STUDENT NAME: Kotadiya Vinay

STUDENT ID: 23101562

GITHUB LINK: *Click Me!*

Convolutional Neural Networks (CNN) for Image Classification

Introduction

Convolutional Neural Networks (CNNs) are a powerful deep learning architecture used primarily for image classification, object detection, and other computer vision tasks. CNNs excel in recognizing spatial patterns and extracting features from images, making them ideal for real-world applications like medical imaging, facial recognition, and autonomous vehicles.

In this project, we use a CNN to classify images from the **CIFAR-10 dataset**, which contains 60,000 color images across **10 categories** (e.g., airplanes, automobiles, birds, cats, etc.). We will explore the **architecture, training process, evaluation, and visualization of results**.

Understanding CNNs

1. Convolutional Layer (Conv2D)

This layer applies multiple filters to the input image to detect edges, colors, and textures. Each filter moves across the image, creating feature maps that capture important details. The deeper the network, the more complex patterns it learns.

$$(\boldsymbol{I} * \boldsymbol{K})(\boldsymbol{x}, \boldsymbol{y}) = \sum_{\boldsymbol{i}=-\boldsymbol{m}}^{\boldsymbol{m}} \sum_{\boldsymbol{j}=-\boldsymbol{n}}^{\boldsymbol{n}} \boldsymbol{I}(\boldsymbol{x}+\boldsymbol{i}, \boldsymbol{y}+\boldsymbol{j}) \boldsymbol{K}(\boldsymbol{i}, \boldsymbol{j})$$

2. Batch Normalization

Batch normalization helps stabilize the learning process by normalizing activations at each layer. This prevents internal covariate shifts and speeds up training by reducing the dependency on careful initialization.

$$\widehat{\boldsymbol{x}} = \frac{\boldsymbol{x} - \boldsymbol{\mu B}}{\sqrt{\boldsymbol{\sigma^2 B} + \boldsymbol{\epsilon}}}$$

3. Activation Function (ReLU)

The Rectified Linear Unit (ReLU) introduces non-linearity into the model, allowing it to learn complex relationships in the data. It replaces all negative values with zero, making the model more efficient.

$$\boldsymbol{f(x)} = \mathbf{max}(\mathbf{0}, \boldsymbol{x})$$

4. Pooling Layer (MaxPooling2D)

Pooling reduces the spatial dimensions of feature maps while retaining important information. Max pooling selects the highest value in a region, helping the model become invariant to small translations in the image.

$$\boldsymbol{P(x,y)} = \boldsymbol{max_{i=0}^{n-1}\ max_{j=0}^{n-1}\ I(x+i, y+j)}$$

5. Dropout

Dropout is a regularization technique that randomly disables neurons during training to prevent overfitting. This forces the model to learn more robust patterns instead of memorizing specific details.

$$\boldsymbol{y' = r.y,} \qquad \boldsymbol{r \sim Bernoulli(p)}$$

6. Fully Connected (Dense) Layer

After extracting features, fully connected layers process them to make final predictions. These layers help the network combine lower-level patterns into higher-level concepts.

$$z = Wx + b$$

7. Softmax Activation

The softmax function converts the model's output into probability scores, ensuring that the sum of all class probabilities is equal to 1. The class with the highest probability is chosen as the final prediction.

$$P(y_i) = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}$$

Step-by-Step Implementation

1. Importing Libraries

We import TensorFlow for deep learning, Keras for model building, Matplotlib for visualization, and NumPy for handling numerical computations. These libraries are essential for data processing and training the CNN model.

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import matplotlib.pyplot as plt
import numpy as np
```

2. Load and Preprocess the Dataset

We load the CIFAR-10 dataset, which consists of 60,000 images divided into 10 categories. The dataset is normalized by dividing pixel values by 255.0, converting them from 0-255 to a 0-1 range, making training more efficient.

```
(x_train, y_train), (x_test, y_test) = keras.datasets.cifar10.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
```

3. Build the CNN Model

This CNN model consists of multiple convolutional layers followed by pooling layers to extract image features. Dropout layers are added to prevent overfitting, and the final fully connected layers classify images into 10 categories.

```
def create_cnn_model():
    model = keras.Sequential([
        layers.Conv2D(32, (3,3), activation='relu', padding='same', input_shape=(32, 32, 3)),
        layers.BatchNormalization(),
        layers.MaxPooling2D((2,2)),
        layers.Dropout(0.2),

        layers.Conv2D(64, (3,3), activation='relu', padding='same'),
        layers.BatchNormalization(),
```

```
    layers.MaxPooling2D((2,2)),
    layers.Dropout(0.3),


    layers.Flatten(),
    layers.Dense(256, activation='relu'),
    layers.BatchNormalization(),
    layers.Dropout(0.5),
    layers.Dense(10, activation='softmax')
  ])
  return model
model = create_cnn_model()
```

4. Compile and Train the Model

The model is compiled using the Adam optimizer, which adapts the learning rate dynamically for better performance. The sparse categorical cross-entropy loss function is used because our target labels are integers, and accuracy is used as the evaluation metric.

```
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
history = model.fit(x_train, y_train, epochs=20, batch_size=64, validation_data=(x_test, y_test))
```

5. Evaluate the Model

The model is evaluated on the test set to measure how well it generalizes to unseen images. The accuracy score provides an estimate of classification performance.

```
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)
print(f"Test Accuracy: {test_acc:.4f}")
```

6. Visualizing Training Performance

Plotting accuracy and loss over epochs helps us understand the training process. We check if the model is overfitting or underfitting by comparing training and validation accuracy.

```
def plot_training_history(history):
    plt.figure(figsize=(12, 4))
    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'], label='Train Accuracy')
    plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.title('CNN Training Performance')

    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'], label='Train Loss')
    plt.plot(history.history['val_loss'], label='Validation Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
```

```
    plt.legend()
    plt.title('CNN Loss Curve')


    plt.show()
plot_training_history(history)
```

Conclusion

In this project, we built a convolutional neural network to classify images from the CIFAR-10 dataset. The CNN used multiple convolutional and pooling layers to extract important features and reduce dimensionality while fully connected layers helped in making accurate predictions. Batch normalization and dropout techniques were used to stabilize training and prevent overfitting. The model was trained using the Adam optimizer and sparse categorical cross-entropy loss function, achieving a high test accuracy. After training, we visualized the model's performance and predictions to assess how well it generalized to unseen data. With further improvements such as data augmentation or fine-tuning hyperparameters, CNN models can be optimized for even better accuracy in real-world applications.

References

- TensorFlow Documentation: https://www.tensorflow.org
- Deep Learning with Python by François Chollet
- CIFAR-10 Dataset: https://www.cs.toronto.edu/~kriz/cifar.html