DESCRIPTION OF DEEP LEARNING NETWORK IMPLEMENTATION EXAMPLES

- dense.py
  - This program trains a neural network to recognize handwritten digits from the MNIST dataset using the Keras library. The code includes the following steps:
    - Data Loading and Preprocessing: Loading the MNIST dataset and converting the images and labels into a suitable format for training the model.
    - Neural Network Definition: Build a multi-layer neural network with two layers, the first with 512 neurons and ReLU activation function, and the second with 10 neurons and Softmax activation function to classify into 10 classes.
    - Model Optimization and Training: Model optimization using the RMSprop optimizer and the categorical cross-entropy cost function, and training for 20 epochs with a batch size of 128.
    - Visualize Results: Create graphs that display loss and accuracy during model training and validation.
- dropout.py
  - This program trains a neural network to recognize handwritten digits from the MNIST dataset, using the Keras library. The network includes a dropout layer to improve generalization. The code includes the following steps:
    - Data Loading and Preprocessing: Loading the MNIST dataset and converting the images and labels into a suitable format for training the model.
    - Neural Network Definition: Create a multi-layer neural network with a dense layer of 512 neurons and ReLU activation function, a dropout layer with a rate of 0.5, and an output layer with 10 neurons and Softmax activation function to classify into 10 classes.
    - Model Optimization and Training: Model optimization using the RMSprop optimizer and the categorical cross-entropy cost function, and training for 20 epochs with a batch size of 128.
    - Visualize Results: Create a graph depicting the loss during model training and validation.
    - Prediction and Evaluation: Make predictions on test images, convert predictions to labels, and display some example predictions. Generate and visualize the confusion matrix to evaluate model performance.
- cifar10.py
  - This program trains and evaluates different kinds of neural networks for image recognition from the CIFAR-10 dataset, using the Keras library. The program includes models designed to show overfitting and underfitting phenomena, as well as tactics for improving model performance. It also includes convolutional neural network (CNN) and data augmentation method. The code includes the following steps:
    - Data Loading and Preprocessing: Loading the CIFAR-10 dataset and converting the images and labels into a suitable format for training the models.
    - Defining and Training an Underfitting Model: Building and training a simple, small neural network that is prone to underfitting.
    - Defining and Training an Overfitting Model: Building and training a deep neural network prone to overfitting.

        - Defining and Training a Normalized Model: Building and training a deep neural network with dropout normalization.
        - Defining and Training a Simple CNN: Building and Training a Convolutional Neural Network for Better Image Recognition Performance.
        - Data Augmentation and CNN Training: Use data augmentation to train the CNN in order to improve the generalization of the model.
        - Visualize Results: Create graphs that depict loss and accuracy during model training and validation, and visualize raw and augmented images.
- cifarGood.py
    - This program trains a convolutional neural network (CNN) for image recognition from the CIFAR-10 dataset, using the Keras library. The network includes multiple layers of convolutions and pooling, as well as dropout layers to improve generalization. The code includes the following steps:
        - Data Loading and Preprocessing: Loading the CIFAR-10 dataset and converting the images and labels into a suitable format for training the model. An ImageDataGenerator is also used for data-driven augmentation.
        - Neural Network Definition: Building a CNN with multiple layers of convolutions (Conv2D) and pooling (MaxPooling2D), as well as dropout layers to improve generalization.
        - Model Optimization and Training: Model optimization using the SGD optimizer with a learning rate of 0.001 and the categorical cross-entropy cost function, and training for 50 epochs with a batch size of 128. The early stopping strategy is also used to avoid overtraining.
        - Visualize Results: Create graphs that display loss and accuracy during model training and validation.