# **Deep Learning: Image Classification with CNN**

**Authors:** Jaime Velez, Larry Davila, Latif,   
**Date:** December 2024

## **1. Loading the Dataset: CIFAR-10**

We explored two approaches to load the CIFAR-10 dataset:

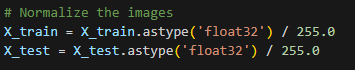
* **Direct Import:** We utilized tensorflow.keras.datasets to import the dataset directly into the Jupyter notebook. This allowed us to split the training and testing sets in a single line of code.



## **2. Preprocessing Steps**

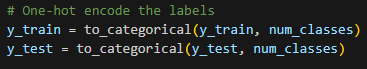
### **a. Normalization**

* Images were converted to float values and scaled to have pixel values between 0 and 1. This normalization helps the model converge faster during training.



### **b. One-Hot Encoding**

* Labels were transformed into categorical format to match the requirements of the categorical cross-entropy loss function.



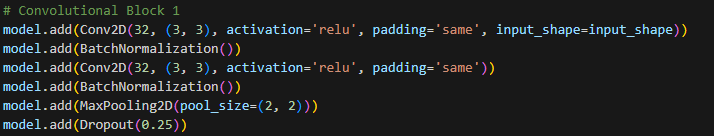
## **3. Model Building and Hyperparameters**

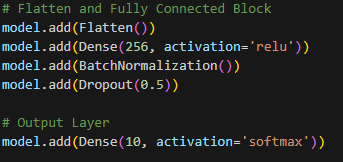
### **Architecture**

* The model consists of 4 convolutional blocks, each with:
  + **Conv2D Layers:** Using ReLU activation and "same" padding to retain input dimensions.
  + **Batch Normalization:** To stabilize and accelerate training.
  + **Max Pooling:** To reduce spatial dimensions.
  + **Dropout Layers:** With an increasing rate across blocks to mitigate overfitting.
* The model concludes with a Flatten layer, a Dense layer for feature combination, and a Softmax layer for multi-class classification.

### **Hyperparameters**

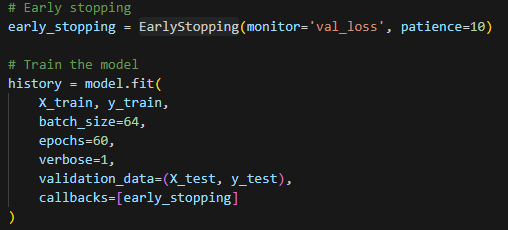
* **Optimizer:** Adam optimizer for adaptive learning.
* **Loss Function:** Categorical cross-entropy.
* **Batch Size:** 64.
* **Epochs:** Set to 60, with early stopping to terminate training when performance stops improving.
* **Sample Size:** 50,000 images for training and 10,000 for testing.





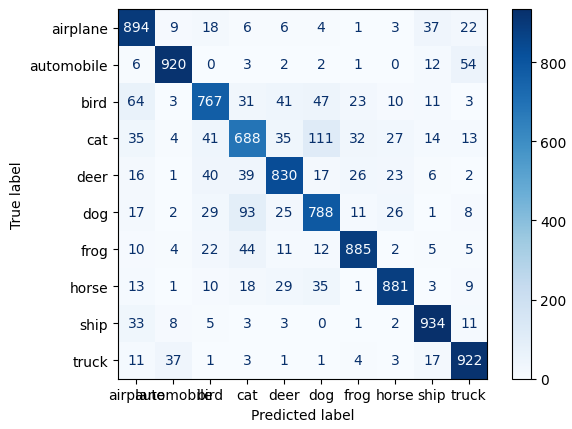
## **4. Training Process**

* The model achieved a training accuracy of **96.82%** in just 11 epochs, thanks to early stopping.
* **Evaluation Metrics:**
  + **Test Accuracy:** 85%
  + **Test Loss:** 0.64



### **Insights from the Confusion Matrix**

* **Strengths:** High accuracy in categories such as airplanes, ships, and trucks.
* **Weaknesses:** Confusion between visually similar classes like cats vs. dogs and trucks vs. automobiles.



## **5. Transfer Learning**

We applied transfer learning using the VGG16 architecture, known for its effectiveness in image classification tasks.

### **Results with VGG16**

* **Test Loss:** 1.716
* **Test Accuracy:** 65.2%
* While accuracy was lower than our custom CNN, the process demonstrated the utility of pre-trained models for smaller datasets like CIFAR-10.

## **6. Model Saving and Deployment**

### **Model Saving**

* The model was saved using TensorFlow's recommended format: model.save('saved\_model/my\_model/1/'), making it compatible with TensorFlow Serving.

### **TensorFlow Serving**

1. Created a Docker container with TensorFlow Serving.
2. Tested the API with a Python client to ensure proper functionality.

### **Hosting on Google Cloud**

1. Uploaded the model to Google Cloud Storage (GCS).
2. Deployed the model using Vertex AI to create an endpoint for predictions.
3. Integrated the endpoint with a client application to send prediction requests.

## **7. Flask Web Application**

### **Features**

1. **Image Upload:** Users can upload one or multiple images through the web interface.
2. **Predictions:** The application processes the images using the pre-trained deep learning model hosted on a remote API, returning class labels and probabilities.
3. **Preprocessed Image Display:** Displays the processed version of the uploaded image to help users understand the model’s input.

### **Technology Stack**

* **Frontend:**
  + **HTML:** Provides the basic structure for web pages.
  + **HTMX:** Enables dynamic updates without full-page reloads.
  + **Tailwind CSS:** Ensures a clean and professional design.
* **Backend:**
  + **Flask (Python):** Manages image uploads, request processing, and result rendering.
* **Model API:** Communicates with the Google AI Platform endpoint for predictions.

### **Workflow**

1. **User Interaction:** Users upload images through the web form.
2. **Image Preprocessing:** The backend preprocesses images to match the model’s input dimensions (e.g., resizing to 32x32 pixels and normalization).
3. **Model Prediction:** Preprocessed images are sent to the remote API, which returns prediction probabilities for each class.
4. **Result Display:** Users see the original, preprocessed images and the predicted labels with probabilities.