**Slide 1: Title Slide**

**Transcript:**  
“Hello, my name is Ajayeb, and today I will present my project on Neural Network Models for Object Recognition using CNNs and Transfer Learning.  
In this project, I focus on applying Convolutional Neural Networks, data augmentation, and fine-tuning pre-trained models on the CIFAR-10 dataset to classify images into 10 object categories.

**Slide 2: Table Content**

**Transcript:**

In this slide, I present the content overview of my presentation. We start with the project objectives and dataset overview, followed by data preprocessing and augmentation. Next, I will explain the CNN architecture and transfer learning approach using VGG16. Then, we cover training strategies, hyperparameters, performance visualization, and evaluation on the test set. Finally, I discuss comparative analysis, strengths, limitations, lessons learned, and conclude with references.

**Slide 3: Objectives**

**Visual:** Bullet points of project objectives  
**Transcript:**  
“The objectives of this project are:

1. Prepare and explore the CIFAR-10 dataset, including training, validation, and test splits.
2. Build a Convolutional Neural Network (CNN) from scratch.
3. Apply data augmentation to improve model generalization.
4. Use transfer learning with VGG16, adding custom layers for CIFAR-10 classification.
5. Fine-tune the top layers of VGG16 to improve performance.
6. Evaluate the model with metrics such as accuracy, precision, recall, confusion matrix, and classification report.
7. Discuss the strengths, limitations, and trade-offs of CNN and transfer learning approaches.

**Slide 4: Dataset – CIFAR-10**

**Visual:** Table summarizing dataset stats + grid of 5x5 sample images  
**Transcript:**  
The CIFAR-10 dataset contains 60,000 color images of size 32x32 pixels across 10 classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck.  
I split the data into 50,000 training images and 10,000 test images, then further split the training set into 40,000 images for training and 10,000 images for validation.  
This split ensures hyperparameter tuning and model evaluation are unbiased.  
Visualizing 25 sample images helps inspect class diversity and confirm data integrity before training.

**Slide 5: Data Preprocessing & Augmentation**

**Visual:** Flowchart: Load → Normalize → Split → Augment  
**Transcript:**  
“This slide presents the data preparation phase, which is the foundation for training a reliable Convolutional Neural Network model.

As shown in the flowchart, the process begins with **loading the dataset**, which includes all the image samples that will be used for training and evaluation.

Next, all images are **normalized from a range of 0–255 to 0–1**. This step ensures that the pixel values are scaled consistently, improving numerical stability and helping the model converge faster during training.

After normalization, the dataset is **split into three subsets** — training, validation, and testing.

* The **training set** is used to teach the model.
* The **validation** helps tune hyperparameters and monitor performance during training.
* The **test set** provides an unbiased evaluation of the final model’s accuracy.

Then, we apply **data augmentation** to the training set.  
This involves transformations such as rotations, width and height shifts, and horizontal flips.  
Data augmentation increases the effective size of the dataset, improves generalization, and helps prevent overfitting by exposing the model to more variations of the input images.

It’s important to note that the **validation and test sets are not augmented**, to ensure the performance metrics remain fair and consistent.

In summary, this phase prepares the data in a way that enhances model learning efficiency and generalization.

**Slide 6: CNN architecture**

**Visual:** Diagram: Conv → Pool → Dense → Dropout → Output  
**Transcript:**  
in this slide, I will explain a brief overview of the CNN architecture shown in the diagram.  
The model begins with convolutional layers, which extract key features such as edges, colors, and textures from the input images.  
Next, the pooling layer reduces spatial dimensions while keeping the most important information, helping to make the model more efficient and less prone to overfitting.  
Then, the dense layer acts as a fully connected network that combines the extracted features to make sense of the learned representations.  
Dropout is added to prevent overfitting by randomly disabling some neurons during training.  
Finally, the output layer uses the softmax activation function to classify the image into its corresponding category.  
Overall, this baseline CNN helps establish a foundation for understanding how the model learns before applying transfer learning techniques

**Slide 7: Transfer Learning Concept**

**Visual:** VGG16 base + custom classifier top layers diagram  
**Transcript:**  
in this slide, I will explain the transfer learning architecture used in my project.  
Transfer learning adapts a pre-trained model to a new task.  
I used VGG16 trained on ImageNet, which has already learned generic image features such as edges, textures, and shapes.  
I removed the original fully connected top layers and added a global average pooling layer, a dense layer with 256 neurons, a dropout layer for regularization, and a 10-class softmax output for CIFAR-10 classification.  
This setup allows the model to combine pre-trained visual knowledge with new task-specific learning efficiently.

**Slide 8: Initial Training (Frozen Base)**

**Visual:** Flowchart showing frozen base → custom layers → output  
**Transcript:**

In this slide, I will explain the initial training phase.  
The base layers of VGG16 were frozen to preserve their pre-trained ImageNet features.  
Only the custom classifier layers were trained for 5 epochs using the Adam optimizer with a learning rate of 0.001 and a batch size of 64.  
Monitoring training and validation accuracy and loss ensured the top layers learned dataset-specific features without overfitting.  
This phase helps the model adapt to the new dataset while retaining the powerful representations from VGG16.

**Slide 9: Fine-Tuning Top Layers**

**Visual:** Highlight top layers of VGG16 being unfrozen  
**Transcript:**  
In this phase, we apply fine-tuning to improve model performance.  
The blue layers of VGG16 remain frozen to retain pre-trained features, while the **orange layers are unfrozen** to adapt high-level representations to CIFAR-10.  
These layers are trained with a **smaller learning rate of 1e-5** to prevent large updates from destroying learned knowledge.  
Fine-tuning allows the model to better capture dataset-specific patterns, improving validation accuracy and reducing loss, especially for challenging classes.  
The diagram highlights which layers are frozen versus unfrozen and shows how the trainable layers feed into the classifier.

**Slide 10: Hyperparameters & Training Strategy**

**Visual:** Table of hyperparameters + Two line plots – Training vs Validation Accuracy & Loss (from Phase 4, Step 8)

| **Parameter** | **CNN** | **VGG16 Transfer** |
| --- | --- | --- |
| Optimizer | Adam | Adam |
| Learning Rate | 0.001 | 0.001 → 1e-5 (fine-tune) |
| Epochs | 10 | 5 → 3 (fine-tune) |
| Batch Size | 64 | 64 |
| Dropout | 0.5 | 0.5 |
| Data Augmentation | Yes | Yes |

**Transcript:**  
On this slide, we summarize the key **hyperparameters** for our two models: a standard CNN and a VGG16 model using transfer learning.

The table shows that both models used the **Adam optimizer**, a **batch size of 64**, and **dropout of 0.5** to reduce overfitting. Data augmentation was applied to both to enhance generalization.

A key difference is the **learning rate and epochs**: for the CNN, we trained for 10 epochs with a learning rate of 0.001. For VGG16, we first trained for 5 epochs at the same rate, then fine-tuned the top layers for 3 more epochs using a much smaller learning rate of 1e-5. This adjustment prevents large updates from destroying the pre-trained weights.

The diagram combining the initial training and fine-tuning phases, we can observe the learning curves. The line plots show:

* **Validation and Training Accuracy**
* **Validation and Training Loss**

You can see that **fine-tuning the top layers of VGG16 with a lower learning rate stabilizes accuracy**, while **dropout and data augmentation help prevent overfitting**, keeping the validation performance consistent.

These curves confirm that the model is learning meaningful features, and fine-tuning improves the classifier’s ability to distinguish CIFAR-10 classes. This diagram comes directly from the output of Phase 4, Step 8 of the code.

**Slide 11: Training & Performance Visualization**

**Visual:** Two line plots – Accuracy vs Epochs and Loss vs Epochs (combined initial + fine-tune)  
**Transcript:**  
In this slide, we can see the learning curves that summarize our model’s training performance.  
The graphs show a steady increase in both training and validation accuracy, along with a consistent decrease in loss, which indicates effective learning.  
Data augmentation played an important role in helping the model generalize — as seen from the stable validation accuracy despite variations during training.  
This confirms that the model successfully learned meaningful features while avoiding overfitting.

**Slide 12: Evaluation on Test Set**

**Visual:** Confusion matrix heatmap + small test accuracy table  
**Transcript:**  
**In this slide**, we evaluate the final performance of the fine-tuned VGG16 model on the test set.

The **table** on the right summarizes key metrics. The model achieved **68% accuracy**, which is a clear improvement compared to the baseline CNN’s 55%. The precision, recall, and F1-score averages are all around **0.67**, indicating balanced performance across classes.

The **confusion matrix** on the left visualizes the model’s predictions.  
The **diagonal cells** show correct classifications — for example, **airplane**, **automobile**, and **ship** are recognized with high accuracy.

In contrast, similar-looking categories like **cat** and **dog** are more frequently misclassified, which is common in CIFAR-10 due to subtle visual similarities.

The **classification report** provides detailed metrics for each class. We can see that airplane and automobile achieve the highest F1-scores (around 0.74–0.77), while cat has the lowest at 0.45.

Overall, this evaluation confirms that **fine-tuning the pre-trained VGG16 and using data augmentation** significantly improved generalization and feature learning compared to training a CNN from scratch.

**Slide 13: Comparative Discussion**

**Visual:** Table comparing Classical ML vs CNN vs Transfer Learning

| **Approach** | **Feature Extraction** | **Training Time** | **Accuracy** | **Pros** | **Cons** |
| --- | --- | --- | --- | --- | --- |
| SVM/KNN | Manual | Long | Low (~30–40%) | Simple | Manual features |
| CNN | Automatic | Medium | Medium (~53%) | Learns features | Needs GPU |
| Transfer Learning | Automatic + Pretrained | Shorter | High (~60%) | Fast + accurate | Source-target mismatch |

**Transcript:**  
In this slide, I compare three methods: SVM/KNN, CNN, and Transfer Learning.

Classical ML like SVM/KNN needs manual features and gives low accuracy.  
CNNs learn features automatically and improve accuracy but need more computing power.  
Transfer Learning, using VGG16, is faster and more accurate since it reuses pre-trained features.

However, performance can drop if the source and target datasets differ too much.

**Slide 14: Strengths of My Approach**

**Visual:** Icons with bullets (speed, accuracy, scalability, robustness)  
**Transcript:**  
In this slide, I summarize the main strengths of my approach.

First, the model performs **automatic feature extraction**, so we don’t need to manually design features.  
Second, **training time is reduced** because the base of VGG16 is pre-trained on ImageNet.  
Third, **data augmentation and fine-tuning** help improve generalization and prevent overfitting.  
Finally, the model is **scalable**, meaning it can be extended to larger datasets or more complex architectures.

**Slide 15: Limitations & Trade-offs**

**Visual:** Icons with bullets  
**Transcript:**

In this slide, I highlight the main limitations of my approach.

First, the model requires **GPU resources** for efficient training.  
Fine-tuning offered **only modest improvements** beyond the frozen layers.  
There’s also **class confusion** for visually similar objects, such as cats and dogs.  
Finally, the model is **sensitive to hyperparameters** like learning rate and batch size, which can affect stability and performance

**Slide 16: Lessons Learned & Conclusion**

**Visual:** Mindmap + model diagram + summary bullets  
**Transcript:**  
In this slide, we summarize the key lessons learned from my project. Using validation sets helped tune hyperparameters and avoid overfitting. Data augmentation improved model robustness, while fine-tuning allowed the model to leverage prior knowledge. Metrics like precision, recall, and F1-score gave us a deeper understanding beyond simple accuracy. Visualizations of learning curves and confusion matrices helped interpret how the model was performing

Here we visualize the model used in the project. Images first enter the pre-trained VGG16 base. The features are pooled globally, then pass through a dense layer and dropout before reaching the output layer with 10 classes. This diagram helps summarize the flow of data and shows how transfer learning and custom layers combine

Finally, the summary points highlight the main outcomes of our project. Data augmentation and fine-tuning improved the model’s performance. We achieved around 60% test accuracy, showing effective feature extraction and classification. For future work, deeper models, alternative pre-trained architectures like ResNet, or self-supervised learning could further improve accuracy

**Slide 17: Reference**

**Transcript:**

In this slide, I list the main references used in this project. These include the original CIFAR-10 dataset paper, the VGG16 architecture by Simonyan and Zisserman, Keras and TensorFlow documentation, and key research on deep learning, data augmentation, and transfer learning. These sources guided the implementation, model design, and evaluation of my CNN and transfer learning approach

**References**

* Krizhevsky, A. & Hinton, G. (2009) *Learning multiple layers of features from tiny images*. [online] Available at: https://www.cs.toronto.edu/~kriz/cifar.html [Accessed 12 October 2025].
* Simonyan, K. & Zisserman, A. (2015) ‘Very deep convolutional networks for large-scale image recognition’, *ICLR*. Available at: https://arxiv.org/abs/1409.1556 [Accessed 12 October 2025].
* Chollet, F. (2015) *Keras: The Python Deep Learning library*. [online] Available at: https://keras.io [Accessed 12 October 2025].
* TensorFlow Developers (2023) *TensorFlow: An end-to-end open source platform for machine learning*. [online] Available at: https://www.tensorflow.org [Accessed 12 October 2025].
* Goodfellow, I., Bengio, Y. & Courville, A. (2016) *Deep Learning*. Cambridge, MA: MIT Press.