Project: An Analysis of Convolutional Neural Network Optimization, Invariance, and Vulnerability

Introduction

Convolutional Neural Networks (CNNs) have become the state-of-the-art in computer vision, capable of achieving super-human performance on classification tasks. However, high accuracy often hides underlying limitations and vulnerabilities of CNN. This project provides an end-to-end analysis of CNNs, following a four-experiment that demonstrates model creation, optimization, property testing, and adversarial "breaking."

This report explains four important experiments:

1. **Experiment 1: Hyperparameter and Architecture Tuning for CIFAR-10 –** It builds a strong model by training a custom AlexNet-style CNN on the CIFAR-10 dataset. It demonstrates that systematic hyperparameter and architecture tuning using the Optuna framework is critical for maximizing performance.
2. **Experiment 2: AlexNet as a Fixed Feature Extractor on Oxford-IIIT Pet Dataset –** It explores CNN as a feature extractor. It uses a pre-trained AlexNet as a fixed feature extractor to show how a simple, classical classifier can achieve high accuracy on the complex 37-class Oxford-IIIT Pet dataset.
3. **Experiment 3: Testing CNN Invariance to Geometric Transformations –** it investigates the inherent properties of CNNs by testing the optimized model's invariance to geometric transformations. It provides a quantitative analysis of a CNN's robustness to translation and flipping, versus its significant weakness to rotation.
4. **Experiment 4: Adversarial Attacks on Convolutional Neural Networks –** It investigates the security and robustness of CNN model. It demonstrates, both visually and quantitatively, how CNNs are highly vulnerable to adversarial attacks (both untargeted FGSM and targeted PGD), forcing misclassifications with high confidence.

Together, these experiments show that while CNNs are powerful pattern-recognition tools, their high performance is highly dependent on optimization, and their decision-making process is fundamentally different from human perception, leaving them vulnerable to simple, imperceptible attacks.

Code Availability: The complete code for all four experiments described in this report is available on GitHub at the following repository: Click Here

1. Hyperparameter and Architecture Tuning for CIFAR-10 Baseline Model

This experiment focuses on the design, implementation, and optimization of a Convolutional Neural Network (CNN) for image classification on the CIFAR-10 dataset. The primary challenge in deep learning is identifying the optimal hyperparameters. This experiment automates that search using the **Optuna** optimization framework. A flexible, dynamically-built CNN was created to allow Optuna to tune not only training parameters (like learning rate) but also architectural parameters (like the number of layers and activation functions). The optimization was validated using **3-Fold Cross-Validation over 50 trials**. The best trial achieved a mean validation accuracy of **77.83%.** This optimal configuration was then retrained on the entire training dataset for **50 epochs**, achieving a final, hold-out test accuracy of **87.38%.** This experiment shows how optimal hyperparameters improves the CNN performance drastically.

1. Introduction & Objective

The objective of the experiment is to build a high-performing CNN model for CIFAR-10 classification. The performance of a CNN on this task is dependent on its architecture and hyperparameters. Manually tuning these parameters (e.g., learning rate, dropout rate, number of layers, optimizer choice) is time-consuming, inefficient, and unlikely to discover the optimal configuration. To solve this, this experiment uses an automated, systematic approach to optimization.

2. Experiment Methodology

2.1. Data Preprocessing & Analysis

1. **Normalization**: The per-channel mean and standard deviation of the entire 50,000-image training set were calculated to normalize the data based on its own statistics.
    - Mean: [0.4914, 0.4822, 0.4465]
    - Standard Deviation: [0.2023, 0.1994, 0.2010]

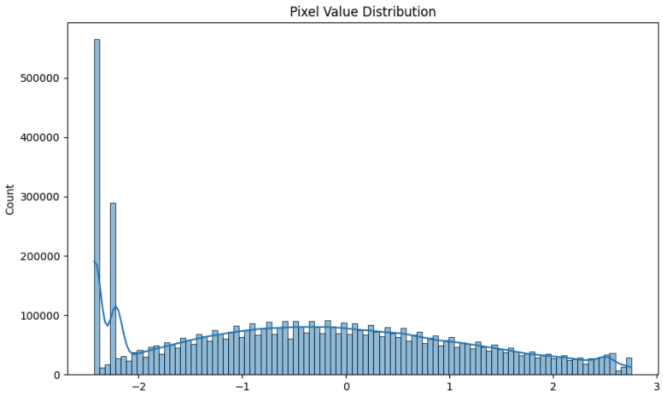


Figure 1 Pixel Value Distribution

2. **Data Augmentation**: To prevent overfitting, the training data was augmented using **RandomHorizontalFlip** and **RandomCrop** (with padding=4). The test set was left un-augmented.

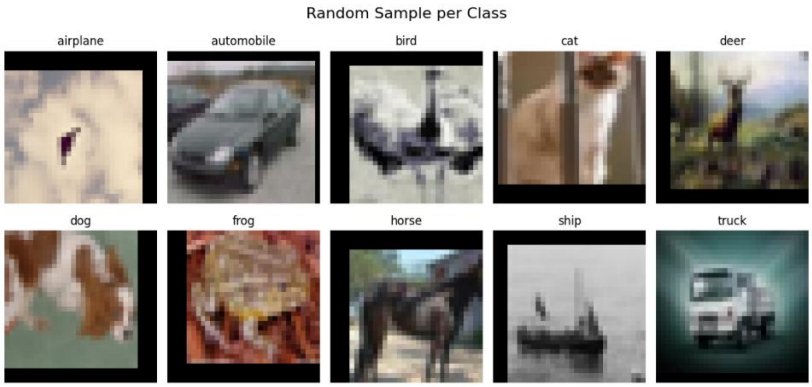


Figure 2 A Sample from 10 Classes

3. **Data Distribution**: The CIFAR-10 dataset is perfectly balanced, with each of the 10 classes representing exactly 10.0% of the training data. This validates "accuracy" as a reliable metric for this problem.

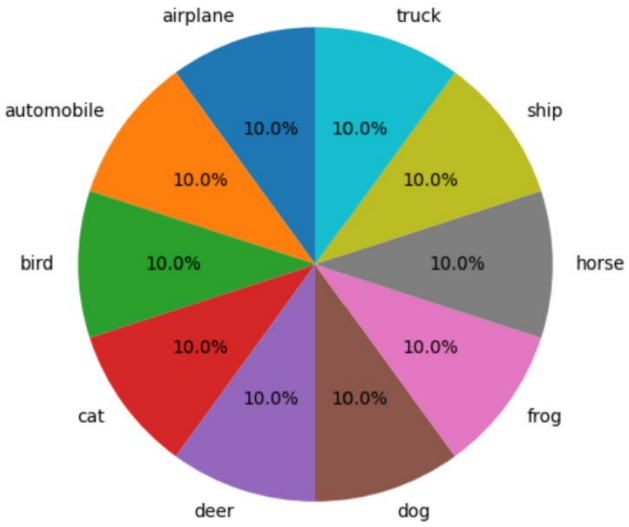


Figure 3 Class Distribution of CIFAR-10 Dataset

2.2. Custom Model Architecture

A custom Flexible CNN PyTorch class was implemented. This class constructs its layers based on configuration (conv_config, fc_config) passed during initialization.

- **Dynamic Convolutional Layer**: The class iterates through the conv_config list to add Conv2d, BatchNorm2d, activation, and MaxPool2d layers.
- **Dynamic Classifier**: It then builds the fully-connected (Linear) and Dropout layers based on the fc_config list.

This design allowed Optuna to treat the model's architecture (e.g., number of layers, layer width, activation function) as hyperparameters to be optimized.

2.3. Hyperparameter Optimization with Optuna

Optuna was used to automate the search for the best model. An objective function was defined to wrap the entire training and validation process.

1. **Hyperparameters Tuned**:
    - Learning Rate (lr): Log-uniform distribution (1e-4 to 1e-1)
    - Dropout Rate (dropout_p): Uniform (0.2 to 0.6)
    - Base Conv Channels (cfg_size): Categorical (32, 64, 128)
    - Activation Function (activation): Categorical (relu, sigmoid, tanh)
    - Optimizer (optimizer): Categorical (SGD, Adam, Adagrad)
    - # of Conv Layers: Integer (2 to 4)
    - # of FC Layers: Integer (1 to 3)
    - FC Hidden Size: Categorical (128, 256, 512)
2. **Validation Strategy**: To ensure a robust performance, each trial was evaluated using 3-Fold Cross-Validation. The model was trained and validated 3 times on different splits of the training data, and the mean validation accuracy was returned as the trial's score.
3. **Optimization**: The Optuna study was run for 50 trials, with each trial building, training, and validating 3 separate models.

3. Experimental Results

3.1. Optimization Phase

The Optuna study successfully identified a high-performing set of hyperparameters. The best result was Trial #43, which achieved a mean 3-fold validation accuracy of 77.83%.

The optimal parameters were found to be:

- Learning Rate: 0.000595
- Optimizer: Adam
- Activation: ReLU
- Dropout Rate: 0.4206
- # Conv Layers: 4
- Base Conv Channels: 64
- # FC Layers: 1
- FC Hidden Size: 512

The optimal architecture derived from these parameters was:

- Conv Architecture: 4x Conv/Pool blocks (output channels: [64, 128, 192, 192])
- FC Architecture: 1x FC layer (output features: 512), followed by dropout.

Table 1 Custom CNN Architecture Optimal for CIFAR-10 Dataset

Layer #	Layer (type)	Output Shape	Kernel Size (k×k)	Stride / Padding	Param #
1	Conv2d (3 → 64)	[64, 32,32]	5×5	stride=1, pad=2	4,800
2	MaxPool2d	[64, 16, 16]	3×3	stride=2, pad=1	0
3	Conv2d (64 → 192)	[192, 16, 16]	5×5	stride=1, pad=2	307,392
4	MaxPool2d	[192, 8, 8]	3×3	stride=2, pad=1	0
5	Conv2d (192 → 384)	[384, 8, 8]	3×3	stride=1, pad=1	663,936
6	Conv2d (384 → 256)	[256, 8, 8]	3×3	stride=1, pad=1	884,992
7	Conv2d (256 → 256)	[256, 8, 8]	3×3	stride=1, pad=1	590,080
8	MaxPool2d	[256, 4, 4]	3×3	stride=2, pad=1	0
9	Linear (FC)	[4096]	— (FC)	input = 256×4×4 = 4096	16,781,312
10	Linear (FC)	[4096]	— (FC)	4096 → 4096	16,781,312
11	Linear (FC)	[10]	— (FC)	4096 → 10	40,970
				Total Parameters	**36,054,794**

3.2. Final Training and Test Performance

Using the best hyperparameters from Trial #43, a single, final model was created. This model was then trained on the entire 50,000-image training set for 50 epochs to allow it to learn from all available data.

This fully-trained model was then evaluated *once* on the unseen 10,000-image test set.

- **Final Test Accuracy: 87.38%**

4. Conclusion

This experiment successfully builds a robust CNN model, achieving 87.38% test accuracy. The automated workflow, combining a flexible CNN class with the Optuna optimization framework, proved highly effective. It efficiently optimized the complex, high-dimensional parameter space to find a strong configuration.

The experiment proves that a flexible CNN architecture combined with a systematic optimizer (Optuna) and a robust validation strategy (K-Fold CV) can produce a highly effective model.

2. AlexNet as a Fixed Feature Extractor on Oxford-IIIT Pet Dataset

The objective of this experiment was to use Convolutional Neural Network (CNN) as a feature extractor. A **pre-trained AlexNet** model, with weights from **ImageNet**, was used to extract the feature of **Oxford-IIIT Pet dataset**. The final classification layer of the AlexNet was removed, and the 4096-dimensional output from the penultimate layer was extracted for each image. These features were then used to train two standard machine learning classifiers: **Logistic Regression** and **Random Forest**. The Logistic Regression model achieved the highest test accuracy at **74.54%,** demonstrating that pre-trained features can be effectively used to train simple, non-deep-learning models.

1. Introduction & Objective

While Experiment 1 focused on training a custom CNN from scratch, this experiment explores a computationally efficient alternative: transfer learning via feature extraction. The Oxford-IIIT Pet dataset, which contains 37 classes, is significantly different from the 10-class CIFAR-10. Training a deep network from scratch on this dataset would require significant time and hyperparameter tuning.

The objective of this experiment is to bypass this complex training by leveraging the powerful, generalized features learned by a model (AlexNet) already trained on the massive ImageNet dataset. The hypothesis is that these features are rich enough to allow a simple classifier to effectively distinguish between the 37 pet categories.

2. Experiment Methodology

2.1. Data & Preprocessing

1. **Dataset**: The Oxford-IIIT Pet dataset was used. The full train-val split served as the training set, and the test split was used for final evaluation.
2. **Transforms**: To be compatible with the pre-trained AlexNet, all images (both train and test) were:
    - Resized to (224, 224) pixels.
    - Converted to PyTorch Tensors.
    - Normalized using the standard ImageNet statistics (mean=[0.485, 0.456, 0.406] and std=[0.229, 0.224, 0.225]).

2.2. Feature Extraction

The core of this experiment was the feature extraction process:

1. **Model**: A pre-trained AlexNet model was loaded from torchvision.models.
2. **Weight Freezing:** All parameters in the AlexNet model were frozen by setting param.requires_grad = False. This ensures that the pre-trained weights are not modified.
3. **Model Modification**: The final Linear layer (which classifies 1000 ImageNet classes) was removed from AlexNet's classifier. The model was then defined to stop at the output of the second-to-last layer, which produces a **4096-dimensional feature vector**.
4. **Extraction**: An extract_feature function was created to pass all images from the train_loader and test_loader through the modified AlexNet. This process converted the entire image dataset into two numerical datasets:

- X_train, y_train: The 4096-dimensional feature vectors and corresponding labels for the training set.
- X_test, y_test: The 4096-dimensional feature vectors and corresponding labels for the test set.

2.3. Classical Model Training

With the image data transformed into feature vectors, the problem was reduced to a standard Machine Learning classification task. The feature and label arrays were converted from PyTorch Tensors to NumPy arrays and used to train two classifiers from Scikit-learn.

3. Experimental Results:

The extracted X_train and X_test feature sets were used to train and evaluate the classifiers. The final performance on the unseen test set was as follows:

- Logistic Regression Accuracy: **74.54%**
- Random Forest Accuracy: **72.39%**

The Logistic Regression model, despite its simplicity, outperformed the more complex Random Forest when trained on the high-dimensional deep features.

4. Conclusion

This experiment successfully demonstrated the power and efficiency of CNN as a feature extractor. By using a pre-trained AlexNet as a fixed feature extractor, we were able to achieve a respectable 74.54% accuracy on the 37-class Oxford-IIIT Pet dataset.

This approach is computationally efficient, as it requires only a single forward pass through the deep network to extract features. The subsequent training of the Logistic Regression model is extremely fast compared to end-to-end deep learning training. This experiment validates feature extraction as a strong method.

3. Testing CNN Invariance to Geometric Transformations

This experiment investigates the **translation and rotation invariance** of a Convolutional Neural Network (CNN). Using the feature extraction pipeline established in Experiment 2 (a pre-trained AlexNet), a single Logistic Regression classifier was trained on the standard Oxford-IIIT Pet training images. This classifier was then evaluated against four different test sets: a baseline (unmodified), a translated (shifted) set, a horizontally flipped set, and a 45-degree rotated set. The results strongly support the Invariance properties of CNNs. The model showed **high invariance to translation (71.1% accuracy) and horizontal flipping (71.8%)**, performing nearly as well as the baseline (71.9%). However, it showed **no invariance to rotation**, with accuracy collapsing to **46.6%**.

1. Introduction & Objective

A key property of Convolutional Neural Networks (CNNs) over simple feed-forward networks is their *translation invariance*. This means the CNN network can recognize an object regardless of where it appears in the image. This experiment also proves that CNNs are generally *not* invariant to other transformations, such as rotation.

The objective of this experiment is to:

1. Quantify the translation invariance of the AlexNet feature extractor.
2. Test the model's invariance to horizontal flipping, a transformation often included in data augmentation.
3. Test the model's invariance to rotation, a transformation for which it was not explicitly trained.

2. Experiment Methodology

2.1. Baseline Model

This experiment builds directly on the pipeline from Experiment 2. The baseline model is a single **Logistic Regression classifier** trained on the 4096-dimensional feature vectors extracted from the *original* train-val split of the Oxford-IIIT Pet dataset. The feature extractor used was the pre-trained and frozen **AlexNet** model.

2.2. Test Dataset

To test invariance, four distinct test datasets were created from the original test split:

1. **Baseline (test_transform):** Standard images, resized to 224x224 and normalized.
2. **Shifted (test_transform_shifted):** Images were translated (shifted) by (20, 20) pixels before being normalized.
3. **Flipped (test_transform_flip):** Images were horizontally flipped (functional.hflip) before being normalized.
4. **Rotated (test_transform_rotated):** Images were rotated by 45 degrees (functional.rotate) before being normalized.

2.3. Evaluation

The evaluation process was as follows:

1. All four test sets were passed through the *same* frozen AlexNet model to extract four different sets of feature vectors.

2. The *single* Logistic Regression classifier (trained *only* on features from the original training set) was used to make predictions on all four feature sets.
3. The accuracy for each transformation was calculated and compared.

3. Experimental Results

The single classifier produced significantly different results depending on the transformation applied to the test images. The bar chart confirms the findings:

- **Baseline Accuracy: 71.9%**
- **Shifted Accuracy: 71.1%**
- **Flipped Accuracy: 71.8%**
- **Rotated Accuracy: 46.6%**

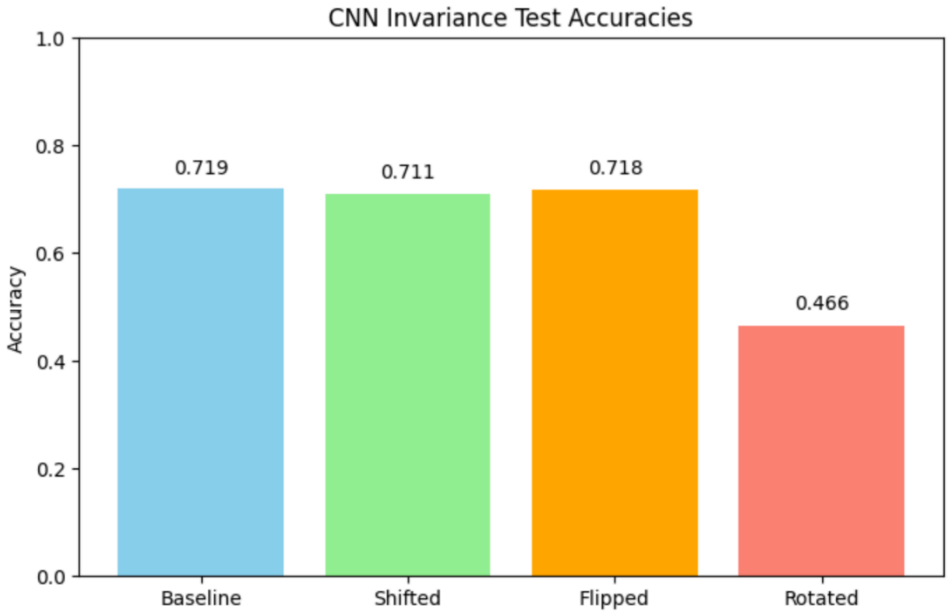


Figure 4 Accuracy of different test set based on different transformation

4. Conclusion

The results of this experiment clearly show the invariance properties of the CNN:

1. **High Translation Invariance:** The minimal drop in accuracy for *shifted* images (71.9% -> 71.1%) confirms that the CNN feature extractor is highly robust to translation. The convolutional layers, by sliding their kernels across the image, successfully generate similar feature vectors for the object regardless of its position.
2. **High Flip Invariance:** The *near-identical* accuracy for *flipped* images (71.9% -> 71.8%) demonstrates invariance to horizontal flipping. This is not an inherent property of CNNs but is likely learned. The AlexNet model, pre-trained on ImageNet, was almost certainly trained with horizontal flip augmentation, teaching it that a "flipped dog" is still a "dog."
3. **Lack of Rotation Invariance:** The *drastic drop in* accuracy for *rotated* images (71.9% -> 46.6%) proves the model is **not rotation-invariant**. The filters that learned to recognize features (e.g.,

an "upright ear" or "vertical leg") fail to activate correctly when those same features are rotated 45 degrees. This results in a completely different feature vector, which the classifier (trained only on upright features) cannot understand. This highlights a significant drawback in standard CNN architectures that must be addressed with data augmentation if rotational invariance is required.

4. Adversarial Attacks on Convolutional Neural Networks

Convolutional Neural Networks (CNNs) high performance is based on a statistical understanding of pixels of image. So, CNN is highly sensitive to pixels and a small change in it with the addition of noise can highly affect the performance of CNN. This is known as **adversarial attacks**. Research has shown that CNNs are highly vulnerable to adversarial attacks.

An adversarial attack involves adding a small, carefully crafted layer of noise (known as "perturbation") to an image. This noise is often not visible to the human eye, but it is specifically designed to exploit the model's logic and cause it to make a completely wrong prediction, often with high confidence.

This section of report explains a series of experiments to demonstrate and visualize the effect of two different types of adversarial attacks on two different CNN architectures:

1. An **untargeted Fast Gradient Sign Method (FGSM) attack** was performed on the custom **AlexNet** model. This attack successfully flipped a 'cat' classification to a 'dog' with high confidence. Then the attack was tested on entire test set to quantify the model's overall accuracy drop and generate confusion matrices.

2. A powerful, **targeted Projected Gradient Descent (PGD) attack** was executed against a pre-trained **ResNet-18** model. This attack successfully and precisely manipulated the model, forcing it to misclassify an image of a 'dog' as an 'ostrich' with 89.1% confidence.

The results from both attacks provide a clear demonstration of how CNNs can be fooled by suitable noise, highlighting CNN's limitation.

2.Experiment Methodology

2.1. Datasets

1. **CIFAR-10:** A dataset of 60,000, (32x32) pixel images in 10 classes ('plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck'). 50,000 images are used for training the custom AlexNet and the rest 10,000 images are used for demonstrating the effect of adversarial attack on AlexNet.

2. **ImageNet:** A large-scale dataset of high-resolution images in 1,000 classes (e.g., 'Samoyed', 'ostrich'). This dataset is used for demonstrating the effect of adversarial attack on ResNet-18.

2.2. CNN Architecture (Model):

1. **AlexNet:** A custom-built **AlexNet** architecture, modified to work with 32x32 inputs. This model was trained from scratch on CIFAR-10. Following table outlines the architecture of custom AlexNet.

Table 2 Custom AlexNet Architecture for CIFAR-10 Dataset

Layer #	Layer (type)	Output Shape	Kernel Size (kxk)	Stride / Padding	Param #
1	Conv2d (3 → 64)	[64, 32, 32]	5×5	stride=1, pad=2	4,800
2	MaxPool2d	[64, 16, 16]	3×3	stride=2, pad=1	0
3	Conv2d (64 → 192)	[192, 16, 16]	5×5	stride=1, pad=2	307,392
4	MaxPool2d	[192, 8, 8]	3×3	stride=2, pad=1	0
5	Conv2d (192 → 384)	[384, 8, 8]	3×3	stride=1, pad=1	663,936
6	Conv2d (384 → 256)	[256, 8, 8]	3×3	stride=1, pad=1	884,992
7	Conv2d (256 → 256)	[256, 8, 8]	3×3	stride=1, pad=1	590,080
8	MaxPool2d	[256, 4, 4]	3×3	stride=2, pad=1	0
9	Linear (FC)	[4096]	(FC)	input = 256×4×4 → 4096	16,781,312

10	Linear (FC)	[4096]	(FC)	4096 → 4096	16,781,312
11	Linear (FC)	[10]	(FC)	4096 → 10	40,970
	Total params		-	-	**36,054,794**

2. **ResNet-18:** A standard, pre-trained **ResNet-18** is used to show the effect of adversarial attack on ImageNet dataset. This modern and highly accurate model is suitable for classifying ImageNet like complex and large dataset. We have used pre-trained model because training the model from scratch on large dataset will take huge amount of time due to lack of resources.

2.3. Attack Algorithms

Two different "white-box" attacks (which require model access) were used.

1. **Fast Gradient Sign Method (FGSM):**
    - Used for the ***untargeted*** **attack** on CIFAR-10**.** Untargeted attack tries to make the model wrong in any way possible.
    - How it Works: FGSM is a single-step attack. It calculates the model's loss to an image and then adds noise to the image in the direction that maximizes this error. It is very fast but less effective than iterative methods.

$$adv\_x = x + \epsilon * \text{sign}(\nabla_x J(\theta, x, y))$$

2. **Projected Gradient Descent (PGD):**
    - Used for the difficult, ***targeted*** attack on ImageNet. Targeted attack tries to make the model predict a *specific, new* target class.
    - How it Works: PGD is a much stronger, iterative version of FGSM. Instead of taking one large step, it takes multiple small steps. After each step, it "projects" the noise back to ensure it stays within the allowed limit (epsilon). This multi-step approach is far more effective at finding the optimal perturbation to reliably fool a model.

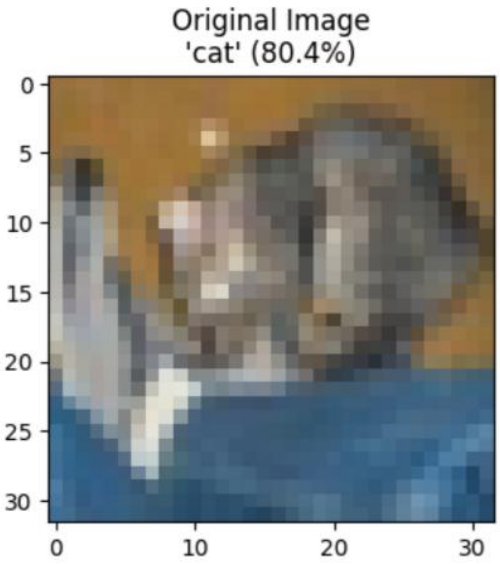
3. Experiment 1: Untargeted Attack on CIFAR-10 (AlexNet)

This experiment was conducted to demonstrate effect of targeted attack on the model's prediction and performance. The **FGSM** method was used to attack the model. Custom AlexNet was used as model. This experiment was divided into two parts: Qualitative Analysis and Quantitative Analysis.

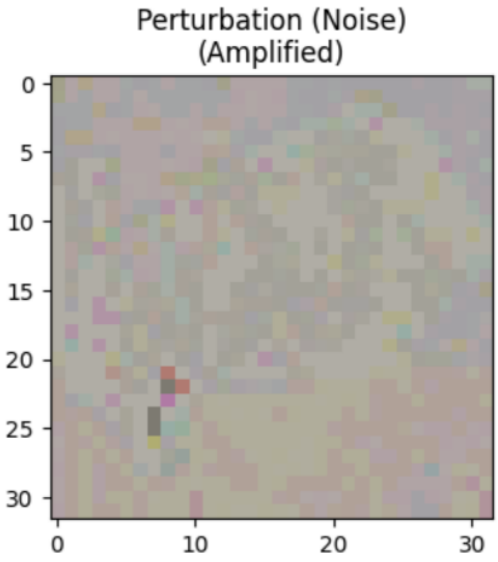
3.1. Qualitative Analysis (Single Image)

A single image is used to show the effect of attack on model's prediction. Using FGSM method, noise was added in a single image. Then the model predicts the clean (original) image and perturbated image. This experiment greatly observes the pixel structure of image before and after attack and shows how the change in pixels forces the model to make wrong prediction, even when the changes are not significant. It also shows how the attack tricks the model to make the wrong prediction with high confidence.

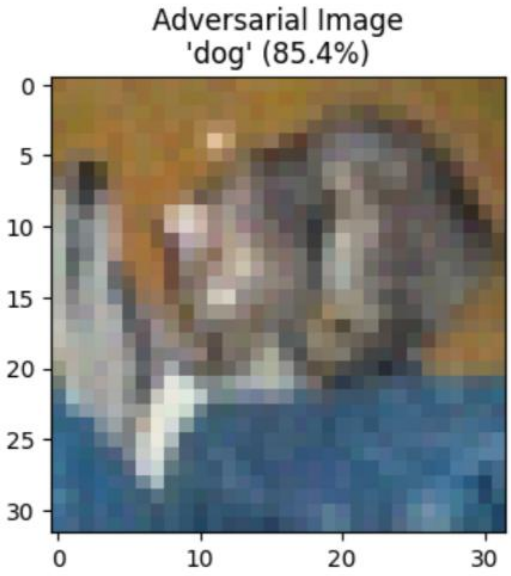
1. **Original Image:** The plot confirmed the model's correct baseline. The model predicted the image as 'Cat' with 80.4% confidence.



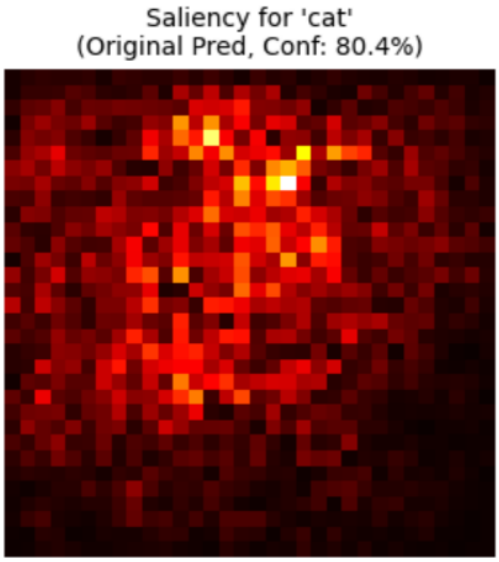
2. **Noise added to Image:** A small amount of noise was added to the image by FGSM. The pot shows the noise.



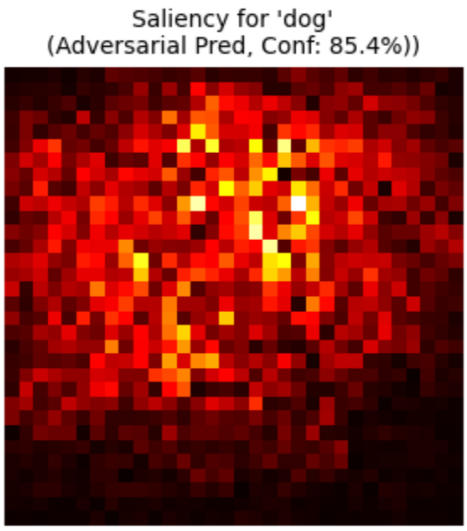
3. **Adversarial Image:** The plot showed the model's new, incorrect prediction. The change in image by noise is not visible to human eye, but the model has made wrong prediction. The image predicted as ‘Dog’. Interesting fact is that the model is very confident in their prediction which makes this attack more dangerous.



4. **Original Saliency Map:** We generated a heatmap showing *which pixels* the model used to make its correct "cat" prediction. It correctly focused in the centre of the image where cat is present. It highly focusses on the location where cat's face is present in the image



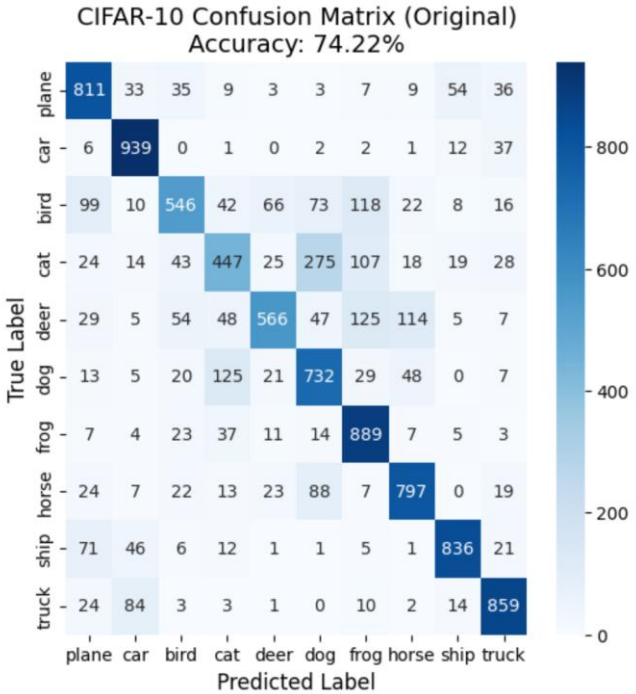
5. **Adversarial Saliency Map:** This map showed that the model's "attention" had completely shifted. It was now focusing on seemingly random pixels in the background that corresponded to the invisible noise, which were "tricking" it into seeing a 'dog'.



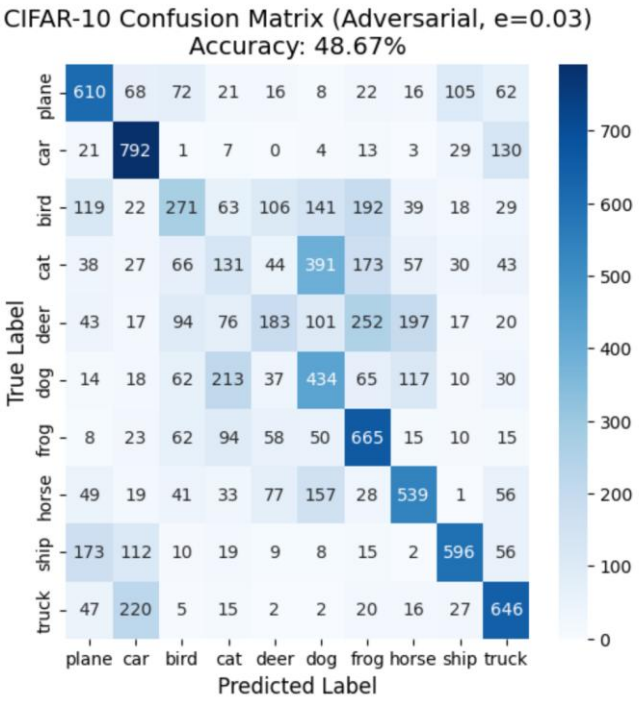
3.2. Quantitative Analysis (Batch Attack)

Quantitative analysis shows the effect of attack on the overall model's performance. In the experiment we ran untargeted FGSM attack on the entire 10,000-image CIFAR-10 test set to measure the large-scale impact.

- **Accuracy Drop:** The model's baseline accuracy on clean images was **74.22%**. After the adversarial attack (with epsilon = 0.1), the accuracy drops to **48.6%**. This demonstrates a big failure of the model due to adversarial attack.
- **Confusion Matrices:** Confusion matrix shows classification capability of the model before and after attack.
    - **"Before" (Clean) Matrix:** The deep coloured diagonal element indicates that model is correctly classifying most of the images. That means model is well trained and can predict accurately.



- **"After" (Adversarial) Matrix:** The intensity of colour along diagonal line faded. The predictions were wrong in almost all class. This proves that the attack didn't just cause a few mistakes; it destroyed the model's entire ability to distinguish between *any* of the 10 classes.



4. Experiment 2: Targeted Attack on ImageNet / ResNet-18

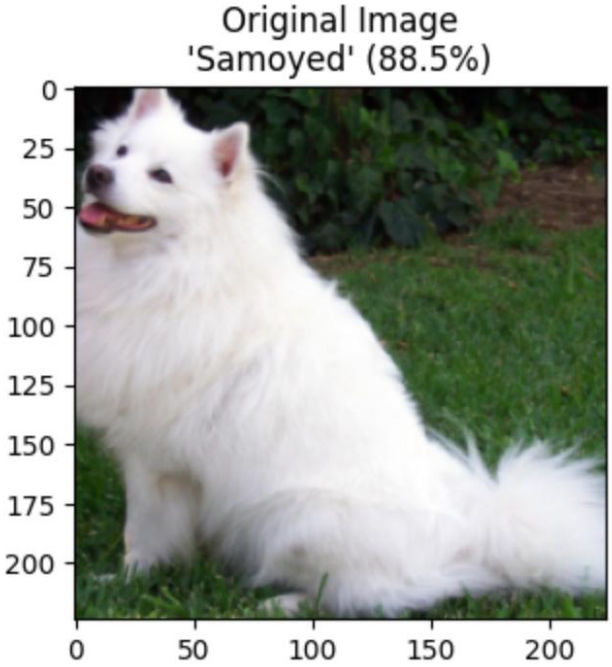
This experiment was designed to simulate a more malicious, specific attack on a high-performance model. ResNet-18 is a robust and highly accurate model. A simple FGSM attack might fail, so a powerful iterative method (PGD) was used to ensure a high success rate. A targeted attack forces the model to classify the image to a targeted class.

4.1. Experiment Process:

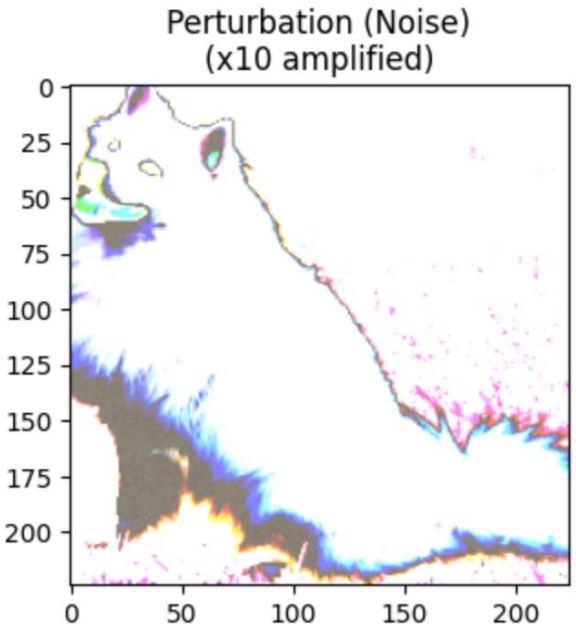
1. A ResNet-18 pre-trained on ImageNet dataset is used as model.
2. At first, a high-resolution image of dog (original image) was fed to the model to predict its class.
3. The **PGD attack** with target ‘**ostrich**’ was run on the image. The algorithm iteratively applied small, targeted noise steps over multiple iterations to find the optimal perturbation.
4. The resulting adversarial image (which still looked like a dog) was fed back into the ResNet-18.
5. The model predicts the class of the adversarial image.

4.2. Results:

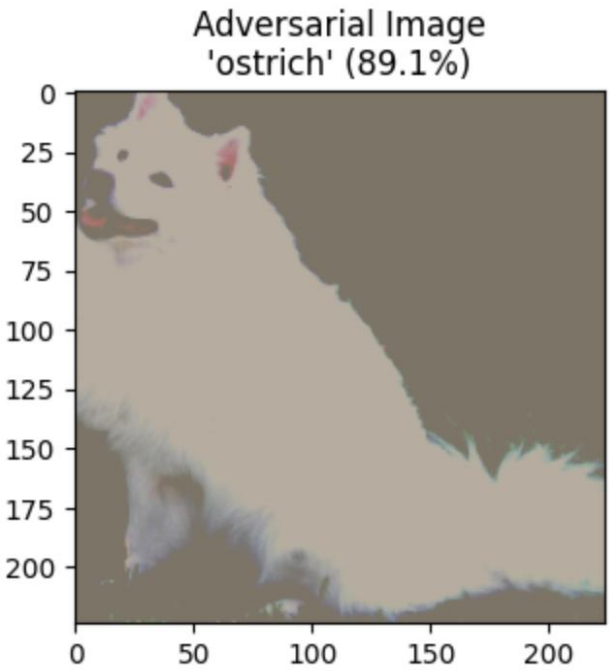
1. **Original Prediction:** The model classifies the dog with 88.5% confidence



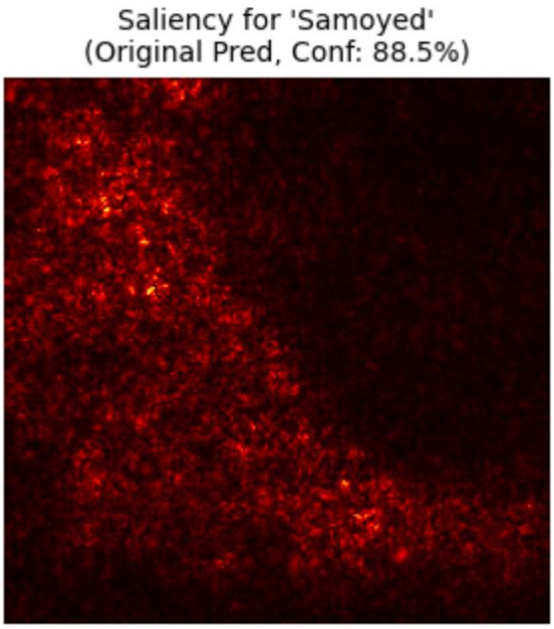
2. **Perturbation plot** shows the targeted noise added to the image



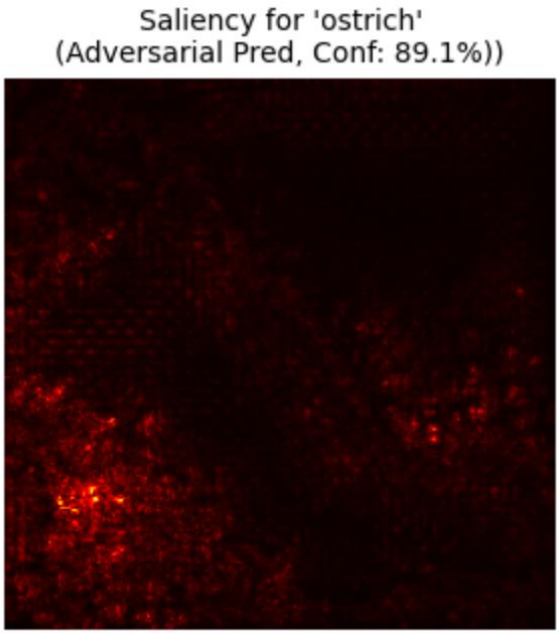
3. **Adversarial Attack Prediction:** The model classified the adversarial image as **'ostrich'** with 89.1% confidence. This shows that even a robust, modern model can be fooled into predicting a specific, arbitrary class with high confidence, provided a sufficiently powerful attack (like PGD) is used.



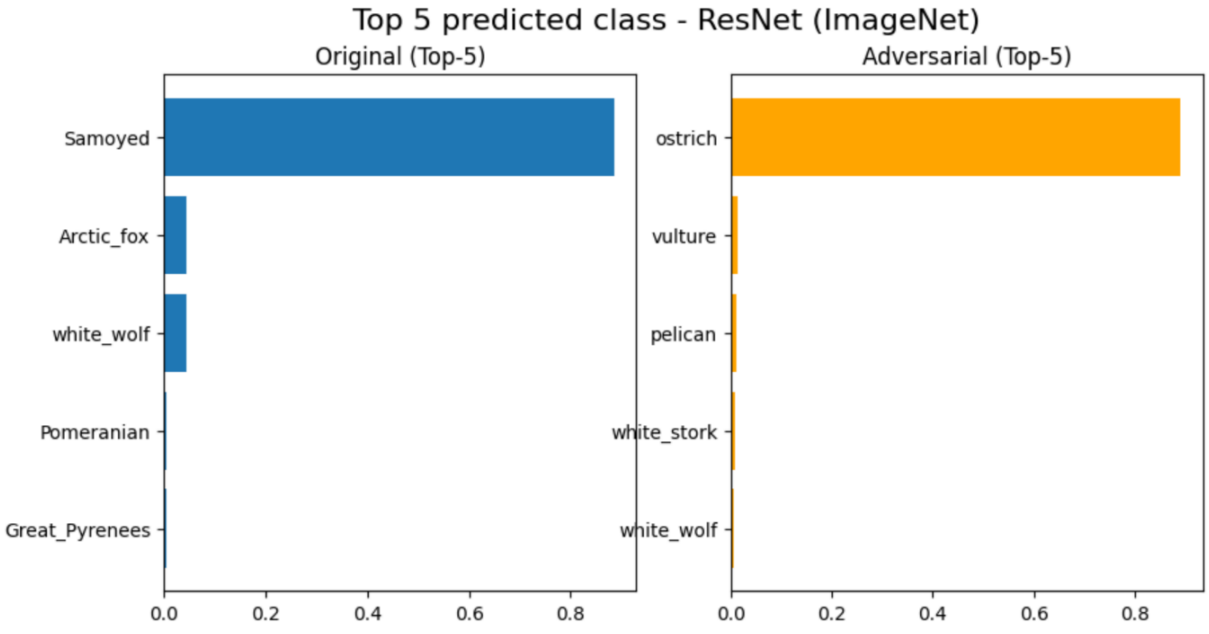
4. **Original Saliency Map**: The heatmap shows the model has mostly focused the region where the dog is present in the image.



5. **Adversarial Saliency Map:** The heatmap is vastly different from the original one. This map showed that the model's "attention" had completely shifted. The attack forces the model to focus on those regions which are important for ostrich class.



6. **Top 5 Predicted Class:** This plot is a powerful demonstration of the attack's precision. It proves that the model's output can be not just "broken" or confused, but precisely manipulated to a completely unrelated category with high certainty.



5. Analysis: Why Adversarial Attacks Happen?

These experiments demonstrate that adversarial attacks are not random flukes but are a fundamental vulnerability resulting from *how* deep learning models learn. The primary reasons these attacks are so effective are:

1. **Models Learn "Non-Robust Features":** Humans classify objects using "robust features" (e.g., a dog's nose, a cat's pointy ears). But CNN learns *any* statistical pattern that is predictive,

including "non-robust features". Subtle or high-frequency textures or pixel combinations that are imperceptible to humans but highly correlated with a label in the training data. The adversarial attack identifies these non-robust features (e.g., a noise pattern for "ostrich") and injects them into the "dog" image. The model, weighing all evidence, sees the "dog" features *and* a strong "ostrich" signal, causing it to misclassify.

2. **Linearity in High-Dimensional Space:** While CNNs are complex, their underlying operations (like convolutions and ReLU activations) are largely linear. An image is a very high-dimensional space. An attacker can make thousands of tiny, imperceptible changes to individual pixels. Because the model is mostly linear, these tiny changes add up cumulatively, creating a massive shift in the final output and pushing the image's feature vector across a decision boundary into a different class.

6. Conclusion

These experiments successfully demonstrated the vulnerability of modern CNNs towards Adversarial attack.

1. **CNNs are not robust:** They are easily confused by inputs that are intentionally crafted to exploit their internal logic.
2. **Attacks are scalable:** A simple, fast attack (FGSM) can cause a model's overall accuracy to collapse, as shown by the 74% -> 48% drop on CIFAR-10.
3. **Attacks can be precise:** A powerful, iterative attack (PGD) can force a robust model to see *anything* the attacker desires (e.g., a "dog" as an "ostrich").
4. **Saliency maps** visually confirmed that the attack works by diverting the model's attention away from the real features of the image and onto the engineered noise.

This final experiment concludes that these powerful models rely heavily on statistical patterns (non-robust features) that can be easily exploited. This vulnerability has significant security implications for any real-world system relying on AI for image recognition, from autonomous vehicles to medical diagnostics.

Overall Project Conclusion

This four-part project provided a comprehensive exploration of Convolutional Neural Networks, from their initial construction to their ultimate failure. The four experiments led to a set of findings:

1. **Performance is Not Automatic (Experiment 1):** A "good" architecture is not enough. We saw that systematic hyperparameter and architectural tuning via Optuna was essential, improving our baseline model's accuracy to a strong **87.38%** on CIFAR-10.
2. **Feature Extraction is a Powerful Tool (Experiment 2):** Training a model from scratch is not always necessary. The features from a pre-trained AlexNet were powerful enough for a simple Logistic Regression model to achieve **74.54%** accuracy on the complex 37-class Oxford-IIIT Pet dataset, demonstrating a highly efficient alternative.
3. **Invariance is Specific, Not General (Experiment 3):** CNNs are not "smart" in a human sense. They are inherently invariant to **translation** (71.1% vs 71.9% accuracy) but not to **rotation**. The model's accuracy collapsed to **46.6%** when faced with a 45-degree rotation, proving that this robustness must be explicitly learned through data augmentation.
4. **High Accuracy Does Not Imply Robustness (Experiment 4):** This was the project's most critical finding. The very logic that allows models to be so accurate also makes them vulnerable. Both the custom AlexNet and the powerful ResNet-18 were easily fooled by imperceptible noise. The successful untargeted and targeted attack proved that a model's high confidence is not a measure of certainty, but merely a measure of its knowledge with statistical patterns that can be easily exploited.

In conclusion, this project provides a complete overview from model creation to optimization and, finally, to "breaking" the model. The key takeaway is that a model's high accuracy should not be mistaken for robust, human-like understanding. These systems are powerful pattern identifier, but their dependency on non-robust, high-dimensional features makes them vulnerable to manipulation.