COEN 240 Machine Learning

Term Project

Alex Cherekdjian - 00001083236
PJ McCurdy - 00001171451

Task 1

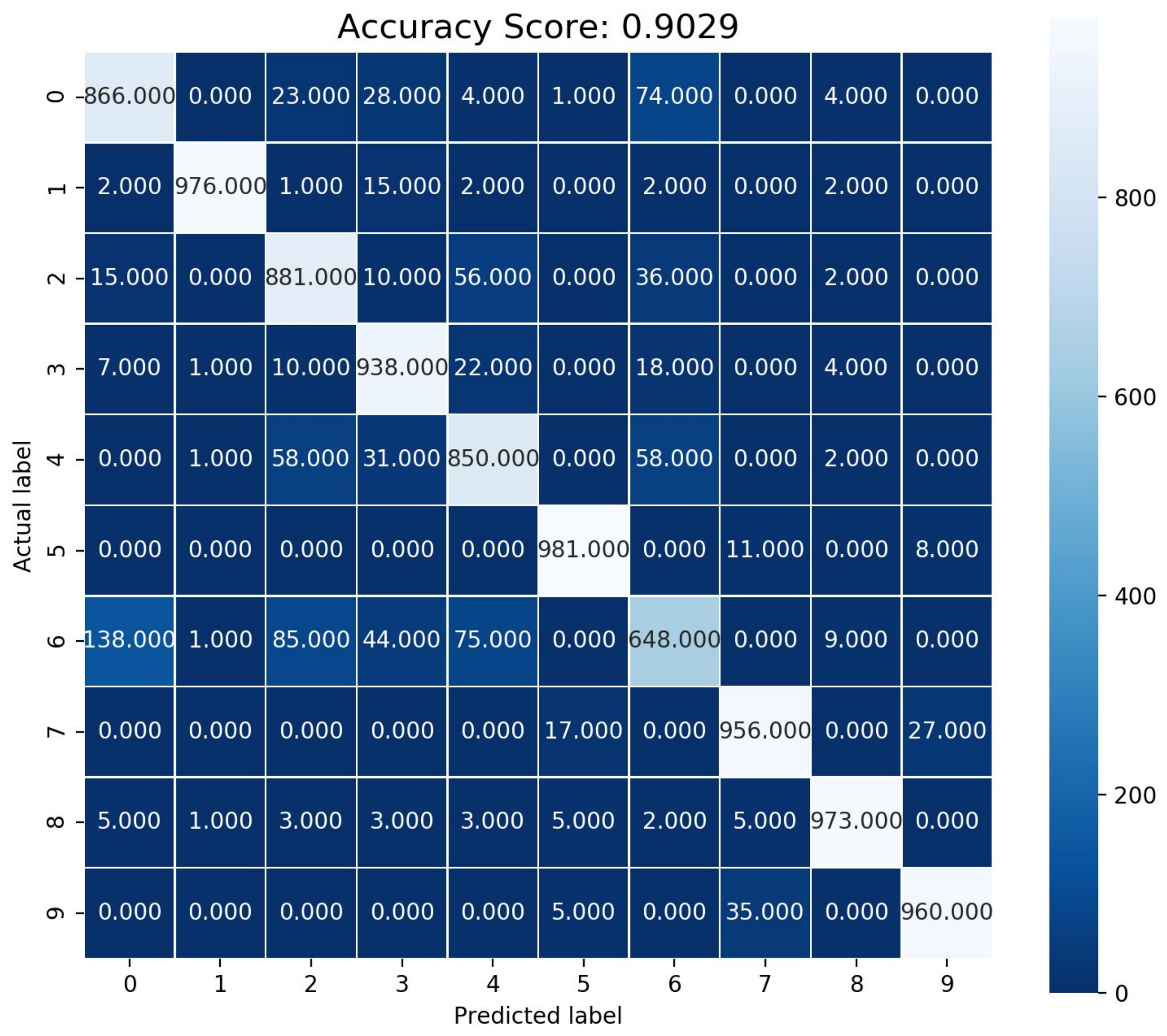

Figure 1. Confusion matrix with the accuracy shown.

Comment: The above figure shows the confusion matrix from the model with an accuracy rating of roughly 90%. The matrix displays some interesting findings of the network especially with classifying the number '0' and '6' as this is what the network really struggled with.

Task 2

The average Peak Signal-to-Noise Ratio (PSNR) values for different P values are listed in Table 1. The PSNR values are shown to increase as the value of P increases.

Table 1. P values and PSNR values for the test set.

P	PSNR
10	18.57
50	21.25
200	24.48

The first 10 test images and their decompressed images with P = 10, 50, and 200 are shown in Figure 2. The first row shows the original 10 test images and rows 2, 3, and 4 show their decompressed images with P =10, 50, and 200 respectively. From the images, it can be shown from the quality that the higher P values preserve a lot of the details within the pictures. This can especially be shown with the jacket (5th image from the left) and the flannel (third image from the right). Within the same P-value, we think the images that are most difficult to compress are those with varying patterns.

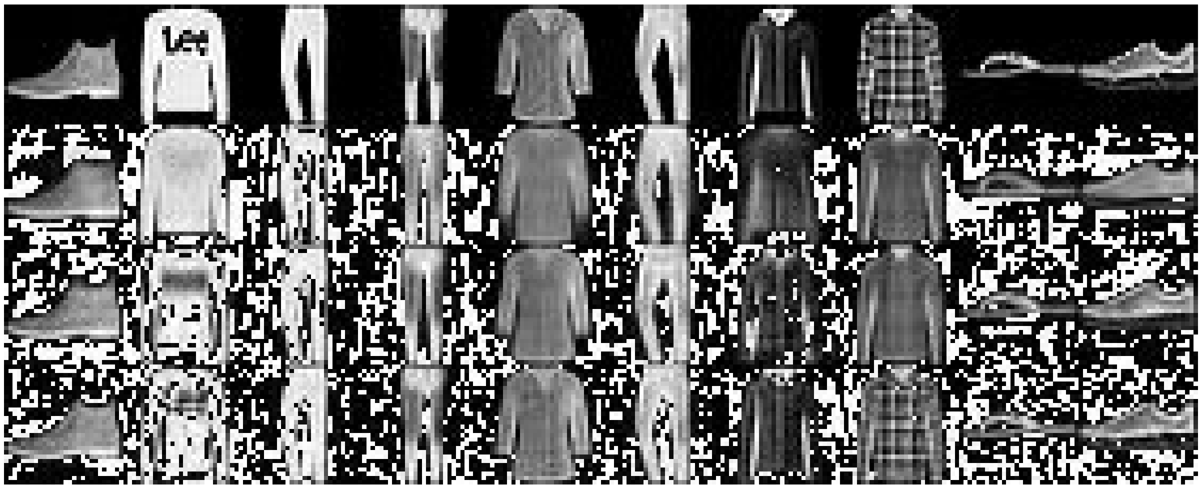

Figure 2. 10 test images and their decompressed images with P = 10, 50, and 200.

Task 3

I. Introduction

The objective of this task is to build a color video compression system to achieve better reconstruction quality at the same compression ratio and to study the trade-off between reconstruction quality, computational complexity, and model size using a convolutional neural network and neural networks. The reconstruction quality is evaluated using the Peak Signal-to-Noise Ratio (PSNR) values and visual qualities.

II. Proposed Methods

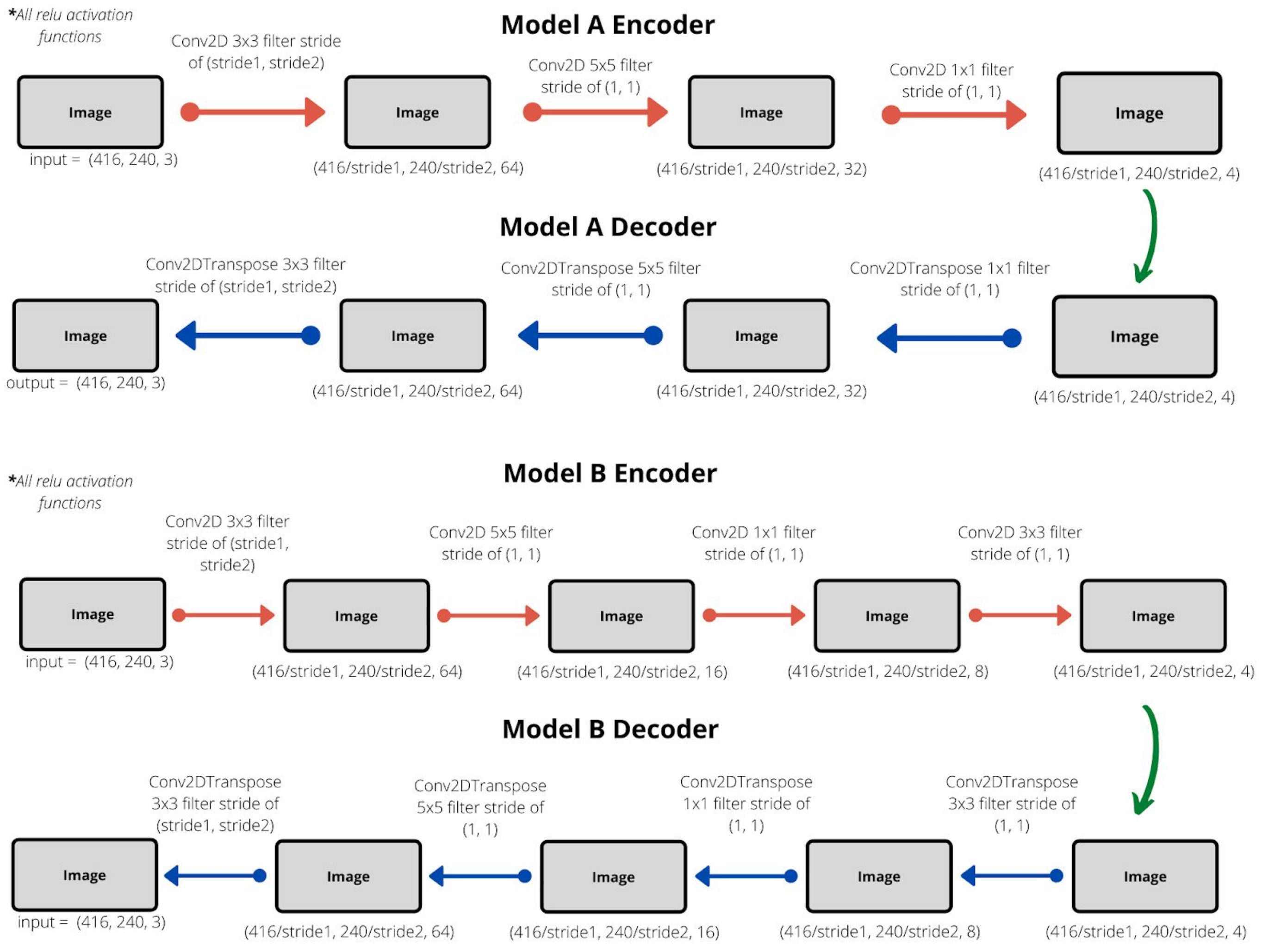
*All relu activation functions
Model A Encoder
Conv2D 3x3 filter stride of (stride1, stride2)
Conv2D 5x5 filter stride of (1, 1)
Conv2D 1x1 filter stride of (1, 1)
Image
input = (416, 240, 3)
(416/stride1, 240/stride2, 64)
(416/stride1, 240/stride2, 32)
(416/stride1, 240/stride2, 4)
Model A Decoder
Conv2DTranspose 3x3 filter stride of (stride1, stride2)
Conv2DTranspose 5x5 filter stride of (1, 1)
Conv2DTranspose 1x1 filter stride of (1, 1)
output = (416, 240, 3)
(416/stride1, 240/stride2, 64)
(416/stride1, 240/stride2, 32)
(416/stride1, 240/stride2, 4)
*All relu activation functions
Model B Encoder
Conv2D 3x3 filter stride of (stride1, stride2)
Conv2D 5x5 filter stride of (1, 1)
Conv2D 1x1 filter stride of (1, 1)
Conv2D 3x3 filter stride of (1, 1)
input = (416, 240, 3)
(416/stride1, 240/stride2, 64)
(416/stride1, 240/stride2, 16)
(416/stride1, 240/stride2, 8)
(416/stride1, 240/stride2, 4)
Model B Decoder
Conv2DTranspose 3x3 filter stride of (stride1, stride2)
Conv2DTranspose 5x5 filter stride of (1, 1)
Conv2DTranspose 1x1 filter stride of (1, 1)
Conv2DTranspose 3x3 filter stride of (1, 1)
input = (416, 240, 3)
(416/stride1, 240/stride2, 64)
(416/stride1, 240/stride2, 16)
(416/stride1, 240/stride2, 8)
(416/stride1, 240/stride2, 4)

Figure 3. Model A vs Model B network architecture

In Figure 3 above, the network architectures we used for the following task are explained in detail. The network structures are both convolution neural networks. Both models have values ‘stride1’ and ‘stride2’ which in combination were used to obtain the compression values necessary for this task. Model A uses three layers to encode the image. The first is a 3x3 filter with a stride to compress the image. Then two layers of 5x5 filtering and 1x1 filtering respectively are added. Dimensionality wise, the layers decrease from a depth of 64 to 32 to 4. The decoder of Model A is the exact opposite of these layers but using a 2D transpose layer. Model A and Model B are very similar however the slight difference between them is the fact that Model B adds an additional layer of filtering. After the same filtering done in Model A is applied, an additional 3x3 filter is used. With this added layer, the dimensionality of the layers also changes from 64 to 16 to 8 to 4. These networks were chosen to see the differences between how a slightly more complex neural network will do against a more simpler approach and whether a more complex network yields better results. These networks do framewise compression, use the ‘relu’ activation function, use the ‘mse’ loss function, and were trained on a training set size of a hundred images with twenty epochs.

III. Experimental Studies

Dataset Description:
The datasets used are the images of the horses, basketball players, and bubbles. Each model was trained on 100 training images and a test set image size of 25. For the basketball player images, the images were resized to the input shape of 416 by 240.

Quantitative Evaluation:
The PSNR curves of the two different network models are plotted against the compression ratios in Figures 4-6 for the blowing bubbles, racehorses, and basketball images respectively. The results show an increasing linear correlation with PSNR and compression ratio.

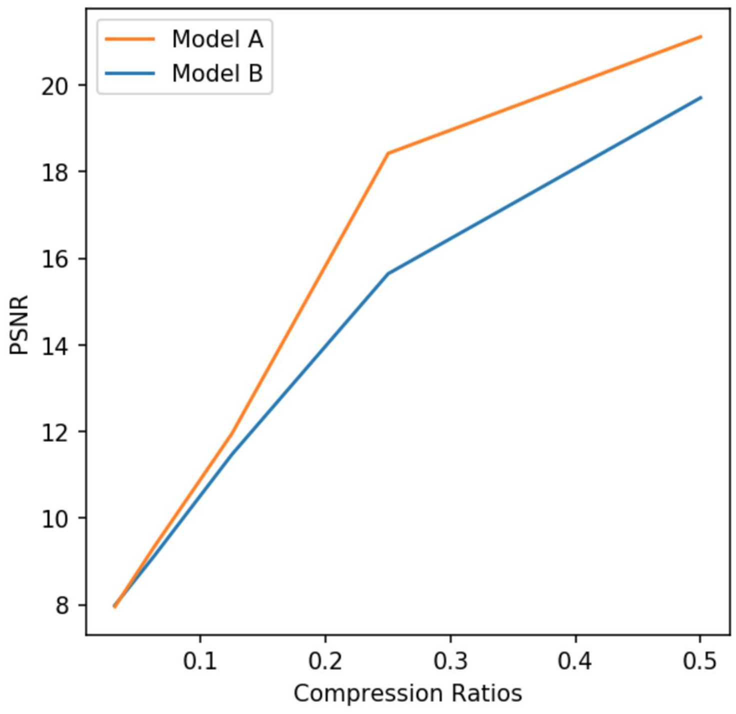

Figure 4. PSNR vs. compression ratios for blowing bubbles images.

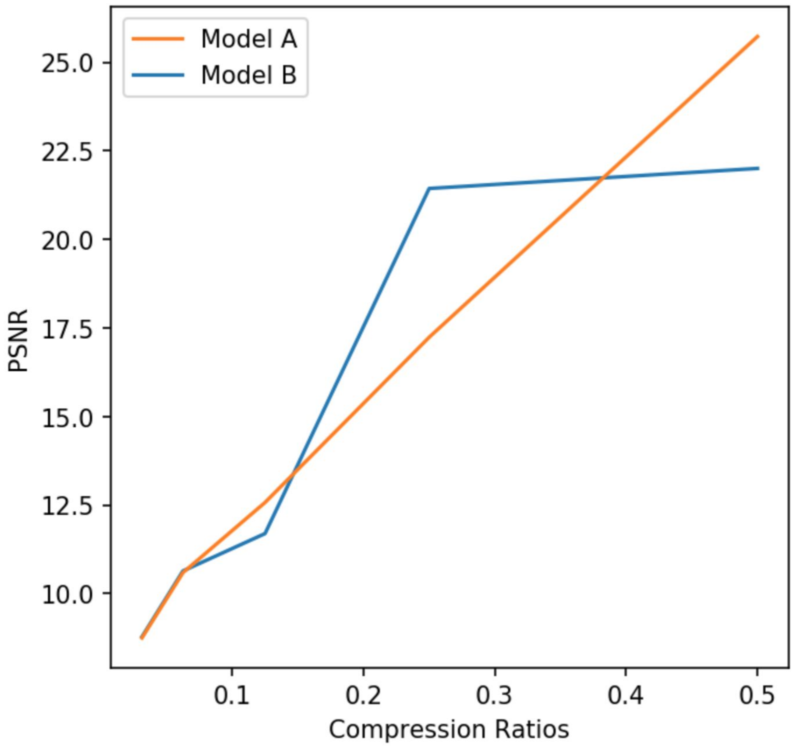

Figure 5. PSNR vs. compression ratios for racehorses images.

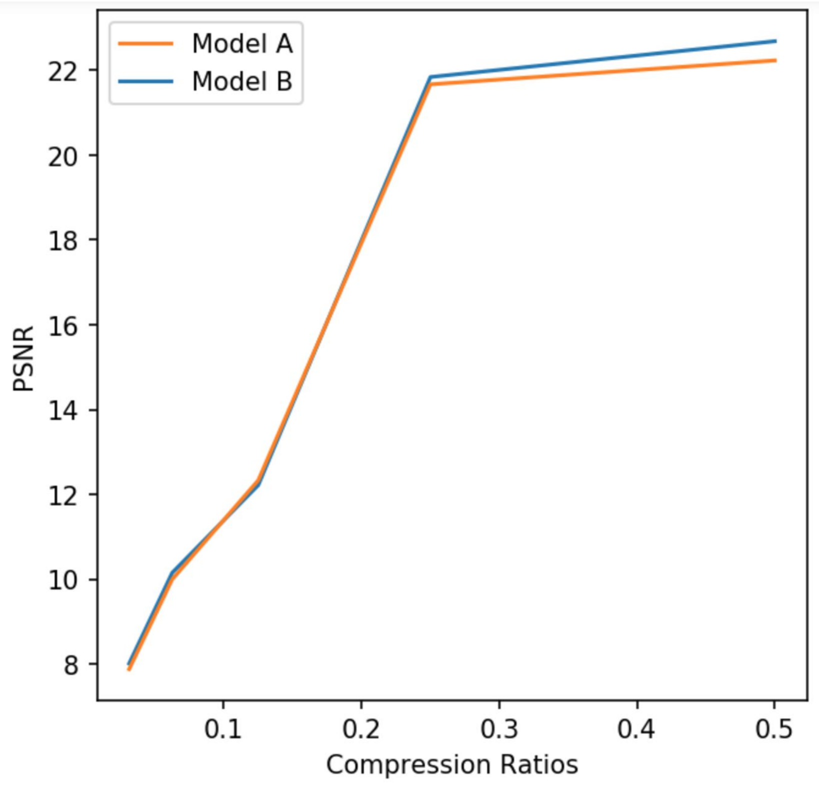

Figure 6. PSNR vs. compression ratio for basketball images.

Perceptual Quality Evaluation:
For each compression ratio, two original frames from each category of images were reconstructed using the two different networks as shown in Figures 7-9. The PSNR values of the reconstructed frames are labeled on the images. The image quality increases with PSNR. Images reconstructed with lower compression ratios resulted in a lower PSNR and lower image quality. For the blowing bubbles images, model A always resulted in higher image quality. For some compression ratios for the racehorse images, model B resulted in better image quality and higher PSNR, but for other compression ratios, model A resulted in better image quality. For the basketball images, the resulting image quality appears to be very similar between the two models.

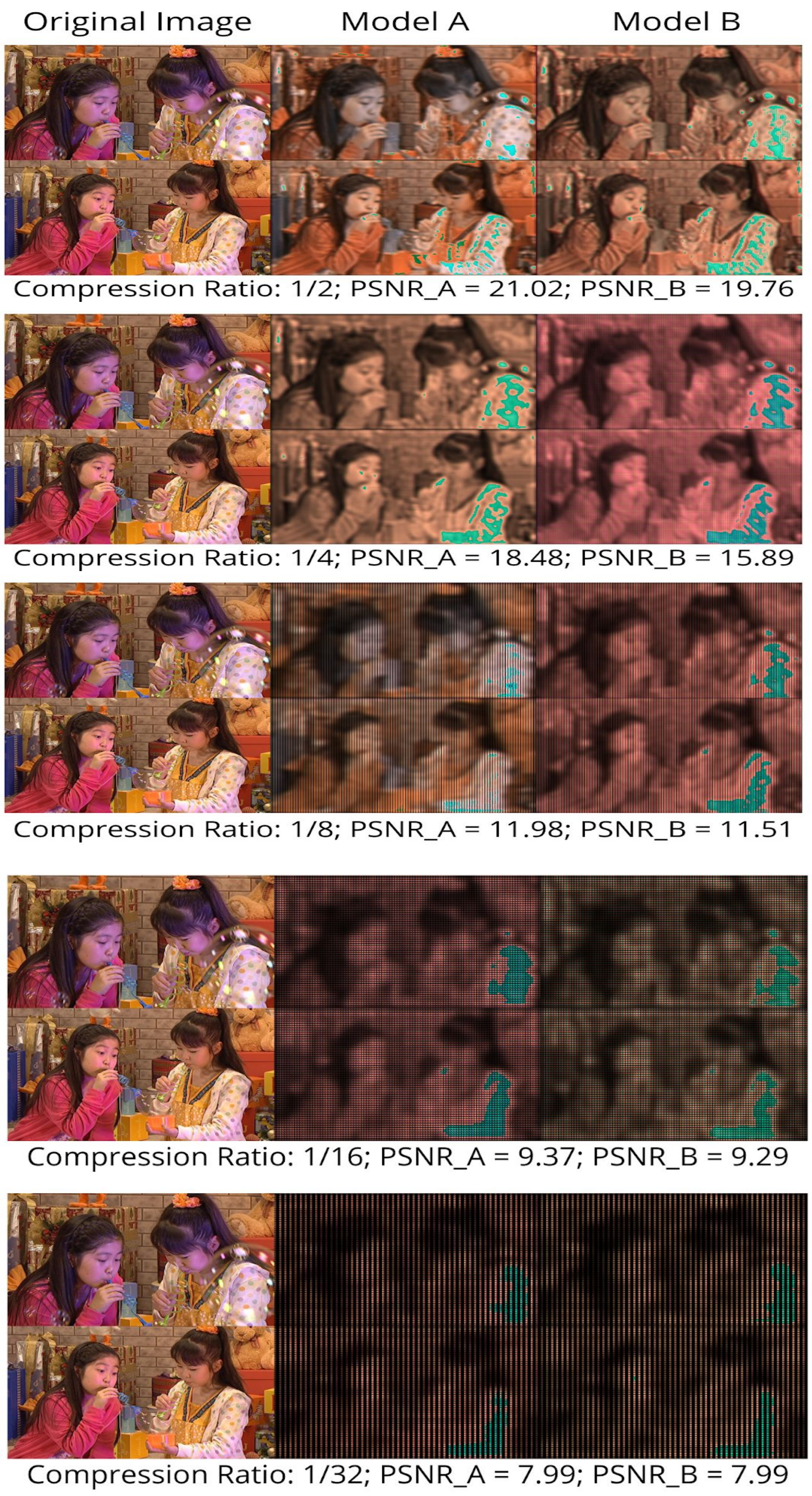
Original Image
Model A
Model B
Compression Ratio: 1/2; PSNR_A = 21.02; PSNR_B = 19.76
Compression Ratio: 1/4; PSNR_A = 18.48; PSNR_B = 15.89
Compression Ratio: 1/8; PSNR_A = 11.98; PSNR_B = 11.51
Compression Ratio: 1/16; PSNR_A = 9.37; PSNR_B = 9.29
Compression Ratio: 1/32; PSNR_A = 7.99; PSNR_B = 7.99

Figure 7. Original and reconstructed blowing bubbles images using the two methods.

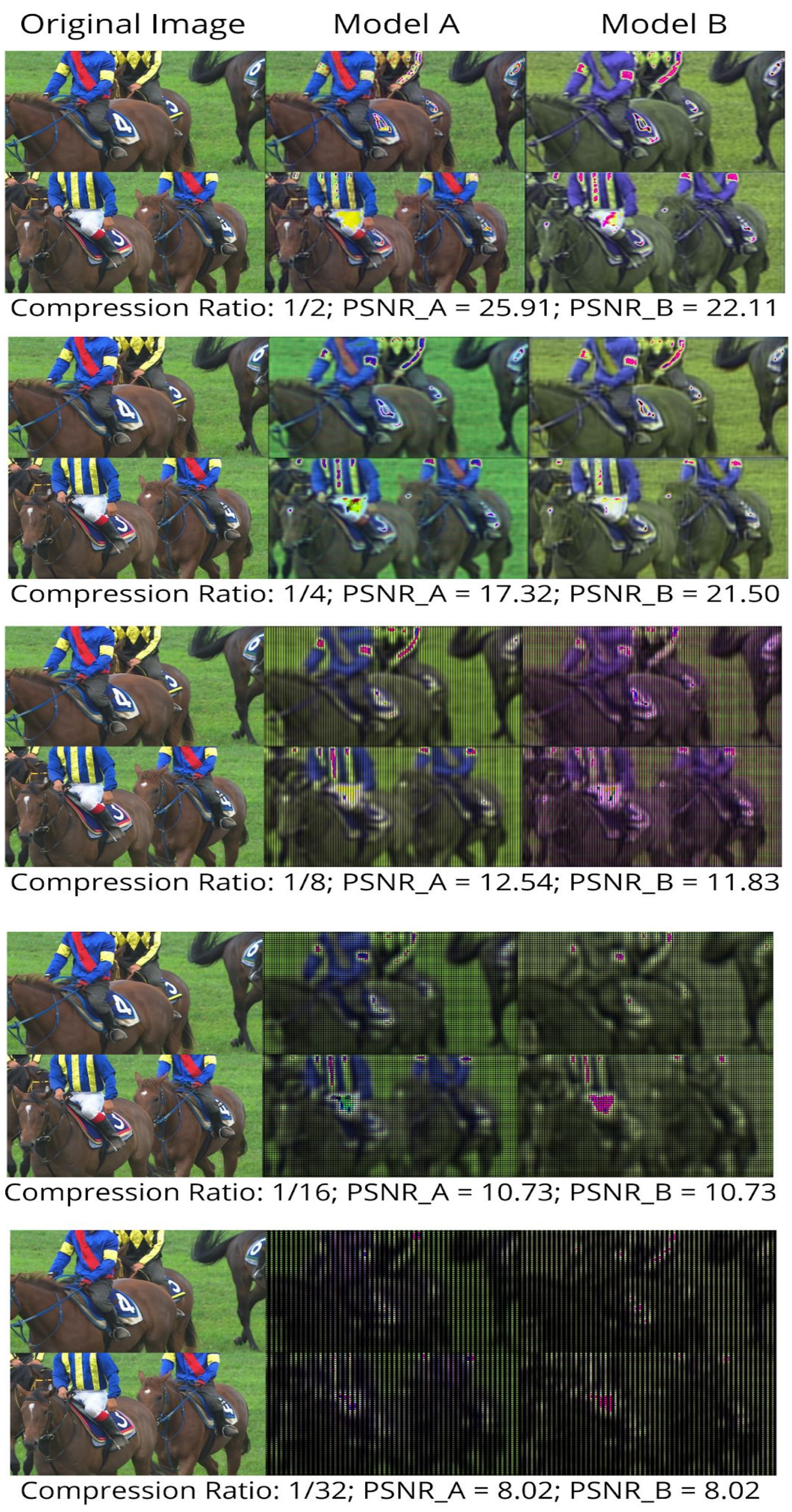
Original Image
Model A
Model B
Compression Ratio: 1/2; PSNR_A = 25.91; PSNR_B = 22.11
Compression Ratio: 1/4; PSNR_A = 17.32; PSNR_B = 21.50
Compression Ratio: 1/8; PSNR_A = 12.54; PSNR_B = 11.83
Compression Ratio: 1/16; PSNR_A = 10.73; PSNR_B = 10.73
Compression Ratio: 1/32; PSNR_A = 8.02; PSNR_B = 8.02

Figure 8. Original and reconstructed racehorses images using the two methods.

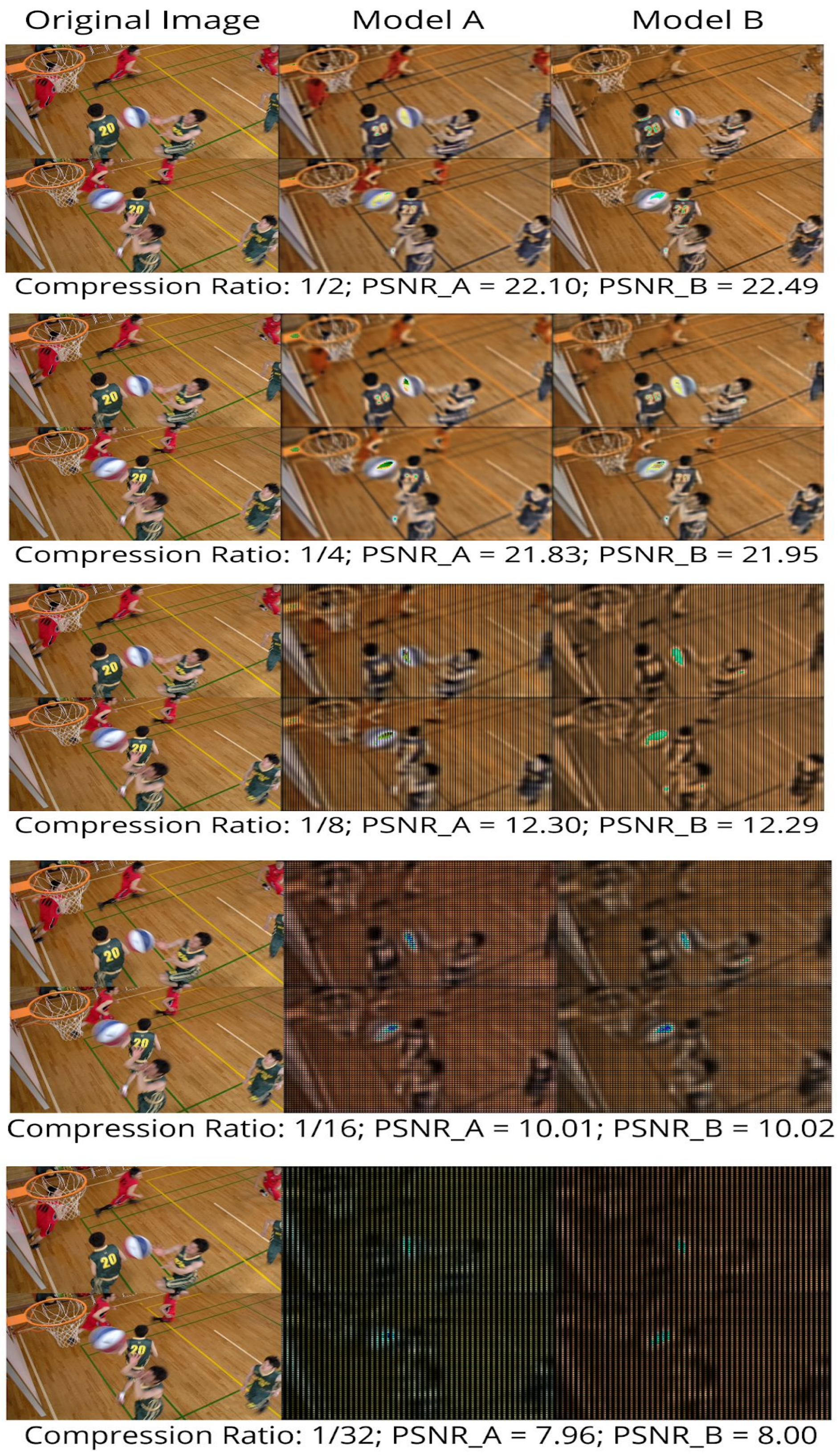
Original Image
Model A
Model B
Compression Ratio: 1/2; PSNR_A = 22.10; PSNR_B = 22.49
Compression Ratio: 1/4; PSNR_A = 21.83; PSNR_B = 21.95
Compression Ratio: 1/8; PSNR_A = 12.30; PSNR_B = 12.29
Compression Ratio: 1/16; PSNR_A = 10.01; PSNR_B = 10.02
Compression Ratio: 1/32; PSNR_A = 7.96; PSNR_B = 8.00

Figure 9. Original and reconstructed basketball images using the two methods.

Complexity and Model Size Analysis:

Table 2. Parameters and Computational Complexity Comparison between Model's A and B

	Model A	Model B
Number of Parameters	106,112	55,488
Computational Complexity	Forward/Inference	Forward/Inference

The general form of the number of parameters for each model due to each layer being a convolutional layer was as follows:

$$\sum_{n=1}^{N} [(filter\ size) \times input\ dimensionality] \times \#\ of\ filters$$

For instance, the first layer of Model A had a 3x3 filter with an input dimension of 3 and 64 filters leading to the number of parameters in that layer to be 1,728. After calculating this number for each layer of the encoder and decoder, we added them all together to get the final results displayed in Table 2. Through calculating the number of parameters, it is shown that even by adding a full layer to Model B, Model A still has more parameters simply due to its architecture. This may be why Model A out-performed model B in certain image compression cases.

IV. Conclusions and Future Work

Overall, it is clear that a more complex network does not necessarily produce higher quality compression. In fact, for some of the images, the simpler Model A outperformed the more complex Model B. For the future, it would be interesting to implement different layers such as max-pooling and maybe even some dense layers to see how adding such layers provides better or worse output.

V. References

None.

Contribution of Team Members

- Contribution of team members (in order 1, 2, 3 or in percentage)
  PJ McCurdy - 50%
  Alex Cherekdjian - 50%

Attachments: (Attached in zip)

Problem 1 code:

```
import tensorflow as tf
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras import datasets, layers, models
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score

# grab a reference to the dataset
fashion_mnist = datasets.fashion_mnist

# load the data set
(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()

# flatten the images
train_images = train_images.reshape((60000, 28, 28, 1))
test_images = test_images.reshape((10000, 28, 28, 1))

# normalize pixel values to be between 0 and 1
train_images, test_images = train_images / 255.0, test_images / 255.0

# create model instance
model = models.Sequential()

# add layers
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPool2D(pool_size=(2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPool2D(pool_size=(2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))

# compile the model
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

```
# fit the model
model.fit(train_images, train_labels, epochs=5)

# calculate predictions
predictions_net = model.predict(test_images)

# finding values from prob percentages of each category
prediction = np.argmax(predictions_net, axis=1)

# create confusion matrix and accuracy score
predictions_cm = confusion_matrix(test_labels, prediction)
recognition_accuracy_rate_net = accuracy_score(test_labels, prediction)

# plot using seaborn to make it look nice
plt.figure(figsize=(9,9))
sns.heatmap(predictions_cm, annot=True, fmt=".3f", linewidths=.5, square = True, cmap =
'Blues_r');
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
all_sample_title = 'Accuracy Score: {0}'.format(recognition_accuracy_rate_net)
plt.title(all_sample_title, size = 15)
plt.show()
```

Problem 2 code:

```
import tensorflow as tf
import numpy as np
import math
from PIL import Image
from tensorflow.python import keras
from tensorflow.python.keras import datasets, layers, models
from tensorflow.python.keras.models import Sequential
from tensorflow.python.keras.layers import Input, Dense, Flatten, Reshape

# p values, image and psnr list
P_values = [10,50,200]
psnr = []
im = Image.new('RGB', (280, 112))

T = 2; # expansion factor
```

```
# image resolution (pixels)
m = 28; # rows
n = 28; # columns

# get data for fashion mnist
fashion_mnist = datasets.fashion_mnist

# split into training and testing
(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()

# Normalize pixel values to be between 0 and 1
train_images, test_images = train_images / 255.0, test_images / 255.0

# print first set of ten test images
test_imgs = np.concatenate((test_images[0:10]), axis = 1)
image = Image.fromarray((test_imgs*255.0).astype('uint8'))
im.paste(image, (0, 0))

# go through all p values
for i in range(0, len(P_values)):
    # Parameters
    P = P_values[i]; # P < m * n; 10, 50, 200

    # define the keras model
    model = Sequential()
    # A flattened input layer with m x n nodes
    model.add(Flatten(input_shape=(m, n)))
    # A compressed layer with P nodes (no activation function)
    model.add(Dense(P))
    # An expansion layer with m x n x T nodes, followed by ReLU activation
    model.add(Dense(m*n*T, activation='relu'))
    # An output layer with m x n nodes (no activation function)
    model.add(Dense(m*n))
    # A reshape layer that convert the 1-D vector output to the m x n 2-dimensional image
    model.add(Reshape((m, n)))

    # compile the model
    model.compile(loss='mean_squared_error', optimizer='adam')
    # fit the keras model on the dataset
    model.fit(train_images, train_images, epochs=10, batch_size=64)

    # evaluate the model and get mse
    mse = model.evaluate(test_images, test_images)

    # calculate psnr
    psnr.append(10*math.log10(1/mse))

    # append predictions to final image
    pred_image = model.predict(test_images)
    test_imgs = np.concatenate((pred_image[0:10]), axis = 1)
    image = Image.fromarray((test_imgs*255.0).astype('uint8'))
    im.paste(image, (0, 28 + i * 28))

# print p vs psnr values
print("P_values " + str(P_values))
print("PSNR " + str(psnr))

# save image
im.save('problem2.png')
print("png saved.")
```

Problem 3 code:

```
import math
import os
import random
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
from PIL import Image
from sklearn.model_selection import train_test_split
from tensorflow.keras import datasets, layers, models
%matplotlib inline

# class definitions for two models
class CNN_A:
    def __init__(self, stride1, stride2):
        self.model = models.Sequential()
        self.model.add(layers.Conv2D(64, kernel_size=(3, 3), padding = "SAME", strides =
(stride1, stride2), activation='relu', input_shape=(240, 416, 3)))
        self.model.add(layers.Conv2D(32, kernel_size=(5, 5), padding = "SAME", strides = (1,1),
activation='relu'))
        self.model.add(layers.Conv2D(4, kernel_size=(1, 1), padding = "SAME", strides = (1,1),
activation='relu'))

        self.model.add(layers.Conv2DTranspose(32, kernel_size=(1, 1), padding = "SAME", strides
= (1,1), activation='relu'))
        self.model.add(layers.Conv2DTranspose(64, kernel_size=(5, 5), padding = "SAME", strides
= (1,1), activation='relu'))
        self.model.add(layers.Conv2DTranspose(3, kernel_size=(3, 3), padding = "SAME", strides
= (stride1, stride2), activation='relu'))

class CNN_B:
    def __init__(self, stride1, stride2):
        self.model = models.Sequential()
        self.model.add(layers.Conv2D(64, kernel_size = (3, 3), padding = "SAME", strides =
(stride1, stride2), activation = 'relu', input_shape = (240, 416, 3)))
        self.model.add(layers.Conv2D(16, kernel_size = (5, 5), padding = "SAME", strides = (1, 1),
activation = 'relu'))
        self.model.add(layers.Conv2D(8, kernel_size = (1, 1), padding = "SAME", strides = (1, 1),
activation = 'relu'))

        self.model.add(layers.Conv2D(4, kernel_size = (3, 3), padding = "SAME", strides = (1, 1),
activation = 'relu'))

        self.model.add(layers.Conv2DTranspose(8, kernel_size = (3, 3), padding = "SAME", strides
= (1, 1), activation = 'relu'))
        self.model.add(layers.Conv2DTranspose(16, kernel_size = (1, 1), padding = "SAME",
strides = (1, 1), activation = 'relu'))
        self.model.add(layers.Conv2DTranspose(64, kernel_size = (5, 5), padding = "SAME",
strides = (1, 1), activation = 'relu'))
        self.model.add(layers.Conv2DTranspose(3, kernel_size = (3, 3), padding = "SAME", strides
= (stride1, stride2), activation = 'relu'))

# pathname of files, change pathname manually for different picture sets
pathname = './BasketballDrill_832x480_50'
```

```
x_samples = []

# read in all files and resize images if needed
for im in os.listdir(pathname):
    img = Image.open(pathname + "/" + str(im)).convert('RGB')
    img = img.resize((416, 240))

    np_img = np.array(img)
    x_samples.append(np_img)

# create np array with all samples and create test train split
x_samples = np.array(x_samples)
x_train, x_test = train_test_split(x_samples, test_size=0.2)


x_train = x_train[:int(x_train.shape[0]/4)]
x_test = x_test[:int(x_train.shape[0]/4)]

# initializing compression ratio lists
compression_ratios = [1/2, 1/4, 1/8, 1/16, 1/32]

# initializing different A models and psnr list
model_A_psnr = []
model_1_2_A = CNN_A(1,2)
model_1_4_A = CNN_A(2,2)
model_1_8_A = CNN_A(2,4)
model_1_16_A = CNN_A(4,4)
model_1_32_A = CNN_A(4,8)

# initializing different B models and psnr list
model_B_psnr = []
model_1_2_B = CNN_B(1,2)
model_1_4_B = CNN_B(2,2)
model_1_8_B = CNN_B(2,4)
model_1_16_B = CNN_B(4,4)
model_1_32_B = CNN_B(4,8)

# using 5 epochs to train
EPOCHS = 20
```

```
# train and evaluate model A
model_1_2_A.model.compile(optimizer='adam', loss='mse')
model_1_2_A.model.fit(x_train, x_train, epochs=EPOCHS)
mse = model_1_2_A.model.evaluate(x_test, x_test)
model_A_psnr.append(10*math.log10((255**2)/mse))

model_1_4_A.model.compile(optimizer='adam', loss='mse')
model_1_4_A.model.fit(x_train, x_train, epochs=EPOCHS)
mse = model_1_4_A.model.evaluate(x_test, x_test)
model_A_psnr.append(10*math.log10((255**2)/mse))

model_1_8_A.model.compile(optimizer='adam', loss='mse')
model_1_8_A.model.fit(x_train, x_train, epochs=EPOCHS)
mse = model_1_8_A.model.evaluate(x_test, x_test)
model_A_psnr.append(10*math.log10((255**2)/mse))

model_1_16_A.model.compile(optimizer='adam', loss='mse')
model_1_16_A.model.fit(x_train, x_train, epochs=EPOCHS)
mse = model_1_16_A.model.evaluate(x_test, x_test)
model_A_psnr.append(10*math.log10((255**2)/mse))

model_1_32_A.model.compile(optimizer='adam', loss='mse')
model_1_32_A.model.fit(x_train, x_train, epochs=EPOCHS)
mse = model_1_32_A.model.evaluate(x_test, x_test)
model_A_psnr.append(10*math.log10((255**2)/mse))

# train and evaluate model B
model_1_2_B.model.compile(optimizer='adam', loss='mse')
model_1_2_B.model.fit(x_train, x_train, epochs=EPOCHS)
mse = model_1_2_B.model.evaluate(x_test, x_test)
model_B_psnr.append(10*math.log10((255**2)/mse))

model_1_4_B.model.compile(optimizer='adam', loss='mse')
model_1_4_B.model.fit(x_train, x_train, epochs=EPOCHS)
mse = model_1_4_B.model.evaluate(x_test, x_test)
model_B_psnr.append(10*math.log10((255**2)/mse))

model_1_8_B.model.compile(optimizer='adam', loss='mse')
```

```
model_1_8_B.model.fit(x_train, x_train, epochs=EPOCHS)
mse = model_1_8_B.model.evaluate(x_test, x_test)
model_B_psnr.append(10*math.log10((255**2)/mse))

model_1_16_B.model.compile(optimizer='adam', loss='mse')
model_1_16_B.model.fit(x_train, x_train, epochs=EPOCHS)
mse = model_1_16_B.model.evaluate(x_test, x_test)
model_B_psnr.append(10*math.log10((255**2)/mse))

model_1_32_B.model.compile(optimizer='adam', loss='mse')
model_1_32_B.model.fit(x_train, x_train, epochs=EPOCHS)
mse = model_1_32_B.model.evaluate(x_test, x_test)
model_B_psnr.append(10*math.log10((255**2)/mse))

# create images
im_1_2 = Image.new('RGB', (1248, 480))
im_1_4 = Image.new('RGB', (1248, 480))
im_1_8 = Image.new('RGB', (1248, 480))
im_1_16 = Image.new('RGB', (1248, 480))
im_1_32 = Image.new('RGB', (1248, 480))

# 2 frames of each
for i in range(0, 2):
    # predict an image from each 1/2 compression rate model
    pred_image_A = model_1_2_A.model.predict(x_test[i].reshape(1,240, 416, 3))
    pred_image_B = model_1_2_B.model.predict(x_test[i].reshape(1,240, 416, 3))
    pred_image_A = pred_image_A.reshape(240, 416, 3)
    pred_image_B = pred_image_B.reshape(240, 416, 3)

    # print predictions to image to display quality difference
    original = Image.fromarray(x_test[i].astype('uint8'))
    image_A = Image.fromarray(pred_image_A.astype('uint8'))
    image_B = Image.fromarray(pred_image_B.astype('uint8'))

    im_1_2.paste(original, (0, 0 + i*240))
    im_1_2.paste(image_A, (416, 0 + i*240))
    im_1_2.paste(image_B, (832, 0 + i*240))

for i in range(0, 2):
    # predict an image from each 1/4 compression rate model
    pred_image_A = model_1_4_A.model.predict(x_test[i].reshape(1,240, 416, 3))
    pred_image_B = model_1_4_B.model.predict(x_test[i].reshape(1,240, 416, 3))
    pred_image_A = pred_image_A.reshape(240, 416, 3)
    pred_image_B = pred_image_B.reshape(240, 416, 3)

    # print predictions to image to display quality difference
    original = Image.fromarray(x_test[i].astype('uint8'))
    image_A = Image.fromarray(pred_image_A.astype('uint8'))
    image_B = Image.fromarray(pred_image_B.astype('uint8'))

    im_1_4.paste(original, (0, 0 + i*240))
    im_1_4.paste(image_A, (416, 0 + i*240))
    im_1_4.paste(image_B, (832, 0 + i*240))

for i in range(0, 2):
    # predict an image from each 1/8 compression rate model
    pred_image_A = model_1_8_A.model.predict(x_test[i].reshape(1,240, 416, 3))
    pred_image_B = model_1_8_B.model.predict(x_test[i].reshape(1,240, 416, 3))
    pred_image_A = pred_image_A.reshape(240, 416, 3)
    pred_image_B = pred_image_B.reshape(240, 416, 3)

    # print predictions to image to display quality difference
    original = Image.fromarray(x_test[i].astype('uint8'))
    image_A = Image.fromarray(pred_image_A.astype('uint8'))
    image_B = Image.fromarray(pred_image_B.astype('uint8'))

    im_1_8.paste(original, (0, 0 + i*240))
    im_1_8.paste(image_A, (416, 0 + i*240))
    im_1_8.paste(image_B, (832, 0 + i*240))

for i in range(0, 2):
    # predict an image from each 1/16 compression rate model
    pred_image_A = model_1_16_A.model.predict(x_test[i].reshape(1,240, 416, 3))
    pred_image_B = model_1_16_B.model.predict(x_test[i].reshape(1,240, 416, 3))
    pred_image_A = pred_image_A.reshape(240, 416, 3)
    pred_image_B = pred_image_B.reshape(240, 416, 3)

    # print predictions to image to display quality difference
    original = Image.fromarray(x_test[i].astype('uint8'))
    image_A = Image.fromarray(pred_image_A.astype('uint8'))
    image_B = Image.fromarray(pred_image_B.astype('uint8'))

    im_1_16.paste(original, (0, 0 + i*240))
    im_1_16.paste(image_A, (416, 0 + i*240))
    im_1_16.paste(image_B, (832, 0 + i*240))

for i in range(0, 2):
    # predict an image from each 1/32 compression rate model
    pred_image_A = model_1_32_A.model.predict(x_test[i].reshape(1,240, 416, 3))
    pred_image_B = model_1_32_B.model.predict(x_test[i].reshape(1,240, 416, 3))
    pred_image_A = pred_image_A.reshape(240, 416, 3)
    pred_image_B = pred_image_B.reshape(240, 416, 3)

    # print predictions to image to display quality difference
    original = Image.fromarray(x_test[i].astype('uint8'))
    image_A = Image.fromarray(pred_image_A.astype('uint8'))
    image_B = Image.fromarray(pred_image_B.astype('uint8'))

    im_1_32.paste(original, (0, 0 + i*240))
    im_1_32.paste(image_A, (416, 0 + i*240))
    im_1_32.paste(image_B, (832, 0 + i*240))

# save images
im_1_2.save('1_2_reconstruct.png')
print("1/2 png saved.")
im_1_4.save('1_4_reconstruct.png')
print("1/4 png saved.")
im_1_8.save('1_8_reconstruct.png')
print("1/8 png saved.")
im_1_16.save('1_16_reconstruct.png')
print("1/16 png saved.")
im_1_32.save('1_32_reconstruct.png')
print("1/32 png saved.")

# plot PSNR vs Compression Ratios
plt.figure(num=None, figsize=(5, 5), dpi=150, facecolor='w', edgecolor='k')
plot_B, = plt.plot(compression_ratios, model_B_psnr)
```

```
plot_A, = plt.plot(compression_ratios, model_A_psnr)
plt.legend([plot_A, plot_B], ['Model A', 'Model B'])
plt.xlabel('Compression Ratios')
plt.ylabel('PSNR')
plt.show()
```