# Deep Learning: Image Classification with CNN

Team G3

Authors: Jaime Velez, Larry Davila, Latif Calderon, and Juan Andujar

Date: 20 Dec 2024

# Project Overview

## Selected Dataset: CIFAR-10

- The CIFAR-10 dataset consists of 60000 32x32 color images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

## Objective:

- Build a Convolutional Neural Network (CNN) model to classify images from a given dataset into predefined categories/classes.

## Hypothesis:

- CNN can effectively classify CIFAR10 images with high accuracy.

## Data Loading Method:

- Direct import using tensorflow.keras.datasets.

## Preprocessing Techniques used:

- Normalization & One-Hot Encoding

## Building Model Architecture

## Model Training & Evaluation

- Accuracy & Loss Results and Confusion Matrix

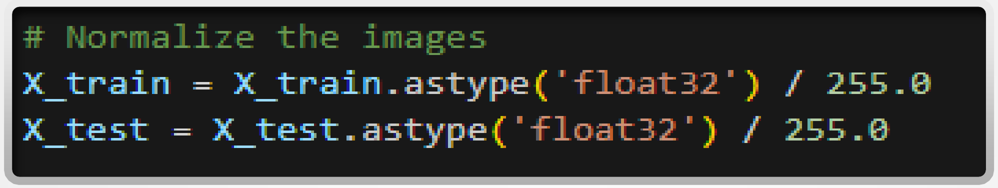
## Transfer Learning

- VGG16 Architecture used

## Bonus: Model Deployment

- Docker Container, Google Cloud Storage, Vertex AI & Flask Web App

# Preprocessing Techniques



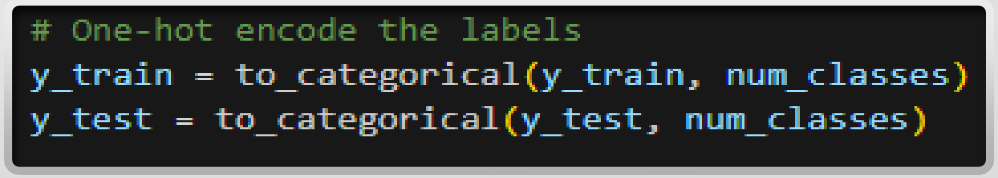
```
# Normalize the images
X_train = X_train.astype('float32') / 255.0
X_test = X_test.astype('float32') / 255.0
```

## Normalization

- Scaled pixel values of images to [0, 1] , converting them to floats, which accelerates model convergence during training.

## One-Hot Encoding:

- Transforms categorical labels into binary vectors. Ensuring compatibility with machine learning models and preventing unintended ordinal relationships between classes.



```
# One-hot encode the labels
y_train = to_categorical(y_train, num_classes)
y_test = to_categorical(y_test, num_classes)
```

# Data Wrangling and Cleaning

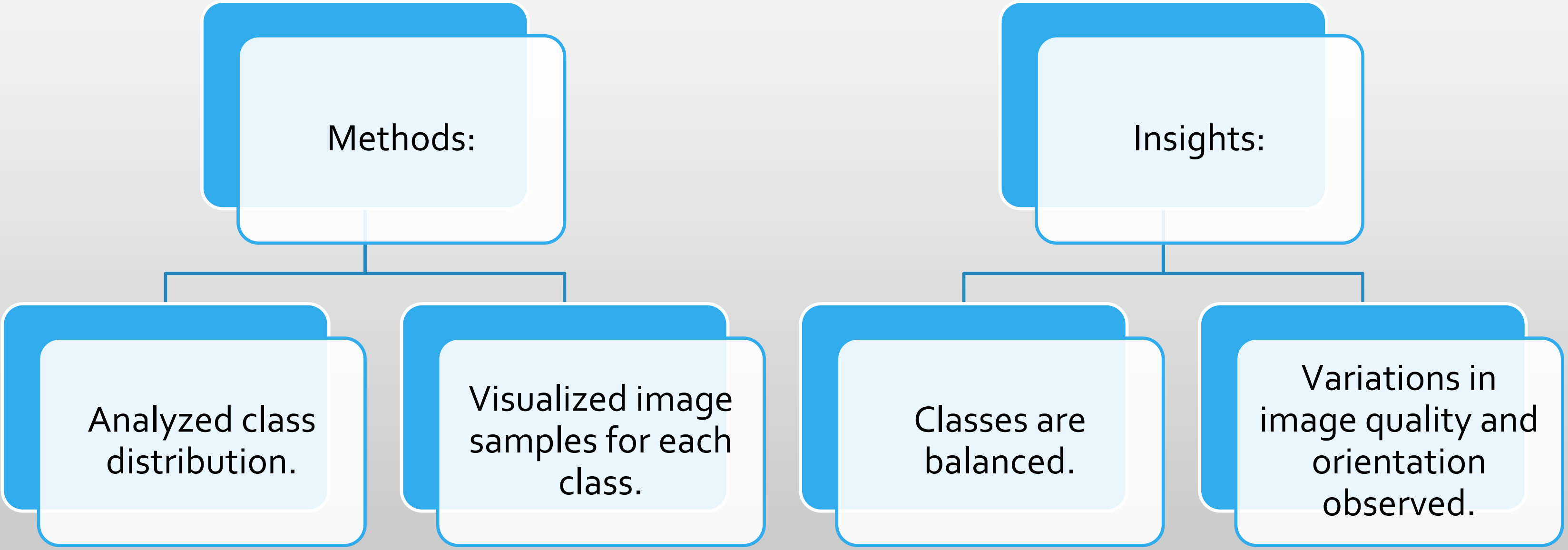
## CHALLENGES:

- HANDLING DATASET SIZE EFFICIENTLY.
- ENSURING DATA WAS NORMALIZED FOR MODEL TRAINING.

## RESOLUTION:

- UTILIZED EFFICIENT PREPROCESSING.

# Exploratory Data Analysis



Methods:

Analyzed class distribution.

Visualized image samples for each class.

Insights:

Classes are balanced.

Variations in image quality and orientation observed.

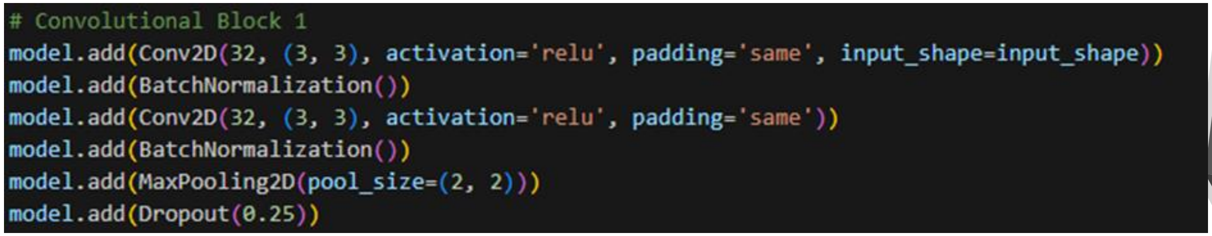
# Model Building

## Architecture:

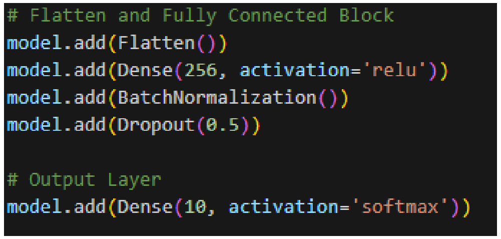
- 4 Convolutional Blocks: ReLU activation, 'same' padding.
- Batch Normalization, Max Pooling, and Dropout layers.
- Final Dense and Softmax layers.

## Hyperparameters:

- Optimizer: Adam
- Loss Function: Categorical Crossentropy
- Batch Size: 64, Epochs: 60 (Early Stopping enabled).



```
# Convolutional Block 1
model.add(Conv2D(32, (3, 3), activation='relu', padding='same', input_shape=input_shape))
model.add(BatchNormalization())
model.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
```



```
# Flatten and Fully Connected Block
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))

# Output Layer
model.add(Dense(10, activation='softmax'))
```

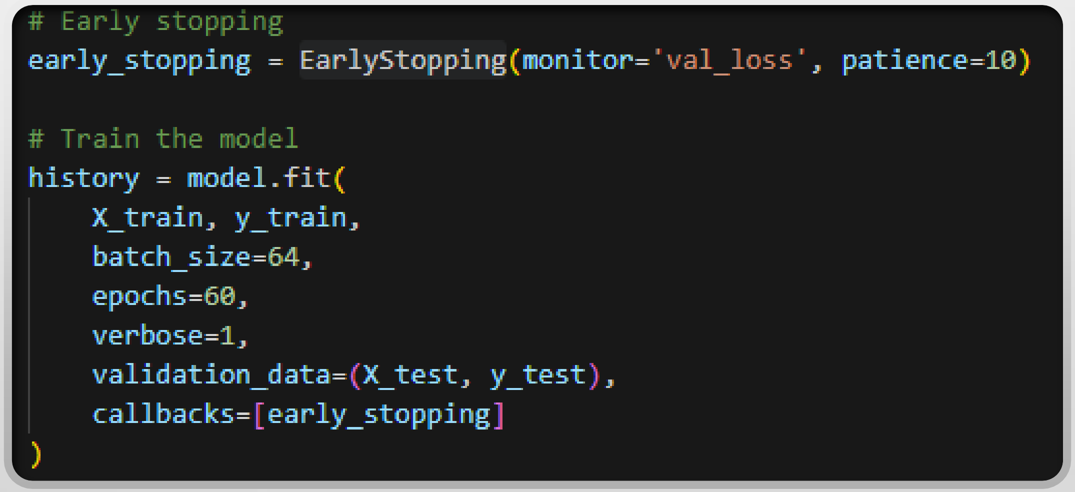
# Training Process

**Training Results:**

- Accuracy: 96.82% (11 epochs)
- Test Accuracy: 85%
- Test Loss: 0.64

**Optimization:**

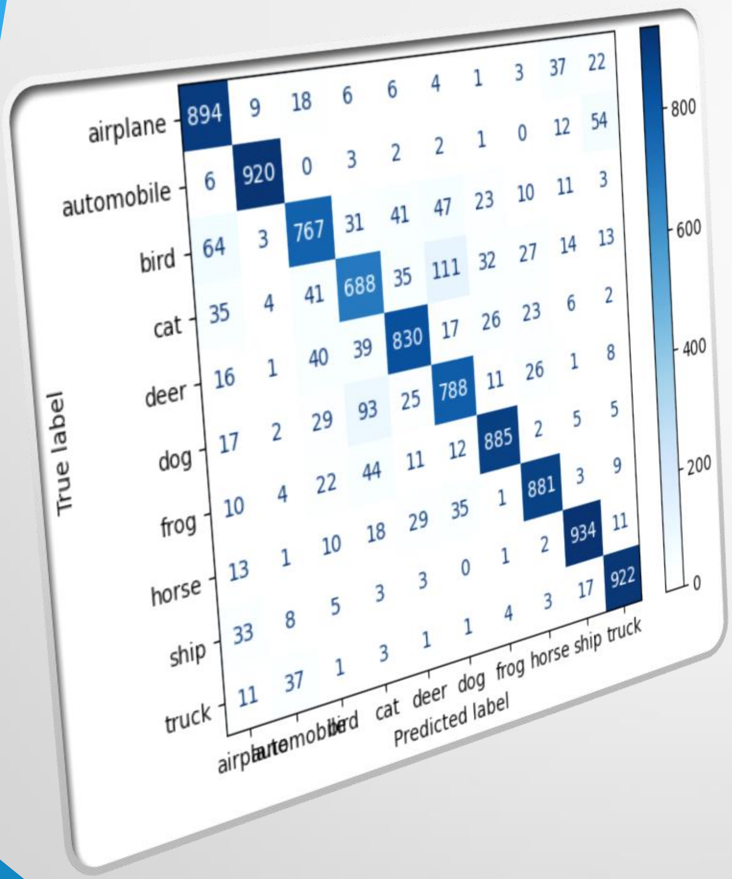
- Early stopping function used for efficiency.



```
# Early stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=10)

# Train the model
history = model.fit(
    X_train, y_train,
    batch_size=64,
    epochs=60,
    verbose=1,
    validation_data=(X_test, y_test),
    callbacks=[early_stopping]
)
```

# Insights from the Confusion Matrix



| True label | airplane | automobile | bird | cat | deer | dog | frog | horse | ship | truck |
|---|---|---|---|---|---|---|---|---|---|---|
| airplane | 894 | 9 | 18 | 6 | 6 | 4 | 1 | 3 | 37 | 22 |
| automobile | 6 | 920 | 0 | 3 | 2 | 2 | 1 | 0 | 12 | 54 |
| bird | 64 | 3 | 767 | 31 | 41 | 47 | 23 | 10 | 11 | 3 |
| cat | 35 | 4 | 41 | 688 | 35 | 111 | 32 | 27 | 14 | 13 |
| deer | 16 | 1 | 40 | 39 | 830 | 17 | 26 | 23 | 6 | 2 |
| dog | 17 | 2 | 29 | 93 | 25 | 788 | 11 | 26 | 1 | 8 |
| frog | 10 | 4 | 22 | 44 | 11 | 12 | 885 | 2 | 5 | 5 |
| horse | 13 | 1 | 10 | 18 | 29 | 35 | 1 | 881 | 3 | 9 |
| ship | 33 | 8 | 5 | 3 | 3 | 0 | 1 | 2 | 934 | 11 |
| truck | 11 | 37 | 1 | 3 | 1 | 1 | 4 | 3 | 17 | 922 |

Predicted label

800
600
400
200
0

Strengths:

- High accuracy for airplanes, ships, trucks.

Weaknesses:

- Highest Confusion between cats vs. dogs.

# Transfer Learning

Architecture: VGG16 pre-trained model.

Results:

Test Accuracy: 65.2%

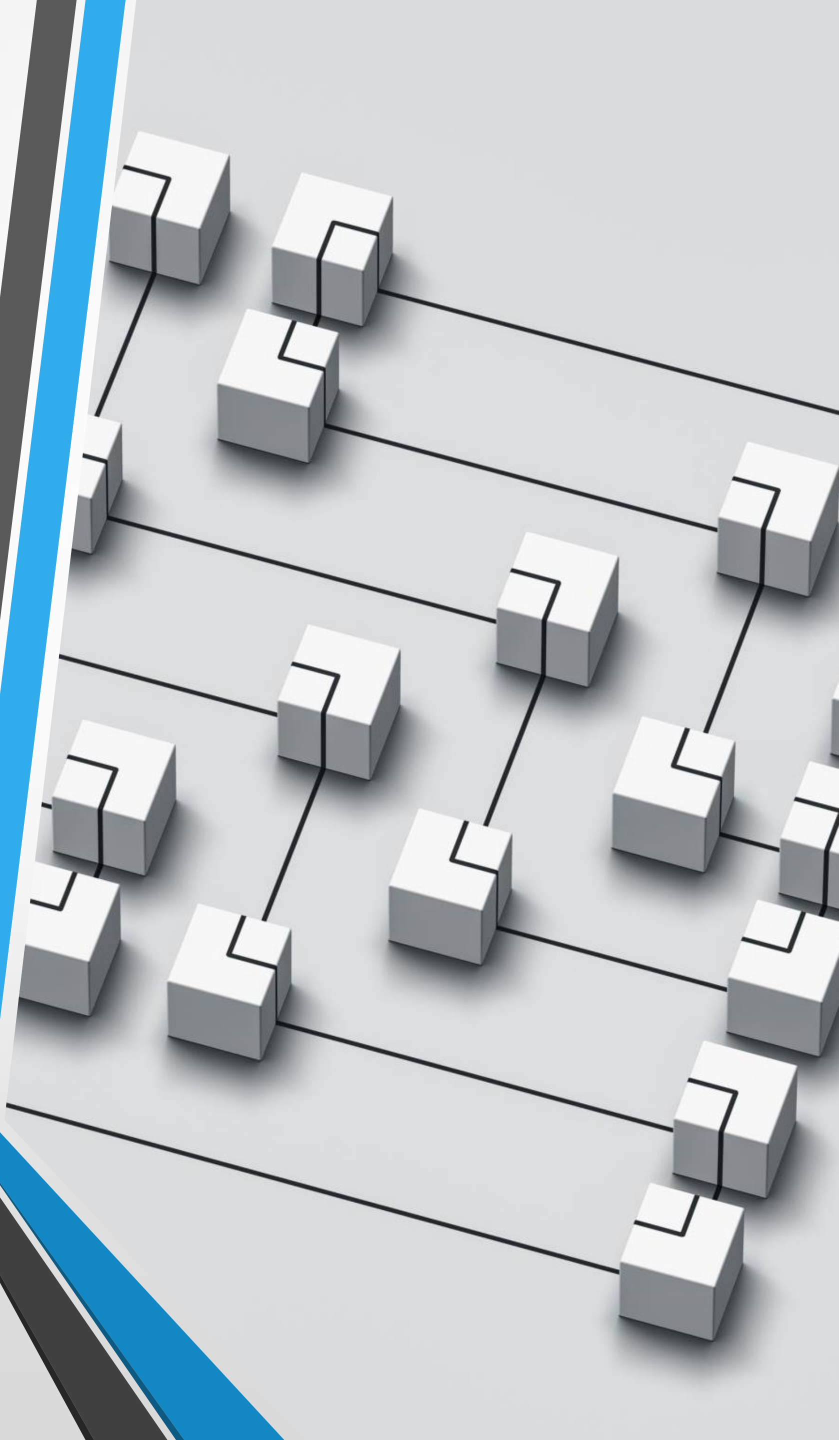
Test Loss: 1.716

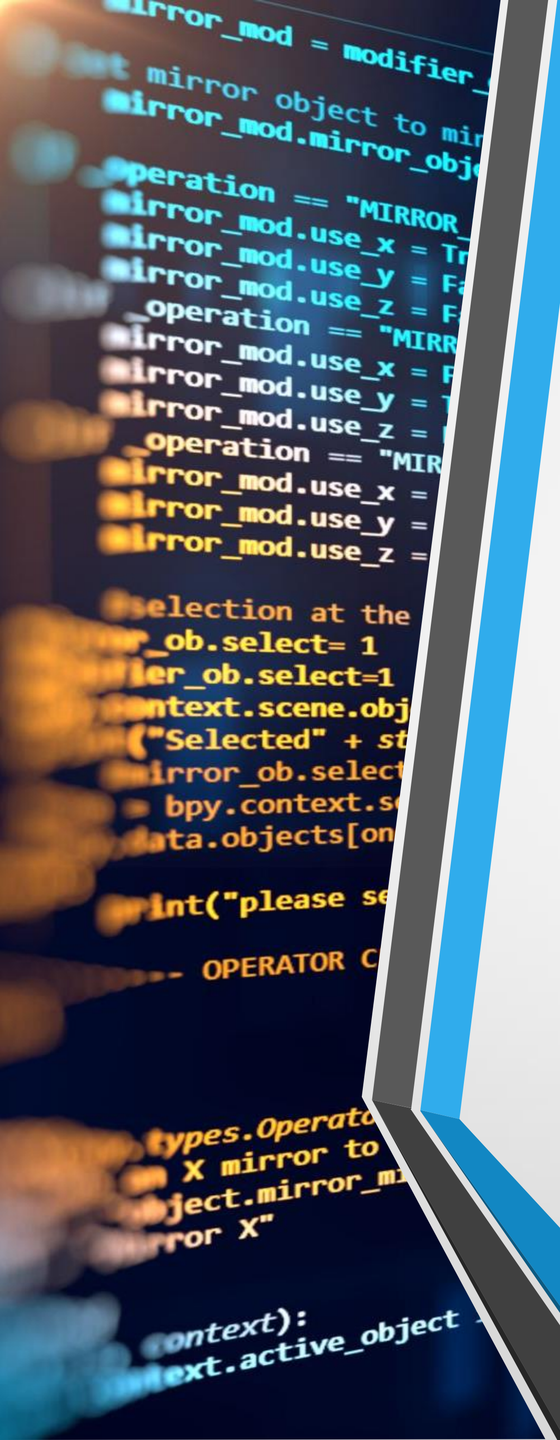
Observation:

Lower accuracy but demonstrated utility of pre-trained models.

# Model Saving and Deployment

- Saving:
  - Saved in TensorFlow format for compatibility.
- Deployment:
  - TensorFlow Serving in Docker.
  - Hosted on Google Cloud using Vertex AI.
- Integration:
  - Endpoint connected to a client application for predictions.

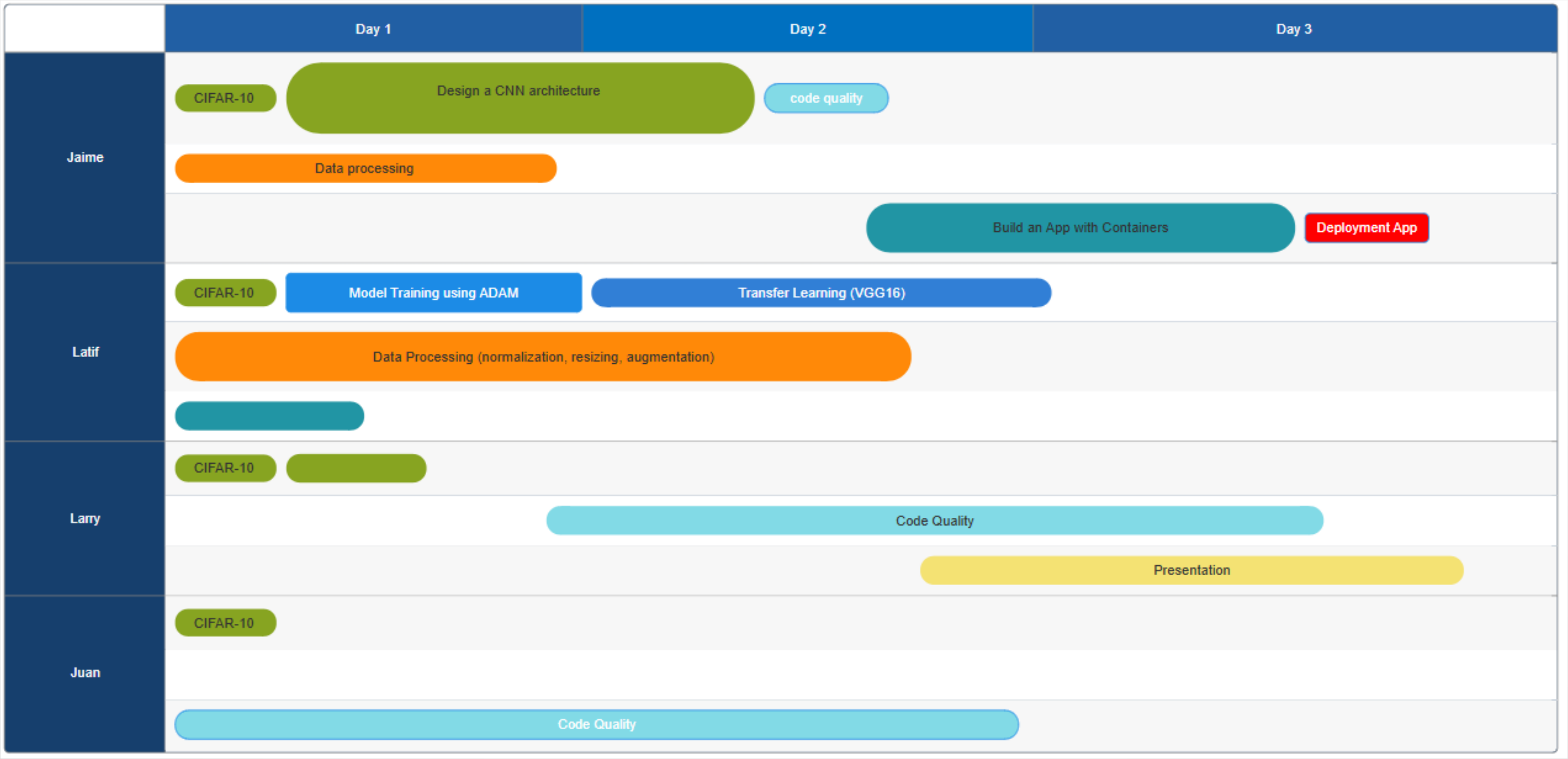




# Flask Web Application

- Features:
  - Image Upload and Prediction.
  - Displays original and preprocessed images.
- Technology Stack:
  - Frontend: HTML, HTMX, Tailwind CSS.
  - Backend: Flask, Model API integration.
- Workflow:
  - Upload, preprocess, predict, and display results.

# Project Management



Day 1
Day 2
Day 3

Jaime
CIFAR-10
Design a CNN architecture
code quality
Data processing
Build an App with Containers
Deployment App

Latif
CIFAR-10
Model Training using ADAM
Transfer Learning (VGG16)
Data Processing (normalization, resizing, augmentation)

Larry
CIFAR-10
Code Quality
Presentation

Juan
CIFAR-10
Code Quality

# Teamwork & Project Management

- Workflow:
  - Followed an iterative refinement process.
- What Worked Well:
  - Clear role distribution.
- Improvements:
  - Enhanced documentation.
- Risk Management:
  - Addressed model overfitting early.

# Major Obstacle

Biggest Obstacle:

Overfitting with initial model design.

Learnings:

Importance of regularization techniques.

Effective use of validation data.

Hindsight:

Earlier implementation of dropout layers.

# Conclusion and Insights

## Hypothesis Outcome:

Supported: Achieved high accuracy with CNN.

## Key Learnings:

Iterative model improvement leads to significant accuracy gains.

Importance of balancing model complexity and computational efficiency.

Deployment Model Processes

# Thank You!