

Term Project Report  
COEN 240: Machine Learning

Michael Dallow

Daniel Okazaki

Stephen Pacwa

## Task 1: Classification of MNIST Dataset.

### Methodology:

We built a Convolutional Neural Network (CNN) for the MNIST dataset that classifies an image as a digit zero through nine. The model includes: an input layer, a 2D-convolutional layer with filter size 3x3 and depth 32 using a ReLU activation function, a 2x2 max pooling layer, a 2D-convolutional layer with filter size 3x3 and depth 64 using a ReLU activation function, a 2x2 max pooling layer, a 2D-convolutional layer with filter size 3x3 and depth 64 with a ReLU activation function, a fully-connected layer of 64 units and using a ReLU activation function, and the output layer, which is a fully-connected layer of 10 units using the softmax activation function. The recognition accuracy was 98.4% showing that the accuracy of this model is extremely good.

### Results

Shown below are the results of this experiment, including the recognition accuracy and the confusion matrix (Figure 1). Our final recognition accuracy rate was approximately 0.984. From the confusion matrix below, we can see that our network was the most confused when attempting to classify the digit 5.

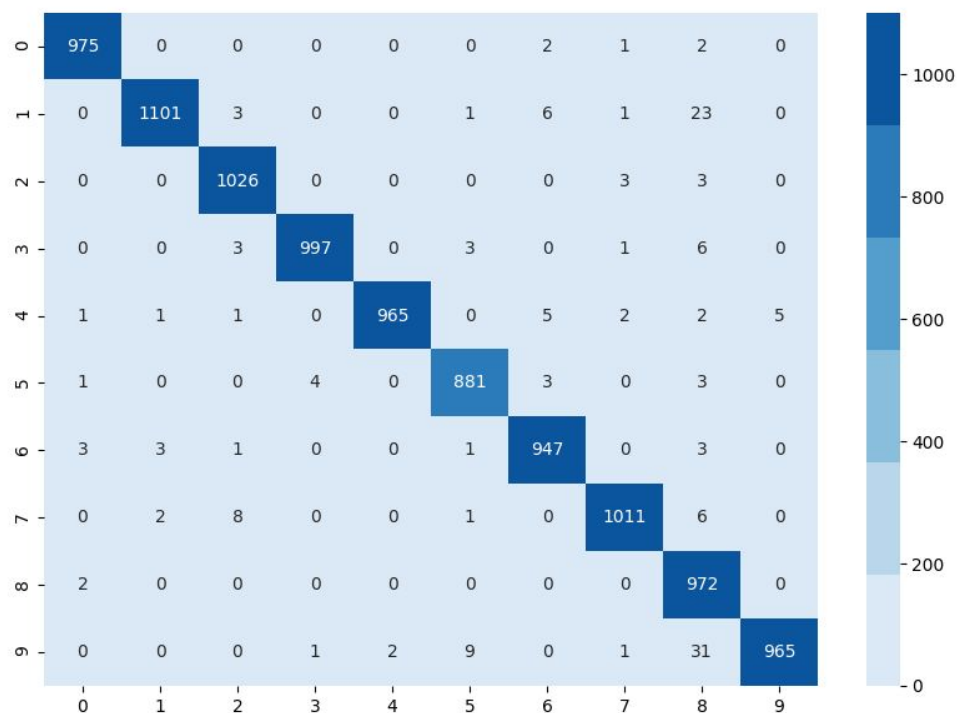


Figure 1: Confusion Matrix for MNIST Classification.

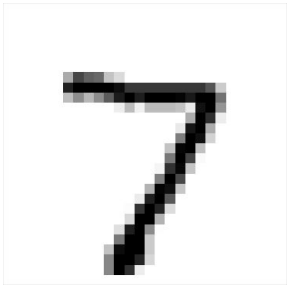
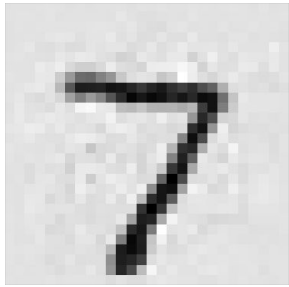
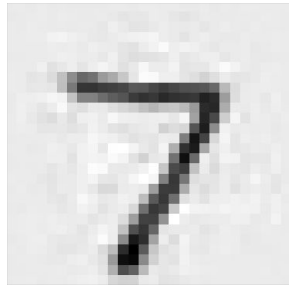
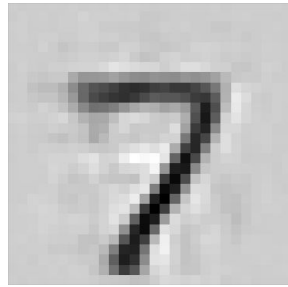
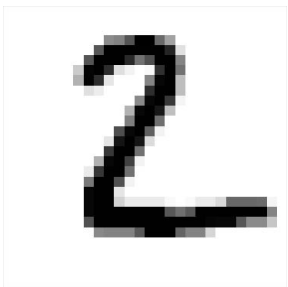
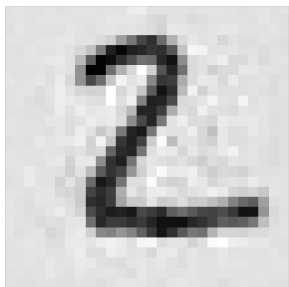
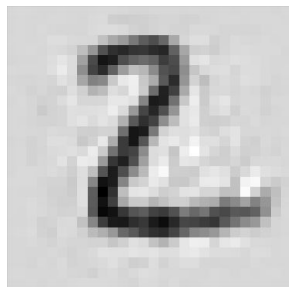
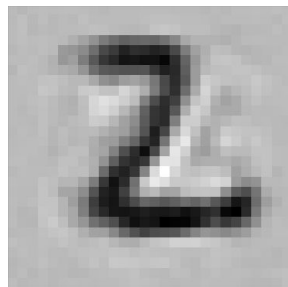
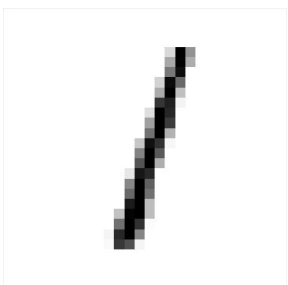
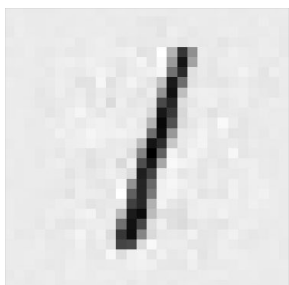
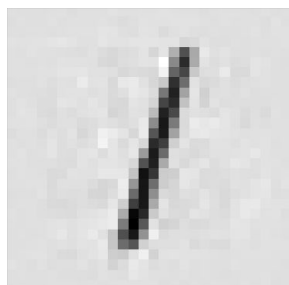
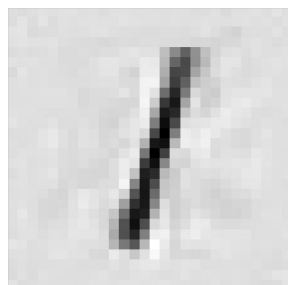
## Task 2: Neural Network Compression of MNIST Dataset.

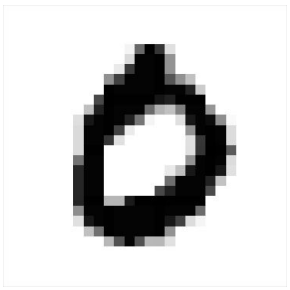
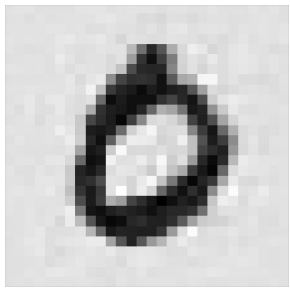
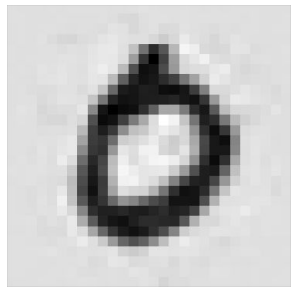
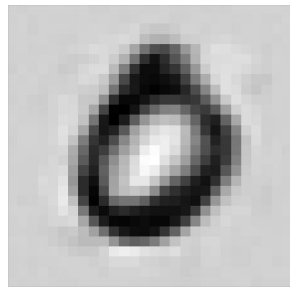
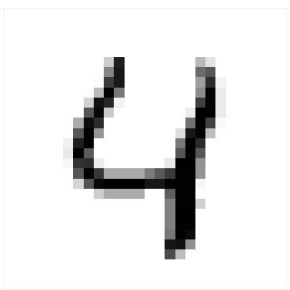
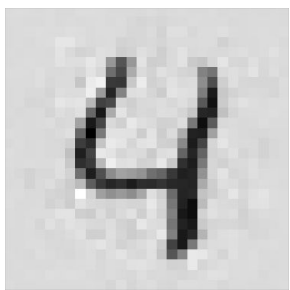
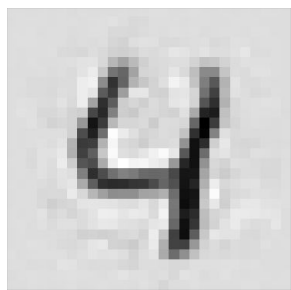
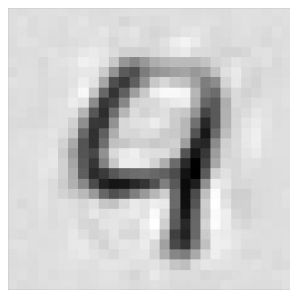
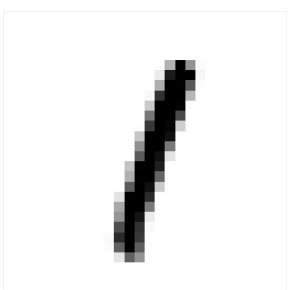
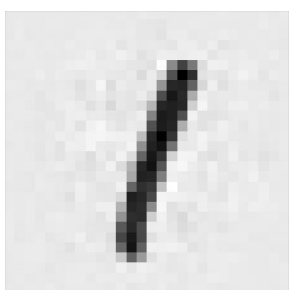
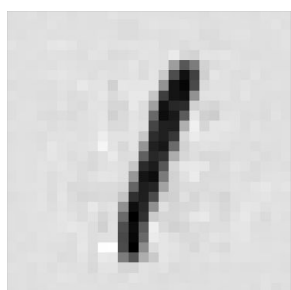
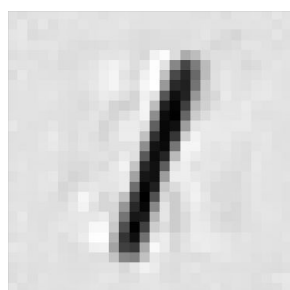
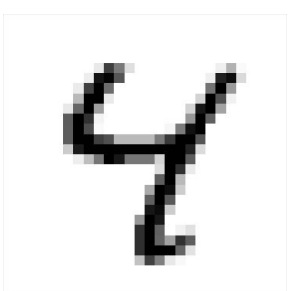
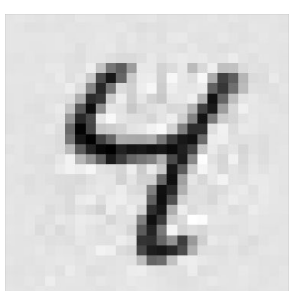
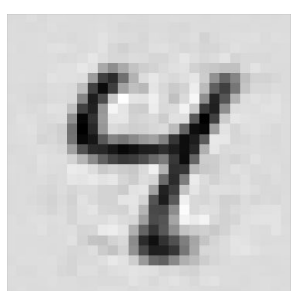
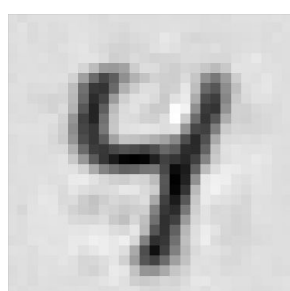
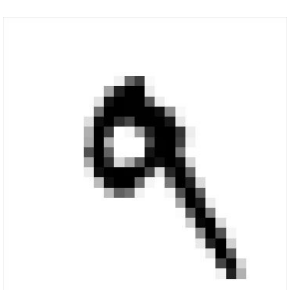
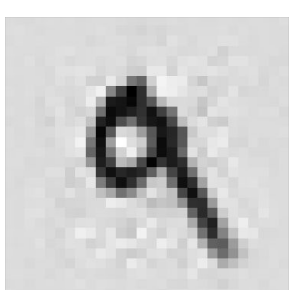
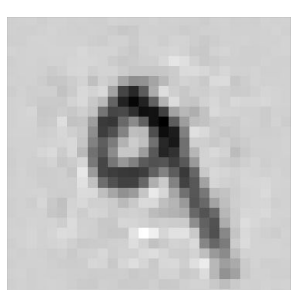
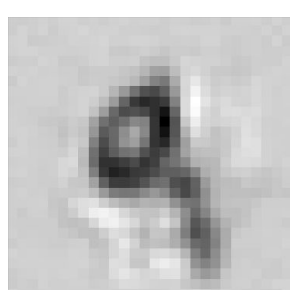
### *Methodology*

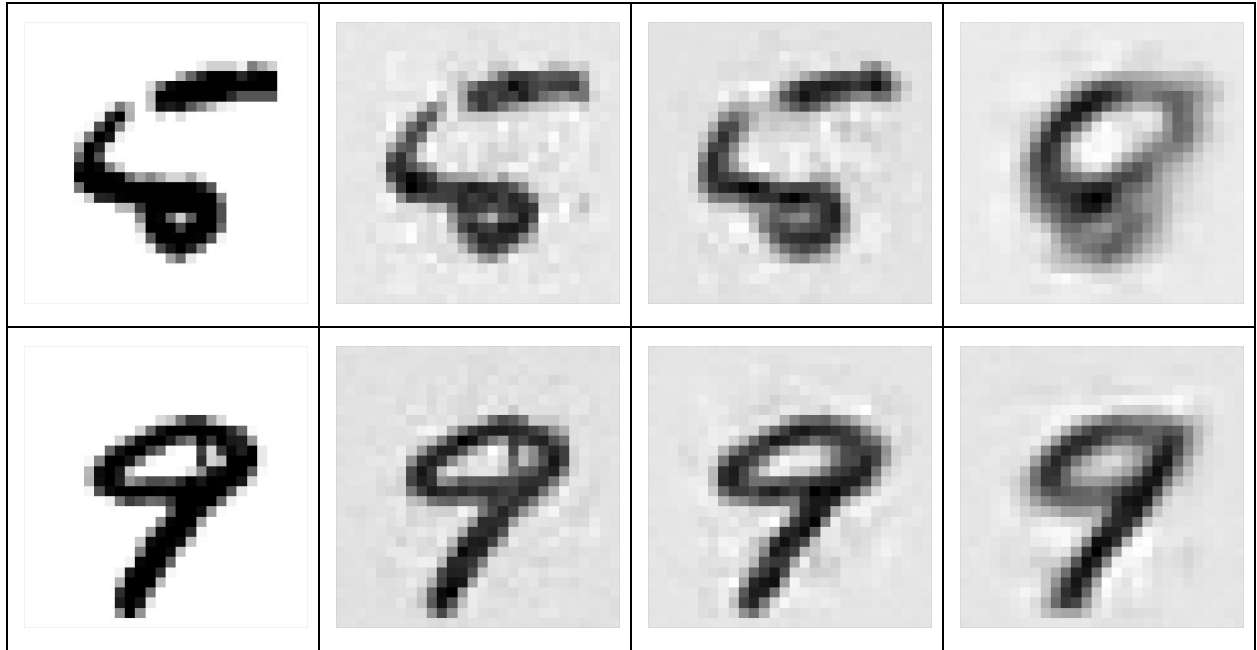
For Task 2, we built a neural network to compress the images that make up the MNIST dataset. These images have a height and width of 28 pixels, and are grayscale. Our model has four total layers, with the first acting as the compressor and the last three acting as the decompressor. The first layer is a dense layer that compresses an image to a specified number of nodes; we used 10, 50, and 200. The second layer (also dense) uses a ReLU activation function, and an expansion factor,  $T$ , of two. This means that the second layer has a total number of nodes equal to  $M \times N \times T$  (where  $M$  and  $N$  are the height and width of the image) or  $28 \times 28 \times 2$ . The third layer is also a dense layer and the output layer with  $M \times N$  nodes. Finally, we use a reshape layer which projects the data to a 2D space. Our model runs over 10 epochs, uses a batch size of 64, and uses Mean Squared Error (MSE) as the loss function.

### *Results*

**Table 1: Original and Compressed Images of MNIST Dataset.**

Original	P = 200	P = 50	P = 10
			
			
			



**Table 2: Average PSNR for Values of P of MNIST Dataset**

Value of P	PSNR
10	17.715
50	22.742
200	26.187

We observe that a higher value of P leads to a higher average PSNR value (Table 2). This is due to the fact that a higher value of P means that less information needs to be lost in order to fit into the compressed size. Overall, the compressed versions of these numbers (Table 1) were extremely close to the originals in terms of shape, with only noticeable differences between the different compressed versions being the shade of color and clarity of the image. As the value of P decreases, the images become more pixelated and color depth is lost, as true blacks become gray. With the same value of P, the images that contain numbers that are similar to each other are harder to decompress. For example, the five in our case looks like an eight, and the four looks similar to the nine. The results show that if there is some ambiguity in the shape of the numbers the compression and decompression will often be much more difficult at smaller numbers. The numbers also display gray smudges on them, especially when P is 10. These can also be attributed to the ambiguity of features and the incredibly small compression size

### Task 3: Neural Network and Convolutional Neural Network Compression of Video.

#### *Introduction*

Traditional machine learning models for classification allow input training data to be converted from images to weights of a specific non-linear function that can be then used to classify target images into different classes when fed through the function. The weights of this training data are calculated by compressing shape and color information into feature maps. This compression process can also be used outside of classification, allowing for images to be compressed into feature maps through an encoder and subsequently blown back up to the original size using a decoder. We use the same process to compress a series of videos from three datasets: blowing bubbles, race horses, and basketball drills using different compression ratios:  $\frac{1}{3}$ ,  $\frac{1}{6}$ ,  $\frac{1}{8}$ ,  $\frac{1}{4}$ ,  $\frac{1}{2}$ . We introduce two different models to achieve this compression, and compare the differences between both models using PSNR as our performance metric. The testing and training images are sized to 416x240 pixels and are in the RGB color space.

#### *Proposed Methods*

We wanted to keep the methodology of our two models as similar as possible to give an accurate comparison between the two. Both models are using a sequential model using conv2d layers with a filter size of (3,3) and “same” padding. The feature maps for each layer are in the following order: 128,64,32,3,32,64,128,3. We use the mean squared error function and the adam optimizer for both models as well. For our training, we use 100 images from one of the three training data folders: blowing bubbles, basketball drills, and race horses as our training data, and 30 images from the aforementioned folder for testing data. We trained a model for each folder, so we get 3 models trained, and test using images from the same folder for each model. We use 10 epochs and a batch size of 10 for each model as well. We use the testing data to calculate the mean squared error as well as the PSNR for each model.

Our first proposed method is increasing the stride of the CNN layers to achieve our wanted compression levels. We use the following strides:

**Table 3: Stride for Model 1**

Ratio	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$
encode layer	(1,2)	(2,2)	(2,2)	(2,2)	(2,2)
encode layer	(1,1)	(1,1)	(1,2)	(2,2)	(2,2)
encode layer	(1,1)	(1,1)	(1,1)	(1,1)	(1,2)
encode layer	(1,1)	(1,1)	(1,1)	(1,1)	(1,1)

decode layer	(1,1)	(1,1)	(1,1)	(1,1)	(1,1)
decode layer	(1,1)	(1,1)	(1,1)	(1,1)	(1,2)
decode layer	(1,1)	(1,1)	(1,2)	(2,2)	(2,2)
decode layer	(1,2)	(1,2)	(2,2)	(2,2)	(2,2)

Our second method is by using max pooling layers to achieve the wanted compression ratios.  
The strides used in the max pooling layer are:

**Table 4: Strides and Max Pooling Size for Model 2**

Ratio	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$
encode layer	(1,1)	(1,1)	(1,1)	(1,1)	(1,1)
max pooling layer	(1,2)	(2,2)	(2,2)	(2,2)	(2,2)
encode layer	(1,1)	(1,1)	(1,1)	(1,1)	(1,1)
max pooling layer	(1,1)	(1,2)	(2,2)	(2,2)	(2,2)
encode layer	(1,1)	(1,1)	(1,1)	(1,1)	(1,1)
max pooling layer	(1,1)	(1,1)	(1,1)	(1,2)	(1,2)
encode layer	(1,1)	(1,1)	(1,1)	(1,1)	(1,1)
max pooling layer	(1,1)	(1,1)	(1,1)	(1,1)	(1,1)
decode layer	(1,1)	(1,1)	(1,1)	(1,1)	(1,1)
upsampling layer	(1,1)	(1,1)	(1,1)	(1,2)	(1,2)
decode layer	(1,1)	(1,1)	(1,1)	(1,1)	(1,1)
upsampling layer	(1,1)	(1,2)	(2,2)	(2,2)	(2,2)
decode layer	(1,1)	(1,1)	(1,1)	(1,1)	(1,1)

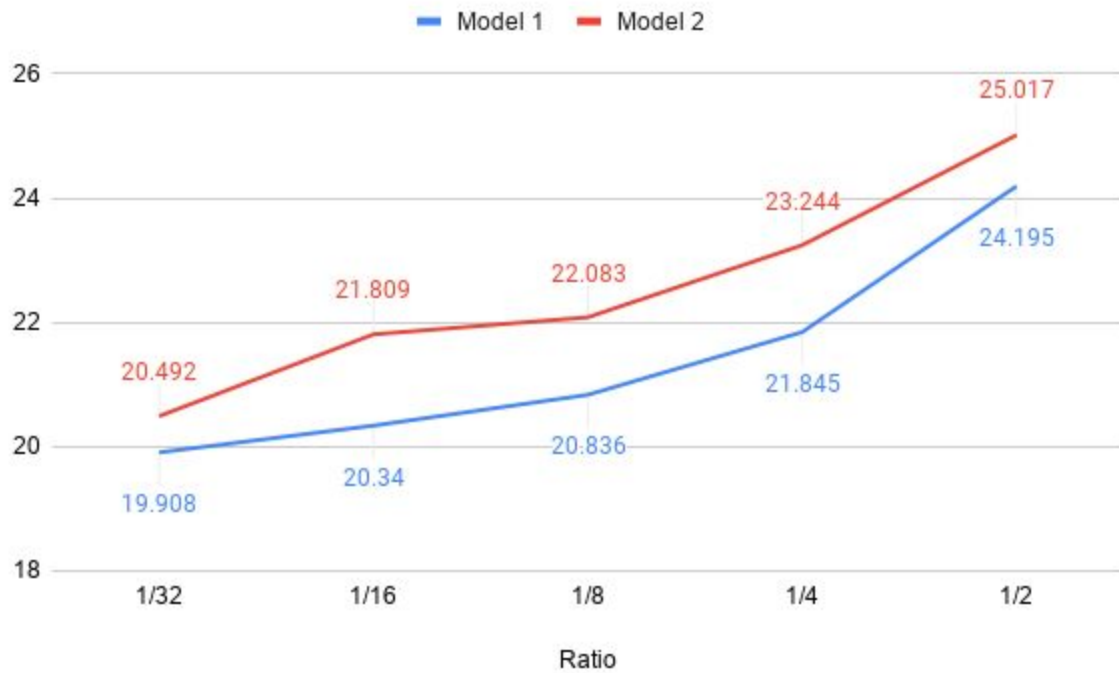
upsampling layer	(1,2)	(2,2)	(2,2)	(2,2)	(2,2)
decode layer	(1,1)	(1,1)	(1,1)	(1,1)	(1,1)

### *Experimental Studies*

**Table 5: Average PSNR Values for each Model, Dataset, and Ratio**

<b>Model 1</b>					
Ratio	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$
Basketball Drill	24.407	21.842	20.607	20.272	19.816
Blowing Bubbles	24.412	22.234	21.349	20.366	19.909
Race Horses	23.766	21.46	20.552	20.383	20.001
<b>Model 2</b>					
Ratio	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$
Basketball Drill	25.905	23.938	21.486	21.01	19.348
Blowing Bubbles	23.072	21.738	22.206	21.503	20.855
Race Horses	26.073	24.058	22.557	22.916	21.273





**Figure 2: Average PSNR vs. Compression Ratio**

From the graph above (Figure 2), we can see that model 2 achieves better PSNR values for every given compression ratio. Model 2 uses MaxPooling layers to compress rather than larger strides (like in Model 1), so while the images are clearer we have more multiplications to perform compared to Model 1. The graph in Figure 2 was made by taking the average PSNR value for each compression ratio, and averaging them over each dataset.

**Table 6: Compression Results for Basketball Dataset**








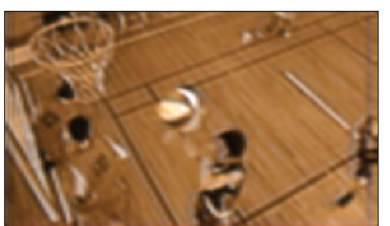

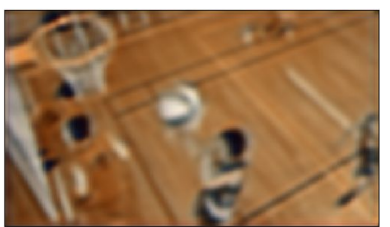



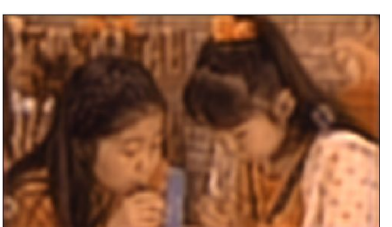



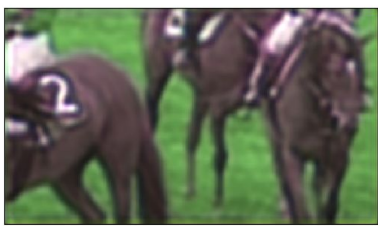
Compression Ratio	Model 1	Model 2
Original		
$\frac{1}{2}$		
$\frac{1}{4}$		
$\frac{1}{8}$		
$\frac{1}{16}$		
$\frac{1}{32}$		

Table 7: Compression Results for Blowing Bubbles Dataset

Compression Ratio	Model 1	Model 2
Original		
$\frac{1}{2}$		
$\frac{1}{4}$		
$\frac{1}{8}$		
$\frac{1}{16}$		
$\frac{1}{32}$		



**Table 8: Compression Results for Race Horses Dataset**

Compression Ratio	Model 1	Model 2
Original		
$\frac{1}{2}$		
$\frac{1}{4}$		
$\frac{1}{8}$		
$\frac{1}{16}$		
$\frac{1}{32}$		

Overall, the higher the compression ratio, the more color depth is lost, and the more pixelated the images get. However, Model 2 is much better at retaining color information than Model 1 in all cases. After the  $\frac{1}{4}$  compression rate, Model 1 starts to lose almost all of its color information, defaulting to a single majority color for every pixel using various shades of the color to represent objects. However, Model 2 is able to retain at least some of the color image information at the lowest compression ratios, while retaining a majority of color information at higher compression ratios. Model 1 seems to default towards more grayish color color tones, while Model 2 retains a shade of the original color in the highest compression ratios.

**Table 9: Multiplicative Complexity of the Models by Compression Ratio**

<b>Model 1</b>		
$\frac{1}{2}$	$(240 \times 208 \times 128)(3 \times 3 \times 3) + (240 \times 208 \times 64)(3 \times 3 \times 128) +$ $(240 \times 208 \times 32)(3 \times 3 \times 64) + (240 \times 208 \times 3)(3 \times 3 \times 32) + (240 \times 208 \times 32)(3 \times 3 \times 3)$ $+ (240 \times 208 \times 64)(3 \times 3 \times 32) + (240 \times 208 \times 128)(3 \times 3 \times 64) +$ $(240 \times 416 \times 3)(3 \times 3 \times 128)$	9,805,086,720
$\frac{1}{4}$	$(120 \times 208 \times 128)(3 \times 3 \times 3) + (120 \times 208 \times 64)(3 \times 3 \times 128) +$ $(120 \times 208 \times 32)(3 \times 3 \times 64) + (120 \times 208 \times 3)(3 \times 3 \times 32) + (120 \times 208 \times 32)(3 \times 3 \times 3)$ $+ (120 \times 208 \times 64)(3 \times 3 \times 32) + (120 \times 208 \times 128)(3 \times 3 \times 64) +$ $(240 \times 416 \times 3)(3 \times 3 \times 128)$	5,075,066,880
$\frac{1}{8}$	$(120 \times 208 \times 128)(3 \times 3 \times 3) + (120 \times 104 \times 64)(3 \times 3 \times 128) +$ $(120 \times 104 \times 32)(3 \times 3 \times 64) + (120 \times 104 \times 3)(3 \times 3 \times 32) + (120 \times 104 \times 32)(3 \times 3 \times 3)$ $+ (120 \times 104 \times 64)(3 \times 3 \times 32) + (120 \times 208 \times 128)(3 \times 3 \times 64) +$ $(240 \times 416 \times 3)(3 \times 3 \times 128)$	3,673,313,280
$\frac{1}{16}$	$(120 \times 208 \times 128)(3 \times 3 \times 3) + (60 \times 104 \times 64)(3 \times 3 \times 128) +$ $(60 \times 104 \times 32)(3 \times 3 \times 64) + (60 \times 104 \times 3)(3 \times 3 \times 32) + (60 \times 104 \times 32)(3 \times 3 \times 3) +$ $(60 \times 104 \times 64)(3 \times 3 \times 32) + (120 \times 208 \times 128)(3 \times 3 \times 64) +$ $(240 \times 416 \times 3)(3 \times 3 \times 128)$	2,972,436,480
$\frac{1}{32}$	$(120 \times 208 \times 128)(3 \times 3 \times 3) + (60 \times 104 \times 64)(3 \times 3 \times 128) +$ $(60 \times 52 \times 32)(3 \times 3 \times 64) + (60 \times 52 \times 3)(3 \times 3 \times 32) + (60 \times 52 \times 32)(3 \times 3 \times 3) +$ $(60 \times 104 \times 64)(3 \times 3 \times 32) + (120 \times 208 \times 128)(3 \times 3 \times 64) +$ $(240 \times 416 \times 3)(3 \times 3 \times 128)$	2,909,537,280

<b>Model 2</b>		
$\frac{1}{2}$	$(240 \times 416 \times 128)(3 \times 3 \times 3) + (240 \times 208 \times 64)(3 \times 3 \times 128) + (240 \times 208 \times 32)(3 \times 3 \times 64) + (240 \times 208 \times 3)(3 \times 3 \times 32) + (240 \times 208 \times 32)(3 \times 3 \times 3) + (240 \times 208 \times 64)(3 \times 3 \times 32) + (240 \times 208 \times 128)(3 \times 3 \times 64) + (240 \times 416 \times 3)(3 \times 3 \times 128)$	9,977,610,240
$\frac{1}{4}$	$(240 \times 416 \times 128)(3 \times 3 \times 3) + (120 \times 208 \times 64)(3 \times 3 \times 128) + (120 \times 208 \times 32)(3 \times 3 \times 64) + (120 \times 208 \times 3)(3 \times 3 \times 32) + (120 \times 208 \times 32)(3 \times 3 \times 3) + (120 \times 208 \times 64)(3 \times 3 \times 32) + (240 \times 208 \times 128)(3 \times 3 \times 64) + (240 \times 416 \times 3)(3 \times 3 \times 128)$	7,174,103,040
$\frac{1}{8}$	$(240 \times 416 \times 128)(3 \times 3 \times 3) + (120 \times 208 \times 64)(3 \times 3 \times 128) + (120 \times 104 \times 32)(3 \times 3 \times 64) + (120 \times 104 \times 3)(3 \times 3 \times 32) + (120 \times 104 \times 32)(3 \times 3 \times 3) + (120 \times 104 \times 64)(3 \times 3 \times 32) + (240 \times 208 \times 128)(3 \times 3 \times 64) + (240 \times 416 \times 3)(3 \times 3 \times 128)$	6,692,474,880
$\frac{1}{16}$	$(240 \times 416 \times 128)(3 \times 3 \times 3) + (120 \times 208 \times 64)(3 \times 3 \times 128) + (60 \times 104 \times 32)(3 \times 3 \times 64) + (60 \times 104 \times 3)(3 \times 3 \times 32) + (60 \times 104 \times 32)(3 \times 3 \times 3) + (60 \times 104 \times 64)(3 \times 3 \times 32) + (120 \times 208 \times 128)(3 \times 3 \times 64) + (240 \times 416 \times 3)(3 \times 3 \times 128)$	4,611,409,920
$\frac{1}{32}$	$(240 \times 416 \times 128)(3 \times 3 \times 3) + (120 \times 208 \times 64)(3 \times 3 \times 128) + (60 \times 104 \times 32)(3 \times 3 \times 64) + (60 \times 52 \times 3)(3 \times 3 \times 32) + (60 \times 52 \times 32)(3 \times 3 \times 3) + (60 \times 104 \times 64)(3 \times 3 \times 32) + (120 \times 208 \times 128)(3 \times 3 \times 64) + (240 \times 416 \times 3)(3 \times 3 \times 128)$	4,606,018,560

We calculated the number of multiplications by adding together the number of multiplications of each layer. The multiplications for each layer is calculated by multiplying the output dimensions of the layer by the filter size.

**Table 10: Number of Parameters per Model**

Model 1	$(3 \times 3 \times 3 \times 128) + (3 \times 3 \times 128 \times 64) + (3 \times 3 \times 64 \times 32) + (3 \times 3 \times 32 \times 3) + (3 \times 3 \times 3 \times 32) + (3 \times 3 \times 32 \times 64) + (3 \times 3 \times 64 \times 128) + (3 \times 3 \times 128 \times 3)$	192,960
Model 2	$(3 \times 3 \times 3 \times 128) + (3 \times 3 \times 128 \times 64) + (3 \times 3 \times 64 \times 32) + (3 \times 3 \times 32 \times 3) + (3 \times 3 \times 3 \times 32) + (3 \times 3 \times 32 \times 64) + (3 \times 3 \times 64 \times 128) + (3 \times 3 \times 128 \times 3)$	192,960

The number of parameters per model were calculated by multiplying the filter size of the current layer times the number of output feature maps, summed over every layer. The number of

parameters stays the same for both models because our filter sizes and number of output feature maps stay the same in both models.

### *Conclusion and Future Work*

In conclusion, our two models both did well in terms of PSNR. However, the second model which incorporates max pooling to reduce image size instead of convolutional filter stride achieved higher PSNR values across the board. From a non-qualitative point of view it can be seen that the max pooling model was able to more consistently preserve the colors that were present in the original image, especially at lower compression ratios.

In the future, it would be beneficial to change the point that we start adjusting the strides in the first model to be one layer further in. This would allow both models to have the same multiplicative complexity and the same number of parameters, allowing a stronger comparison to be drawn between the two models on the merits of their design alone.

### **Contribution of Team Members**

Stephen Pacwa: 40%

Michael Dallow: 30%

Daniel Okazaki: 30%