Documentation for cnn.py

This is a Python code for a Convolutional Neural Network (CNN) that is trained to classify images from the CIFAR-10 dataset. The CIFAR-10 dataset consists of 60,000 32x32 color images in
10 classes, with 6,000 images per class. The goal of the model is to classify these images into their corresponding class labels.

The code begins with importing the necessary modules, including the Keras library for building deep learning models. The code then loads the CIFAR-10 dataset into two sets, one for training the model and the other for testing it.

The pixel values of the images are normalized by calculating the mean and standard deviation of the pixel values of the training dataset, and then subtracting the mean and dividing by the standard deviation. This is done to make sure that the input values are in the same range, which helps the model learn better.

Next, the labels are converted to one-hot encoded vectors using the np_utils.to_categorical() function. This is a common step in deep learning image classification tasks, where each label is represented by a vector of length equal to the number of classes, with a 1 in the position corresponding to the class label and 0's in all other positions.

The CNN model is defined using the Sequential() class of the Keras library. It consists of several convolutional layers with increasing number of filters, followed by max-pooling layers and dropout regularization to prevent overfitting. The layers are connected using the Flatten() and Dense() functions. The model is then compiled with a loss function, optimizer, and accuracy metric.

Data augmentation is performed using the ImageDataGenerator() function, which randomly applies transformations to the images to increase the size of the training dataset and improve the generalization of the model. These transformations include rotation, horizontal and vertical flipping, and shifting.

The model is then trained using the fit_generator() function, which takes in the training and validation data, as well as the batch size and number of epochs. The model is trained using different learning rates and number of epochs to find the best hyperparameters.

Finally, the trained model is evaluated on the test dataset and the accuracy and loss values are printed. The model is then saved in a .h5 file for future use.

Overall, this code demonstrates how to build and train a CNN model for image classification using the CIFAR-10 dataset.

Documentation for test_image.py
This code is a Python script that loads a trained CIFAR-10 image classification model and uses it to make predictions on images downloaded from the internet.

The first few lines of code disable certificate verification, which is a security feature that verifies the authenticity of certificates for secure connections. This is done to allow the script to download images from websites that may not have a valid certificate.

The script then imports several libraries including TensorFlow, Matplotlib, pathlib, and certifi. It also defines a list of class names that correspond to the 10 different image classes in the CIFAR-10 dataset.

The script then defines a function `load_image` that takes a filename as input, resizes the image to 32x32 pixels, converts it to an array, normalizes the pixel values between 0 and 1, and returns the resulting image array.

Next, the script loads a pre-trained CIFAR-10 model from a saved file `cifar10_normal_rms_ep125.h5`.

The script then proceeds to download three images from the internet using their respective URLs. For each image, it loads the image file using the `load_image` function, feeds it into the pre-trained model using the `model.predict` function, and prints out the predicted image class. It also displays the downloaded image using Matplotlib's `plt.imshow` function.

Finally, the script prints out the predicted image class for each of the three downloaded images.

Documentation for app.py
This is a Python program that uses OpenAI's text-generation API to generate names for humans, animals, and other objects. It also uses Flask, a web application framework, to create a basic web application to work with user inputs.

The first line imports the os module, which provides a way to interact with the operating system. The second line imports the openai module, which provides access to OpenAI's API. The third line imports the Flask module, which is used to create the web application.

The next line initializes the Flask application with the name of the current module.

The next line sets the API key for OpenAI's API. The key is stored in an environment variable called "OPENAI_API_KEY".

The next block of code defines a route for the application. The route is "/", which means the root of the website. The route accepts both GET and POST requests. When a GET request is

received, it renders the index.html template with the result argument set to None. When a POST request is received, it takes the value of the animal input from the form, generates a prompt using the generate_prompt function, sends the prompt to OpenAI's API using the create method of the Completion object, and then redirects the user back to the index page with the result set to the first suggestion returned by the API.

The generate_prompt function takes an animal as an argument and returns a string that contains a prompt for generating superhero names for that animal. The string contains placeholders for the animal and the four names to be generated.

Finally, the program starts the Flask application with the run method.

Documentation for chatgpt_query.py
This code is a Python script that utilizes OpenAI's GPT-3.5 language model to generate responses to user prompts in natural language.

Before using the script, the user must sign up for an OpenAI account and obtain an API key, which should be placed in a .env file. The .env file is loaded using the `dotenv` library. The script imports the `openai` and `os` libraries. The former is used to access the GPT-3.5 API, while the latter is used to access environment variables, including the OpenAI API key stored in the .env file.

The `generate_response(prompt)` function takes a user prompt as input and generates a response using the GPT-3.5 model. The function first sets the model engine to "text-davinci-003," which is one of the most capable models in the GPT-3.5 family. It then creates a `completions` object using the `openai.Completion.create()` method, passing in the model engine, the user prompt, the maximum number of tokens (set to 2000), the number of completions to generate (set to 1), and the temperature (set to 0.5).

The function then prints the generated response to the console.

The `while (True):` loop prompts the user to enter a question or prompt. If the user enters "quitme", the loop breaks and the script terminates. Otherwise, the script calls the `generate_response(prompt)` function to generate a response, which is printed to the console.