identified the limitations of pruning in transfer learning models. Due to the substantial depth and parameter count of EfficientNet B7, the application of pruning techniques is significantly constrained under limited computational resources. Weight pruning is suitable for optimizing model storage and computational requirements, while also enhancing the model's generalization ability within a moderate pruning range. Filter pruning, on the other hand, requires more precise control over sparsity rates to strike a balance between performance and compression. Both pruning methods demonstrate potential in resource-constrained environments, particularly in applications requiring efficient deployment of deep learning models, such as field monitoring and ecological research. In the future, we plan to test the performance and efficiency of pruned models in real-world scenarios, such as deployment on edge devices or embedded systems, to validate their adaptability and reliability for practical tasks.

References

- Dataset and baseline models architecture: A. Corrado, Animals-10 dataset [Online]. Available: https://www.kaggle.com/datasets/alessiocorrado99/animals10. [Accessed: Nov. 12, 2024].
- Weight pruning: M. McAteer, Model Pruning Exploration [Online]. Available: https://github.com/matthew-mcateer/Keras_pruning/blob/master/Model_pruning_exploration.ipynb. [Accessed: Nov. 23, 2024].
- Luo, J. H., Zhang, H., Zhou, H. Y., Xie, C. W., Wu, J., & Lin, W. (2018). ThiNet: Pruning CNN filters for a thinner net. IEEE transactions on pattern analysis and machine intelligence, 41(10), 2525-2538.