# Author : Cristopher McIntyre Garcia
# Email : cmcin019@uottawa.ca

## Introduction:

In this assignment we evaluate and compare different CNN architectures while performing object recognition. The basic CNN is comprised of a 5x5 kernel and a valid convolution. The forward propagation goes through two convolutions with relu as the activation function, and after each, are sent through a pooling layer. Following this are three linear layers, the first two also using relu as an activation function.

The architecture of the model can be seen bellow:

**Original CNN (Valid - 5x5 kernel)**

```
Net(
    (conv1): Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1))
    (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (conv2): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
    (fc1): Linear(in_features=400, out_features=120, bias=True)
    (fc2): Linear(in_features=120, out_features=120, bias=True)
    (fc3): Linear(in_features=120, out_features=10, bias=True)
)
```

After running the initial model, here are the test accuracies respective to each epoch:

```
Epoch=1     Test Accuracy=46.430
Epoch=2     Test Accuracy=52.890
Epoch=3     Test Accuracy=58.460
Epoch=4     Test Accuracy=61.270
Epoch=5     Test Accuracy=60.000
Epoch=6     Test Accuracy=61.080
Epoch=7     Test Accuracy=61.920
Epoch=8     Test Accuracy=61.840
Epoch=9     Test Accuracy=61.850
Epoch=10    Test Accuracy=63.010
```

And bellow is figure 1, where these accuracies are plotted with respect to their epoch.

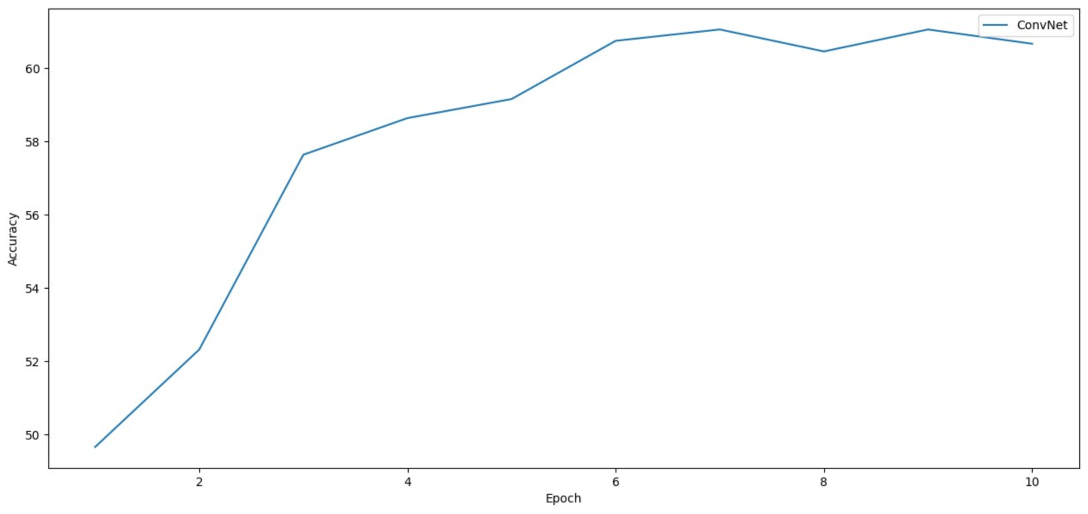

Figure 1. ConvNet with 5x5 kernel, relu activation, and valid convolution

In every part of this assignment, we use this CNN to compare with other models. Please refer to the epochs and figure above for reference when accuracies are discussed in the coming parts.

## Part 01.

In this part we will take a look at how the ConvNet compares to a verity of fully connected (FC) models without any convolution layers. The fully connected models will be comprised of 0, 1, 2, 3, and 4 hidden layers with 120 nodes and relu as their activation function.

The architecture of the fully connected models can be seen bellow:

**FC Net**

```
FC_Net(
        (ffn): Linear(in_features=3072, out_features=10, bias=True)
        (fc_in): Linear(in_features=3072, out_features=120, bias=True)
        (fc): Linear(in_features=120, out_features=120, bias=True)
        (fc_out): Linear(in_features=120, out_features=10, bias=True)
)
```

Depending on if there are any hidden layers in the model, either a combination of fc_in, a certain number of fc, and fc_out will be used, or simply ffn if there are no hidden layers. An FC network with 2 hidden layers would look as follows:

```
x = relu(     fc_in(x)      )
x = relu(     fc(x)         )
x = relu(     fc_out(x)     )
```

The first and final few epochs for each combination are seen bellow (the rest are at the end of the report in the results section):

**FC Net – 0 hidden layers**

```
Epoch=1     Test Accuracy=31.630
Epoch=2     Test Accuracy=32.790
...
Epoch=9     Test Accuracy=32.910
Epoch=10    Test Accuracy=33.050
```

**FC Net – 1 hidden layer**

```
Epoch=1     Test Accuracy=47.110
Epoch=2     Test Accuracy=46.890
...
Epoch=9     Test Accuracy=48.740
Epoch=10    Test Accuracy=48.620
```

**FC Net – 2 hidden layers**

```
Epoch=1     Test Accuracy=47.230
Epoch=2     Test Accuracy=49.180
...
Epoch=9     Test Accuracy=51.270
Epoch=10    Test Accuracy=51.950
```

**FC Net – 3 hidden layers**

```
Epoch=1     Test Accuracy=46.170
Epoch=2     Test Accuracy=48.490
..
Epoch=9     Test Accuracy=51.750
Epoch=10    Test Accuracy=52.630
```

**FC Net – 4 hidden layers**

```
Epoch=1     Test Accuracy=44.110
Epoch=2     Test Accuracy=48.200
...
Epoch=9     Test Accuracy=50.950
Epoch=10    Test Accuracy=51.160
```

And bellow is figure 2, where these accuracies are plotted with respect to their epoch.

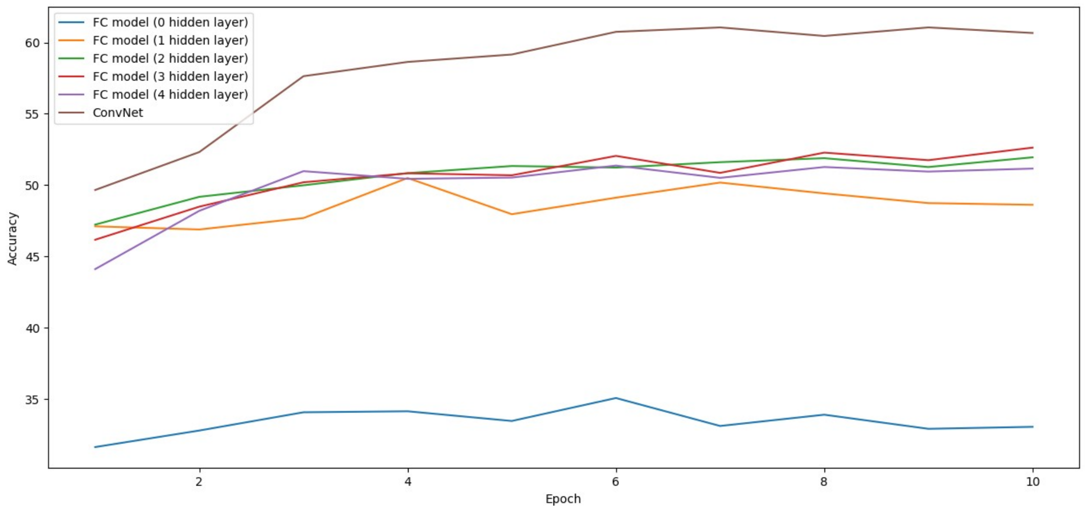

figure 2. CNN vs FC networks

Here we see that regardless of the parameters chosen for the FC networks, they are unable to out perform the CNN. This is most likely do to the fact that the CNN's convolution present an inductive bias to the model. It is able to better understand the structure of the image, and uses regions to process neighbouring pixels at a time. The performance of the FC networks do improve with their complexity (number of layers), however they do seem to start performing worse after 3 hidden layers. This is probably do to over fitting, and so more data would be needed for these larger networks. Therefore from these observations we note, when creating a model for image tasks, it is important to use a convolution window which will group neighbouring pixels together. This convolution window allows positional properties of the image to influence the output of the network, given the inductive structural bias of the model.

## Part 02.

In this part we compare CNN with different activation functions. We replace every activation function, including those used in the linear layers. The first CNN uses relu as the activation function and the second uses sigmoid. Relu will return the number fed into it if positive, else it will return 0, meaning that it is only bounded bellow. Sigmoid on the other hand will return a number between 0 and 1, meaning that the numbers will always be smaller than 1.

The architecture of the model using sigmoid can be seen bellow (the crossed part is not used in the forward function):

**CNN with Sigmoid**

```
CNN_Net(
        (conv1): Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1), padding=valid)
        (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
        (conv2): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1), padding=valid)
        (fc1): Linear(in_features=400, out_features=120, bias=True)
        ~~(fc1_same): Linear(in_features=1024, out_features=120, bias=True)~~
        (fc2): Linear(in_features=120, out_features=120, bias=True)
        (fc3): Linear(in_features=120, out_features=10, bias=True)
        (activation): Sigmoid()
)
```

After running the model using sigmoid, here are the test accuracies respective to each epoch:

| | |
|---|---|
| Epoch=1 | Test Accuracy=10.000 |
| Epoch=2 | Test Accuracy=10.000 |
| Epoch=3 | Test Accuracy=10.000 |
| Epoch=4 | Test Accuracy=10.000 |
| Epoch=5 | Test Accuracy=10.000 |
| Epoch=6 | Test Accuracy=10.000 |
| Epoch=7 | Test Accuracy=10.000 |
| Epoch=8 | Test Accuracy=10.000 |
| Epoch=9 | Test Accuracy=18.820 |
| Epoch=10 | Test Accuracy=19.500 |

And bellow is figure 3, where these accuracies are plotted with respect to their epoch.

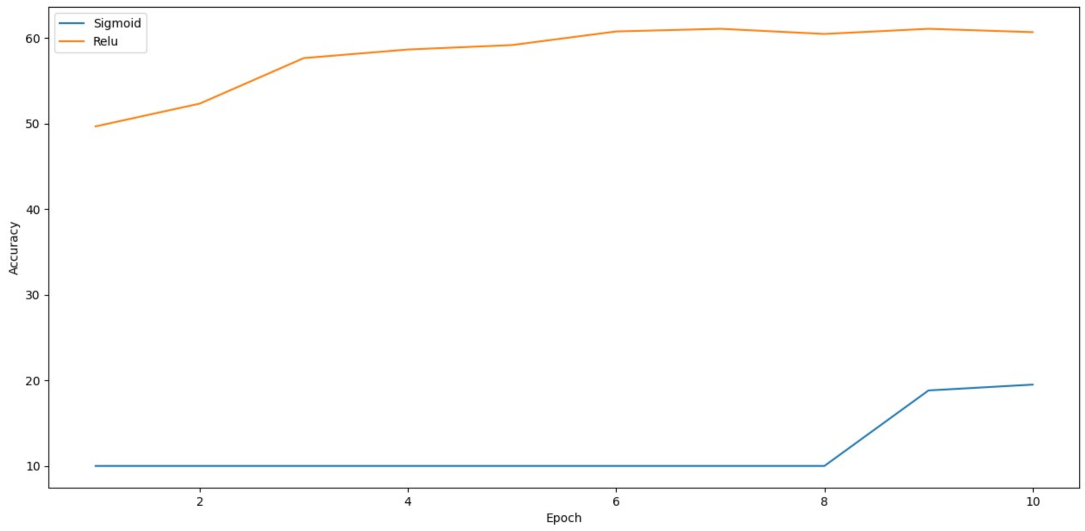

Figure 3. Sigmoid vs Relu

The results seem really bad, however the reason behind this is because the CNN using sigmoid is learning much slower than the CNN using relu. This I assume, is because of the bounds that limit the output of the sigmoid activation function to bellow 1. In the assignment instructions, it states not to change the learning rate, however for this section I was curious to see how the model would hold up with a larger learning rate.  And so, I modified the learning rate to be 0.1 to see how the CNN using sigmoid would perform, if it would hit a plateau or surpass the model using relu. We see the results of this in figure 4 found bellow.

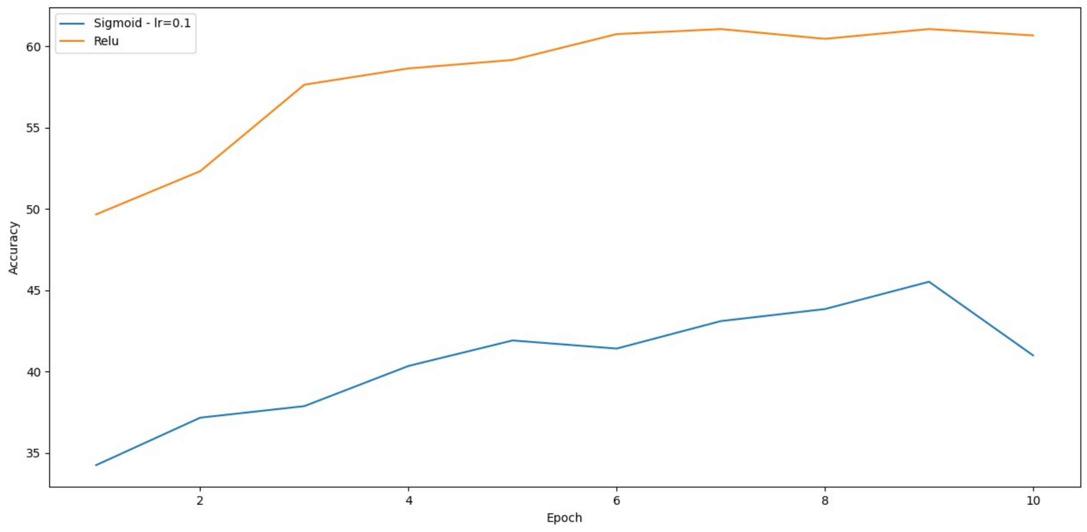

Figure 4. Sigmoid lr=0.1 vs Relu lr=0.001

As can be seen, the model using relu still outperforms the model using sigmoid. On top of this, it seems that the prior model starts to lose performance after the 9th epoch. This might be a fluke, and if we were to train for another epoch the model might have corrected itself. That said, training with relu seems to provide better results at the beginning, and the improvements are proportional to the model using sigmoid and a larger learning rate.

## Part 03.

In this part we compare CNN models with different kernel sizes and padding. The kernel sizes will be 5x5 and 3x3, and the padding will be either “same” or “valid.” When using padding of “same,” the output of the convolutions will have the same size as their input, this will be donne by padding the output with zeros. With valid padding, the output of the convolution layer will be lowered depending on the kernel size and stride (1x1 in our case). Including the original CNN, we will compare 4 models comprised of a 5x5 kernel with valid convolution, 5x5 kernel with same convolution, 3x3 kernel with valid convolution, and 3x3 kernel with same convolution.

The architecture of the models can be seen bellow:

**CNN - valid 5x5**

```
Net(
    (conv1): Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1))
    (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (conv2): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
    (fc1): Linear(in_features=400, out_features=120, bias=True)
    (fc2): Linear(in_features=120, out_features=120, bias=True)
    (fc3): Linear(in_features=120, out_features=10, bias=True)
)
```

**CNN - same 5x5**

```
CNN_Net_5x5_Same(
	(conv1): Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1), padding=same)
	(pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
	(conv2): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1), padding=same)
	(fc1): Linear(in_features=1024, out_features=120, bias=True)
	(fc2): Linear(in_features=120, out_features=120, bias=True)
	(fc3): Linear(in_features=120, out_features=10, bias=True)
)
```

**CNN - valid 3x3**

```
CNN_Net_3x3_Valid(
	(conv1): Conv2d(3, 6, kernel_size=(3, 3), stride=(1, 1), padding=valid)
	(pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
	(conv2): Conv2d(6, 16, kernel_size=(3, 3), stride=(1, 1), padding=valid)
	(fc1): Linear(in_features=576, out_features=120, bias=True)
	(fc2): Linear(in_features=120, out_features=120, bias=True)
	(fc3): Linear(in_features=120, out_features=10, bias=True)
)
```

**CNN - same 3x3**

```
CNN_Net_3x3_Same(
	(conv1): Conv2d(3, 6, kernel_size=(3, 3), stride=(1, 1), padding=same)
	(pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
	(conv2): Conv2d(6, 16, kernel_size=(3, 3), stride=(1, 1), padding=same)
	(fc1): Linear(in_features=1024, out_features=120, bias=True)
	(fc2): Linear(in_features=120, out_features=120, bias=True)
	(fc3): Linear(in_features=120, out_features=10, bias=True)
)
```

After running the model, here are the first and last few test accuracies (the rest are at the end of the report in the results section) respective to each epoch (final 3 models):

**Same & Filter 5x5**

```
Epoch=1		Test Accuracy=53.030
Epoch=2		Test Accuracy=62.090
...
Epoch=9		Test Accuracy=65.960
Epoch=10	Test Accuracy=65.910
Finished Training
```

**Valid & Filter 3x3**

```
Epoch=1		Test Accuracy=49.510
Epoch=2		Test Accuracy=56.780
...
Epoch=9		Test Accuracy=63.780
Epoch=10	Test Accuracy=63.130
Finished Training
```

**Same & Filter 3x3**

```
Epoch=1		Test Accuracy=52.040
```

```
Epoch=2     Test Accuracy=59.980
...
Epoch=9     Test Accuracy=64.560
Epoch=10    Test Accuracy=64.250
```

And bellow is figure 5, where these accuracies are plotted with respect to their epoch.

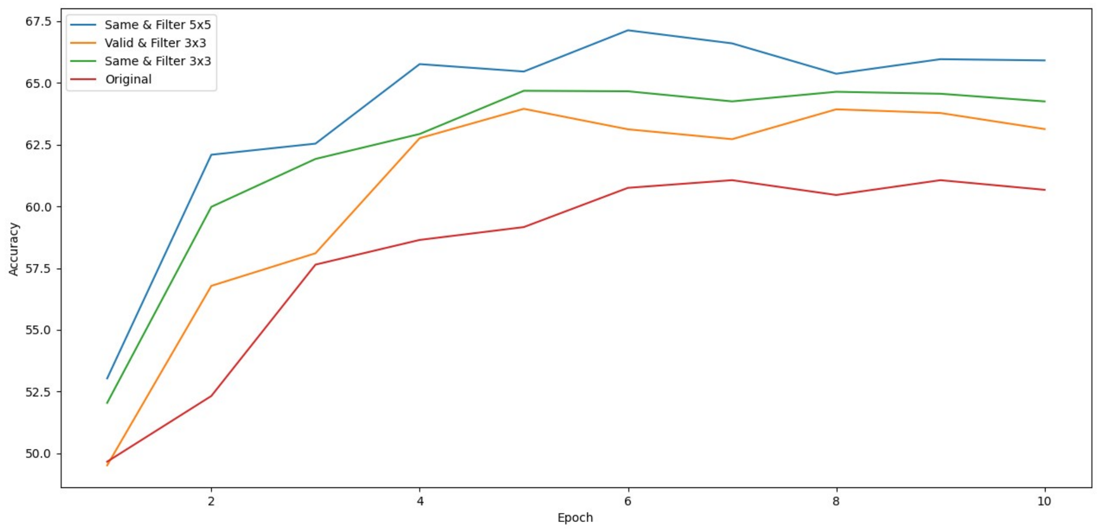

Figure 5. CNNs with different kernel size and padding

The above figure shows how every new combination outperformed the original model. The best performer was the CNN model using same convolutions with a 5x5 kernel. Augmenting the kernel size was likely able to give the model more of the inductive bias that made the original model perform better than the FC networks. With the larger kernel, more neighbouring pixels are being processed by the network at a time. The relationships between these pixels are therefor better understood, because their positional properties are being kept. This is clearly reflected in the figure above, where the best performing model has a large kernel size. However there is a catch, and it being that same convolution must be used with larger kernels.

Using same convolutions allows the model to evaluate every pixel equally, whereas with valid convolutions the pixels on the edges might not be represented properly. This becomes more apparent with a larger kernel size because a larger section of the outer pixels will be seen less than those closer to the centre. Using smaller kernel sizes seems to be a good idea if using valid convolutions, but even when the kernel size is small, same convolutions perform better according to the figure. The worst scenario is using valid convolutions with a larger kernel size. The convolution window won’t slide as much across the image, which causes difficulty for the model to properly understand the image. Though it may better understand the positional properties of the centre pixels, the outer pixels are way less understood, which affects the accuracy drastically.

End of report

## Execution results:

```
(torch) :~/Assignment 01$  python3 run.py
Original CNN (Valid - 5x5 kernel)
Net(
    (conv1): Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1))
    (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (conv2): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
    (fc1): Linear(in_features=400, out_features=120, bias=True)
    (fc2): Linear(in_features=120, out_features=120, bias=True)
    (fc3): Linear(in_features=120, out_features=10, bias=True)
)

FC Net
FC_Net(
  (ffn): Linear(in_features=3072, out_features=10, bias=True)
  (fc_in): Linear(in_features=3072, out_features=120, bias=True)
  (fc): Linear(in_features=120, out_features=120, bias=True)
  (fc_out): Linear(in_features=120, out_features=10, bias=True)
)

CNN with Sigmoid
CNN_Net(
  (conv1): Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1), padding=valid)
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1), padding=valid)
  (fc1): Linear(in_features=400, out_features=120, bias=True)
  (fc1_same): Linear(in_features=1024, out_features=120, bias=True)
  (fc2): Linear(in_features=120, out_features=120, bias=True)
  (fc3): Linear(in_features=120, out_features=10, bias=True)
  (activation): Sigmoid()
)

CNN - same 5x5
CNN_Net_5x5_Same(
  (conv1): Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1), padding=same)
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1), padding=same)
  (fc1): Linear(in_features=1024, out_features=120, bias=True)
  (fc2): Linear(in_features=120, out_features=120, bias=True)
  (fc3): Linear(in_features=120, out_features=10, bias=True)
)

CNN - valid 3x3
CNN_Net_3x3_Valid(
  (conv1): Conv2d(3, 6, kernel_size=(3, 3), stride=(1, 1), padding=valid)
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(6, 16, kernel_size=(3, 3), stride=(1, 1), padding=valid)
  (fc1): Linear(in_features=576, out_features=120, bias=True)
  (fc2): Linear(in_features=120, out_features=120, bias=True)
  (fc3): Linear(in_features=120, out_features=10, bias=True)
)

CNN - same 3x3
CNN_Net_3x3_Same(
  (conv1): Conv2d(3, 6, kernel_size=(3, 3), stride=(1, 1), padding=same)
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(6, 16, kernel_size=(3, 3), stride=(1, 1), padding=same)
  (fc1): Linear(in_features=1024, out_features=120, bias=True)
  (fc2): Linear(in_features=120, out_features=120, bias=True)
  (fc3): Linear(in_features=120, out_features=10, bias=True)
)

cuda:0
Original CNN
Epoch=1        Test Accuracy=49.660
Epoch=2        Test Accuracy=52.320
Epoch=3        Test Accuracy=57.640
Epoch=4        Test Accuracy=58.640
Epoch=5        Test Accuracy=59.160
Epoch=6        Test Accuracy=60.750
Epoch=7        Test Accuracy=61.060
Epoch=8        Test Accuracy=60.460
Epoch=9        Test Accuracy=61.060
Epoch=10       Test Accuracy=60.670
Finished Training


Question 01
FC model (0 hidden layer)
Epoch=1        Test Accuracy=31.630
Epoch=2        Test Accuracy=32.790
Epoch=3        Test Accuracy=34.070
Epoch=4        Test Accuracy=34.140
Epoch=5        Test Accuracy=33.460
Epoch=6        Test Accuracy=35.070
Epoch=7        Test Accuracy=33.110
Epoch=8        Test Accuracy=33.900
Epoch=9        Test Accuracy=32.910
Epoch=10       Test Accuracy=33.050
Finished Training

FC model (1 hidden layer)
Epoch=1        Test Accuracy=47.110
Epoch=2        Test Accuracy=46.890
Epoch=3        Test Accuracy=47.690
Epoch=4        Test Accuracy=50.500
```

```
Epoch=5      Test Accuracy=47.960
Epoch=6      Test Accuracy=49.120
Epoch=7      Test Accuracy=50.180
Epoch=8      Test Accuracy=49.420
Epoch=9      Test Accuracy=48.740
Epoch=10     Test Accuracy=48.620
Finished Training

FC model (2 hidden layer)
Epoch=1      Test Accuracy=47.230
Epoch=2      Test Accuracy=49.180
Epoch=3      Test Accuracy=49.990
Epoch=4      Test Accuracy=50.840
Epoch=5      Test Accuracy=51.340
Epoch=6      Test Accuracy=51.230
Epoch=7      Test Accuracy=51.610
Epoch=8      Test Accuracy=51.890
Epoch=9      Test Accuracy=51.270
Epoch=10     Test Accuracy=51.950
Finished Training

FC model (3 hidden layer)
Epoch=1      Test Accuracy=46.170
Epoch=2      Test Accuracy=48.490
Epoch=3      Test Accuracy=50.200
Epoch=4      Test Accuracy=50.830
Epoch=5      Test Accuracy=50.690
Epoch=6      Test Accuracy=52.050
Epoch=7      Test Accuracy=50.860
Epoch=8      Test Accuracy=52.280
Epoch=9      Test Accuracy=51.750
Epoch=10     Test Accuracy=52.630
Finished Training

FC model (4 hidden layer)
Epoch=1      Test Accuracy=44.110
Epoch=2      Test Accuracy=48.200
Epoch=3      Test Accuracy=50.980
Epoch=4      Test Accuracy=50.440
Epoch=5      Test Accuracy=50.530
Epoch=6      Test Accuracy=51.370
Epoch=7      Test Accuracy=50.510
Epoch=8      Test Accuracy=51.270
Epoch=9      Test Accuracy=50.950
Epoch=10     Test Accuracy=51.160
Finished Training
```

CNN - original

Question 02

CNN - Sigmoid

```
Epoch=1        Test Accuracy=10.000
Epoch=2        Test Accuracy=10.000
Epoch=3        Test Accuracy=10.000
Epoch=4        Test Accuracy=10.000
Epoch=5        Test Accuracy=10.000
Epoch=6        Test Accuracy=10.000
Epoch=7        Test Accuracy=10.000
Epoch=8        Test Accuracy=10.000
Epoch=9        Test Accuracy=18.820
Epoch=10       Test Accuracy=19.500
Finished Training
```

Question 02.5

CNN - Sigmoid

```
Epoch=1        Test Accuracy=34.240
Epoch=2        Test Accuracy=37.160
Epoch=3        Test Accuracy=37.870
Epoch=4        Test Accuracy=40.340
Epoch=5        Test Accuracy=41.910
Epoch=6        Test Accuracy=41.410
Epoch=7        Test Accuracy=43.100
Epoch=8        Test Accuracy=43.840
Epoch=9        Test Accuracy=45.520
Epoch=10       Test Accuracy=40.990
Finished Training
```

Question 03

Same & Filter 5x5

```
Epoch=1        Test Accuracy=53.030
Epoch=2        Test Accuracy=62.090
Epoch=3        Test Accuracy=62.540
Epoch=4        Test Accuracy=65.760
Epoch=5        Test Accuracy=65.460
Epoch=6        Test Accuracy=67.130
Epoch=7        Test Accuracy=66.600
Epoch=8        Test Accuracy=65.370
Epoch=9        Test Accuracy=65.960
Epoch=10       Test Accuracy=65.910
Finished Training
```

Valid & Filter 3x3

```
Epoch=1        Test Accuracy=49.510
Epoch=2        Test Accuracy=56.780
Epoch=3        Test Accuracy=58.100
Epoch=4        Test Accuracy=62.760
```

```
Epoch=5      Test Accuracy=63.950
Epoch=6      Test Accuracy=63.120
Epoch=7      Test Accuracy=62.720
Epoch=8      Test Accuracy=63.930
Epoch=9      Test Accuracy=63.780
Epoch=10     Test Accuracy=63.130
Finished Training

Same & Filter 3x3
Epoch=1      Test Accuracy=52.040
Epoch=2      Test Accuracy=59.980
Epoch=3      Test Accuracy=61.920
Epoch=4      Test Accuracy=62.930
Epoch=5      Test Accuracy=64.680
Epoch=6      Test Accuracy=64.660
Epoch=7      Test Accuracy=64.250
Epoch=8      Test Accuracy=64.640
Epoch=9      Test Accuracy=64.560
Epoch=10     Test Accuracy=64.250
Finished Training
```