**HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY AND EDUCATION**

**FACULTY FOR HIGH QUALITY TRAINING**



PROJECT REPORT

**DESIGN NETWORK**

**COURSE: AI Foundations and Applications**

**MAJOR: COMPUTER ENGINEERING TECHNOLOGY**

Students: **PHAN NHƯ KHÔI**

ID: 20119189

Advisor: PGS.TS Huynh The Thien

Ho Chi Minh City, MaY, 2023

## **Overview about my network**

This is a neural network implementation using TensorFlow and Keras that uses the ResNet-like architecture to perform image classification. The ResNet-like architecture used here consists of a series of convolutional layers followed by a series of residual blocks, which enable the network to learn more complex representations of the input images.

The network is trained on a dataset consisting of images divided into a training set and a validation set. The image data is loaded using the tf.keras.utils.image\_dataset\_from\_directory function, which loads the images from the directory and preprocesses them for use in the neural network. The model is compiled using the Adam optimizer and SparseCategoricalCrossentropy loss function, and the accuracy metric is used to evaluate the model during training.

Callbacks are used during training to stop the training early if the validation loss does not improve for a certain number of epochs, and to reduce the learning rate if the validation loss plateaus.

After training, the model is evaluated on a test set using the evaluate function, and the accuracy of the model on the test set is printed. Finally, the training and validation loss and accuracy are plotted using the Matplotlib library to provide a visual representation of the model's performance during training.

**Explain**

The code begins with importing necessary libraries such as matplotlib, numpy, PIL, and TensorFlow. Then, it defines two paths to the training and test datasets, counts the number of images in each directory, sets the batch size, image height and width, and creates training and validation datasets using the tf.keras.utils.image\_dataset\_from\_directory() method.

After creating the datasets, the code defines several functions that will be used to build the ResNet-like neural network architecture. These functions include conv\_block() which applies a convolutional layer, batch normalization, and ReLU activation; identity\_block() which is used for residual connections in the network; and ResNet\_like() which builds the ResNet-like architecture.

Next, the code compiles the model with the Adam optimizer and sparse categorical cross-entropy loss. It also sets accuracy as a metric for evaluation. The summary of the model is printed using the model.summary() method.

Then, the code defines two callbacks: early\_stop and reduce\_lr, which will be used during training to stop the training early if the validation loss does not improve for a certain number of epochs, and to reduce the learning rate if the validation loss does not improve for a certain number of epochs, respectively.

The model is trained using the model.fit() method, with the training and validation datasets, number of epochs, and callbacks as inputs. The training history is then plotted using matplotlib.

Finally, the model is evaluated on the test set using the model.evaluate() method, and the test accuracy is printed.

Diagram, table

Description automatically generatedDiagram, table

Description automatically generatedDiagram

Description automatically generatedDiagram

Description automatically generatedDiagram

Description automatically generatedDiagram

Description automatically generatedDiagram, table

Description automatically generatedTable

Description automatically generated

**Why I choose this netwwork architecture ?**The ResNet-like architecture implemented in your code is a deep convolutional neural network designed for image classification tasks.

The network starts with two convolutional layers followed by an identity block, which is a building block of the ResNet architecture. An identity block consists of two convolutional layers with the same number of filters and kernel size followed by batch normalization and a skip connection that bypasses the convolutional layers and adds the input directly to the output of the second convolutional layer. This helps to alleviate the problem of vanishing gradients and allows the network to be trained deeper.

After the identity block, the network continues with a convolutional layer with stride of 2, followed by two more identity blocks. This pattern of a convolutional layer followed by two identity blocks is repeated once more, with the number of filters increasing from 16 to 32.

Then, the network continues with another convolutional layer with stride of 2 followed by three more identity blocks. The number of filters in these identity blocks increases from 32 to 64.

Finally, the output of the last identity block is passed through a global average pooling layer to reduce the spatial dimensions of the output to a vector of length 64. This is followed by a fully connected layer with the number of units equal to the number of classes in the dataset.

The ResNet-like architecture implemented in your code has a total of 15 convolutional layers and 1 fully connected layer. The number of filters in the first convolutional layer is 8, which is relatively low, while the number of filters in the subsequent convolutional layers and identity blocks is higher. This is because the first convolutional layer is meant to extract simple features from the input images, while the subsequent layers build on these features to extract more complex and abstract features. The choice of 64 filters in the final identity block is a common choice in ResNet architectures.

**Using callbacks**

Callbacks are used in deep learning to improve the training process and avoid overfitting. They are functions that are called at certain points during training, allowing you to monitor the training process, modify the learning rate or stop training early if necessary.

Callbacks are used to monitor the training of a neural network and perform certain actions during training. In this code, two callbacks are used:

EarlyStopping: This callback is used to stop the training if the validation loss stops improving for a certain number of epochs (defined by the patience parameter). This helps prevent overfitting and reduces the training time.

ReduceLROnPlateau: This callback reduces the learning rate when the validation loss stops improving for a certain number of epochs (defined by the patience parameter). Reducing the learning rate can help the optimizer find a better minimum of the loss function, especially if it's oscillating around the minimum.

The result after training

Table

Description automatically generatedGraphical user interface

Description automatically generated with low confidence