Annaelle TOURET
Anthony PHILIPPE

Project Report:

Convolutional Neural Network for CIFAR-100 Classification

Introduction:

The aim of the project is to design and train a convolutional neural network (CNN) for classifying images from the CIFAR-100 dataset. The CIFAR-100 dataset consists of 60,000 32x32 colour images divided into 100 classes.
The project uses the Keras deep learning library as well as techniques for data pre-processing, data augmentation and a carefully designed CNN architecture.

Data Preprocessing:

The CIFAR-100 dataset is loaded and pre-processed to scale the pixel values between 0 and 1, known as normalization. To improve the model, data augmentation is employed using the Keras ImageDataGenerator, allowing rotation, horizontal and vertical shifts, shearing and zooming. This strategy increases the variability of the training dataset, improving model generalisation.

Keras is a neural network API in Python. It simplifies the development of artificial neural networks by providing a modular interface. Keras can be used to rapidly prototype and experiment with neural network architectures.

Model Architecture:

We opted for a convolutional neural network (CNN) because of its ability to capture spatial hierarchies and complex patterns in visual data. The translation invariance and parameter sharing characteristics of the CNN improve robust feature extraction and minimise over-fitting, making it particularly well suited to image classification tasks. The proven success of CNNs in various image-related applications and their adaptability to different input sizes reinforced our choice, demonstrating the model's effectiveness in handling the unique features of the CIFAR-100 dataset.

The CNN model is built using the Keras Sequential API. The architecture consists of three convolutional blocks, each composed of convolutional layers, batch normalisation, maximum pooling and dropout for regularisation. The final layers include a flattening operation, fully connected layers and softmax activation for multi-class classification.
The network ends with dense layers, incorporating batch normalisation and dropout to improve performance.
The last layer uses softmax activation for multi-class classification.
The model architecture can be visualised using Keras' plot_model function, giving a concise overview of the network structure.

Annaelle TOURET
Anthony PHILIPPE

Training Strategy:

The model is trained using Adam's optimizer. The Adam optimizer is chosen for its adaptive learning rate properties and its effectiveness in neural network optimization.
To adaptively adjust the learning rate during training, an exponentially decaying learning rate scheme is implemented. Early stopping is used to control validation loss and avoid over-adaptation. The training process stops if no improvement is observed after a certain number of epochs. In addition, a custom learning rate programming function is used to fine-tune the learning rate over time.

The SaveBestAccuracyModel callback function plays a central role in our training process by saving the model with the highest validation accuracy. Configured with parameters such as the file path for storage and the metric to be monitored (by default, validation accuracy), the callback evaluates and compares accuracy at the end of each epoch. If an improvement is detected, it updates and saves the model, allowing the best performing version to be retained. The verbosity parameter controls the level of detail in the printed output, providing valuable information about the model's progress during training. This reminder is essential for automating the selection and preservation of the most accurate model for future use.

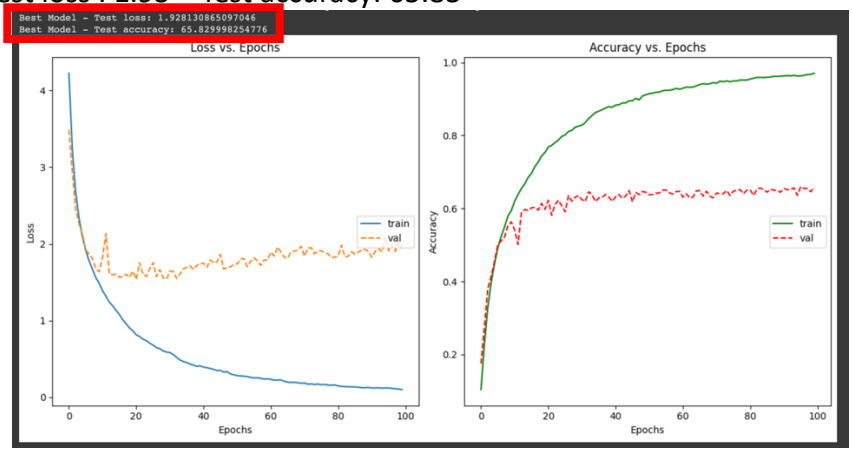
Model Evaluation:

The trained model is evaluated on a separate test set to assess its performance. Measurements of test loss and accuracy are reported. If there is a previous record of the model's accuracy, the current accuracy is compared and the model with the best accuracy is selected. This strategy is used to track and save the best performing model.

Results and Visualization:

The learning history, including loss and accuracy over time, is visualised using matplotlib. This provides a better understanding of the convergence and generalisation of the model. The plotted graphs help to analyse the trade-off between training and validation performance.

Our result : Test loss : 1.93 – Test accuracy: 65.83



Annaelle TOURET
Anthony PHILIPPE

Problem :

At the beginning, to get good results, our trainings included 350 epochs. This took a long time to execute. However, we obtained better results and accuracy was close to 70%.
We were also limited by Google collab on the number of GPU executions.

Conclusion:

The project demonstrates the successful implementation of a CNN for image classification on the CIFAR-100 dataset. Achieving 65% accuracy underlines the effectiveness of the chosen architecture and learning strategies. The difficulties encountered provide valuable pointers for future improvements, including the exploration of more sophisticated architectures and the optimisation of computing efficiency. This project makes a significant contribution to our understanding of convolutional neural networks and their application to image classification tasks.