॥ त्वं ज्ञानमयो विज्ञानमयोऽसि ॥

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY, JODHPUR

EEL3090: Embedded Systems

ES8 : Conversion of DL models from high-level (Python libraries) to C++ descriptions

By

Roll Number	Name
B21EE014	Atharva Ganesh Pade
B21EE030	Kalbhavi Vadhi Raj

I. OBJECTIVE:

In this Project, our objective is to convert DL models from high-level (Python libraries) to C++. We will be converting some of the popular models, like ResNet50, Mobilenetv3 Small, etc.

II. FUNCTIONS IMPLEMENTED:

We have implemented all the functions mentioned below from scratch based on the standard known algorithms that were discussed in the Deep Learning course (CSL4020). These implemented layers/functions are utilised for implementing the neural networks that we will discuss.

Activation Layer	
ReLu	Applies the rectified linear unit activation function.
Leaky ReLu	Leaky relu activation function.
Sigmoid	It is defined as: sigmoid(x) = 1 / (1 + exp(-x))
Tanh	Hyperbolic tangent activation function.
Softmax	Softmax converts a vector of values to a probability distribution.
Hard Swish	Hard sigmoid activation function.
Convolution Layer	
Padding	Convolution operation with padding on an input image
Add padding in the middle	Adds padding to the input array, with zeros placed in the middle of each patch.
Convolution	Performs 2D convolution on an input image with given kernel weights and biases.
Transposed Convolution	Performs transposed convolution by padding the input, then applying regular convolution with adjusted parameters.
Depth wise Convolution	This function performs depthwise convolution.
Pooling Layers	
Max Pooling	Downsamples the input along its spatial dimensions (height and width) by taking the maximum value over an input window for each

	channel of the input.
Average Pooling	Downsamples the input along its spatial dimensions (height and width) by taking the average value over an input window for each channel of the input.
Fully Connected Layers	
Linear	Computes a linear transformation by applying weights to the input and adding biases, producing an output of a specified size.
Add Noise	Adds Gaussian noise to each element of the input array.

III. ResNet50:

In our course project, we have implemented ResNet50 in Cpp. ResNet-50 is a variant of the Residual Network (ResNet) architecture; it consists of 50 layers. These layers include convolutional layers, pooling layers, and fully connected layers.

We have implemented ResNet50 as mentioned in "Deep Residual Learning for Image Recognition," [1]. The paper provides detailed archietecture on how to implement the ResNet model. Table-1 provides detail for each individual layer.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Table-1

RESULTS:

Test image:

Mask on (class = 1)

Mask on (class = 0)

Predicted Output:
Image 1: 1
Image 2: 0

IV. Mobilenetv3 Small:

MobileNetV3 Small is a variant of the MobileNetV3 architecture that is made specifically for mobile and edge computing devices where computational resources are limited. We used the Imagenet dataset [3] to verify the model. The details of the model can be seen in the table-2

Input	Operator	exp size	#out	SE	NL	s
$224^2 \times 3$	conv2d, 3x3	-	16	-	HS	2
$112^2 \times 16$	bneck, 3x3	16	16	✓	RE	2
$56^2 \times 16$	bneck, 3x3	72	24	-	RE	2
$28^2 \times 24$	bneck, 3x3	88	24	-	RE	1
$28^2 \times 24$	bneck, 5x5	96	40	✓	HS	2
$14^2 \times 40$	bneck, 5x5	240	40	✓	HS	1
$14^2 \times 40$	bneck, 5x5	240	40	✓	HS	1
$14^2 \times 40$	bneck, 5x5	120	48	✓	HS	1
$14^2 \times 48$	bneck, 5x5	144	48	✓	HS	1
$14^2 \times 48$	bneck, 5x5	288	96	✓	HS	2
$7^2 \times 96$	bneck, 5x5	576	96	✓	HS	1
$7^2 \times 96$	bneck, 5x5	576	96	✓	HS	1
$7^2 \times 96$	conv2d, 1x1	-	576	✓	HS	1
$7^2 \times 576$	pool, 7x7	-	-	-	-	1
$1^2 \times 576$	conv2d 1x1, NBN	-	1024	-	HS	1
$1^2 \times 1024$	conv2d 1x1, NBN	-	k	-	-	1

Table-2

RESULTS:

Test image:

Class = 0, Tench (n01440764_tench.JPEG) Output: 0

VII. OTHER MODELS:

We have implemented the rest of the neural networks; ASL Quant, Handwritten CNN, and MNIST.

VI. INNOVATION:

After the use of a variable is over, we are freeing the allocated value. Freeing the allocated value ensures that memory is released for other operations. This is good practice, as it helps prevent memory leaks and optimises memory usage within the program.

VII. FUTURE WORK:

Implement YoloV4 Tiny in cpp, using already existing layers.

References:

[1] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, USA, 2016, pp. 770-778, doi: 10.1109/CVPR.2016.90.

[2] https://arxiv.org/abs/1905.02244

[3] Imagenet Dataset