EE 5545 Assignment 4

Ethan Glaser (eg492)

1. Conv2D Implementation Details:
    a. Im2col - Iterated through all starting locations of convolution (top left indices) and extracted all values in each dot product of the convolution, flattening and appending to an array that was then used in matrix multiplication before reshaping to the desired shape
    b. Winograd – padded input and defined transformation matrices for 3x3 kernel with 4x4 sections of padded input. Iterated through 4x4 blocks using stride 2 and performed matrix multiplications on input, kernel, and output transformations then remove padding.
    c. FFT –kernel to the same size as input and then flip axes and reorient such that the center of the kernel is in the top left, then multiply their Fast Fourier Transforms and take the inverse FFT and select center to match dimensions of output.

2. Matmul Implementation Details:
    a. SVD: took svd of both input matrices, then selected columns from U, values from sigma, and rows from V according to the corresponding ranks of A and B, then reconstructed each and matrix multiplied them together.
    b. Log – iterate through row/column combinations and add log values and use exponent to determine magnitude. Sign is also used to determine whether product of 2 values will be negative and this result is used as a mask on magnitudes to determine final result.

3. The following figures show the resulting reconstruction errors of each of the 3 cases examined. For Winograd, there was a significant magnitude difference between float and int reconstruction errors. The first figure shows the raw scale with both float values being invisible due to the extremely small magnitude. Even on a log scale they are much smaller, but the relative difference between the 2 is show in the 3rd image. There weren't consistent results among the int values for the cases tested so the four of those were approximately the same, whereas the reconstruction error for the float32 was consistently about twice that of the float64 which makes sense due to the precision.

   Figure 2 shows the reconstruction error of FFT. Running the script several times yields extremely low reconstruction errors, which is encouraging and suggests that the implementation achieves the desired goal. The float64 error is significantly lower than the float32 and int values, which are relatively comparable.

   Figure 3 shows the reconstruction error of Log MatMul. In this case there is not much difference among the various integers across several experiments, however the float values yield higher errors relative to the integers, with float32 again being significantly larger than float64.

   These experiments were conducted using random integers from -3 to 3 and random uniform floats selected and multiplied by 3 for consistency. Raising the magnitude past a certain level led to high reconstruction error in int8 due to the size of the results not being able to be captured.

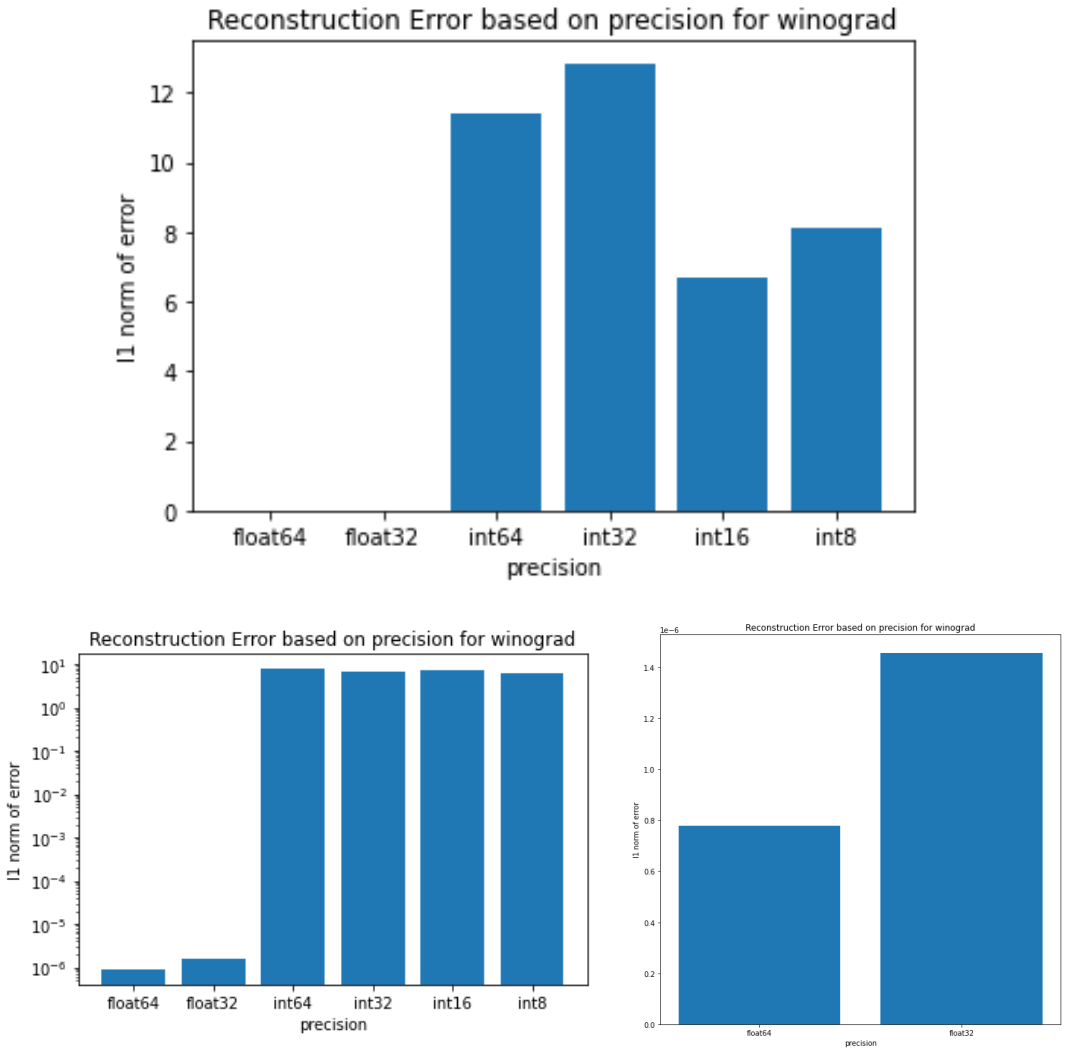


Figure 1: Reconstruction Error for Winograd (top is original, bottom left is log scale, and bottom right is just the float comparison)

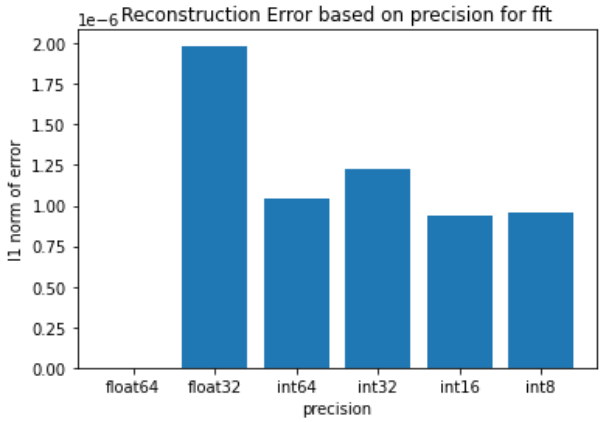


Figure 2: FFT Reconstruction Error

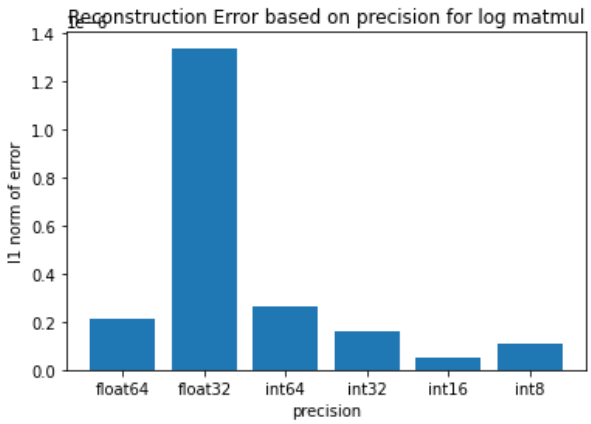
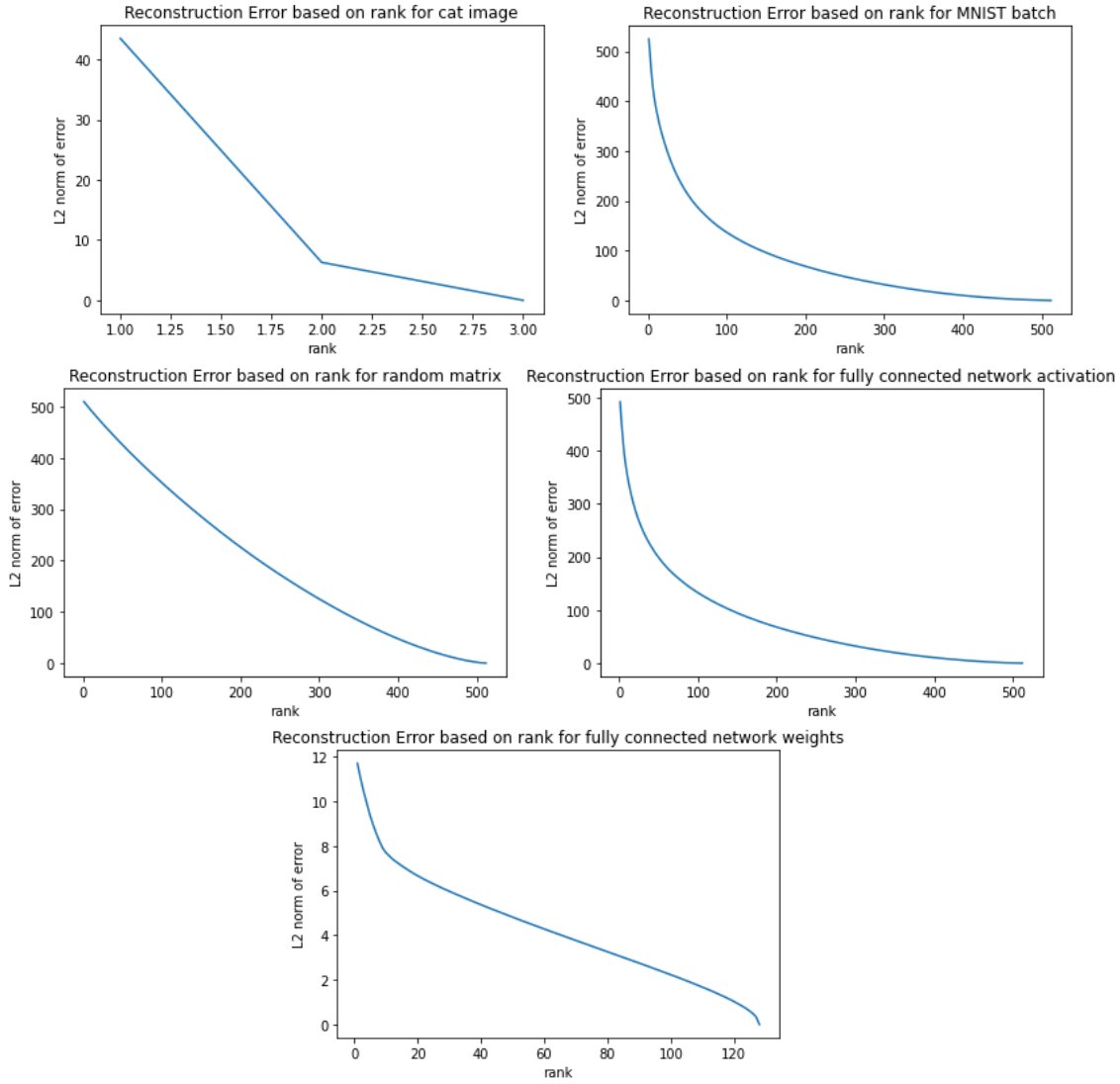
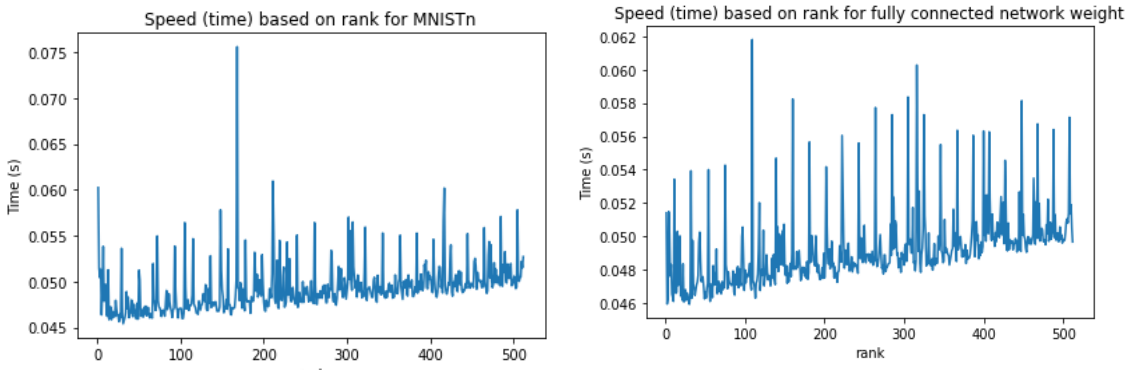


Figure 3: Log MatMul Reconstruction Error

4. The following figures display the performance of SVD MatMul across various situations in terms of accuracy and speed as a result of rank. These plots generally decrease at a decreasing rate as rank increases. The reason for this is that smaller ranks mean less information from the original matrices are accounted for, so naturally higher ranks will yield better results. However, the most important information is prioritized, so the addition of a rank will yield decreasing boosts in accuracy compared to the previous ranks. The plots in Figures 9-10 are relatively messy due to outliers at various ranks, but if these are ignored the general trend shows an approximately linear increase in compute time per additional rank. This is due to the fact that each additional rank leads to one additional constant being multiplied by a row in V, one additional column in U multiplying by that row, etc. so each rank introduces the same level of new computation, thus the linear relationship makes sense.



Figures 4-8: SVD accuracy based on rank value for several situations



Figures 9-10: Computation time as a function of rank

5. In order to test the rank impact on model performance, a base model was trained to determine the weights. Then a new class was created called SVDNet, which takes a rank argument. The model has a similar architecture as the base net (with weights from the original copied into its parameters) but each of the forward passes have matrix multiplications using the svd function with the rank of the weights passed in (instead of just running a traditional PyTorch Linear layer). The code for this class is displayed below to better understand the difference between the original model.

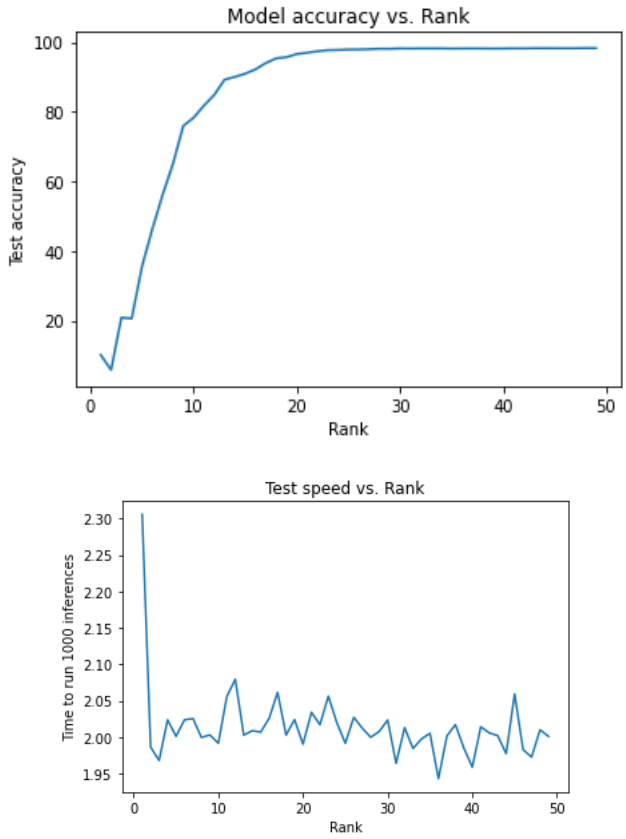


Figure 11: Model accuracy and inference speed as a function of rank

```python
class NormalNet(nn.Module):
    def __init__(self):
        super(NormalNet, self).__init__()
        self.fc1 = nn.Linear(28 * 28, 512)
        self.fc2 = nn.Linear(512, 128)
        self.fc3 = nn.Linear(128, 10)

    def forward(self, x):
        bs = x.size(0)
        x = x.view(bs, -1)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.fc2(x)
        x = F.relu(x)
        x = self.fc3(x)
        output = F.log_softmax(x, dim=1)
        return output
```

```python
class SVDNet(nn.Module):
    def __init__(self, rank):
        super(SVDNet, self).__init__()
        self.fc1 = nn.Linear(28 * 28, 512)
        self.fc2 = nn.Linear(512, 128)
        self.fc3 = nn.Linear(128, 10)
        self.rank = rank

    def forward(self, x):
        bs = x.size(0)
        x = x.view(bs, -1)
        x = svd(x, self.fc1.weight.T, rank_B=self.rank)
        x = F.relu(x)
        x = svd(x, self.fc2.weight.T, rank_B=self.rank)
        x = F.relu(x)
        x = svd(x, self.fc3.weight.T, rank_B=self.rank)
        output = F.log_softmax(x, dim=1)
        return output
```

Figure 12: Difference in implementation of SVD class to account for rank