Answer for question 1:

https://colab.research.google.com/drive/1PzOzpArXCKZk2-8tRD1l1qsjA0B1FVhl?usp=sharing

Answer for question 2:

This convolution results in reducing the size of the image as illustrated in the example as the new image size will follow the following equations:

$$M' = \left\lfloor \frac{M+2p-r}{s} \right\rfloor + 1$$

$$N' = \left\lfloor \frac{N+2p-r}{s} \right\rfloor + 1$$

And

$$\begin{aligned} &S = 1(\#\ of\ strides) \\ &P = 0(padding) \\ &r = 3(size\ of\ kernel) \\ &M = N = 6(original\ image\ size). \end{aligned}$$

So that results in:

$$\begin{aligned} M' &= 4 \\ N' &= 4 \end{aligned}$$

Also for the effect of the kernel on the values of the image, here is the plot for the original image to visualize it and to see what happens after convolution:

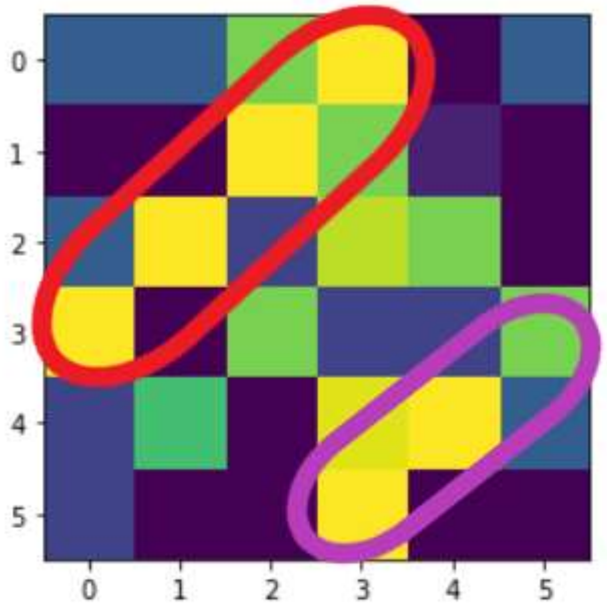


Figure 1: Original image

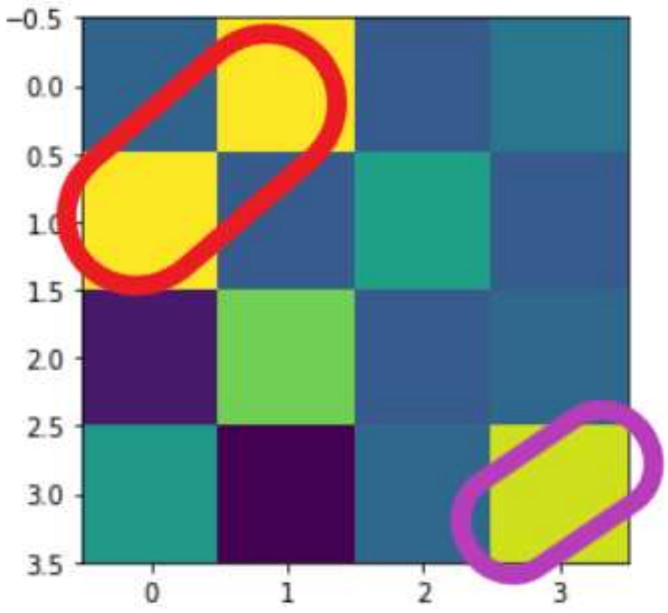


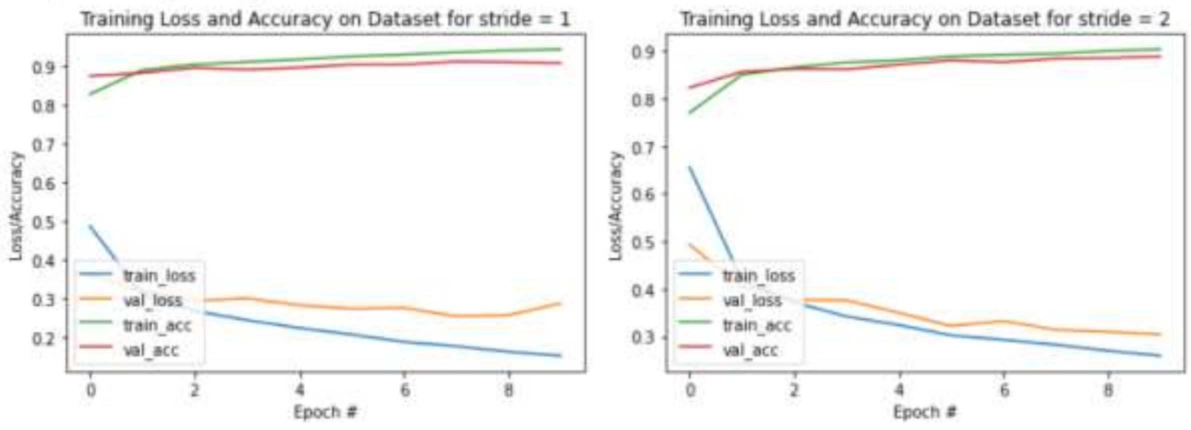
Figure 2: New image (after convolution)

And here after the convolution, the effect of the kernel prevails as it make the edges with 45 degree angle shines while diminishes the other edges.

Answer for question 3:

https://colab.research.google.com/drive/1ELcONXQDCroktIAodGddclwcHgdcsg5N?usp=sharing

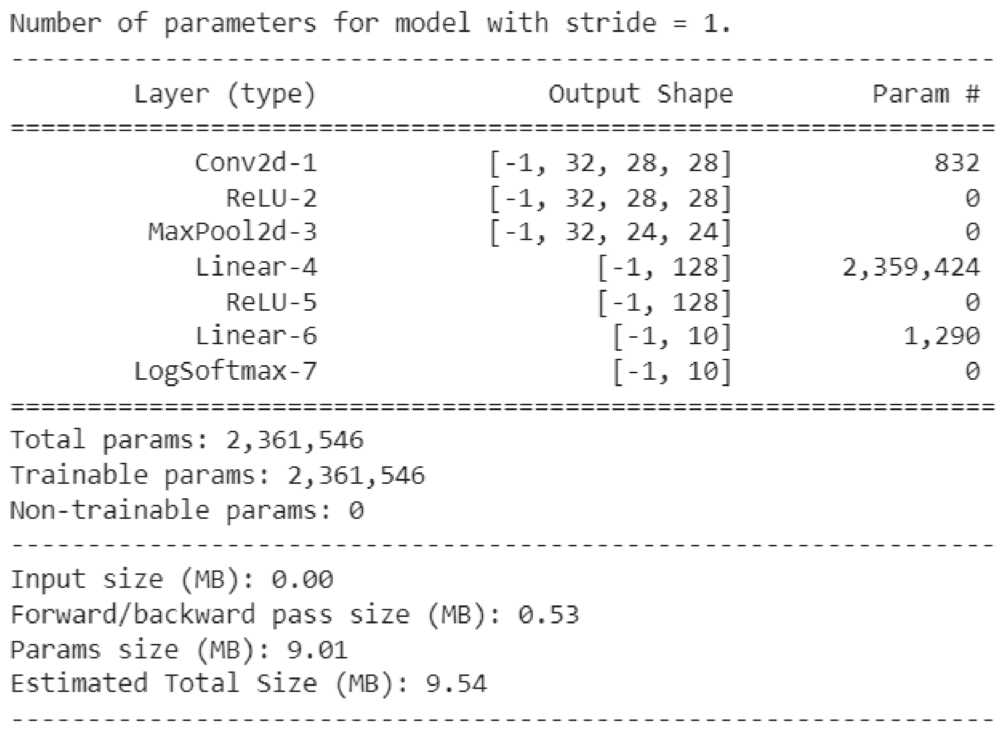
Answer for question 4:



These curves show that there is no much difference between the results provided by the model with stride = 1 vs. the model with results from model with stride = 2.

But taking in consideration the number of parameters gives a whole new dimension for this case.

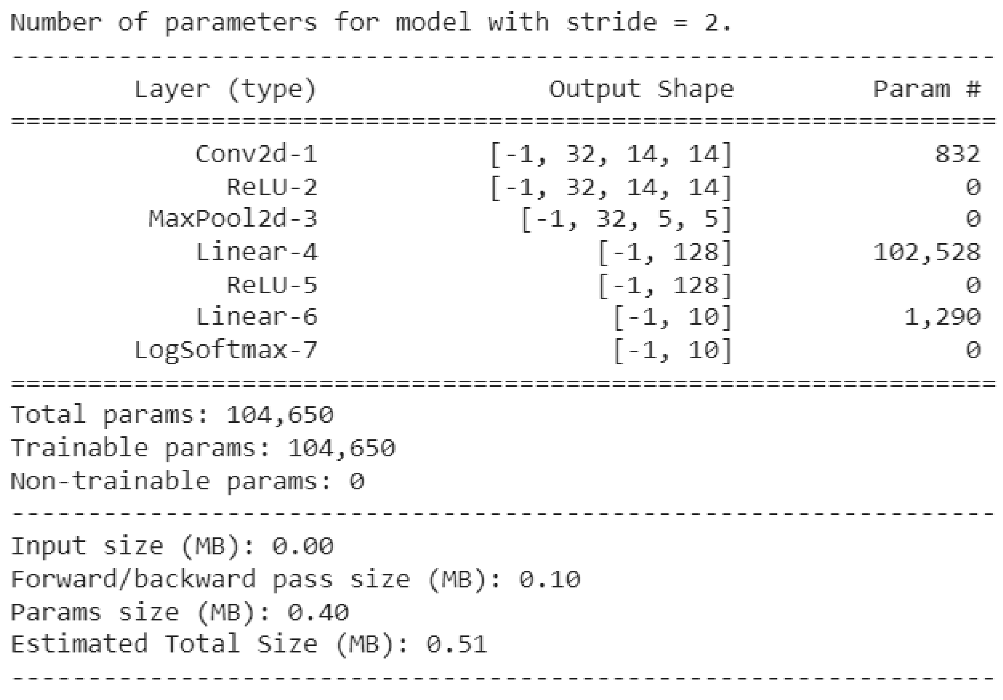
As figure 3 says, the total number of parameters with one stride is 2,361,546.



```
Number of parameters for model with stride = 1.
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 32, 28, 28]             832
              ReLU-2           [-1, 32, 28, 28]               0
         MaxPool2d-3           [-1, 32, 24, 24]               0
            Linear-4                  [-1, 128]       2,359,424
              ReLU-5                  [-1, 128]               0
            Linear-6                   [-1, 10]           1,290
        LogSoftmax-7                   [-1, 10]               0
================================================================
Total params: 2,361,546
Trainable params: 2,361,546
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.53
Params size (MB): 9.01
Estimated Total Size (MB): 9.54
----------------------------------------------------------------
```

Figure 3: Number of parameters for model with stride = 1.

While the total number of parameters with the model using stride = 2, as shown in figure 4, is 104,650.



```
Number of parameters for model with stride = 2.
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 32, 14, 14]             832
              ReLU-2           [-1, 32, 14, 14]               0
         MaxPool2d-3             [-1, 32, 5, 5]               0
            Linear-4                  [-1, 128]         102,528
              ReLU-5                  [-1, 128]               0
            Linear-6                   [-1, 10]           1,290
        LogSoftmax-7                   [-1, 10]               0
================================================================
Total params: 104,650
Trainable params: 104,650
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.10
Params size (MB): 0.40
Estimated Total Size (MB): 0.51
----------------------------------------------------------------
```

Figure 4: Number of parameters for model with stride = 2.

And this huge difference in number of parameters also maps in the size needed for the parameters to be stored which is 9.54MB vs 0.51MB which is almost 19 times greater.

Also, the time consumed during training the model decreases with using higher stride as the number of calculations decreases.

```
[INFO] total time taken to train the model with stride = 1: 73.95s

[INFO] total time taken to train the model with stride = 2: 64.47s
```

Answer for question 5:

Regression loss functions:

Example:
Solving regression problems like weather forecasting and house prices.

1. MSE (Mean Squared Error)

$$MSE = \frac{1}{N}\sum_{i}^{N}(Y - Y')^2$$

Advantages:

- Easy to interpret.
- Always differential because of the square.
- Only one global minima.

Disadvantages:

- Not robust to outlier.

Yes, it could be used as a surrogate loss function because it's differentiable.

2. MAE (Mean Absolute Error)

$$MAE = \frac{1}{N}\sum_{i}^{N}|Y - Y'|$$

Advantages:

- Easy to interpret.
- Robust to outlier

Disadvantages:

- Not differential (cannot use gradient descent directly).

No, it can't be used as a surrogate loss function because it's not differentiable.

Classification loss functions:

Example:
Solving classification problems like animal detection and classification and characters detection.

1. Binary cross-entropy

$$Binary\ cross\ entropy = -\frac{1}{N}\sum_{i=1}^{N} y * \log(y') + (1-y) * \log(1-y')$$

Advantages:

- cost function is a differential.

Disadvantages:

- Multiple local minima.

Yes, it could be used as a surrogate loss function because it's differentiable.

2. Categorical cross-entropy:

$$\text{Categorical cross}-\text{entropy} = \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{K} y_{ij} * \log(y'_{ij})$$

Yes, it could be used as a surrogate loss function because it's differentiable.

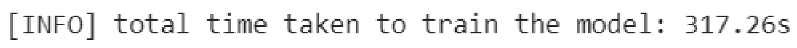
Answer for question 6:

[CIT690E Deep learning Assignment 2 Q6.ipynb - Colaboratory (google.com)]

The effect of increasing the batch size on the speed, accuracy, and loss values:

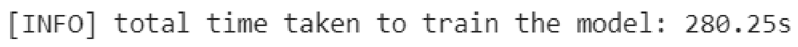
❖ **The speed:**
The speed of training increases when changing it from 32 to 1000 but after changing to 10,000 it's slowed down with respect to 1000 but still faster than the 32 batch size and the explanation for this is that it's supposed to be faster when increasing the batch size but what happened when using batch size equals to 10,000 because the available memory to store the data in would make that decrease in the speed but if we have the adequate amount of memory to deal with such data during the processing it would increase the speed as expected.
And this could be shown from the time consumed in training for each case:

- For batch size = 32:



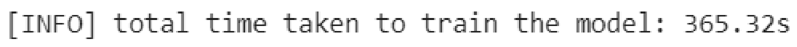
```
[INFO] total time taken to train the model: 317.26s
```

- For batch size = 1000:



```
[INFO] total time taken to train the model: 280.25s
```

- For batch size = 10,000:



```
[INFO] total time taken to train the model: 365.32s
```

❖ **Accuracy:**

higher batch sizes lead to lower accuracy.

A higher batch size does not usually achieve high accuracy, and the learning rate and the optimizer used will have a significant impact as well.

❖ **loss values:**

Increasing batch size increases loss values.