Image Classification Using Convolutional Neural Networks on CIFAR-10 Dataset

Submitted by: A.PRIYADHARSHINI

Position: AI & ML Engineer

Company: Fastack.ai

---

1. Introduction

Image classification is one of the most essential applications of computer vision, and Convolutional Neural Networks (CNNs) are widely used in solving such tasks due to their ability to capture spatial hierarchies in image data. This report details the process of building a CNN model using TensorFlow/Keras to classify images from the CIFAR-10 dataset into 10 distinct categories. The implementation aims to achieve a balance between simplicity and effectiveness while demonstrating the core principles of deep learning for image classification.

2. CIFAR-10 Dataset Overview

1. **Origin and Purpose**:
   The CIFAR-10 dataset was created by the Canadian Institute for Advanced Research. It is primarily used as a benchmark for evaluating machine learning algorithms, especially in the fields of image classification and computer vision.
2. **Image Format**:
   Each image in the dataset is a 32x32 pixel RGB image, meaning each image has 3 color channels (Red, Green, Blue), and every channel has 1024 pixel values (32×32). So, each image is represented by 3072 values ($32 \times 32 \times 3$).
3. **Dataset Split**:
   - **Training set**: 50,000 images
   - **Test set**: 10,000 images
     There is no official validation set, but users typically split the training set (e.g., 80% train, 20% validation) during experimentation.
4. **Balanced Classes**:
   The dataset is completely balanced, with exactly 6,000 images per class. This makes it ideal for evaluating classification models, as there's no class imbalance to skew the performance.
5. **Variability and Difficulty**:
   Despite the small size of the images, CIFAR-10 poses a non-trivial classification problem due to:

- Background clutter and varying contexts
- Intra-class variation (e.g., different types of trucks or birds)
- Low resolution (32x32 makes feature extraction harder)
- Presence of occlusion, lighting variation, and scale changes

6. **Use in Research**:
   CIFAR-10 is widely used in academic papers and deep learning research to test novel architectures such as:
   - Convolutional Neural Networks (CNNs)
   - Residual Networks (ResNets)
   - Capsule Networks
   - Data augmentation strategies and optimization techniques

7. Availability:

The dataset is openly available through tensorflow.keras.datasets, PyTorch torchvision.datasets, and as a downloadable set in NumPy format from the official website.

8. **Comparison with CIFAR-100**:

CIFAR-10 is a simplified version of the CIFAR-100 dataset, which contains 100 classes with 600 images each. CIFAR-10 is preferred for quicker training and evaluation in prototyping stages.

3. Data Preprocessing

3.1 Loading the Dataset

The dataset was loaded using tensorflow.keras.datasets.cifar10, which provides an easy interface to access and split the dataset into training and test sets.

```
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
```

3.2 Data Exploration

- The training set contains 50,000 samples and the test set contains 10,000 samples.
- Each image has the shape (32, 32, 3), representing RGB color images.
- Sample images were visualized using Matplotlib to ensure correct loading and labeling.

3.3 Normalization

Pixel values were normalized by dividing by 255.0 to scale them between 0 and 1. This improves the stability and convergence rate during training.

```
x_train = x_train / 255.0
x_test = x_test / 255.0
```

3.4 One-Hot Encoding

Although the sparse_categorical_crossentropy loss function can handle integer labels directly, labels were one-hot encoded using to_categorical for generalization and compatibility with future multi-output tasks.

```
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)
```

3.5 Data Augmentation

To improve the generalization of the model and reduce overfitting, real-time data augmentation was applied using ImageDataGenerator. Techniques used include random rotations, horizontal flips, and shifts.

4. CNN Architecture

The CNN model was built using Keras' Sequential API. The architecture was chosen to balance performance and computational efficiency. Here's a breakdown of the layers:

Model Layers:

1. **Conv2D (32 filters, 3x3 kernel)** – with ReLU activation
2. **MaxPooling2D (2x2 pool size)**
3. **Conv2D (64 filters, 3x3 kernel)** – with ReLU activation
4. **MaxPooling2D (2x2 pool size)**

5. **Conv2D (64 filters, 3x3 kernel)** – with ReLU activation
6. **Flatten Layer** – to convert 2D feature maps into 1D feature vector
7. **Dense (64 units)** – with ReLU activation
8. **Dropout (rate = 0.5)** – for regularization
9. **Dense (10 units)** – with Softmax activation for multi-class classification

Model Summary:

- Total Parameters: ~1.2 million
- Activation functions: ReLU, Softmax
- Regularization: Dropout

5. Model Compilation and Training

5.1 Compilation

The model was compiled using the **Adam** optimizer, which adapts learning rates during training. The loss function used was categorical_crossentropy due to one-hot encoded labels.

```
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

5.2 Training

- **Epochs:** 30
- **Batch Size:** 64
- **Validation Split:** 20% of the training set
- **Callbacks Used:** EarlyStopping (patience=3) to prevent overfitting.

The model was trained using the .fit() method with data augmentation:

```
history = model.fit(datagen.flow(x_train, y_train, batch_size=64),
                    epochs=30,
                    validation_data=(x_test, y_test),
                    callbacks=[EarlyStopping(patience=3, restore_best_weights=True)])
```

6. Evaluation and Results

The trained model was evaluated on the test dataset:

```
loss, accuracy = model.evaluate(x_test, y_test)
```

Results:

- **Test Loss:** 0.5896
- **Test Accuracy:** 80.34%

Prediction Samples:

Visual comparisons of predicted vs. actual labels were made using matplotlib.pyplot. Most of the misclassifications occurred between visually similar categories (e.g., cats vs. dogs, trucks vs. automobiles).

Confusion Matrix:

A confusion matrix was plotted using sklearn.metrics.confusion_matrix and seaborn.heatmap, which showed high accuracy on airplane, ship, and automobile classes, and some confusion on animal classes.

7. Observations and Experiments

Experiments:

- Adding more convolutional layers increased accuracy by ~5%.

- Applying dropout helped prevent overfitting.
- Data augmentation improved model generalization significantly.
- Switching from SGD to Adam optimizer resulted in faster convergence and higher final accuracy.

Observations:

- CIFAR-10 is complex due to small image size and high intra-class variability.
- Even a relatively shallow CNN model can achieve over 80% accuracy.
- Additional tuning (e.g., learning rate scheduling or deeper models) can push accuracy beyond 85–90%.

8. Challenges and Resolutions

Challenge 1:

Slow training on CPU.
Solution:
Used GPU via Google Colab for faster training cycles.

Challenge 2:

Model overfitting after 15+ epochs.
Solution:
Added dropout layers and implemented early stopping.

9. Conclusion

This project demonstrated the development of a CNN-based image classifier using TensorFlow/Keras on the CIFAR-10 dataset. The model achieved a test accuracy of over 80%, showcasing the effectiveness of CNNs for image classification tasks. With further experimentation and tuning, such as deeper architectures or transfer learning, even higher accuracy can be achieved. This assignment strengthened practical understanding of image classification pipelines and convolutional network design.