Depth Estimation Project with more complicated structure

This project estimates the depth of 2D images using a Convolutional Neural Network (CNN). The project is implemented using PyTorch, PyTorch Lightning and uses Weights & Biases (wandb) for experiment tracking.

Differences with previous version

This version of the model includes several changes compared to the previous versions:

- This model utilizes a deeper architecture with more Convolutional layers.
- The model uses transposed convolutions to generate depth maps.
- The model includes data augmentation in the form of vertical flipping and perspective distortion.
- The DataLoader is now wrapped in a PyTorch Lightning DataModule for easier data handling.
- The model uses the AdamW optimizer with a One Cycle Learning Rate scheduler.
- The model training and evaluation are now handled within a PyTorch LightningModule, making the code more modular and easier to use.
- Logging and model saving is now handled with Weights & Biases.

Prerequisites

- Python 3.6 or later
- PyTorch 1.8 or later
- PyTorch Lightning
- Wandb
- Google Colab
- Google Drive

Installation

You can install the necessary libraries by running the following commands:

```
pip install torch torchvision torchaudio
pip install torchsummary
pip install pytorch-lightning
pip install wandb
```

The script also requires an NVIDIA GPU. You can check your NVIDIA System Management Interface with the following command:

```
nvidia-smi
```

Weights and Biases

The script uses Weights and Biases (wandb) for experiment tracking. You need to log in to wandb using your API key:
wandb login YOUR_API_KEY
Replace YOUR_API_KEY with your actual API key from wandb.

Dataset

The dataset should be stored on Google Drive. The dataset consists of pairs of RGB images and their corresponding depth maps. The path to the dataset is specified in the data_dir variable.

The data used in this script is stored on Google Drive. You need to mount your Google Drive to Colab with the following command:

from google.colab import drive
drive.mount('/content/gdrive')

Usage

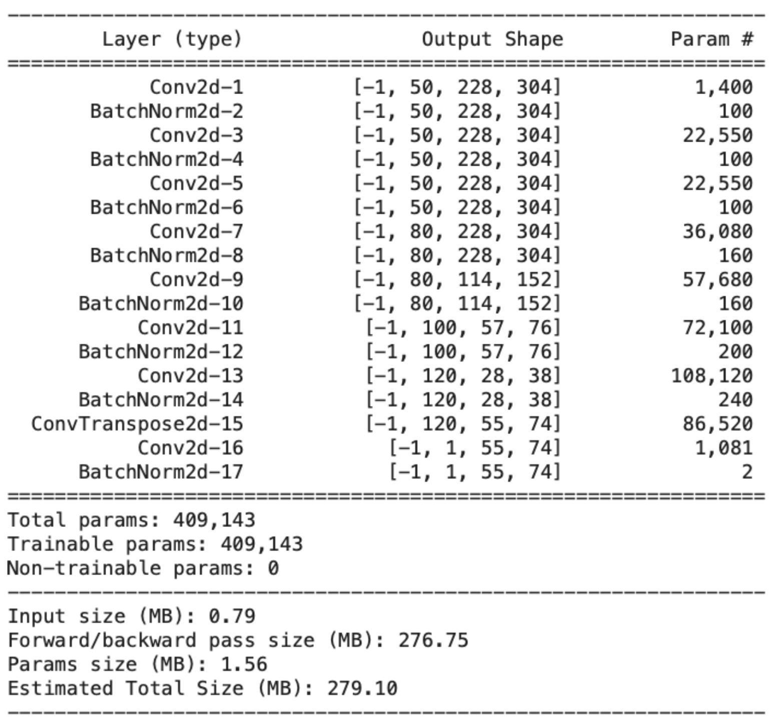
Firstly, ensure that the Google Drive containing the dataset is mounted. This can be done using the following command in Google Colab:

```
from google.colab import drive
drive.mount('/content/gdrive')
```

The main class of the project is Model, which inherits from pl.LightningModule. This class encapsulates the entire ML pipeline: the CNN model, the loss function and the optimizers. It also contains methods for the training, validation and test steps.
The Datamodule class is used to manage the data. It loads the images and their corresponding depth maps from the dataset, applies the necessary transformations and splits the data into training, validation and test sets.
To train the model, you need to create an instance of the Model class and the Datamodule class and pass them to the fit method of a pl.Trainer instance.

Model

The model used in this script is a CNN with a certain architecture. The model and its architecture are defined in the CNN class.



```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1        [-1, 50, 228, 304]           1,400
       BatchNorm2d-2        [-1, 50, 228, 304]             100
            Conv2d-3        [-1, 50, 228, 304]          22,550
       BatchNorm2d-4        [-1, 50, 228, 304]             100
            Conv2d-5        [-1, 50, 228, 304]          22,550
       BatchNorm2d-6        [-1, 50, 228, 304]             100
            Conv2d-7        [-1, 80, 228, 304]          36,080
       BatchNorm2d-8        [-1, 80, 228, 304]             160
            Conv2d-9        [-1, 80, 114, 152]          57,680
      BatchNorm2d-10        [-1, 80, 114, 152]             160
           Conv2d-11         [-1, 100, 57, 76]          72,100
      BatchNorm2d-12         [-1, 100, 57, 76]             200
           Conv2d-13         [-1, 120, 28, 38]         108,120
      BatchNorm2d-14         [-1, 120, 28, 38]             240
  ConvTranspose2d-15         [-1, 120, 55, 74]          86,520
           Conv2d-16           [-1, 1, 55, 74]           1,081
      BatchNorm2d-17           [-1, 1, 55, 74]               2
================================================================
Total params: 409,143
Trainable params: 409,143
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.79
Forward/backward pass size (MB): 276.75
Params size (MB): 1.56
Estimated Total Size (MB): 279.10
----------------------------------------------------------------
```

CNN model structure:

A CNN (Convolutional Neural Network) is a type of deep learning model primarily used for image processing tasks. In the code you provided, the CNN is defined in the class CNN.
The architecture of this CNN model is as follows:

- **Convolutional layer 1** (conv1): This layer has 3 input channels (corresponding to the RGB channels of an input image), and 50 output channels. The kernel size is 3 and padding is 1. The output of this layer goes through a ReLU activation function and then Batch Normalization.
- **Convolutional layer 2** (conv2): This layer has 50 input channels and 50 output channels. The kernel size is 3 and padding is 1. The output of this layer goes through a ReLU activation function and then Batch Normalization.
- **Convolutional layer 3** (conv3): This layer has 50 input channels and 80 output channels. The kernel size is 3 and padding is 1. The output of this layer goes through a ReLU activation function, then Batch Normalization, and then MaxPooling with a size of 2.
- **Convolutional layer 4** (conv4): This layer has 80 input channels and 80 output channels. The kernel size is 3 and padding is 1. The output of this layer goes through a ReLU activation function, then Batch Normalization, and then MaxPooling with a size of 2.
- **Convolutional layer 5** (conv5): This layer has 80 input channels and 100 output channels. The kernel size is 3 and padding is 1. The output of this layer goes through a ReLU activation function, then Batch Normalization, and then MaxPooling with a size of 2.

- **Convolutional layer 6** (conv6): This layer has 100 input channels and 120 output channels. The kernel size is 3 and padding is 1. The output of this layer goes through a ReLU activation function, then Batch Normalization, and then a Transposed Convolution operation to increase the spatial dimensions of the feature map.
- **Convolutional layer 7** (conv7): This layer has 120 input channels and 1 output channel. The kernel size is 3 and padding is 1. The output of this layer goes through a ReLU activation function and then Batch Normalization.

The model uses a ReLU (Rectified Linear Unit) activation function after each Convolutional layer. ReLU is a popular choice for CNNs because it introduces non-linearity without affecting the receptive fields of the convolution layers.
The model also uses Batch Normalization after each ReLU activation. Batch Normalization makes models faster and more stable through normalization of the layers' inputs by re-centering and re-scaling.
This CNN model structure is a typical example of a deep convolutional neural network, with several convolutional layers of increasing depth, interspersed with activation functions and normalization, and pooling layers to reduce spatial dimensions while increasing the depth (number of channels) of the feature maps.

Program structure:

Environment setup: The code installs necessary libraries like torchsummary, wandb, and checks if a GPU is available for training.

Dataset Definition: CustomImageDataset class is defined, which inherits from PyTorch's Dataset class. This class is responsible for loading the image and corresponding labels, applying necessary transformations, and defining the way to access each data sample.

DataLoader Definition: Datamodule class is defined, which inherits from PyTorch Lightning's LightningDataModule class. It's responsible for preparing the train, validation, and test data loaders, which will later feed data into the model.

Model Definition: CNN class is defined for the convolutional neural network architecture. It consists of multiple convolutional layers, batch normalization, and max-pooling layers. The output of the network is a depth map of the same size as the input image.

Model Training: Model class is defined, which inherits from PyTorch Lightning's LightningModule class. It encapsulates the model, the loss function, and the optimizer. It also defines the training, validation, and test steps.

Model Training Execution: The code sets up the model with a learning rate scheduler, trains it using the train data loader, and validates it using the validation data loader. The training progress is logged using the wandb (Weights & Biases) logger.

Model Evaluation: The code tests the trained model with the test data loader. It also visualizes the predicted depth maps from the model.

Training

The script uses PyTorch Lightning for training the model. The trainer is defined in the Model class. You can specify the number of epochs and learning rate for training. The default values are 1 epoch and a learning rate of 0.001.

Logging and Monitoring

The script uses Weights and Biases for logging and monitoring the training process. You can view the training progress on your wandb dashboard.

Testing

After training, the script tests the model with test data. The test data is loaded from the same directory as the training data. The testing results are logged to wandb.

Visualization

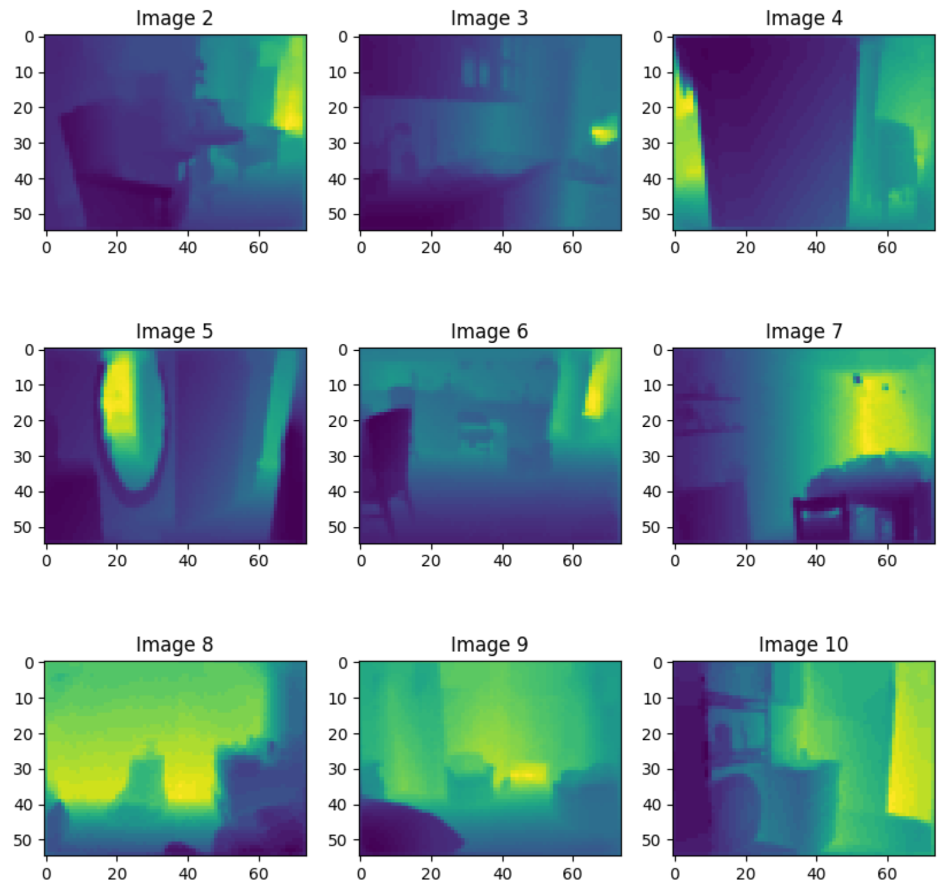
The script includes functions for visualizing the training and testing results. You can view these visualizations on your wandb dashboard or in the Python script's output.

```
Remember to replace the placeholders with your actual data and configurations.
```

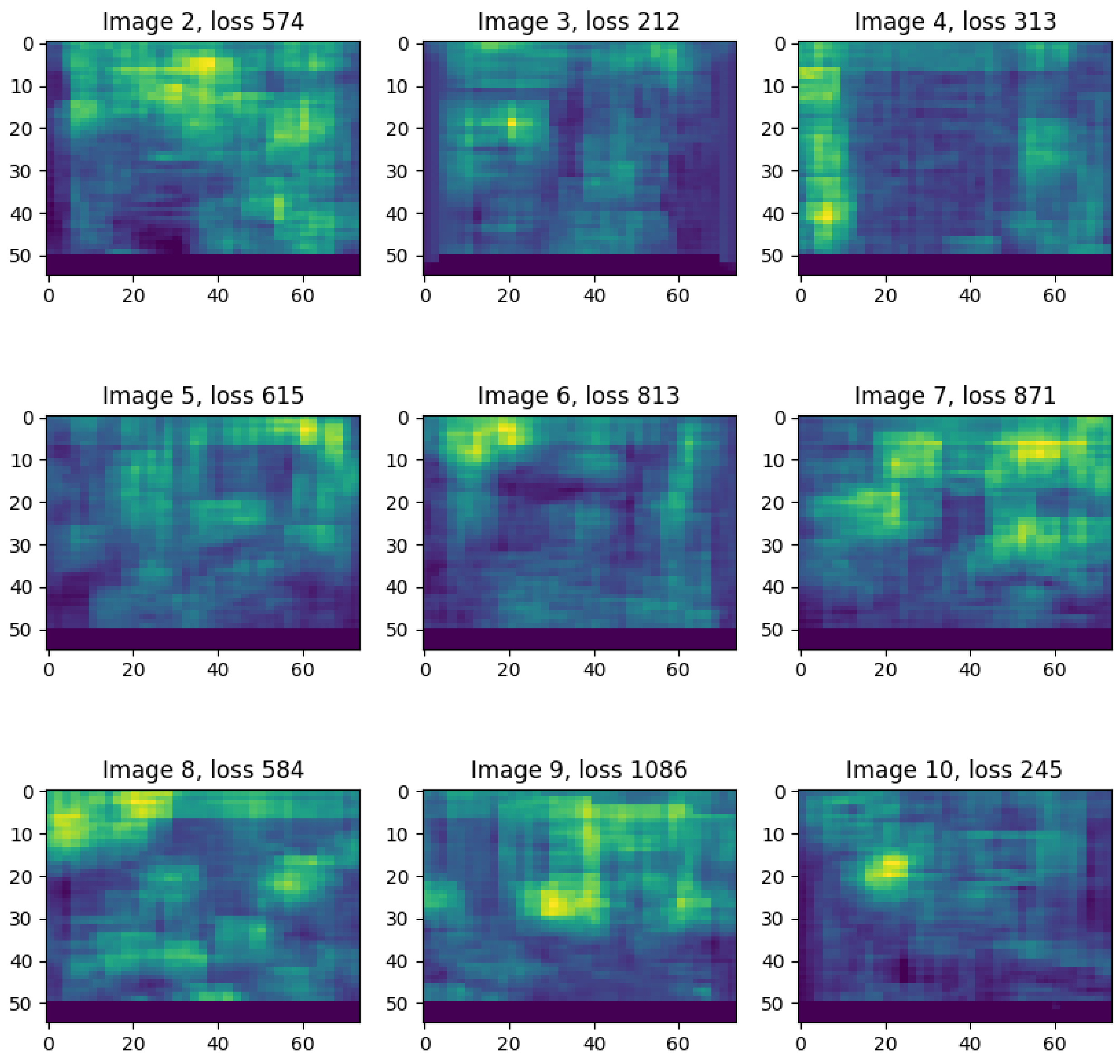
Notes

- The model is trained on a GPU if one is available. If not, it falls back to using a CPU.
- The model uses a custom CNN architecture. The architecture consists of multiple convolutional layers with batch normalization, followed by max pooling layers and finally a transposed convolutional layer.
- The model uses the Mean Squared Error (MSE) loss function and the AdamW optimizer with a learning rate scheduler.
- The model is logged to Weights & Biases for experiment tracking.
- The vis function is used to visualize the input images, their corresponding depth maps and the estimated depth maps.

Results:
Original:



predict results:



Original code from:
https://www.kaggle.com/code/nguynththanhho/depth-estimation-from-single-image-cnn/notebook