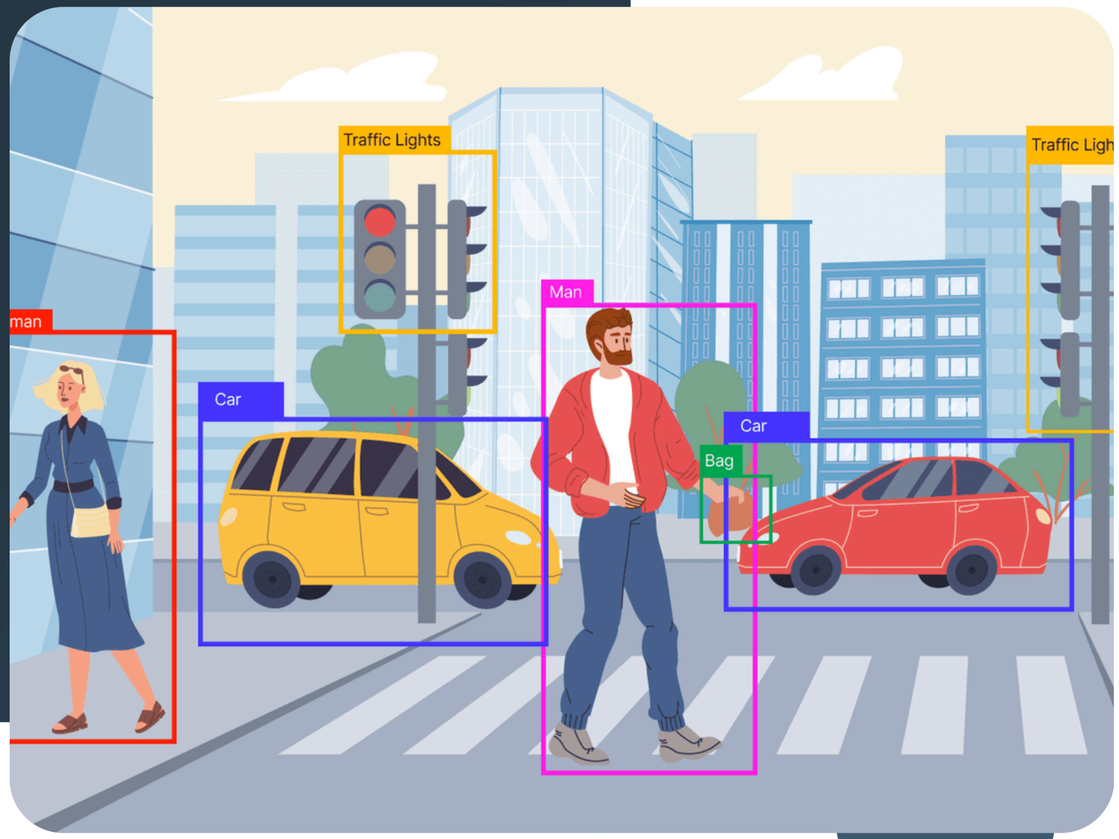
NEURAL NETWORK MODELS FOR OBJECT RECOGNITION

CIFAR-10

KANAN AKPAROV



WHY OBJECT RECOGNITION MATTERS

A core part of modern AI, powering:

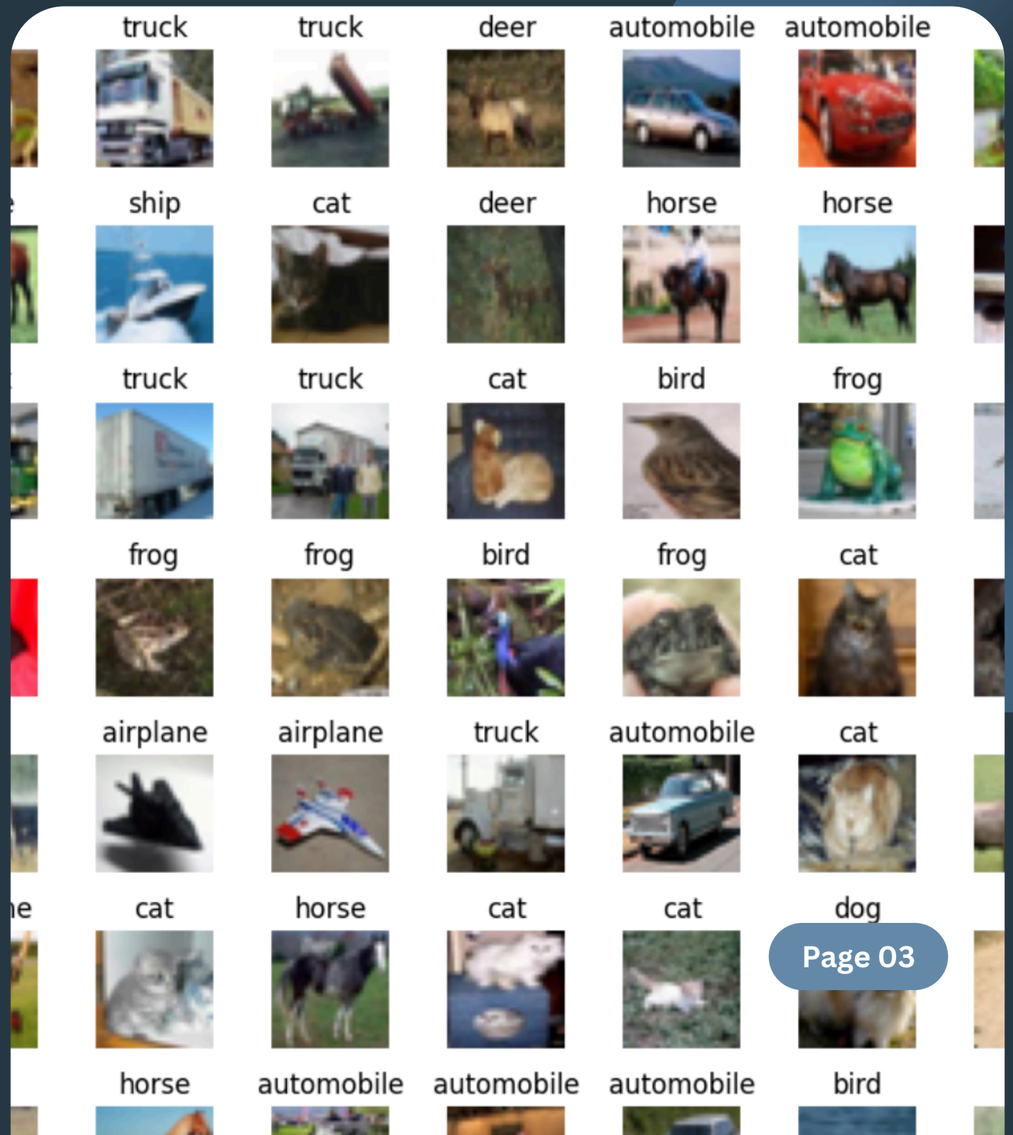
➔ Smart vehicles
➔ Facial recognition
➔ Quality control
➔ Healthcare imaging
➔ Retail automation

DATASET OVERVIEW

The dataset contains 60,000 color images with a resolution of 32×32 pixels, split into:

01.

➔ 50,000 for training
➔ 10,000 for testing
➔ 5,000 reserved for validation



```python
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.utils import to_categorical

class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck']

(X_train, y_train), (X_test, y_test) = cifar10.load_data()
X_train = X_train.astype('float32') / 255.0
X_test = X_test.astype('float32') / 255.0

# One-hot encode labels
y_train_cat = to_categorical(y_train, 10)
y_test_cat = to_categorical(y_test, 10)


print("Training data shape:", X_train.shape, y_train.shape)
print("Test data shape:", X_test.shape, y_test.shape)
print("Number of classes:", len(class_names))

# Display
def plot_sample_images(X, y, class_names):
    plt.figure(figsize=(8, 8))
    for i in range(49):
        plt.subplot(7, 7, i+1)
        plt.imshow(X[i])
        plt.title(class_names[int(y[i])])
        plt.axis('off')
    plt.tight_layout()
    plt.show()
plot_sample_images(X_train, y_train, class_names)
```

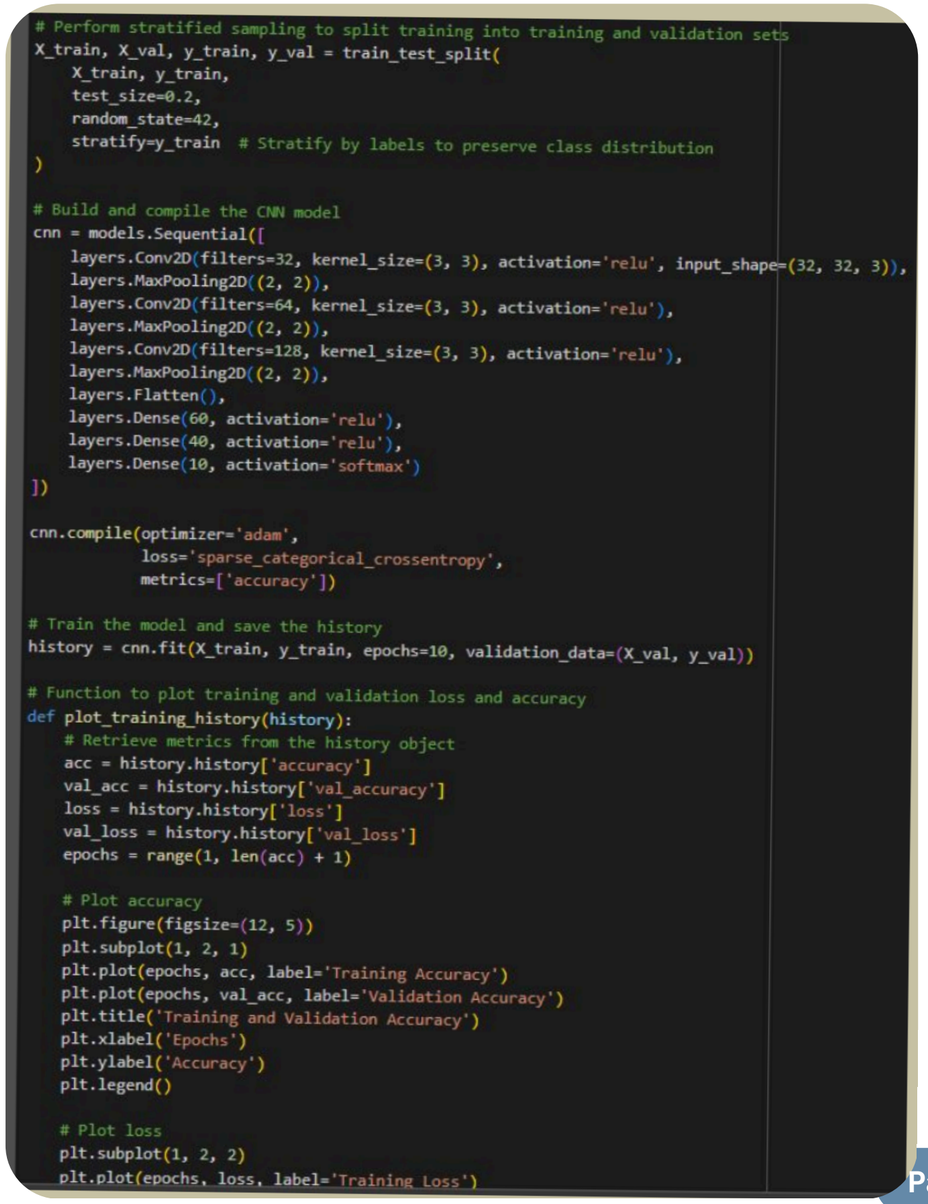
DATASET METADATA & SPLITTING

The original 50,000 images are split to evaluate model generalization on unseen data.

Each image has 3,072 features, providing comprehensive input for learning.

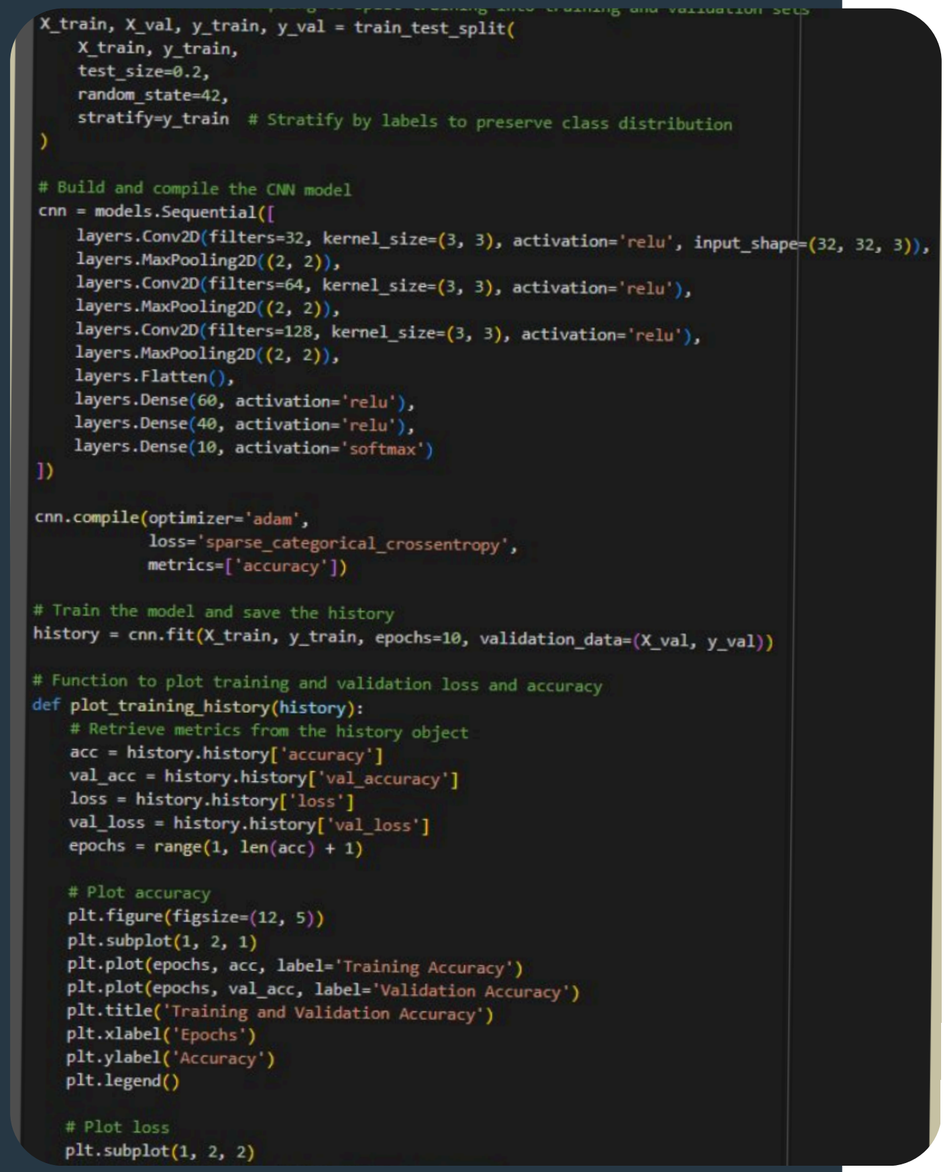
IMPORTANCE OF VALIDATION

➔ Offers feedback during model development
➔ Helps tune parameters like learning rate and dropout rate
➔ Detects overfitting
➔ Provides an unbiased assessment of model performance



```
# Perform stratified sampling to split training into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(
    X_train, y_train,
    test_size=0.2,
    random_state=42,
    stratify=y_train  # Stratify by labels to preserve class distribution
)

# Build and compile the CNN model
cnn = models.Sequential([
    layers.Conv2D(filters=32, kernel_size=(3, 3), activation='relu', input_shape=(32, 32, 3)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(filters=128, kernel_size=(3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(60, activation='relu'),
    layers.Dense(40, activation='relu'),
    layers.Dense(10, activation='softmax')
])

cnn.compile(optimizer='adam',
            loss='sparse_categorical_crossentropy',
            metrics=['accuracy'])

# Train the model and save the history
history = cnn.fit(X_train, y_train, epochs=10, validation_data=(X_val, y_val))

# Function to plot training and validation loss and accuracy
def plot_training_history(history):
    # Retrieve metrics from the history object
    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']
    loss = history.history['loss']
    val_loss = history.history['val_loss']
    epochs = range(1, len(acc) + 1)

    # Plot accuracy
    plt.figure(figsize=(12, 5))
    plt.subplot(1, 2, 1)
    plt.plot(epochs, acc, label='Training Accuracy')
    plt.plot(epochs, val_acc, label='Validation Accuracy')
    plt.title('Training and Validation Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()

    # Plot loss
    plt.subplot(1, 2, 2)
    plt.plot(epochs, loss, label='Training Loss')
```

CNN MODEL ARCHITECTURE OVERVIEW & KEY MODEL COMPONENTS



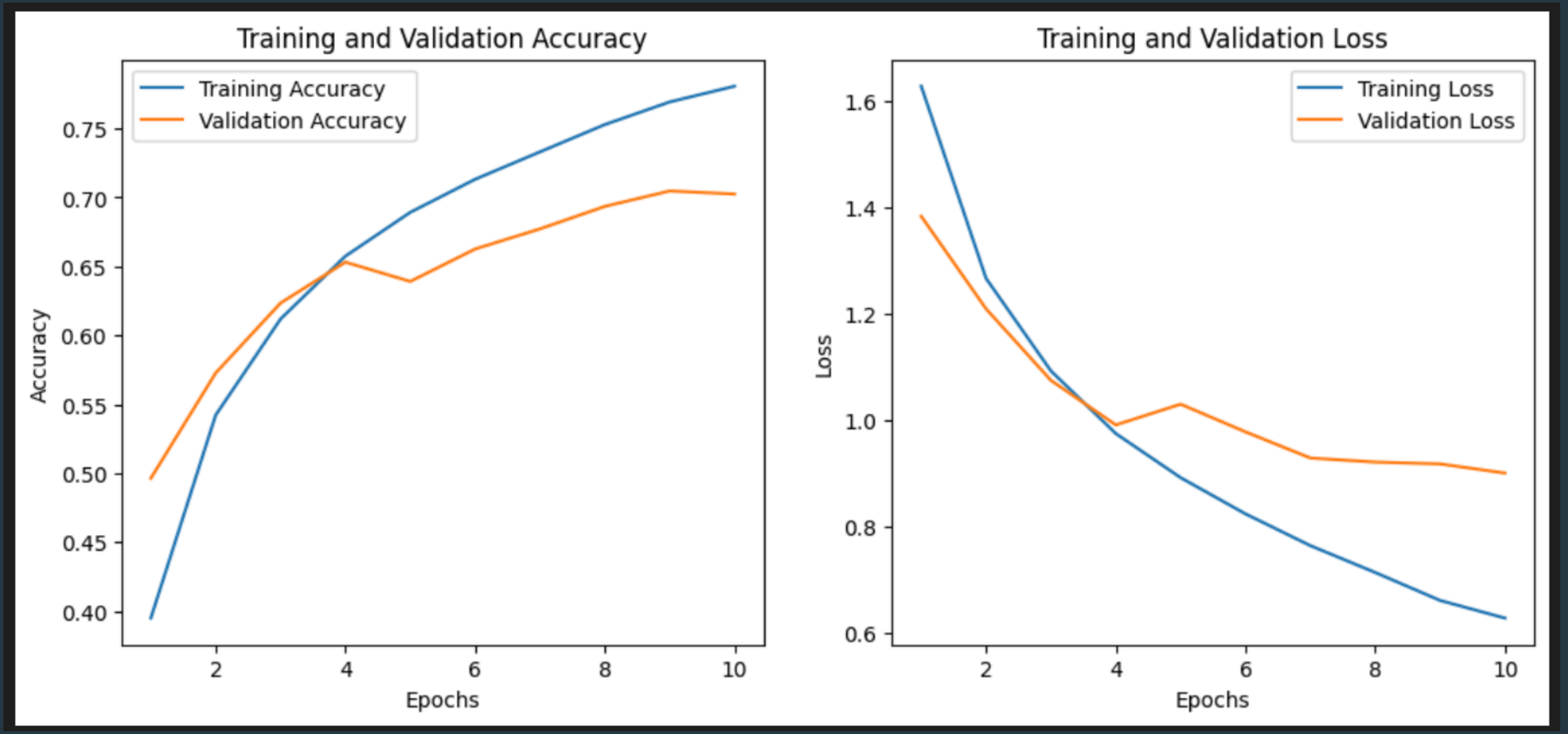
```
X_train, X_val, y_train, y_val = train_test_split(
    X_train, y_train,
    test_size=0.2,
    random_state=42,
    stratify=y_train  # Stratify by labels to preserve class distribution
)

# Build and compile the CNN model
cnn = models.Sequential([
    layers.Conv2D(filters=32, kernel_size=(3, 3), activation='relu', input_shape=(32, 32, 3)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(filters=128, kernel_size=(3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(60, activation='relu'),
    layers.Dense(40, activation='relu'),
    layers.Dense(10, activation='softmax')
])

cnn.compile(optimizer='adam',
            loss='sparse_categorical_crossentropy',
            metrics=['accuracy'])

# Train the model and save the history
history = cnn.fit(X_train, y_train, epochs=10, validation_data=(X_val, y_val))

# Function to plot training and validation loss and accuracy
def plot_training_history(history):
    # Retrieve metrics from the history object
    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']
    loss = history.history['loss']
    val_loss = history.history['val_loss']
    epochs = range(1, len(acc) + 1)

    # Plot accuracy
    plt.figure(figsize=(12, 5))
    plt.subplot(1, 2, 1)
    plt.plot(epochs, acc, label='Training Accuracy')
    plt.plot(epochs, val_acc, label='Validation Accuracy')
    plt.title('Training and Validation Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()

    # Plot loss
    plt.subplot(1, 2, 2)
```

- Input: 32×32×3 RGB images (CIFAR-10)
- 3 Convolutional layers (32, 64, 128 filters) — extract visual features
- MaxPooling after each Conv — reduces spatial size
- Flatten layer — converts feature maps to vector
- Dense layers (60, 40 neurons) — learn complex patterns
- Output layer (10 neurons, softmax) — class prediction
- Optimizer: Adam | Loss: Categorical Crossentropy | Metric: Accuracy
- Trained for 10 epochs with validation split (20%)
- Plotted accuracy & loss to monitor model performance

TRAINING PARAMETERS

- Optimizer: Adam
- Loss Function: Categorical Crossentropy
- Metric: Accuracy
- Epochs: 10
- Validation Split: 20% (using train_test_split)
- Batch Size: (default — likely 32)
- Input Shape: (32, 32, 3)
- Output Classes: 10

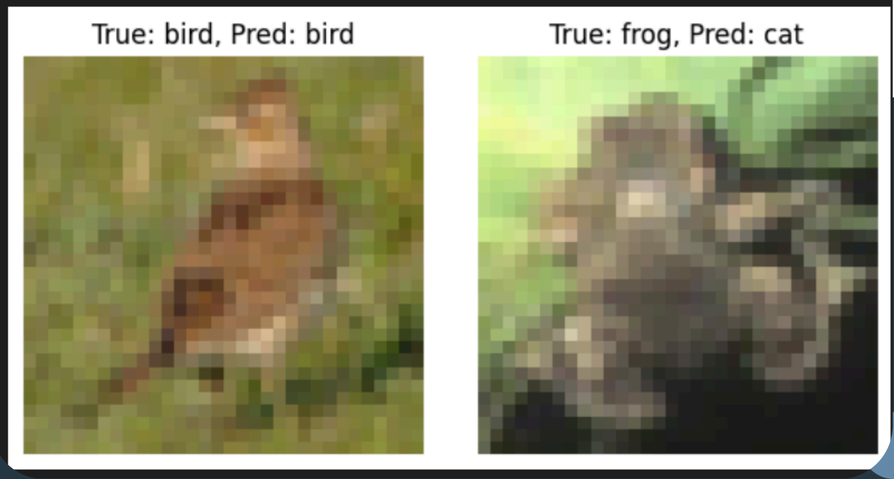
ACCURACY AND LOSS OVER EPOCHS



TEST SET EVALUATION

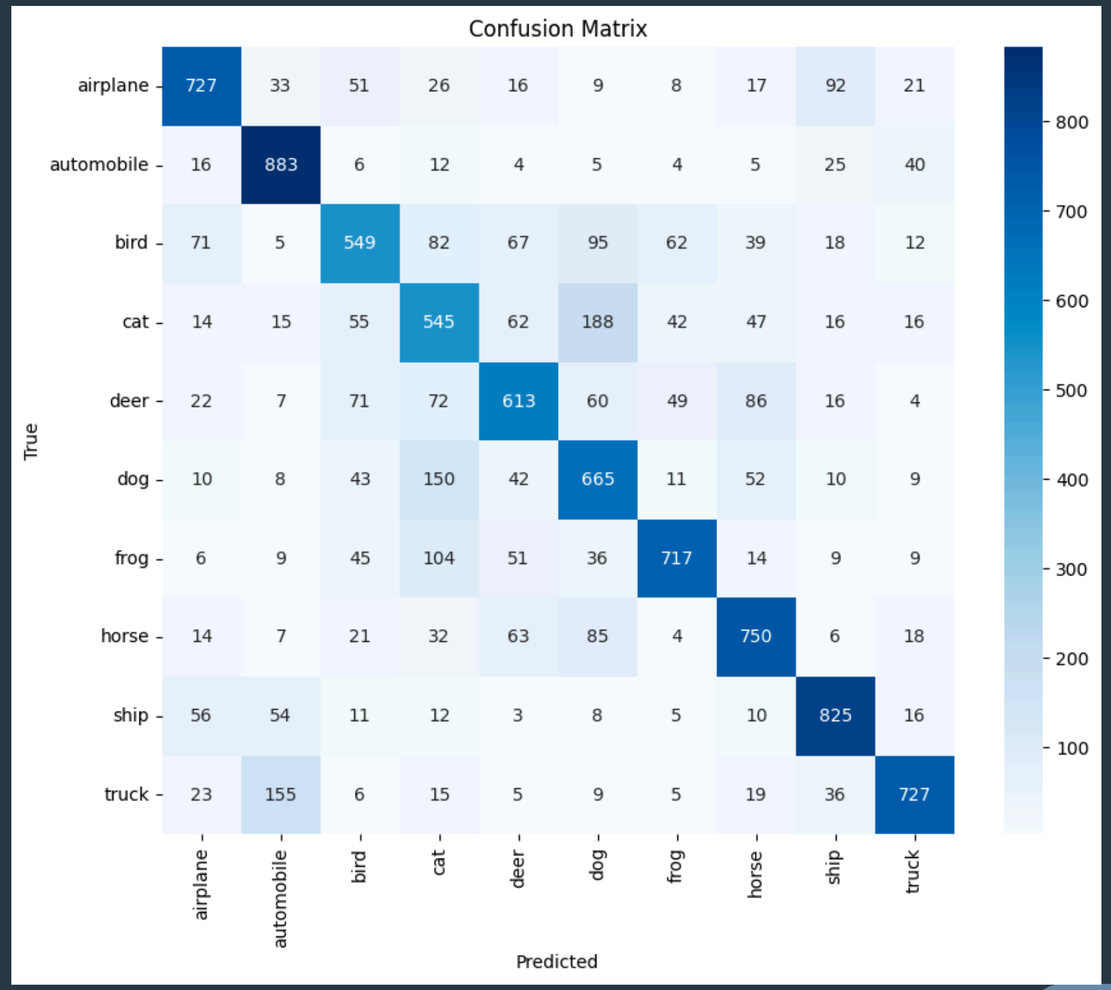
- Final Accuracy: 70%
- Model performed well but can be improved

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step
Test Loss: 0.9032906293869019
Test Accuracy: 0.6983000040054321
Test Precision: 0.706622169683003
Test Recall: 0.6982999999999999
Test F1-score: 0.6965683529846298
Classification Report:

              precision    recall  f1-score   support

    airplane       0.70      0.80      0.75      1000
  automobile       0.81      0.84      0.83      1000
        bird       0.71      0.49      0.58      1000
         cat       0.47      0.58      0.52      1000
        deer       0.71      0.55      0.62      1000
         dog       0.63      0.57      0.60      1000
        frog       0.81      0.73      0.77      1000
       horse       0.62      0.84      0.72      1000
        ship       0.79      0.82      0.80      1000
       truck       0.80      0.76      0.78      1000

    accuracy                           0.70     10000
   macro avg       0.71      0.70      0.70     10000
weighted avg       0.71      0.70      0.70     10000
```



CONFUSION MATRIX & CLASS ANALYSIS

- Highest accuracy achieved on trucks,automobile,frogs,airplane
- Lowest accuracy observed on cats and birds
- Potential improvements: address underperforming classes



POSSIBLE IMPROVEMENTS

- Add more convolutional layers
- Apply Batch Normalization for stability
- Use Transfer Learning with a pre-trained model (e.g., VGG16, ResNet)

```python
from tensorflow.keras import models, layers, applications
import matplotlib.pyplot as plt

# Pre-trained VGG16 as feature extractor
base_model = applications.VGG16(weights='imagenet', include_top=False, input_shape=(32,32,3))
base_model.trainable = False

# Improved CNN model
cnn_improved = models.Sequential([
    base_model,
    layers.GlobalAveragePooling2D(),
    layers.Dense(128, activation='relu'),
    layers.BatchNormalization(),
    layers.Dense(10, activation='softmax')
])

# Compile the model
cnn_improved.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

# Train the model
history_improved = cnn_improved.fit(
    X_train, y_train,
    epochs=10,
    validation_data=(X_val, y_val),
    batch_size=32
)
```

DESIGN DECISIONS & STRATEGY

- Built a simple yet effective CNN for CIFAR-10
- Prototype has enough depth to learn features while remaining interpretable
- Explored dropout, batch normalization, and increased filter counts to improve performance
- Consider future improvements: transfer learning, automated hyperparameter tuning, and advanced regularization techniques

CHALLENGES ENCOUNTERED

- Balanced training time versus model complexity
- Tuned batch size and learning rate for stability
- Observed occasional overfitting on some classes
- Some noisy or inconsistent predictions in underperforming categories
- Memory management

REAL WORLD APPLICATIONS

- Systems for recognizing and interpreting visual information
- Tools for diagnosing medical conditions
- Monitoring and managing agricultural environments
- Enhancing roadway safety
- Ensuring security at airports
- Starting with minor, focused tasks that scale into solving more complex, high-impact challenges

REFLECTIONS AND LEARNING OUTCOMES

- Connects conceptual understanding with hands-on application
- Highlights the critical role of testing, validation, and performance assessment
- Emphasizes that machine learning is an iterative process, not the pursuit of a single flawless model
-

REFERENCES

- Abadi, M. et al., 2016. TensorFlow: A system for large-scale machine learning. [online] Available at: https://www.tensorflow.org [Accessed 12 October 2025].
- Chollet, F. et al., 2015. Keras. [online] Available at: https://keras.io [Accessed 12 October 2025].
- Goodfellow, I., Bengio, Y. & Courville, A., 2016. Deep Learning. Cambridge, MA: MIT Press.
- Krizhevsky, A., Nair, V. & Hinton, G., 2014. The CIFAR-10 Dataset. [online] Available at: https://www.cs.toronto.edu/~kriz/cifar.html [Accessed 12 October 2025].
- LeCun, Y., Bottou, L., Bengio, Y. & Haffner, P., 1998. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), pp.2278–2324.

THANK YOU

FOR YOUR ATTENTION