Ayushi Sapru
COE379L
Project 4 Final Report

Exploring Neural Network Architecture Search on Clean vs Noisy Image Data

Introduction and Project Statement:

Image classification is a core task in computer vision and machine learning. Convolutional neural networks (CNNs) are widely used in this area due to their ability to capture spatial patterns in images. In clean and controlled datasets, CNNs often perform very well. However, their performance tends to drop when the input data is distorted or degraded. This issue becomes important in real-world applications, where images can be affected by noise, compression, or blur due to poor lighting, motion, or low-resolution sensors.

The goal of this project is to analyze how different types of noise affect CNN classification accuracy. I trained a CNN model using the CIFAR-10 dataset, which contains 60,000 images across 10 classes. After training the model on clean data, I tested its performance on several noisy test sets. These included images corrupted by Gaussian noise, Gaussian blur, JPEG compression, and Salt & Pepper noise. Each type of distortion was selected because it reflects common problems in digital imaging.

To improve the model's ability to handle these distortions, I applied hyperparameter tuning using Keras Tuner. This created a second version of the model, optimized to better generalize across different conditions. I compared the performance of both models - Version 1 (baseline) and Version 2 (tuned) - on clean and noisy data. This comparison helped me understand how much model tuning can reduce performance loss due to image noise.

This project addresses two key questions: how sensitive CNNs are to different types of noise, and whether tuning can help make them more robust. These questions are relevant in fields like robotics, surveillance, or even healthcare, where reliable image classification is needed even under imperfect conditions. By measuring the model’s accuracy across multiple noise types and configurations, I aimed to draw conclusions that are practical for future applications.

Data Sources and Technologies Used:

For this project, I used the CIFAR-10 dataset, which is a standard benchmark in image classification tasks. It contains 60,000 color images with a resolution of 32x32 pixels, split into 10 different classes such as airplanes, automobiles, birds, cats, and other everyday objects. The dataset is divided into 50,000 training images and 10,000 test images. I chose CIFAR-10 because it is widely supported in machine learning libraries, and contains enough variety to evaluate model performance across a diverse set of visual patterns.

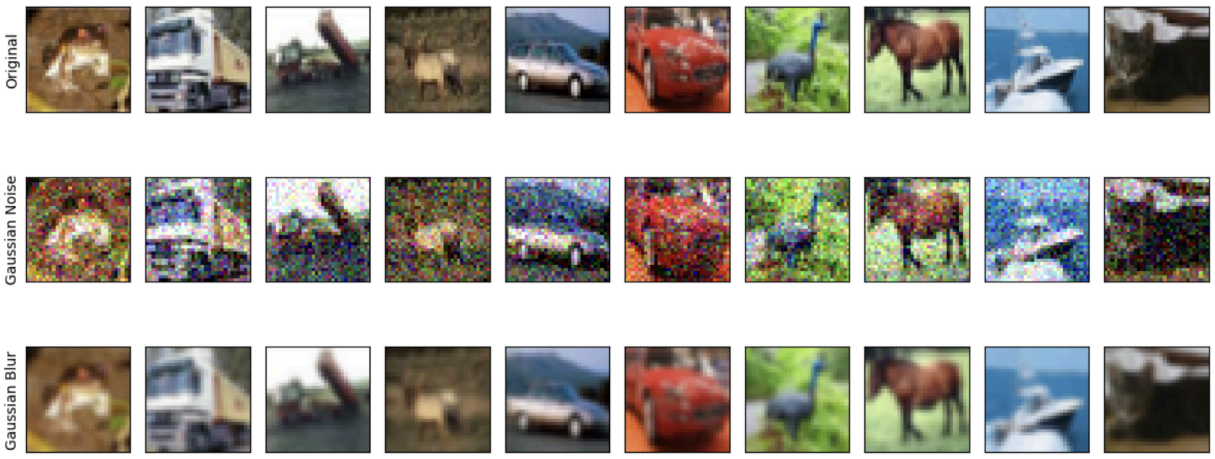
To simulate real-world conditions, I introduced four different types of image noise and distortion to the dataset: Gaussian noise, Gaussian blur, JPEG compression, and Salt & Pepper noise. These types were selected because they represent common problems that occur in real image pipelines. Gaussian noise mimics sensor-level random interference, Gaussian blur simulates motion or out-of-focus images, JPEG compression reflects lossy file compression artifacts, and Salt & Pepper noise imitates pixel dropouts or transmission errors. Each transformation was applied using Python-based libraries such as NumPy and OpenCV.

All experiments were carried out using Python with TensorFlow and Keras as the main machine learning frameworks. For model optimization and architecture tuning, I used Keras Tuner, which allowed me to define a flexible search space for hyperparameters. This included tuning the number of convolutional layers, kernel sizes, number of filters, dense units, dropout rates, and optimizer types.

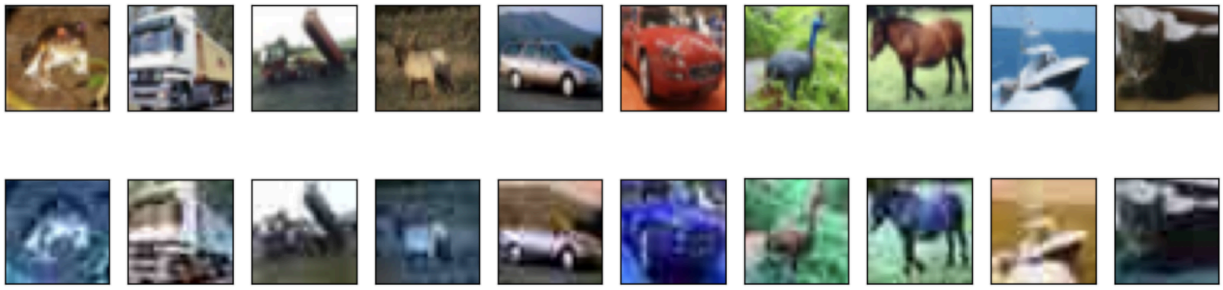
Noise transformations were applied to the training and testing datasets before training each model. I saved each noisy version as .npy files for reuse during model tuning and evaluation. This made it possible to train multiple models across different noise conditions without regenerating the corrupted images each time. For each version of the model, I ran a separate hyperparameter search on each dataset variant to ensure that the model was being tuned specifically for that type of distortion.

Overall, the tools and data used in this project were chosen for their flexibility, performance, and ease of integration into an end-to-end image classification workflow. The combination of CIFAR-10, TensorFlow, Keras Tuner, and custom data augmentation provided a solid foundation for investigating how model architecture responds to noisy input data.

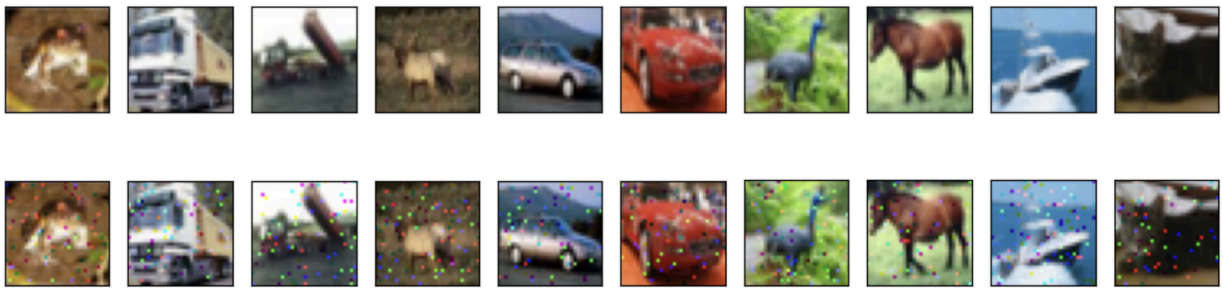
Image Comparison: Original vs Gaussian Noise vs Gaussian Blur



Top: Original | Bottom: JPEG Compression (quality = 20)



Top: Original | Bottom: Salt-and-Pepper Noise



Methods Employed:

The project was divided into two main phases: building and evaluating a baseline model (Version 1), and creating an improved model using neural architecture search (Version 2). Each version was tested independently across five conditions: clean images and images with four different noise types. This allowed for a direct comparison of how architecture tuning impacts model robustness under distortion.

The Version 1 model was built using a straightforward CNN architecture. It consisted of a few convolutional layers followed by a max pooling layer, flattening, one dense hidden layer, dropout, and a final softmax layer for classification. I used this model as a baseline because it reflects common introductory CNN designs and allowed me to observe how a basic architecture behaves under clean and noisy conditions without additional complexity.

The training data was normalized by scaling pixel values to the range [0, 1], and the model was trained using sparse categorical cross entropy loss and the Adam optimizer. I used the Keras Tuner library to perform basic hyperparameter search over parameters such as the number of convolutional filters, kernel size, and dense layer size. For each noise condition, a separate search was run, and the best model was selected based on validation accuracy.

To simulate noisy environments, I applied four types of distortions to the CIFAR-10 dataset. Each distortion was applied to both the training and test datasets before model training.

The transformations included:

- Gaussian noise: Random pixel noise was added using a normal distribution with defined mean and standard deviation.
- Gaussian blur: A blurring kernel was applied using OpenCV to simulate motion blur or soft-focus effects.
- JPEG compression: Images were compressed at low quality using PIL to simulate lossy transmission or storage.
- Salt & Pepper noise: Random pixels were set to pure black or white to mimic transmission errors or sensor dropouts.

These transformations were implemented using NumPy, OpenCV, and PIL. For each distortion type, I generated and saved the resulting datasets as .npy files to avoid regenerating them during each run. This also ensured consistent evaluation across model versions.

In the second phase, I extended the search space in the model tuning process. The Version 2 model allowed for more flexibility in architecture design. I added batch normalization as a tunable option, allowed for a wider range of convolutional blocks, and introduced optimizer choice as a hyperparameter. I also tuned the learning rate on a logarithmic scale and allowed for variable dropout rates.

Each version of the model was trained separately on each noise variant and evaluated on its corresponding test set. Keras Tuner was again used to explore the search space using random search with a fixed number of trials. This ensured that the performance improvement in Version 2 came from architectural changes, not from simply training longer or on more data.

To evaluate performance, I used test set accuracy as the primary metric. For each version and noise type, the model was trained on the noisy training set and then evaluated on the matching noisy test set. This process was repeated for all five datasets (clean, Gaussian noise, Gaussian blur, JPEG compression, Salt & Pepper). After each run, I recorded the best model's test accuracy to compare across conditions.

By isolating the architecture search for each noise type, I was able to investigate how optimal CNN architectures differ depending on the type of image degradation. This approach also helped me understand whether certain distortions require deeper networks, more regularization, or alternative optimization strategies.

Results and Conclusion:

The results of this project are based on evaluating the test accuracy of two CNN models - Version 1 (baseline) and Version 2 (tuned) - on clean and noisy CIFAR-10 datasets. Each model was trained and evaluated separately for five input conditions: clean images, Gaussian noise, Gaussian blur, JPEG compression, and Salt & Pepper noise. The goal was to measure how image distortions affect performance, and whether model tuning could mitigate that impact.

The Version 1 model used a relatively simple architecture and limited hyperparameter search. Its performance on the clean dataset served as a baseline, while evaluations on noisy inputs highlighted the effect of each distortion. The table below summarizes the results:

```
Model Accuracy Summary (Version 1):
--------------------------------------
Clean             : 0.5920
Gaussian Noise    : 0.6086
Gaussian Blur     : 0.6036
JPEG Compression  : 0.6233
Salt & Pepper     : 0.6380
```

Interestingly, some noisy conditions performed better than clean data. Salt & Pepper noise yielded the highest accuracy (0.6380), while clean images performed the worst. One possible explanation is that the added randomness acted as a form of regularization, helping the model avoid overfitting. JPEG compression also showed relatively high performance, while Gaussian blur and Gaussian noise slightly outperformed clean data.

Version 2 introduced a more flexible architecture search. The search space included batch normalization, optimizer selection, and learning rate tuning. This resulted in significant performance gains across most input conditions.

```
Model Accuracy Summary (Version 2):
_____________________________________________
Clean             : 0.6834
Gaussian Noise    : 0.6539
Gaussian Blur     : 0.6483
JPEG Compression  : 0.6787
Salt & Pepper     : 0.5921
```

In contrast to Version 1, the clean dataset performed best in Version 2, showing that the tuned model better leveraged structure in the clean images. JPEG compression and Gaussian noise also showed strong improvements. However, Salt & Pepper accuracy dropped significantly (from 0.6380 to 0.5921). This suggests that the Version 2 architecture may have overfit to clean signal patterns and lacked the robustness required to handle sparse, high-variance corruption like Salt & Pepper.

To directly compare the improvements, I created a side-by-side table:

```
Comparison of Accuracy (V1 vs V2):
_____________________________________________________
Clean             : V1 = 0.5920 | V2 = 0.6834
Gaussian Noise    : V1 = 0.6086 | V2 = 0.6539
Gaussian Blur     : V1 = 0.6036 | V2 = 0.6483
JPEG Compression  : V1 = 0.6233 | V2 = 0.6787
Salt & Pepper     : V1 = 0.6380 | V2 = 0.5921
```

The largest gain came from clean data, with nearly 10% improvement. Gaussian noise and blur also showed steady gains. JPEG compression showed a solid 5.5% gain. The only performance drop occurred with Salt & Pepper noise, which might indicate that aggressive tuning favored more structured inputs over ones with isolated pixel corruption.

These results demonstrate that hyperparameter tuning can significantly improve model robustness for most real-world noise types. However, they also suggest that no single architecture is optimal across all distortions. A model tuned for smooth degradations like blur or JPEG compression may not generalize well to random high-variance noise like Salt & Pepper.

Overall, this project highlights the importance of adapting CNN architectures to the conditions in which they are deployed. By comparing performance across both clean and noisy datasets, I showed that architecture tuning has a meaningful impact on robustness—but not in every case. While Version 2 significantly improved accuracy in four of the five test conditions, it failed to outperform the simpler Version 1 model on Salt & Pepper noise. This suggests that general-purpose tuning may not be sufficient for all distortion types. Future work could explore combining NAS with distortion-specific strategies, such as targeted data augmentation or robustness-focused loss functions, to further improve model reliability in unpredictable environments.

References:

Krizhevsky, A. (2009). Learning Multiple Layers of Features from Tiny Images. [CIFAR-10 Dataset]. University of Toronto. https://www.cs.toronto.edu/~kriz/cifar.html