PREVIOUS PROJECTS IMPLEMENTATION OVERVIEW

Projects That I have worked on:

1) Real-time Edge Detection
2) Single Image Depth Estimation

1) REAL-TIME EDGE DETECTION

AIM: To interface OV7670 camera module to Xilinx Zedboard Zynq-7000 and output an edge-detected video stream.

IMPLEMENTATION:

1) Interfaced the OV7670 with the Zedboard.
   i) The RGB565, horizontal reference and vertical sync values are obtained from the camera pins along with other clock signals.

   ii) The RGB565 is converted to RGB888 format so that it can be compatible with the readily available AXI-Stream IP.
   The pins of the camera were mapped according to [2].

2) The input stream is then converted to AXI4-Stream using the default Video-In to AXI4-Stream IP block. This stream data is then sent to the custom edge detection IP block (Sobel edge detection).

3) The Edge detection IP does the following:
   i) Convert the AXI Stream to HLSMat format
   ii) RGB to GrayCode conversion
   iii) Apply Sobel operation.
   iv) GrayCode to RGB conversion
   v) HLSMat to AXI Stream format

4) Converted the AXI Stream to RGB video out format using the AXI4-Stream to Video-Out IP. Finally RGB_to_VGA IP is used to stream data in VGA format to display it on screen.

   The corresponding constraints for input and output pin mapping to Zedboard Zynq-7000 were written.

REFERENCES:
[1] Codes and implementation details:
http://web-pcm.cnfm.fr/wp-content/uploads/2017/04/Workbook-Digilent_ZYBO_Video_Workshop.pdf
[2] OV7670 pin configuration and interfacing:
https://www.instructables.com/Connect-Camera-to-Zybo-Board
[3] Other references:
https://people.ece.cornell.edu/land/courses/ece5760/FinalProjects/f2007/hc454_gtc32/hc454_gtc32/index.html

2) SINGLE IMAGE DEPTH ESTIMATION

AIM: To improve performance by adding better alternative loss functions to predict the depth of various pixels in a given image using deep learning techniques.

TOOLS/ PACKAGES USED: Python, Keras, Sci-kit learn; The entire process of training was carried out on Google Colab.

IMPLEMENTATION:

1) **Network Architecture**:
   Uses an encoder decoder architecture.
   Encoder: DenseNet-169 which is a pretrained network on ImageNet is used as the encoder. The input RGB image is encoded into a feature vector using this network.

   Decoder: The feature vector obtained from encoder output is fed to a series of up-sampling layers and skip connections, which forms the decoder.

   Each up-sampling layer is made of the following layers:
   - ➢ Bilinear upsampling
   - ➢ Concatenation layer used as skip connection
   - ➢ 2D Conv Layer (kernel size=3x3, stride=1, padding=none)
   - ➢ LeakyReLu Activation Layer
   - ➢ 2D Conv Layer (kernel size=3x3, stride=1, padding=none)
   - ➢ LeakyReLu Activation Layer

   Finally there is a convolution layer at the end as part of decoder to extract depth

LAYER	OUTPUT	FUNCTION
INPUT	$480 \times 640 \times 3$	
CONV1	$240 \times 320 \times 64$	DenseNet CONV1
POOL1	$120 \times 160 \times 64$	DenseNet POOL1
POOL2	$60 \times 80 \times 128$	DenseNet POOL2
POOL3	$30 \times 40 \times 256$	DenseNet POOL3
...	...	...
CONV2	$15 \times 20 \times 1664$	Convolution 1×1 of DenseNet BLOCK4
UP1	$30 \times 40 \times 1664$	Upsample 2×2
CONCAT1	$30 \times 40 \times 1920$	Concatenate POOL3
UP1-CONVA	$30 \times 40 \times 832$	Convolution 3×3
UP1-CONVB	$30 \times 40 \times 832$	Convolution 3×3
UP2	$60 \times 80 \times 832$	Upsample 2×2
CONCAT2	$60 \times 80 \times 960$	Concatenate POOL2
UP2-CONVA	$60 \times 80 \times 416$	Convolution 3×3
UP2-CONVB	$60 \times 80 \times 416$	Convolution 3×3
UP3	$120 \times 160 \times 416$	Upsample 2×2
CONCAT3	$120 \times 160 \times 480$	Concatenate POOL1
UP3-CONVA	$120 \times 160 \times 208$	Convolution 3×3
UP3-CONVB	$120 \times 160 \times 208$	Convolution 3×3
UP4	$240 \times 320 \times 208$	Upsample 2×2
CONCAT3	$240 \times 320 \times 272$	Concatenate CONV1
UP2-CONVA	$240 \times 320 \times 104$	Convolution 3×3
UP2-CONVB	$240 \times 320 \times 104$	Convolution 3×3
CONV3	$240 \times 320 \times 1$	Convolution 3×3

2) **Loss Function**:

Main work involved trying out various alternatives for the SSIM loss function presented in the paper and improve performance.

The additionally proposed loss functions were:

i) Scale Invariant Error (Lsi)
ii) Using Sobel operator (L_Sobel) as an additional loss for edge detection
iii) Using Gradient Magnitude Similarity Deviation (GMSD) as a replacement for SSIM

3) Proposed 2 models incorporating different combinations of the above mentioned loss functions. The main aim was to compare which was able to capture object boundaries effectively.

4) **DATA SET**:
i) The model was trained on the NYU Depth v2 dataset. The dataset provides depth maps for various indoor scenes captured at 640 x 480 resolution.
ii) The model was trained on 50K images and validated on 654 testing samples. The depth maps have an upper bound of 10m.
iii) The network produces results at 320 x 240 resolution.

5) **Benchmarks**:
The benchmarks used were training loss, validation loss, relative error, rms error and log10 error metrics.

REFERENCES:
[1] Alhashim, Ibraheem, and Peter Wonka. "High Quality Monocular Depth Estimation via Transfer Learning." arXiv preprint arXiv:1812.11941 (2018).
[2] Base code:- https://github.com/ialhashim/DenseDepth
[3] Xue, Wufeng, et al. "Gradient magnitude similarity deviation: A highly efficient perceptual image quality index." IEEE Transactions on Image Processing 23.2 (2013): 684-695.
[4] Google Colab : https://colab.research.google.com/
[5] Li, Zhengqi, et al. "Learning the Depths of Moving People by Watching Frozen People." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition . 2019.