Image compression with deep learning

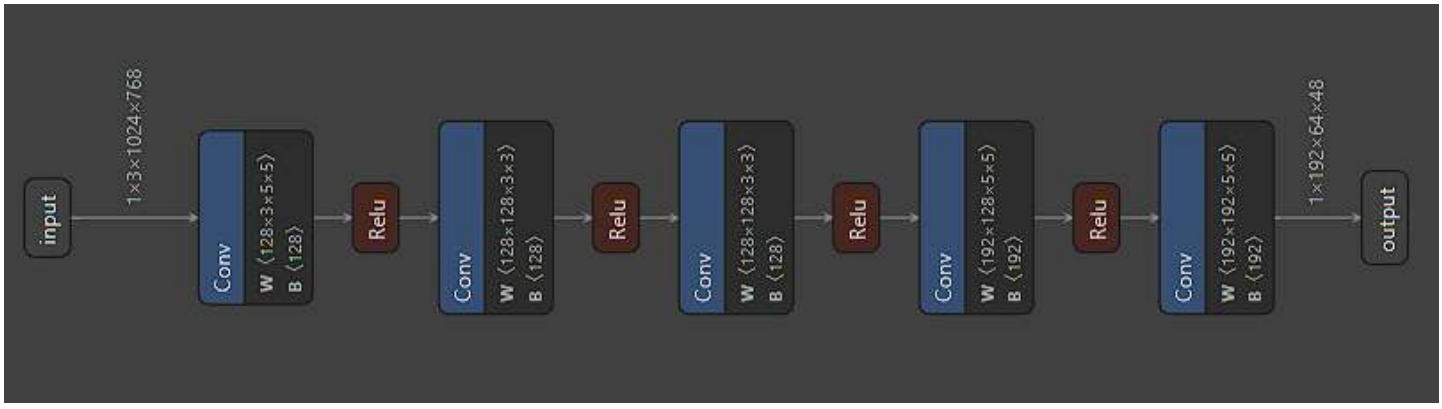
Dr. Binod Kumar
Udit Agarwal (B20ME076)
Sourav Banerjee (M21EE013)

Task: The task assigned was to test the already existing .onnx model and check if the reconstructed images are of good quality, if not to make and train a model from scratch to perform image compression and reconstruction tasks.

The task can be divided into 2 sub tasks, making the .onnx model and testing it and making and training the scratch model.

Task 1: The encoder decoder model's .onnx files were provided already but there is no already existing library or method that can convert the .onnx models to tensorflow models directly therefore we came up with a new technique to convert the models.

Netorn (website) is an already existing web platform where we can upload a .onnx model to visualize its weights and bias vector sizes. Using netron we were able to obtain the weights and bias vector sizes and from them we were able to make out the tensorflow model layers.



(Encoder Model weights and bias size obtained from Netron)

For 1st convolutional layer the weights vector size is (128x3x5x5) and since the input is of size 3x1024x768, this can be interpreted that the kernel size is 5x5, and it has 3 channels and total 128 filters channels, similar approach can be applied to all the layers to make the enoder model and similarly the decoder model.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 512, 384, 128)     9728

 conv2d_1 (Conv2D)           (None, 256, 192, 128)     147584

 conv2d_2 (Conv2D)           (None, 256, 192, 128)     147584

 conv2d_3 (Conv2D)           (None, 128, 96, 192)      614592

 conv2d_4 (Conv2D)           (None, 64, 48, 192)       921792

=================================================================
Total params: 1,841,280
Trainable params: 1,841,280
Non-trainable params: 0
_________________________________________________________________
None
```

(Made model architecture)

Now that we have the model we need to upload the weights, for the purpose the individual weights and bias were extracted as numpy array and stored in dictionaries.

```
weights = {}
for initializer in model.graph.initializer:
    name = initializer.name
    print (name)
    tensor = numpy_helper.to_array(initializer)
    print (tensor.shape)
    weights[name] = tensor
```

```
g_a.0.weight: (128, 3, 5, 5)
g_a.0.bias: (128,)
g_a.2.weight: (128, 128, 3, 3)
g_a.2.bias: (128,)
g_a.4.weight: (128, 128, 3, 3)
g_a.4.bias: (128,)
g_a.6.weight: (192, 128, 5, 5)
g_a.6.bias: (192,)
g_a.8.weight: (192, 192, 5, 5)
g_a.8.bias: (192,)
```

The individual layer weights and bias are loaded in tensorflow model, using for loop.

Task 2: The results were not up to the mark from the pretrained model, hence another model was prepared from scratch and tested. The model was made of almost similar architecture because we want to have the model on edge devices. The previous model was good but because of its smaller architecture it couldn't learn all the features in the image and reconstruct effectively. From the understanding of convolutional neural networks, we know the model learns in the form of kernels, the kernel size in the middle 2 convolutional layers is 3x3 which can be changed to 5x5, so as to achieve more learning, the strides for the new model are updated to (2,2) for all the layers since increased kernel size can involved more pixel and hence information less due to intermixing of pixel data can lead to information loss.

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_5 (Conv2D)           (None, 512, 384, 128)     9728

 conv2d_6 (Conv2D)           (None, 256, 192, 128)     409728

 conv2d_7 (Conv2D)           (None, 256, 192, 128)     409728

 conv2d_8 (Conv2D)           (None, 128, 96, 192)      614592

 conv2d_9 (Conv2D)           (None, 64, 48, 192)       921792

=================================================================
Total params: 2,365,568
Trainable params: 2,365,568
Non-trainable params: 0
```

(Newly trained model)

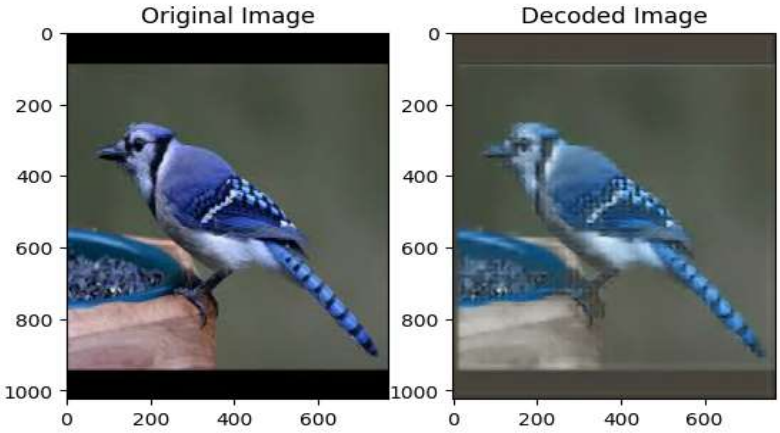
The model was trained on a custom dataset prepared from a game video, the dataset was prepared by extracting frames from a video publicly available. The optimizer used for training is Adam and the loss function used in Mean Absolute Error. The dataset prepared was of almost 10,000 images, training was done on a batch size of 16 and for 5 epochs.

```
Epoch 1/5
659/659 [==============================] - 1689s 2s/step - loss: 0.1224 - accuracy: 0.7034
Epoch 2/5
659/659 [==============================] - 1451s 2s/step - loss: 0.0697 - accuracy: 0.7410
Epoch 3/5
659/659 [==============================] - 1451s 2s/step - loss: 0.0636 - accuracy: 0.7551
Epoch 4/5
659/659 [==============================] - 1454s 2s/step - loss: 0.0399 - accuracy: 0.8481
Epoch 5/5
659/659 [==============================] - 1456s 2s/step - loss: 0.0338 - accuracy: 0.8746
<keras.callbacks.History at 0x7f1871683820>
```

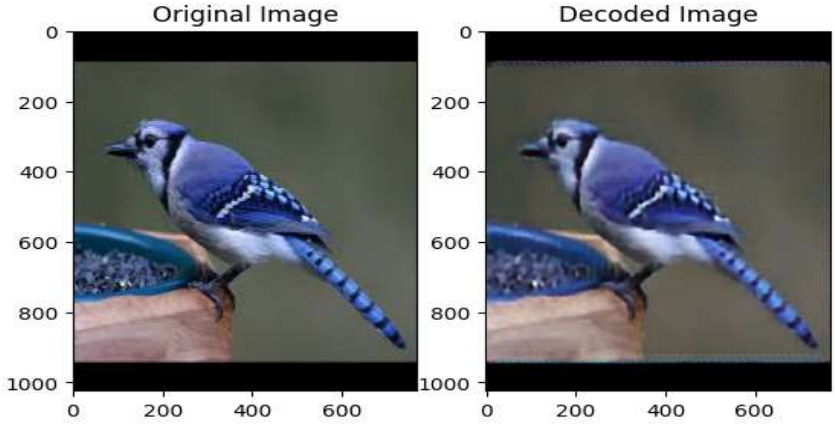
(accuracy with each epoch)

The maximum accuracy achieved is 87.46 percent.

Results:



(Reconstruction with pretrained model)



(Reconstruction results from new model)