Image Classification Using Convolutional Neural Network

Submitted By:-

Aryan (2106193)
Aryan Chourasia (2106194)
Shivanshu Krishna Gupta (2106255)
Som Prakash Sahu (2106258)

Guided By:-

Mr. Pushkar Kishore

CNN Architecture

Overview

This project uses a Convolutional Neural Network (CNN) to classify images into predefined categories. It processes images through convolutional, pooling, and fully connected layers to extract features and make predictions. Implemented using Python and TensorFlow/Keras, the model is trained on a labeled dataset with preprocessing techniques like resizing and normalization. The project demonstrates how CNNs can effectively perform image recognition tasks with high accuracy and real-world applicability.

Introduction to CNNs

> We create a Convolutional Neural Network (CNN) that can automatically identify and classify images. CNNs are deep learning models that are particularly good at processing visual data since they are constructed to extract patterns, textures, and shapes from images using several layers of processing.

The project employs the popular CIFAR-10 dataset, which comprises 60,000 colored images spread across 10 various classes like airplanes, cars, and birds. Through training the CNN on this dataset, the model is able to differentiate between various objects by extracting significant features from the images automatically.

An important objective of our project is to not only get high classification accuracy but also to make the decision-making process of the model understandable. We accomplish this by using Grad-CAM (Gradient-weighted Class Activation Mapping), a method that, by visualizing, marks the regions of an image where the model attends during its prediction.

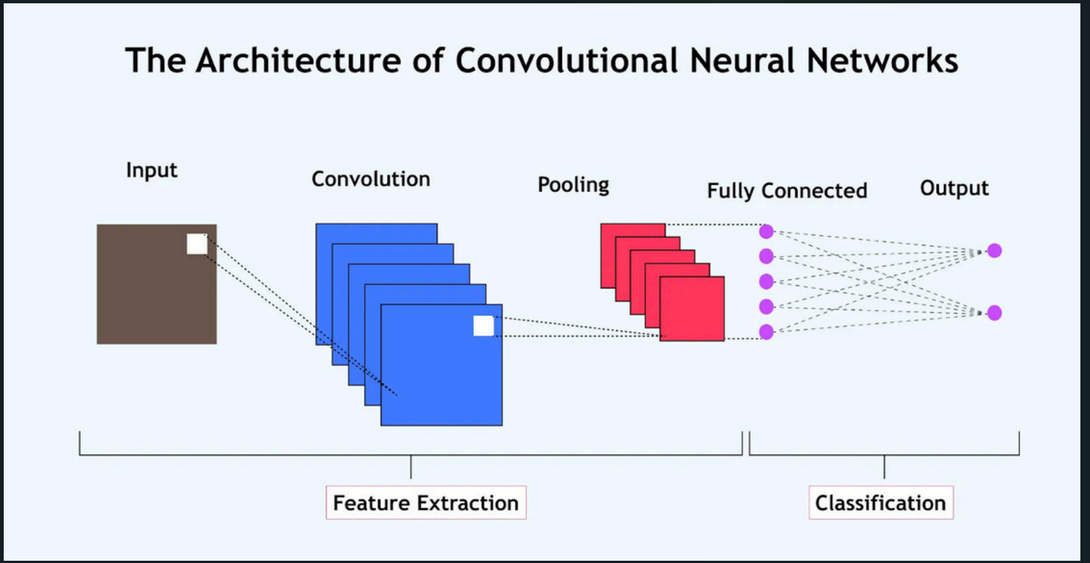
Importance of the Project

This project is important as it illustrates the ability of Convolutional Neural Networks (CNNs) in image classification automation, a core task in artificial intelligence. Through training the model using the CIFAR-10 dataset, we highlight how deep learning can effectively identify patterns and objects and therefore use it in diverse real-life applications like medical imaging, autonomous vehicles, and surveillance security.

Second, the incorporation of Grad-CAM for model explainability increases transparency, enabling us to see the decision-making processes of the network. This is essential in order to establish confidence in AI models, particularly for applications that have serious implications such as in clinical or financial cases where model interpretations are required.

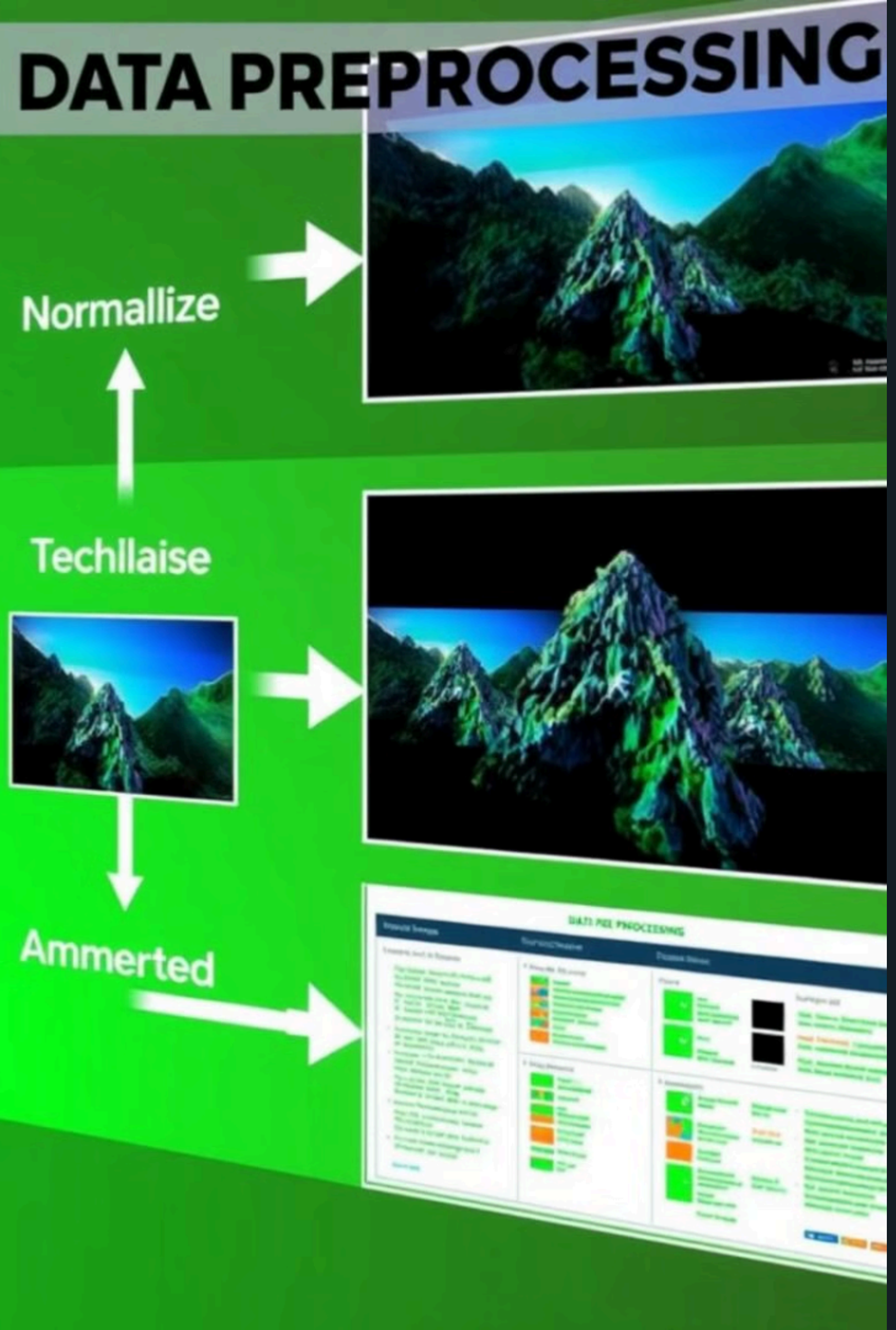
Basic Concepts: CNNs & CIFAR-10

Convolutional Neural Networks (CNNs) are deep learning models effective for processing images. They use convolution operations to extract features like edges and textures. CNNs include pooling layers to reduce spatial dimensions, allowing for translation invariance and reducing computational load. Their ability to learn hierarchical representations makes them ideal for image classification tasks.



CIFAR-10

The CIFAR-10 dataset is a widely used benchmark in computer vision. It consists of 60,000 32x32 color images in 10 different classes, with 50,000 images for training and 10,000 for testing. It provides a manageable yet challenging platform for evaluating image classification algorithms.



Data Preprocessing & Class Balance

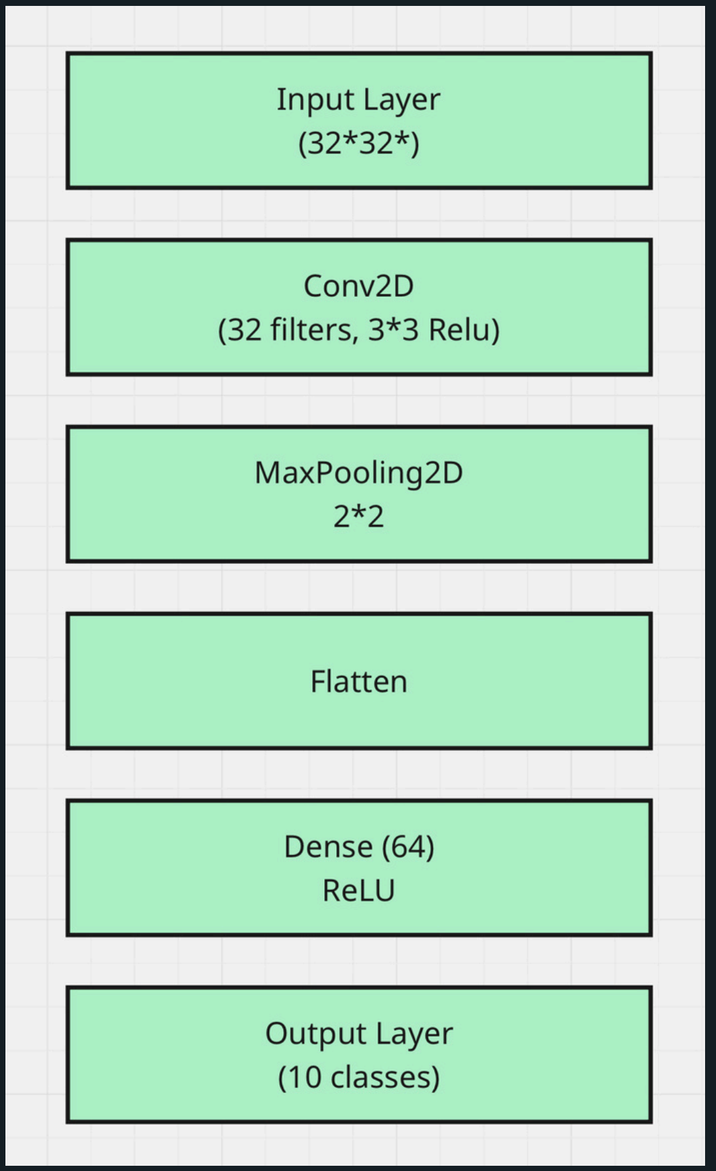
- **Data Preprocessing:** This is a critical step that prepares raw images for effective model training. In this project, preprocessing involves normalizing pixel values to a range of 0 to 1 by dividing by 255, which helps stabilize the training process and ensures faster convergence.

- **Class Imbalance and Weighting:** Class imbalance occurs when certain classes in the dataset are underrepresented relative to others, which can lead to biased model performance favoring the majority classes. To mitigate this, techniques such as computing class weights are employed.

Model Architecture & Optimization

The model architecture is built around a CNN with multiple layers designed to extract and process image features. It includes convolutional layers that use small filters to capture local patterns, interleaved with pooling layers that reduce the spatial dimensions and control overfitting. Activation functions like ReLU are applied after each convolutional layer to introduce non-linearity.

Optimization techniques are used to adjust the model's parameters to minimize the loss function during training. In this project, optimizers such as Adam or its variants are employed due to their ability to handle sparse gradients and adapt the learning rate dynamically. Techniques like learning rate scheduling and early stopping are also integrated.



Methodology

- **Data Preprocessing:** Normalized CIFAR-10 images, applied data augmentation (rotation, flipping, zoom), and used class weighting to handle class imbalance.
- **Training & Optimization:** Trained the model using the Adam optimizer with categorical cross-entropy loss, early stopping, and learning rate scheduling to improve accuracy and prevent overfitting.
- **Evaluation & Explainability:** Evaluated model performance using accuracy, precision, recall, F1-score, and confusion matrix; applied Grad-CAM and threshold tuning for model interpretability and decision refinement.

```
# Normalize and one-hot encode data
x_train, x_test = x_train / 255.0, x_test / 255.0
num_classes = 10
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

```
# Train Model with Class Weights
model.fit(x_train, y_train, epochs=10
          , validation_data=(x_test
                             , y_test)
                             , class_weight=class_weights_dict)
```

```
early_stopping = keras.callbacks.EarlyStopping(
    monitor='val_loss', patience=3, restore_best_weights=True
)

# PASSED CALLBACK TO FIT()
model.fit(
    x_train,
    y_train,
    epochs=10,
    validation_data=(x_test, y_test),
    class_weight=class_weights_dict,
    callbacks=[early_stopping]
)
```

```python
### Define CNN Model
def create_cnn_model():
    inputs = keras.Input(shape=(32, 32, 3))
    x = layers.Conv2D(32, (3,3), activation='relu', name="conv1")(inputs)
    x = layers.MaxPooling2D((2,2))(x)
    x = layers.Conv2D(64, (3,3), activation='relu', name="conv2")(x)
    x = layers.MaxPooling2D((2,2))(x)
    x = layers.Conv2D(128, (3,3), activation='relu', name="conv3")(x)
    x = layers.Flatten()(x)
    x = layers.Dense(128, activation='relu', name="dense1")(x)
    x = layers.Dropout(0.5)(x)
    outputs = layers.Dense(num_classes, activation='softmax', name="output")(x)
    model = keras.Model(inputs=inputs, outputs=outputs)
    return model
```

```python
### Layer-wise and Cumulative Layer Analysis
import gc

# Use a smaller subset
x_subset = x_test[:2000]
y_subset = y_test[:2000]

layer_accuracies = []
layer_outputs = []

for i, layer in enumerate(model.layers):
    if 'conv' in layer.name:
        sub_model = keras.Model(inputs=model.input, outputs=layer.output)
        features = sub_model.predict(x_subset, batch_size=128)

        gap = tf.reduce_mean(features, axis=[1, 2])  # Global Average Pooling
        features_flattened = gap.numpy()

        if layer_outputs:
            features = np.concatenate([layer_outputs[-1], features_flattened], axis=1)
        else:
            features = features_flattened

        layer_outputs.append(features)

        temp_model = keras.Sequential([
            layers.Input(shape=(features.shape[1],)),
            layers.Dense(num_classes, activation='softmax')
        ])
        temp_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
        temp_model.fit(features, y_subset, epochs=1, verbose=0)
        loss, acc = temp_model.evaluate(features, y_subset, verbose=0)
        layer_accuracies.append((layer.name, acc))

        del temp_model  # ◪ Memory cleanup
        gc.collect()
```

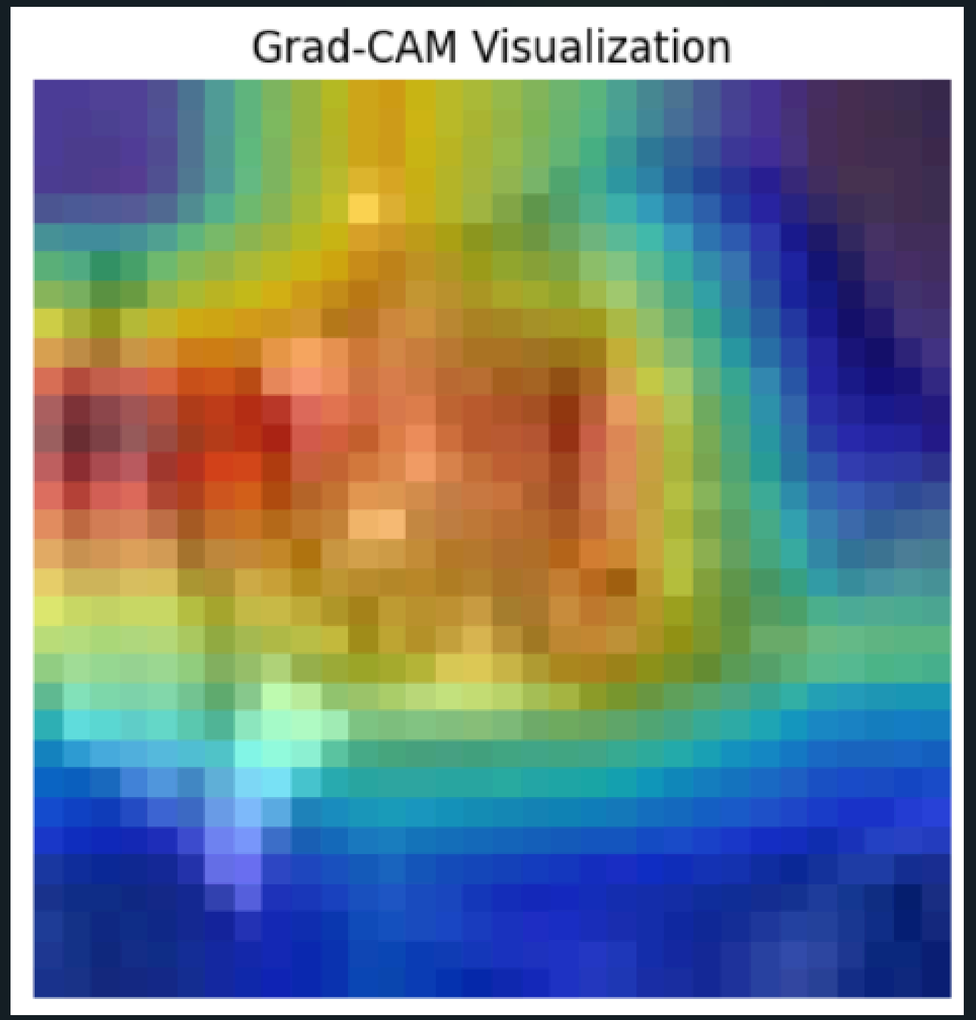
- **Model Design:** Built a CNN with convolutional, pooling, dropout, and fully connected layers; used ReLU and Softmax activation functions for feature extraction and classification.

- **Layer-wise Feature Map Analysis:** Took the output of the first convolutional layer to evaluate accuracy based on its feature map, helping to understand early-stage feature extraction.
- Comparative Layer Evaluation: Compared outputs of individual layers and analyzed accuracy after cumulatively summing each layer’s output to identify the most impactful layers in classification

Performance Metrics & Grad-CAM

Evaluating the performance of a CNN involves multiple metrics that collectively offer a comprehensive picture of model accuracy and reliability. Common metrics include overall accuracy, precision and recall, and the F1-score. Additional evaluation tools such as the confusion matrix provide detailed insights into how well each class is being predicted.

Grad-CAM (Gradient-weighted Class Activation Mapping) is an interpretability technique that helps visualize which parts of an input image are most influential in the model's decision. By computing the gradients of the predicted class score with respect to the feature maps of the last convolutional layer, Grad-CAM generates a heatmap that highlights key regions.



Airplane	0.69	0.77	0.73	1000
Automobile	0.82	0.82	0.82	1000
Bird	0.57	0.62	0.59	1000
Cat	0.56	0.52	0.54	1000
Deer	0.69	0.57	0.62	1000
Dog	0.66	0.57	0.61	1000

```
Optimal threshold for class 0: 0.463223934173584
Optimal threshold for class 1: 0.34410989284515380
Optimal threshold for class 2: 0.3562357723712921
Optimal threshold for class 3: 0.16881532967090607
Optimal threshold for class 4: 0.20995156466960907
Optimal threshold for class 5: 0.27900806069374084
```

```
Layer: conv1, Cumulative Accuracy: 0.1070
Layer: conv2, Cumulative Accuracy: 0.1955
Layer: conv3, Cumulative Accuracy: 0.3310
```

```
Layer: conv1, Accuracy: 0.1070
Layer: conv2, Accuracy: 0.1955
Layer: conv3, Accuracy: 0.3310
```

Testing & Quality Assurance

Quality assurance measures included overfitting prevention through dropout layers, data augmentation, and early stopping. Error analysis was performed on misclassified images, and the model was designed to be scalable and adaptable for future integration into larger systems.

The model underwent rigorous testing and validation to assess its performance. The dataset was split into 80% training and 20% testing to ensure a balanced evaluation. Performance metrics included accuracy, precision, recall, F1-score, and a confusion matrix. K-fold validation was performed to ensure model generalization on different data splits.

Conclusion & Future Scope

The project successfully developed and trained a CNN model for image classification on the CIFAR-10 dataset. The model was implemented using various convolutional layers, pooling layers, fully connected layers, and activation functions, which facilitate effective feature extraction and correct classification. The methods applied in this project can be generalized to a range of practical applications.

Future enhancements include fine-tuning with transfer learning, expanding to larger datasets, optimizing for real-time applications, addressing class imbalance, implementing attention mechanisms, and robust model explainability. These improvements will further enhance the model's performance and applicability.