Part 1

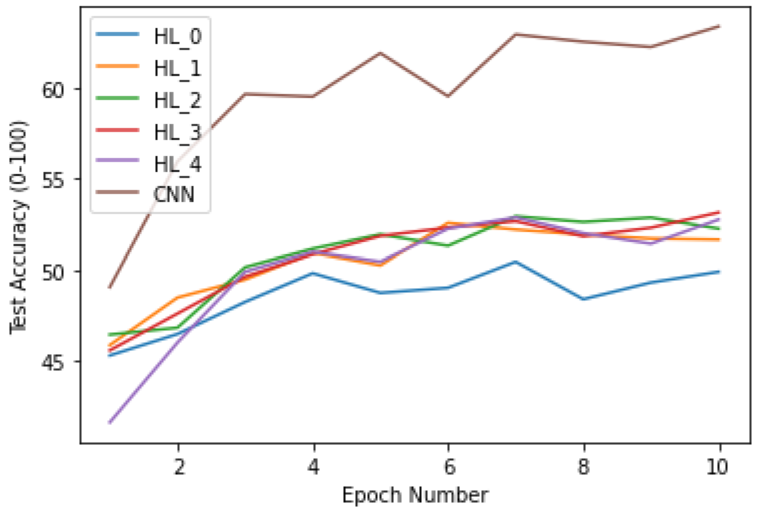


Figure 1: Comparison of a CNN with 4 Simple Networks with a varying number of hidden layers (0-4)

In this figure it is evident that the CNN outperforms all the simple networks without convolution and pooling layers. This is a result of the convolution and pooling layers providing more powerful representations of the raw pixel data, by reducing the number of inputs to the network. This allows the CNN to reduce the high dimensionality of images without losing information. Allowing it to extract information from the data more efficiently and thus train the network more efficiently.

With regard to the simple networks. There is less spread between each simple network than the distance (w.r.t. Test Accuracy) from the simple networks to the CNN. However, it is evident that the network with no hidden layer performs notably worse than the rest. This is a result of there being no non-linear transformation occurring between the first and final layer. As a result the network with no hidden layers is itself a linear classifier. While the other networks with one or more hidden layers are non-linear classifiers. Thus, they have greater capacity to approximate the underlying distribution of the data.

Furthermore, with regard to the other simple networks. We observe a general pattern where the accuracy of the model increases slightly with each additional hidden layer. This is a result of each subsequent model (with more hidden layers) having a greater capacity to fit the data. As a result, these models are able to approximate the underlying distribution on the training data more effectively and thus the test accuracy increases.

The only exception to this rule were the networks with 3 and 4 hidden layers respectively. Where the network with 3 hidden layers outperforms the network with 4 hidden layers. This is likely an indication that the network with 4 hidden layers needs more training time (epochs) in order to outperform the simple network with 3 hidden layers. As the capacity of the last network is greater, it may also require more data in order to avoid overfitting to the noise in the training data. Alternatively, it may also require the use of a regularization technique.

Part 2

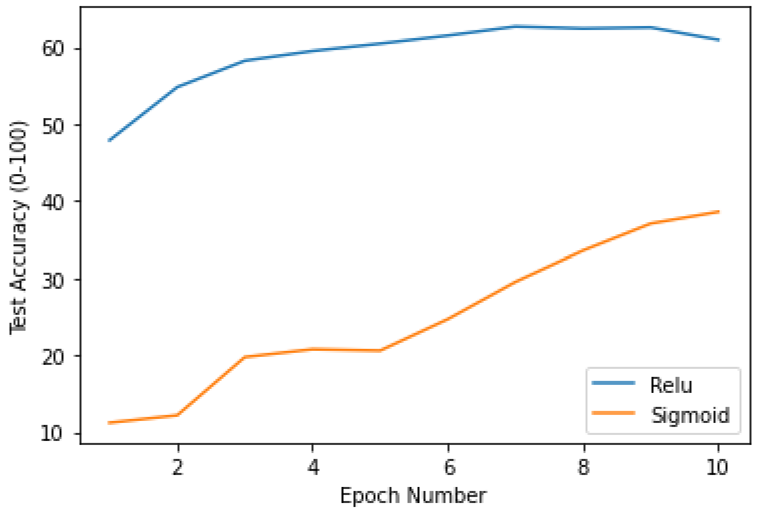


Figure 2: Comparison of using sigmoid and relu activation functions in a CNN

In the second question we see the result of using Sigmoid units vs. using Relu units as the non-linear activation function. As is evident in both the theory and the empirical results, the Relu activation function outperforms the Sigmoid activation function. This is a result of the saturation thresholds of both functions when taking the gradient. As a result the Sigmoid function becomes quickly saturated when the input value from the sum over the weights and activations of the previous layer is largely positive or largely negative. As a result the calculated gradient is very small and therefore does not update the previous layers as efficiently.

Alternatively, the Relu activation function is much more sensitive to the calculated gradient as it is a linear function when positive. Therefore, so long as the initial value is not less than 0, the Relu function will more efficiently update the model parameters and therefore, more quickly learn the parameters that best fit the data.

Part 3

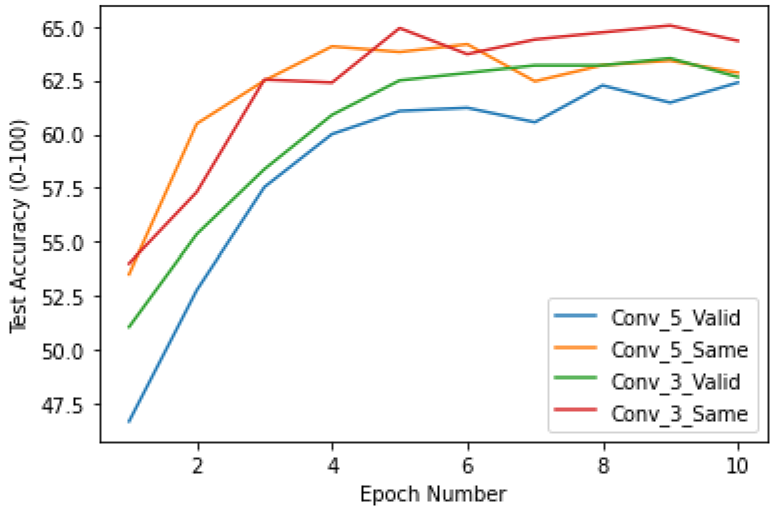
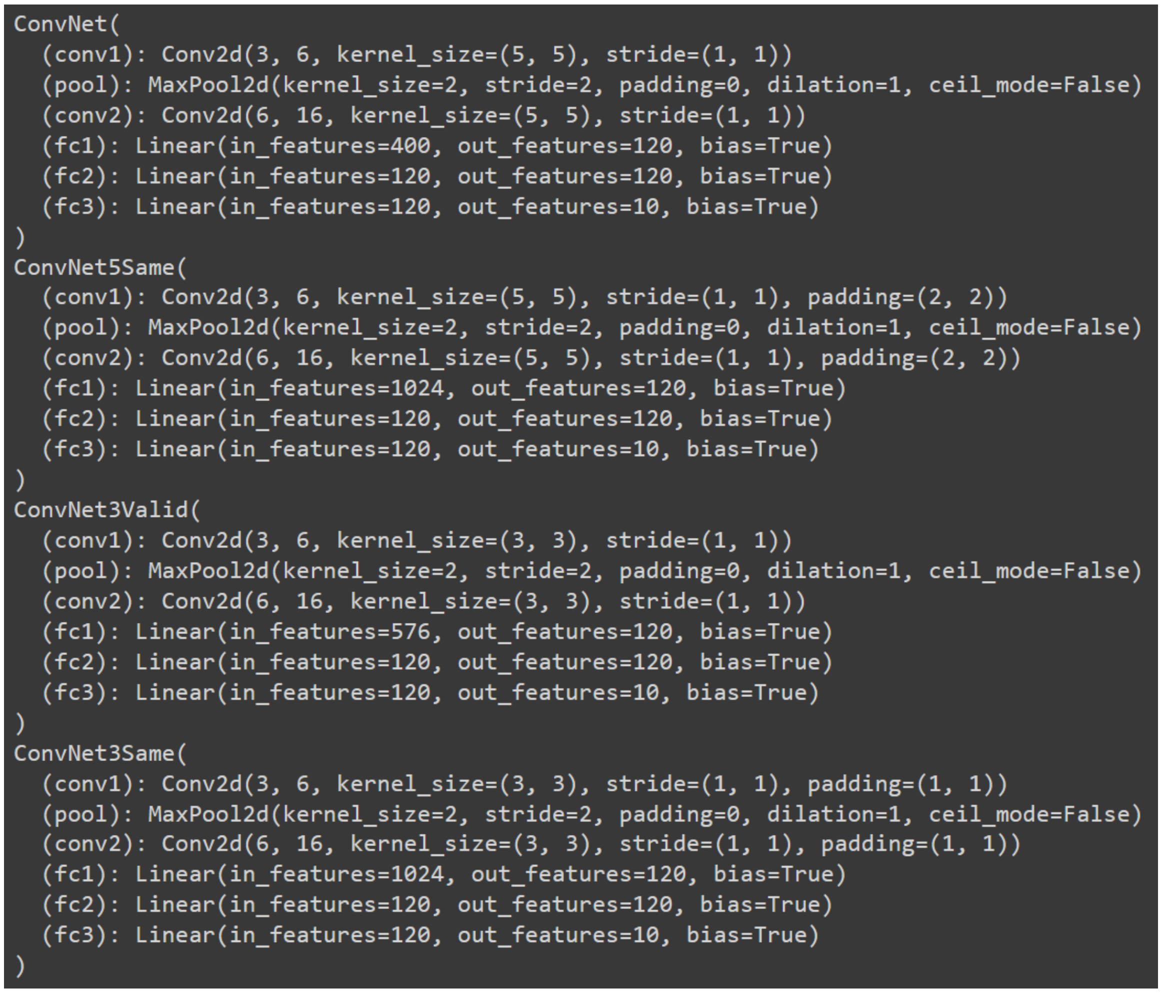


Figure 3: Comparison of CNN models with varying kernel and zero-padding sizes

In the final question we see the result of using four different architectures, by varying the convolution filter and padding size. It is clear from the figure that the best performing model is the model with a convolution filter of size 3x3 with zero-padding on the original image. As a result this model does not lose any pixel data from the original convolution. It is only in the max-pooling layer that a downsampling occurs. By reducing the amount of information lost in the original convolution, the model is able to use more information to approximate the underlying distribution in the training data. Increasing its accuracy on the test data.

The second best performing model is the model with a convolution filter size of 5x5 with zero-padding on the original image. This reinforces the notion that adding zero-padding to the image, allows the model to retain vital pixel data until the max-pooling layer down samples the data.

Finally, the model with a convolution filter size of 3x3 and no zero-padding outperforms the model with convolution size 5x5 without zero-padding. The smaller filter size allows the model to once again retain more pixel data from each convolution. This gives the model a larger amount of data in order to learn the underlying statistical distribution of the classification task on the training set. As the number of epochs is quite low, we noticeably avoid overfitting as the test accuracy tends upwards in general for all models.



```
ConvNet(
  (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)
)
ConvNet5Same(
  (conv1): Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (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=(2, 2))
  (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)
)
ConvNet3Valid(
  (conv1): Conv2d(3, 6, kernel_size=(3, 3), stride=(1, 1))
  (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))
  (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)
)
ConvNet3Same(
  (conv1): Conv2d(3, 6, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (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=(1, 1))
  (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)
)
```

Figure 4: Structure of the four CNN models with varying kernel and zero-padding sizes