Homework 2, STATS 295

The data was generated under the following instructions:

You are tasked with generating simulated data to represent microscopic images of subjects, some of whom have been diagnosed with cancer. The data generation process involves creating a 32x32 matrix for each subject to simulate microscopic images, using the following specifications:

For each subject, a label yi indicates the presence (1) or absence (0) of cancer. This label is generated using a Bernoulli distribution with a probability of 0.5. The image matrix $\boldsymbol{Xi}$ for each subject is produced as

$$\boldsymbol{Xi} = \boldsymbol{Bi} + \boldsymbol{\epsilon i} ,$$

where $\boldsymbol{Bi}$ represents the signal matrix and ϵi is random noise following a normal distribution $N(0, 0.04)$. The signal matrix $\boldsymbol{Bi}$ is generated by first determining the number of microvesicle pixels mi using a Poisson distribution with parameters dependent on the subject's cancer status, and then randomly assigning these pixels in the matrix:

• Generate the number of microvesicle pixels mi : mi follows Poisson distribution with parameter $\mu cyi + \mu n(1 - yi)$. In other words, if the i th subject has cancer, its associated mi follows Poisson(μc); Otherwise, its associated mi follows Poisson(μn). For this simulation, we consider $\mu n = 5$, and $\mu c = 10$, 20, 30, respectively.

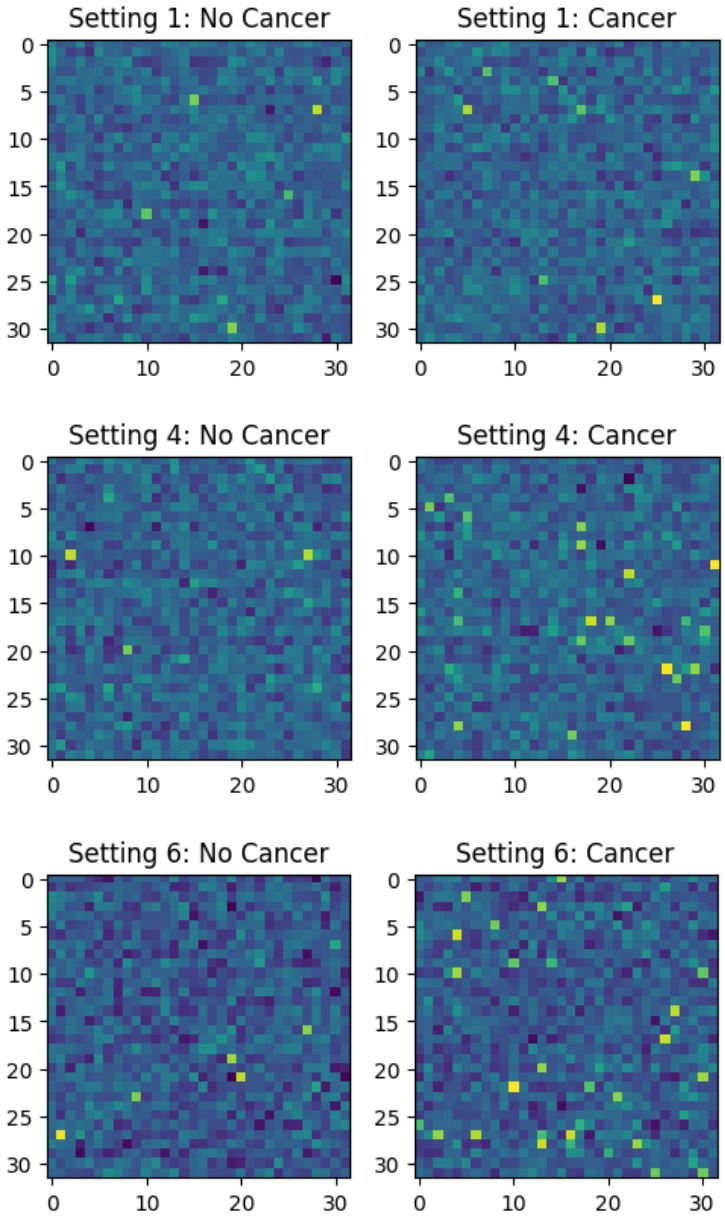
• Randomly select mi pixels from $\boldsymbol{Bi}$ as 1, and set all other pixels as 0.

The following 9 simulation settings were considered for this experiment:

Setting	N	μn	μc
1	200	5	10
2	500	5	10
3	1000	5	10
4	200	5	20
5	500	5	20
6	1000	5	20
7	200	5	30
8	500	5	30
9	500	5	30

1. Data visualization: Visualize the generated images for settings 1, 4, and 7. For each setting, display one image from a subject without cancer $yi = 0$ and one from a subject with cancer $yi = 1$. In total, six images should be visualized, highlighting the differences in the simulated microscopic images between normal and cancerous subjects.

Upon generating the data, I found the following for settings 1, 4, 7. We can clearly see that there are more bright pixels when μc is higher, given the other data generating parameters are fixed (N, μn). (The following code can be found in the appendix under HW2pt1).



2. CNN training: Train a Convolutional Neural Network on the simulated data for each of the nine simulation settings. The goal is to use the CNN to predict the cancer status yi based on the simulated images Xi . Additionally, generate a test set of 1000 subjects using the same data generation process and evaluate the CNN's performance in terms of classification accuracy. You are free to build a CNN with arbitrary hyperparameter settings. Conduct at least 10 independent experiments for each setting by generating new datasets each time, and report the hyperparameters for the CNN, the mean and standard deviation of the classification accuracy achieved by your CNN model.

Given the task to choose arbitrary hyperparameter settings for a CNN, I began investigating different possible architectures that may be suitable as a model. I had four competing models in total, and decided to go with the model that performed the best with a training data of smallest size and low μc value (n = 200, μc = 10; data generation setting 1), and another training data of the biggest size and highest μc value ((n = 1000, μc = 30; data generation setting 9) . For the training process, I made a validation set with the same size as the training data to observe how the model was training. The validating set was not used for training the model, only to see how it was performing after each batch optimizer processing. I considered the model that generalized the best for both data settings in terms of lowest binary cross entropy loss and highest validation accuracy. I used a fixed learning rate of .001, and used an adam optimizer in the training process.

These were the following competing model architectures:

Model 0 (baseline architecture, 1 conv layer):

```
model0 = torch.nn.Sequential()
model0.add_module('conv1', torch.nn.Conv2d(in_channels=1, out_channels=2,
                                           kernel_size = 3, padding = 1))
model0.add_module('relu1', torch.nn.ReLU())
model0.add_module('pool1', torch.nn.MaxPool2d(kernel_size = 2))


model0.add_module('Flatten', torch.nn.Flatten())

model0.add_module('fc1', torch.nn.Linear(512, 10))
model0.add_module('relu1', torch.nn.ReLU())
model0.add_module('fc2', torch.nn.Linear(10, 1))

model0.add_module('sigmoid', torch.nn.Sigmoid())
```

Model 1 (2 conv layers):

```
model1 = torch.nn.Sequential()
model1.add_module('conv1', torch.nn.Conv2d(in_channels=1, out_channels=2,
                                           kernel_size = 3, padding = 1))
model1.add_module('relu1', torch.nn.ReLU())
model1.add_module('pool1', torch.nn.MaxPool2d(kernel_size = 2))

model1.add_module('conv2', torch.nn.Conv2d(in_channels=2, out_channels=4,
                                           kernel_size = 3, padding = 1))
model1.add_module('relu2', torch.nn.ReLU())
model1.add_module('pool2', torch.nn.MaxPool2d(kernel_size = 2))

model1.add_module('Flatten', torch.nn.Flatten())

model1.add_module('fc1', torch.nn.Linear(256, 10))
model1.add_module('relu7', torch.nn.ReLU())
model1.add_module('fc2', torch.nn.Linear(10, 1))

model1.add_module('sigmoid', torch.nn.Sigmoid())
```

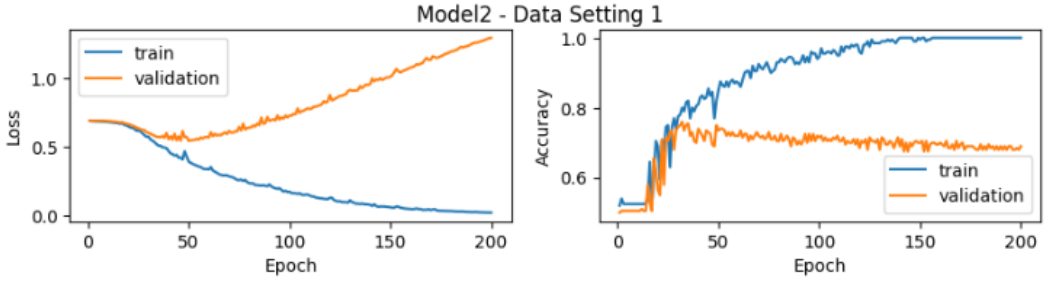
Model 3 (3 conv layers):

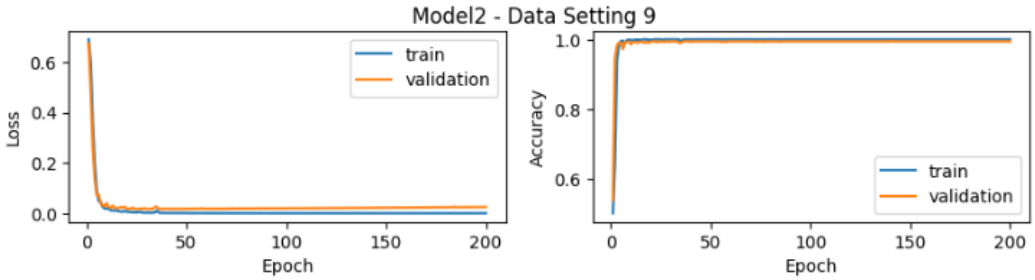
```
model2 = torch.nn.Sequential()
model2.add_module('conv1', torch.nn.Conv2d(in_channels=1, out_channels=2,
                                           kernel_size = 3, padding = 1))
model2.add_module('relu1', torch.nn.ReLU())
model2.add_module('pool1', torch.nn.MaxPool2d(kernel_size = 2))


model2.add_module('conv2', torch.nn.Conv2d(in_channels=2, out_channels=4,
                                           kernel_size = 3, padding = 1))
model2.add_module('relu2', torch.nn.ReLU())
model2.add_module('pool2', torch.nn.MaxPool2d(kernel_size = 2))

model2.add_module('conv3', torch.nn.Conv2d(in_channels=4, out_channels=8,
                                           kernel_size = 3, padding = 1))
model2.add_module('relu3', torch.nn.ReLU())
model2.add_module('pool3', torch.nn.MaxPool2d(kernel_size = 2))
model2.add_module('Flatten', torch.nn.Flatten())
model2.add_module('fc1', torch.nn.Linear(128, 10))
model2.add_module('relu7', torch.nn.ReLU())
model2.add_module('fc2', torch.nn.Linear(10, 1))
model2.add_module('sigmoid', torch.nn.Sigmoid())
```

Model 4 (2 conv layers with more filters)

```
model3 = torch.nn.Sequential()
model3.add_module('conv1', torch.nn.Conv2d(in_channels=1, out_channels=32,
                                           kernel_size = 3, padding = 1))
model3.add_module('relu1', torch.nn.ReLU())
model3.add_module('pool1', torch.nn.MaxPool2d(kernel_size = 2))

model3.add_module('conv2', torch.nn.Conv2d(in_channels=32, out_channels=64,
                                           kernel_size = 3, padding = 1))
model3.add_module('relu2', torch.nn.ReLU())
model3.add_module('pool2', torch.nn.MaxPool2d(kernel_size = 2))

model3.add_module('Flatten', torch.nn.Flatten())

model3.add_module('fc1', torch.nn.Linear(4096, 10))
model3.add_module('relu7', torch.nn.ReLU())
model3.add_module('fc2', torch.nn.Linear(10, 1))

model3.add_module('sigmoid', torch.nn.Sigmoid())
```

I selected Model 2 to run all the different data generation settings due to how it performed with both data settings 1 and 9. As we can see with the Model2 architecture, the model was learning pretty well on data setting 1, with the model performing above baseline and peaking around .76 validation accuracy before eventually overfitting on the training data. When training on data setting 9, we see that the model converges and generalizes well on the validation data due to the rich information of the dataset. Therefore, I proceeded to run the experiment using this model architecture. (The full process of this investigation can be found in the appendix under HW1pt1).



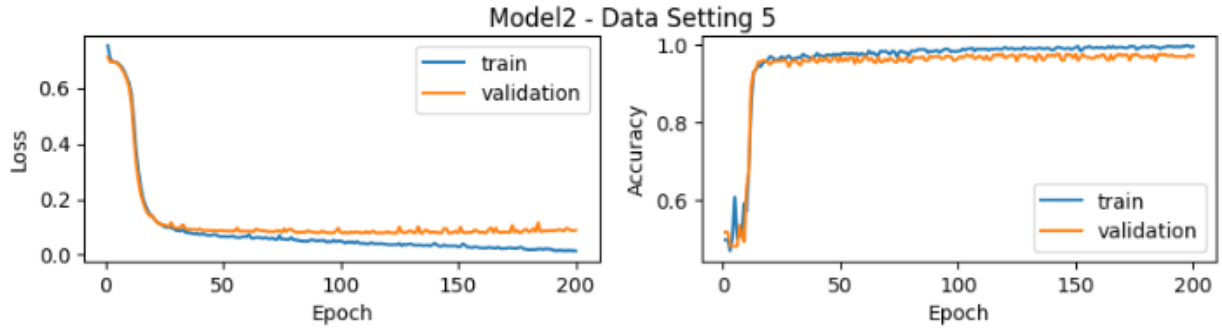


After making the previous observation, I made ten new datasets for all nine different generating settings along with corresponding validation sets of the same size. So overall, I trained the model ten times per each data generation setting. I saved the model weights that obtained the lowest binary cross entropy loss with the validation set, and considered that as the best model for that training cycle. I then generated test sets (n = 1000) for each training cycle, and evaluated the model performance with it. After finishing the training of the datasets, I obtained the mean and standard deviation of the test set accuracy scores. These are the following results for each data setting after repeating the training and test evaluation process ten times:

	n	mu_n	mu_c	mean_accuracy	std_accuracy
0	200	5	10	0.6604	0.104083
1	500	5	10	0.7303	0.072259
2	1000	5	10	0.6649	0.129779
3	200	5	20	0.9505	0.008559
4	500	5	20	0.8718	0.183053
5	1000	5	20	0.9645	0.007393
6	200	5	30	0.8897	0.201759
7	500	5	30	0.8923	0.198259
8	1000	5	30	0.9457	0.142924

Surprisingly, under the model architecture, dataset setting 5 achieved highest performance after ten training cycles with an average test score of .9645 and a small standard deviation of .007. I would have expected dataset 9 (accuracy = .9457) to obtain a better score due to more information provided in the dataset, but I didn't obtain that result in this experiment. Overall, we can observe that those with low information quality with μc = 10 had lower test accuracies compared to the rest of the data. Additionally, for each fixed μc and μn combination, the dataset with n = 1000 always obtained a better generalization than that on those with n = 200 (although not always substantial). (The code for the training process can

be found in the appendix under HW2pt1, and the code for the test set evaluation scores can be found under HW2pt2). The following is an example of 1 of 10 training cycles of data setting 5:



(The training of the models can be found in appendix under HW2pt2, and the test evaluation scores can be found under HW2pt3).

3. CNN tuning: Focus on simulation setting 7 to optimize your CNN for improved classification performance on the test set. Generate an independent validation set of 1000 subjects to assist in tuning. Define a search space for the number of convolution layers, the width of the final fully connected layer, and the optimizer's learning rate. Conduct a grid search over the defined search space to identify the best hyperparameter combination. Report the search space, selected hyperparameters, and the classification accuracy for each combination tested.

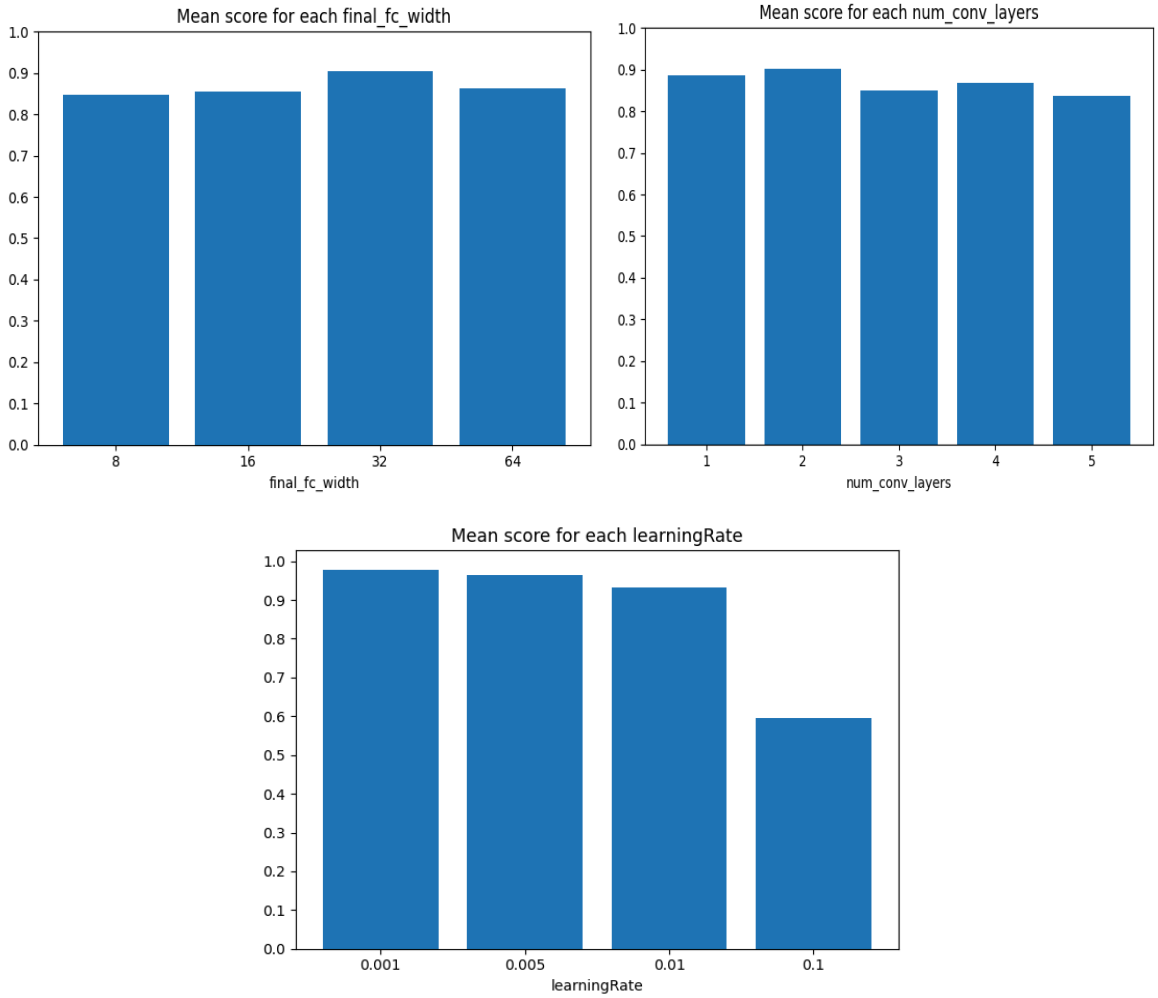
I followed the same training cycle process when tuning for dataset 7. Along with its training set, I generated a validation and test set (n = 1000 each). I compared across different hyperparameters for the model: the number of convolutional layers, width of the final fully connected layer, and the optimizer's learning rate. For convolutional layers, the architecture was very similar to that used in problem 2. I just increased the number of output channels by 2 every time a new layer was added, and made the last fully connected layer a parameter as well. The grid search consisted of the following:

```
{'final_fc_width': [8, 16, 32, 64],
 'num_conv_layers': [1, 2, 3, 4, 5],
 'learningRate': [0.001, 0.005, 0.01, 0.1]}
```

I selected the best model as I did previously: while training on the training set, the model weights that yielded that lowest binary cross entropy loss on the validation set was considered the best model. The test set was then used to evaluate the model's final performance. The following was the best hyperparameters for the model:

Model for dataset 7 performance peaked when we used two convolutional layers, had a final fully connected layer of 16 nodes, and used a learning rate of .005 or .01 for the adam optimizer (two parameter combinations tied). The test accuracy was .998 for these two hyperparameter combination settings. There were also a few instances where the accuracy was .997 for some parameter combinations. When observing the average test accuracy for each feature (bar plots below), we can see that a final width of 32, 2 convolutional layers, and a learning rate of .001 obtained the highest scores, respectively;

however, the combination of all of these features did not yield the best accuracy, but was pretty close (accuracy of .995). (The training and full accuracy table of these parameter combinations can be found in the appendix under HWpt4). The scores for all parameter combinations can be found in the next page.



This is the full table for the different hyperparameter combinations.

final_fc_width	num_conv_layers	learningRate	accuracy
8	1	0.001	0.947
8	1	0.005	0.988
8	1	0.01	0.979
8	1	0.1	0.97
8	2	0.001	0.997
8	2	0.005	0.997
8	2	0.01	0.993
8	2	0.1	0.499
8	3	0.001	0.989
8	3	0.005	0.995
8	3	0.01	0.499
8	3	0.1	0.664
8	4	0.001	0.993
8	4	0.005	0.988
8	4	0.01	0.992
8	4	0.1	0.499
8	5	0.001	0.984
8	5	0.005	0.499
8	5	0.01	0.994
8	5	0.1	0.501
16	1	0.001	0.838
16	1	0.005	0.941
16	1	0.01	0.499

16	**1**	**0.1**	**0.904**
16	**2**	**0.001**	**0.997**
16	**2**	**0.005**	**0.998**
16	**2**	**0.01**	**0.998**
16	**2**	**0.1**	**0.499**
16	**3**	**0.001**	**0.993**
16	**3**	**0.005**	**0.993**
16	**3**	**0.01**	**0.995**
16	**3**	**0.1**	**0.499**
16	**4**	**0.001**	**0.992**
16	**4**	**0.005**	**0.985**
16	**4**	**0.01**	**0.983**
16	**4**	**0.1**	**0.499**
16	**5**	**0.001**	**0.992**
16	**5**	**0.005**	**0.995**
16	**5**	**0.01**	**0.995**
16	**5**	**0.1**	**0.499**
32	**1**	**0.001**	**0.963**
32	**1**	**0.005**	**0.986**
32	**1**	**0.01**	**0.863**
32	**1**	**0.1**	**0.927**
32	**2**	**0.001**	**0.995**
32	**2**	**0.005**	**0.99**
32	**2**	**0.01**	**0.997**
32	**2**	**0.1**	**0.972**

32	**3**	**0.001**	**0.995**
32	**3**	**0.005**	**0.991**
32	**3**	**0.01**	**0.994**
32	**3**	**0.1**	**0.499**
32	**4**	**0.001**	**0.989**
32	**4**	**0.005**	**0.992**
32	**4**	**0.01**	**0.99**
32	**4**	**0.1**	**0.499**
32	**5**	**0.001**	**0.98**
32	**5**	**0.005**	**0.989**
32	**5**	**0.01**	**0.992**
32	**5**	**0.1**	**0.499**
64	**1**	**0.001**	**0.96**
64	**1**	**0.005**	**0.973**
64	**1**	**0.01**	**0.931**
64	**1**	**0.1**	**0.501**
64	**2**	**0.001**	**0.993**
64	**2**	**0.005**	**0.995**
64	**2**	**0.01**	**0.995**
64	**2**	**0.1**	**0.499**
64	**3**	**0.001**	**0.996**
64	**3**	**0.005**	**0.989**
64	**3**	**0.01**	**0.995**
64	**3**	**0.1**	**0.499**
64	**4**	**0.001**	**0.99**

64	**4**	**0.005**	**0.993**
64	**4**	**0.01**	**0.988**
64	**4**	**0.1**	**0.499**
64	**5**	**0.001**	**0.991**
64	**5**	**0.005**	**0.993**
64	**5**	**0.01**	**0.989**
64	**5**	**0.1**	**0.499**

APPENDIX

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.nn.init as init


from torch.utils.data import Dataset, DataLoader
```

Data generation: You are tasked with generating simulated data to represent microscopic images of subjects, some of whom have been diagnosed with cancer. The data generation process involves creating a 32x32 matrix for each subject to simulate microscopic images, using the following specifications: For each subject, a label yi indicates the presence (1) or absence (0) of cancer. This label is generated using a Bernoulli distribution with a probability of 0.5. The image matrix $\boldsymbol{Xi}$ for each subject is produced as $\boldsymbol{Xi} = \boldsymbol{Bi} + \boldsymbol{\epsilon i}$, where $\boldsymbol{Bi}$ represents the signal matrix and ϵi is random noise following a normal distribution $N(0, 0.04)$. The signal matrix $\boldsymbol{Bi}$ is generated by first determining the number of macrovesicle pixels mi using a Poisson distribution with parameters dependent on the subject's cancer status, and then randomly assigning these pixels in the matrix:

Determine the number of macrovesicle pixels m_i using a poisson distribution with parameters dependent on subject cancer status, and then randomly assigning these pixels in the matrix:

• Generate number of macrovesicle pixels mi: mi follows Poisson distribution with parameter $\mu cyi + \mu n(1 - yi)$. In other words, if the i th subject has cancer, its associated mi follows Poisson(μc); Otherwise, its associated mi follows Poisson(μn). For this simulation, we consider μn = 5, and μc = 10, 20, 30, respectively.

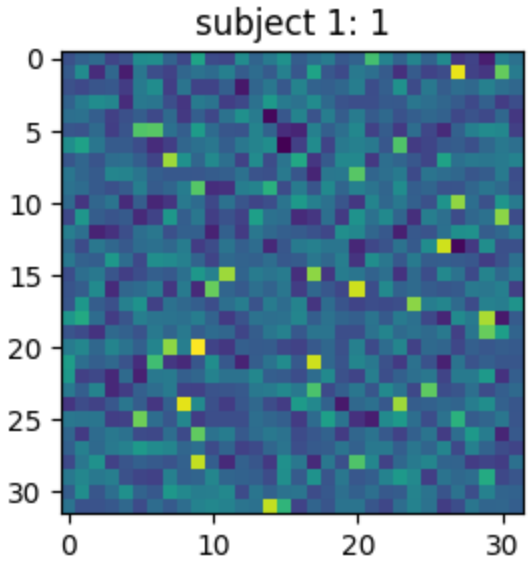
• Randomly select mi pixels from $\boldsymbol{Bi}$ as 1, and set all other pixels as 0.

In [2]:
```
def simulateData(n, mu_c, mu_n):

    y = np.random.choice([0, 1], size = n, p = [0.5, 0.5])
    m_i = np.random.poisson(lam = mu_c, size = n) * y + np.random.poisson(lam = mu_n,

    simulated_data = np.zeros([n, 32, 32])
    for i in range(n):
        random_indices = np.random.choice(32 * 32, m_i[i], replace = False)
        row_indices, col_indices = np.unravel_index(random_indices, (32, 32))
        Bi = np.zeros([32, 32])
        Bi[row_indices, col_indices] = 1
        epsilon_i = np.random.normal(loc = 0, scale = np.sqrt(0.04), size = (32, 32))
        simulated_data[i] = Bi + epsilon_i

    return y, simulated_data
```

In [3]:
```
n = [200, 500, 1000, 200, 500, 1000, 200, 500, 1000]
mu_n = [5, 5, 5, 5, 5, 5, 5, 5, 5]
mu_c = [10, 10, 10, 20, 20, 20, 30, 30, 30]
```

In [4]:
```
simulated_datasets_list = []
simulated_y_list = []
for i in range(9):

    y, simulated_data = simulateData(n = n[i],
                                     mu_c = mu_c[i],
                                     mu_n = mu_n[i])

    simulated_datasets_list.append(simulated_data)
    simulated_y_list.append(y)
```

In [5]:
```
subject = 0
plt.figure(figsize=(3, 3))
plt.imshow(simulated_datasets_list[8][subject, :, :])
plt.title('subject ' + str(subject + 1) + ': ' + str(y[subject]))
plt.show()
```



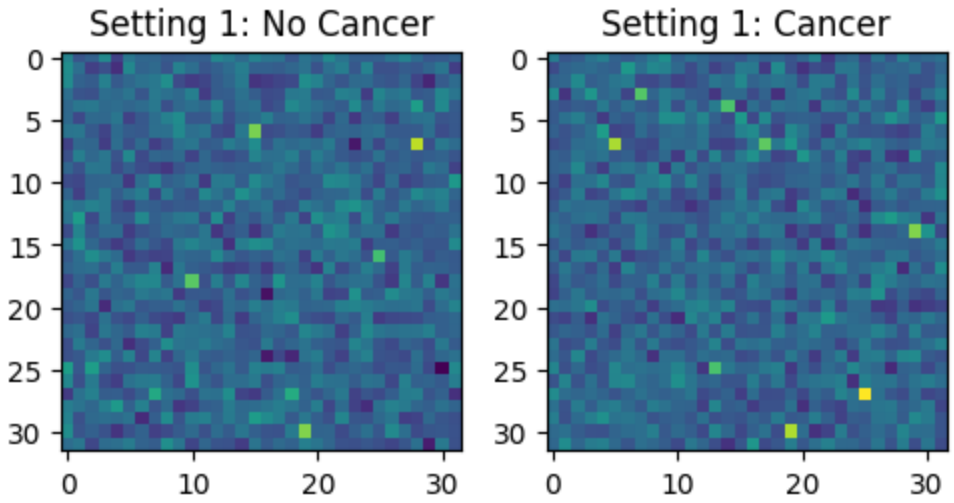
Question 1

Data visualization: Visualize the generated images for settings 1, 4, and 7. For each setting, display one image from a subject without cancer yi = 0 and one from a subject with cancer yi = 1. In total, six images should be visualized, highlighting the differences in the simulated microscopic images between normal and cancerous subjects.

Setting 1: (N μn μc) = (200 5 10)

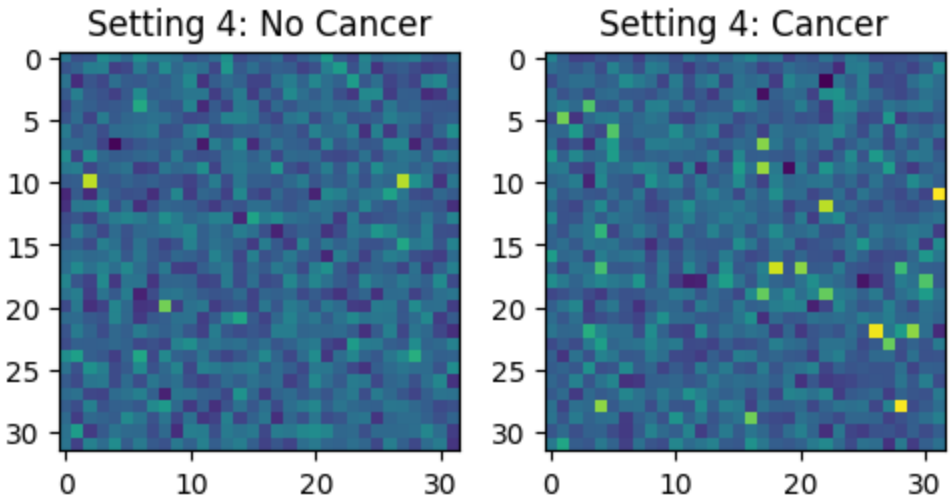
In [6]:
```
setting1_data = simulated_datasets_list[0]
setting1_y = simulated_y_list[0]

indices_0_y = np.where(setting1_y == 0)
setting1_example_0 = setting1_data[indices_0_y][0, :, :]

indices_1_y = np.where(setting1_y == 1)
setting1_example_1 = setting1_data[indices_1_y][0, :, :]

min_range = np.min([setting1_example_0.min(), setting1_example_1.min()])
```

```
max_range = np.max([setting1_example_0.max(), setting1_example_1.max()])

plt.figure(figsize=(5, 5))
plt.subplot(1, 2, 1)
plt.imshow(setting1_example_0, vmin=min_range, vmax=max_range)
plt.title('Setting 1: No Cancer')

plt.subplot(1, 2, 2)
plt.imshow(setting1_example_1, vmin=min_range, vmax=max_range)
plt.title('Setting 1: Cancer')

plt.tight_layout()
```



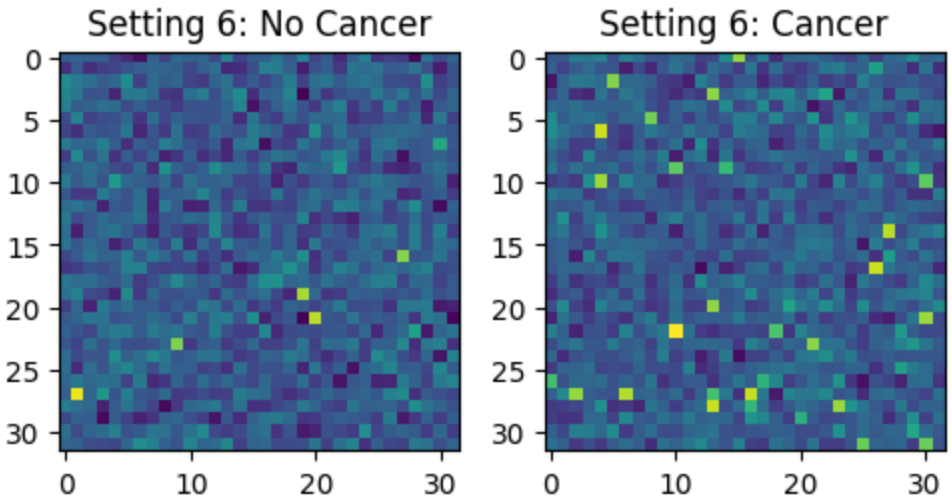
Setting 4: (N μn μc) = (200 5 20)

In [7]:
```
setting4_data = simulated_datasets_list[3]
setting4_y = simulated_y_list[3]

indices_0_y = np.where(setting4_y == 0)
setting4_example_0 = setting4_data[indices_0_y][0, :, :]

indices_1_y = np.where(setting4_y == 1)
setting4_example_1 = setting4_data[indices_1_y][0, :, :]

min_range = np.min([setting4_example_0.min(), setting4_example_1.min()])
max_range = np.max([setting4_example_0.max(), setting4_example_1.max()])

plt.figure(figsize=(5, 5))
plt.subplot(1, 2, 1)
plt.imshow(setting4_example_0, vmin=min_range, vmax=max_range)
plt.title('Setting 4: No Cancer')

plt.subplot(1, 2, 2)
plt.imshow(setting4_example_1, vmin=min_range, vmax=max_range)
plt.title('Setting 4: Cancer')
plt.tight_layout()
```



Setting 7: (N μn μc) = (200 5 30)

In [8]:
```
setting7_data = simulated_datasets_list[6]
setting7_y = simulated_y_list[6]

indices_0_y = np.where(setting7_y == 0)
setting7_example_0 = setting7_data[indices_0_y][0, :, :]

indices_1_y = np.where(setting7_y == 1)
setting7_example_1 = setting7_data[indices_1_y][0, :, :]

min_range = np.min([setting7_example_0.min(), setting7_example_1.min()])
max_range = np.max([setting7_example_0.max(), setting7_example_1.max()])

plt.figure(figsize=(5, 5))
plt.subplot(1, 2, 1)
plt.imshow(setting7_example_0, vmin=min_range, vmax=max_range)
plt.title('Setting 6: No Cancer')

plt.subplot(1, 2, 2)
plt.imshow(setting7_example_1, vmin=min_range, vmax=max_range)
plt.title('Setting 6: Cancer')
plt.tight_layout()
```



Question 2

CNN training: Train a Convolutional Neural Network on the simulated data for each of the nine simulation settings. The goal is to use the CNN to predict the cancer status yi based on the simulated images Xi . Additionally, generate a test set of 1000 subjects using the same data generation process and evaluate the CNN's performance in terms of classification accuracy. You are free to build a CNN with arbitrary hyperparameter setting. Conduct at least 10 independent experiments for each setting by generating new datasets each time, and report the hyperparameters for the CNN, the mean and standard deviation of the classification accuracy achieved by your CNN model.

Data Simulation: Train, Validation, and Test sets

Make test set for each one now. I am also going to make a validation set, the same size as the training set!

In [9]:
```
n_test = 1000
n = [200, 500, 1000, 200, 500, 1000, 200, 500, 1000]
mu_n = [5, 5, 5, 5, 5, 5, 5, 5, 5]
mu_c = [10, 10, 10, 20, 20, 20, 30, 30, 30]
```

In [10]:
```
simulated_val_datasets_list = []
simulated_val_y_list = []
for i in range(9):

    y, simulated_data = simulateData(n = n[i],
                                     mu_c = mu_c[i],
                                     mu_n = mu_n[i])

    simulated_val_datasets_list.append(simulated_data)
    simulated_val_y_list.append(y)
simulated_datasets_list[0].shape, simulated_val_datasets_list[0].shape
```

Out[10]:
```
((200, 32, 32), (200, 32, 32))
```

In [11]:
```
simulated_test_datasets_list = []
simulated_test_y_list = []
for i in range(9):
    y, simulated_data = simulateData(n = n_test,
                                     mu_c = mu_c[i],
                                     mu_n = mu_n[i])

    simulated_test_datasets_list.append(simulated_data)
    simulated_test_y_list.append(y)

simulated_datasets_list[0].shape, simulated_test_datasets_list[0].shape
```

Out[11]:
```
((200, 32, 32), (1000, 32, 32))
```

Example image to see how a single data sample will look like going through the models.

In [12]:
```
exampleImg = simulated_datasets_list[0][0, :, :].reshape([1, 1, 32, 32])
exampleImg = torch.from_numpy(exampleImg).float()
exampleImg.shape
```

Out[12]:
```
torch.Size([1, 1, 32, 32])
```

Going to be making and testing different models!

MODEL 0 - Baseline Model!

In [13]:
```
model0 = torch.nn.Sequential()
model0.add_module('conv1', torch.nn.Conv2d(in_channels=1, out_channels=2, kernel_size
model0.add_module('relu1', torch.nn.ReLU())
model0.add_module('pool1', torch.nn.MaxPool2d(kernel_size = 2))


model0.add_module('Flatten', torch.nn.Flatten())

model0.add_module('fc1', torch.nn.Linear(512, 10))
model0.add_module('relu1', torch.nn.ReLU())
model0.add_module('fc2', torch.nn.Linear(10, 1))

model0.add_module('sigmoid', torch.nn.Sigmoid())

print(model0(exampleImg).shape)
```

```
torch.Size([1, 1])
```

MODEL 1

In [14]:
```
model1 = torch.nn.Sequential()
model1.add_module('conv1', torch.nn.Conv2d(in_channels=1, out_channels=2, kernel_size
model1.add_module('relu1', torch.nn.ReLU())
model1.add_module('pool1', torch.nn.MaxPool2d(kernel_size = 2))

model1.add_module('conv2', torch.nn.Conv2d(in_channels=2, out_channels=4, kernel_size
model1.add_module('relu2', torch.nn.ReLU())
model1.add_module('pool2', torch.nn.MaxPool2d(kernel_size = 2))

model1.add_module('Flatten', torch.nn.Flatten())

model1.add_module('fc1', torch.nn.Linear(256, 10))
model1.add_module('relu7', torch.nn.ReLU())
model1.add_module('fc2', torch.nn.Linear(10, 1))

model1.add_module('sigmoid', torch.nn.Sigmoid())

print(model1(exampleImg).shape)
```

```
torch.Size([1, 1])
```

MODEL 2

In [15]:
```
model2 = torch.nn.Sequential()
model2.add_module('conv1', torch.nn.Conv2d(in_channels=1, out_channels=2, kernel_size
model2.add_module('relu1', torch.nn.ReLU())
model2.add_module('pool1', torch.nn.MaxPool2d(kernel_size = 2))
```

```
model2.add_module('conv2', torch.nn.Conv2d(in_channels=2, out_channels=4, kernel_size
model2.add_module('relu2', torch.nn.ReLU())
model2.add_module('pool2', torch.nn.MaxPool2d(kernel_size = 2))


model2.add_module('conv3', torch.nn.Conv2d(in_channels=4, out_channels=8, kernel_size
model2.add_module('relu3', torch.nn.ReLU())
model2.add_module('pool3', torch.nn.MaxPool2d(kernel_size = 2))


model2.add_module('Flatten', torch.nn.Flatten())

model2.add_module('fc1', torch.nn.Linear(128, 10))
model2.add_module('relu7', torch.nn.ReLU())
model2.add_module('fc2', torch.nn.Linear(10, 1))

model2.add_module('sigmoid', torch.nn.Sigmoid())

print(model2(exampleImg).shape)
```

```
torch.Size([1, 1])
```

MODEL 3

In [16]:
```
model3 = torch.nn.Sequential()
model3.add_module('conv1', torch.nn.Conv2d(in_channels=1, out_channels=32, kernel_size
model3.add_module('relu1', torch.nn.ReLU())
model3.add_module('pool1', torch.nn.MaxPool2d(kernel_size = 2))

model3.add_module('conv2', torch.nn.Conv2d(in_channels=32, out_channels=64, kernel_siz
model3.add_module('relu2', torch.nn.ReLU())
model3.add_module('pool2', torch.nn.MaxPool2d(kernel_size = 2))

model3.add_module('Flatten', torch.nn.Flatten())

model3.add_module('fc1', torch.nn.Linear(4096, 10))
model3.add_module('relu7', torch.nn.ReLU())
model3.add_module('fc2', torch.nn.Linear(10, 1))

model3.add_module('sigmoid', torch.nn.Sigmoid())

print(model3(exampleImg).shape)
```

```
torch.Size([1, 1])
```

DataLoaders for training and validation sets

In [17]:
```
class dataSetPytorch(Dataset):
    def __init__(self, x, y):
        self.x = torch.from_numpy(x.reshape([-1, 1, 32, 32])).float()
        self.y = torch.from_numpy(y)
    def __len__(self):
        return len(self.x)
    def __getitem__(self, idx):
        return self.x[idx], self.y[idx]

# Setting 1: train and validation loader data.
```

```
datasetSetting1_train = dataSetPytorch(simulated_datasets_list[0], simulated_y_list[0]
dataLoader_setting1_train = DataLoader(datasetSetting1_train, batch_size=25, shuffle =

datasetSetting1_validation = dataSetPytorch(simulated_val_datasets_list[0], simulated_
dataLoader_setting1_validation = DataLoader(datasetSetting1_validation, batch_size=25,

# Setting 9: train and validation loader data.
datasetSetting9_train = dataSetPytorch(simulated_datasets_list[8], simulated_y_list[8]
dataLoader_setting9_train = DataLoader(datasetSetting9_train, batch_size=25, shuffle =

datasetSetting9_validation = dataSetPytorch(simulated_val_datasets_list[8], simulated_
dataLoader_setting9_validation = DataLoader(datasetSetting9_validation, batch_size=25,
```

In [18]:
```
datax, labely = next(iter(dataLoader_setting9_train))
print(datax.shape, labely.shape)
model0(datax).shape
```

Out[18]:
```
torch.Size([25, 1, 32, 32]) torch.Size([25])
torch.Size([25, 1])
```

Training the models

In [19]:
```
def reset_weights(model):
    for m in model.modules():
        if isinstance(m, nn.Conv2d) or isinstance(m, nn.Linear):
            init.xavier_uniform_(m.weight)
    return
```

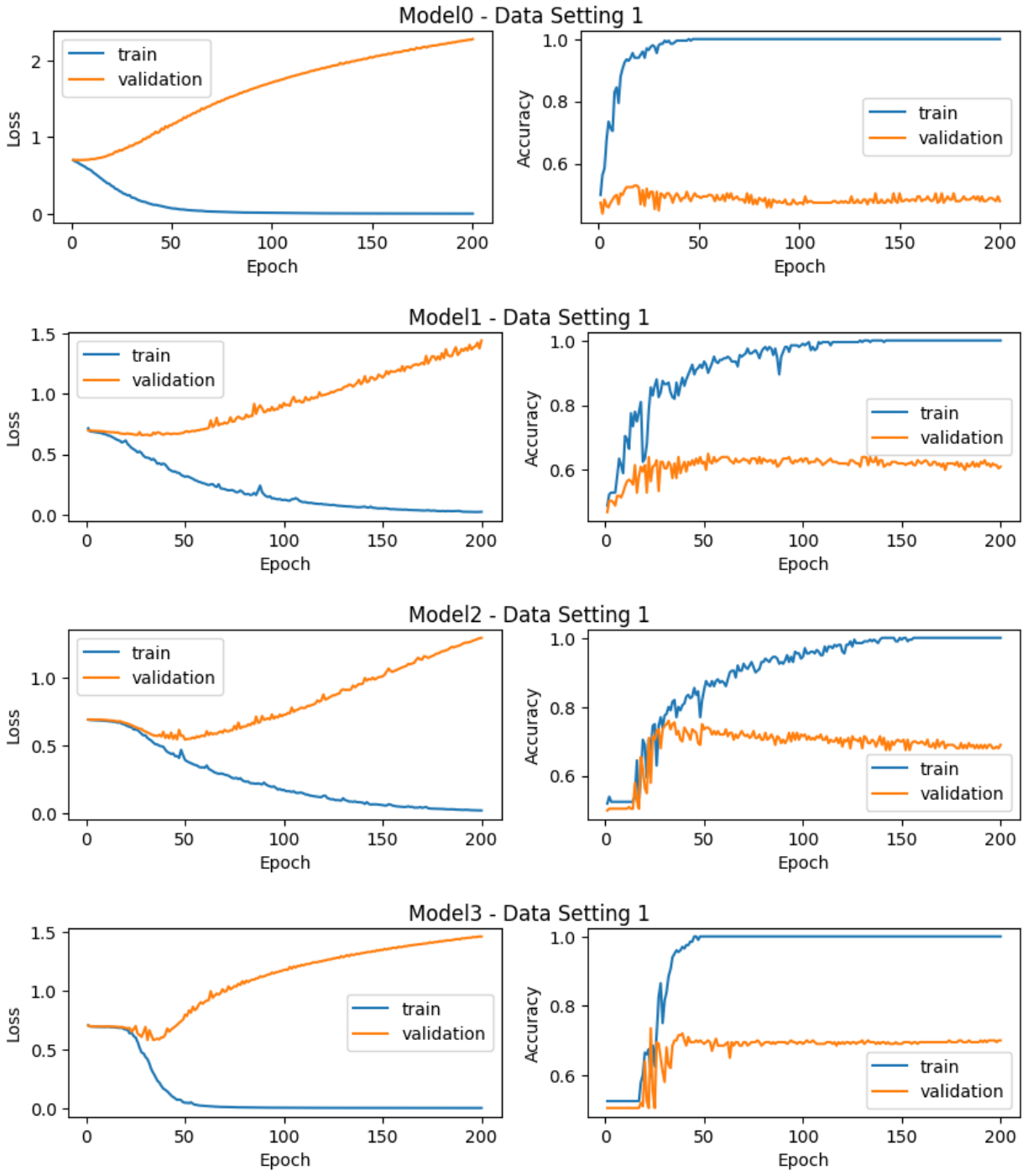
In [20]:
```
def train(name, model, train_dl, valid_dl, num_epochs = 200):
    # reinitialize weights!
    reset_weights(model)

    loss_fn = torch.nn.BCELoss()
    optimizer = torch.optim.Adam(model.parameters(), lr = 0.001)
    loss_hist_train = [0] * num_epochs
    accuracy_hist_train = [0] * num_epochs
    loss_hist_valid = [0] * num_epochs
    accuracy_hist_valid = [0] * num_epochs

    best_loss = torch.inf

    for epoch in range(num_epochs):
        model.train()

        for x_batch, y_batch in train_dl:
            pred = model(x_batch)[:, 0]
            loss = loss_fn(pred, y_batch.float())
#             print("pred", pred, "observed", y_batch)
            loss.backward()
            optimizer.step()
            optimizer.zero_grad()
            loss_hist_train[epoch] += loss.item() * y_batch.size(0)

            is_correct = ((pred >= 0.5).float() == y_batch).float()
#             print(is_correct)
            accuracy_hist_train[epoch] += is_correct.sum()
        loss_hist_train[epoch] /= len(train_dl.dataset)
        accuracy_hist_train[epoch] /= len(train_dl.dataset)
```

```
        model.eval()
        with torch.no_grad():
            for x_batch, y_batch in valid_dl:
                pred = model(x_batch)[:, 0]
                loss = loss_fn(pred, y_batch.float())
                loss_hist_valid[epoch] += loss.item() * y_batch.size(0)

                is_correct = ((pred >= 0.5).float() == y_batch).float()
                accuracy_hist_valid[epoch] += is_correct.sum()
        loss_hist_valid[epoch] /= len(valid_dl.dataset)
        accuracy_hist_valid[epoch] /= len(valid_dl.dataset)


    return loss_hist_train, loss_hist_valid, accuracy_hist_train, accuracy_hist_valid


def plotModelPerformance(modelNum, dataSettingNum, loss_hist_train, loss_hist_valid,
                         accuracy_hist_train, accuracy_hist_valid):
    epochsX = np.arange(len(loss_hist_train)) + 1
    plt.figure(figsize=(10, 2))
    plt.subplot(1, 2, 1)
    plt.plot(epochsX, loss_hist_train, label = "train")
    plt.plot(epochsX, loss_hist_valid, label = "validation")

    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend()

    plt.subplot(1, 2, 2)
    epochsX = np.arange(len(accuracy_hist_train)) + 1
    plt.plot(epochsX, accuracy_hist_train, label = "train")
    plt.plot(epochsX, accuracy_hist_valid, label = "validation")
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.legend()

    plt.suptitle("Model" + str(modelNum) + " - Data Setting " + str(dataSettingNum))
    plt.show()
    return
```

In [21]:

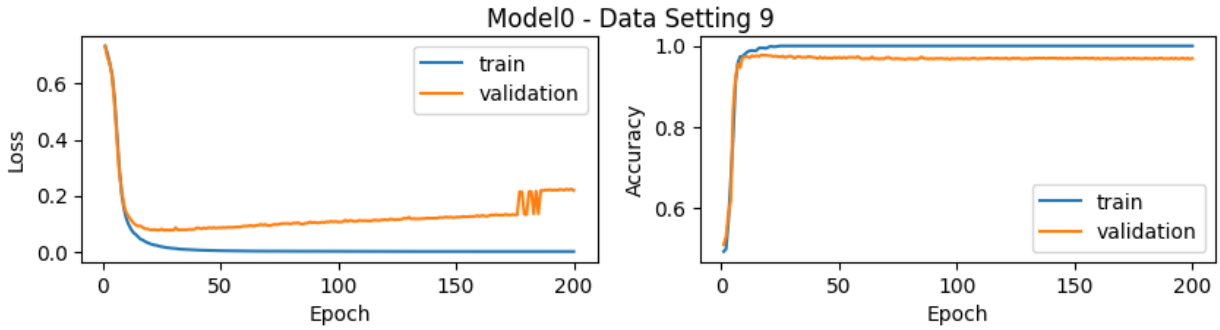
```
loss_hist_train, loss_hist_valid, accuracy_hist_train, accuracy_hist_valid = train("mo
                                                                                   dat
                                                                                   dat
                                                                                   num
plotModelPerformance(modelNum = 0, dataSettingNum = 1, loss_hist_train = loss_hist_tra
                         accuracy_hist_train = accuracy_hist_train, accuracy_hist_vali


loss_hist_train, loss_hist_valid, accuracy_hist_train, accuracy_hist_valid = train("mo
                                                                                   dat
                                                                                   dat
                                                                                   num
plotModelPerformance(modelNum = 1, dataSettingNum = 1, loss_hist_train = loss_hist_tra
                         accuracy_hist_train = accuracy_hist_train, accuracy_hist_vali

loss_hist_train, loss_hist_valid, accuracy_hist_train, accuracy_hist_valid = train("mo
                                                                                   dat
```

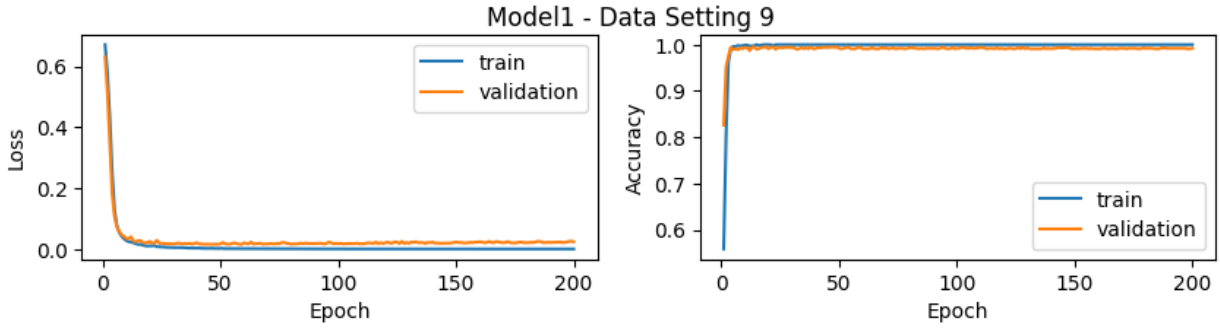
```
                                                                                dat
                                                                                num
plotModelPerformance(modelNum = 2, dataSettingNum = 1, loss_hist_train = loss_hist_tra
                          accuracy_hist_train = accuracy_hist_train, accuracy_hist_vali

loss_hist_train, loss_hist_valid, accuracy_hist_train, accuracy_hist_valid = train("mo
                                                                                dat
                                                                                dat
                                                                                num
plotModelPerformance(modelNum = 3, dataSettingNum = 1, loss_hist_train = loss_hist_tra
                          accuracy_hist_train = accuracy_hist_train, accuracy_hist_vali
```

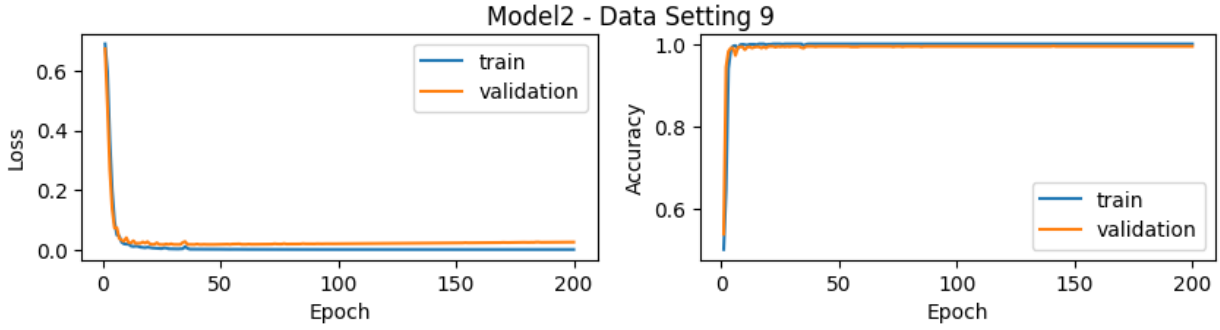


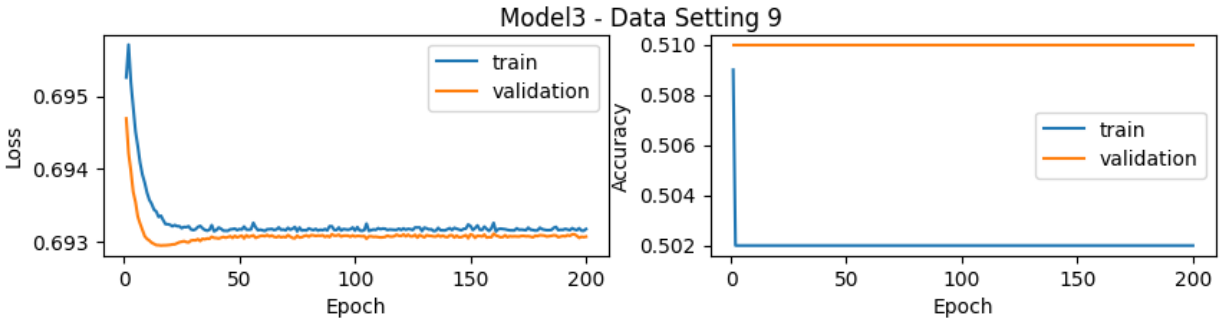
In [22]:
```
loss_hist_train, loss_hist_valid, accuracy_hist_train, accuracy_hist_valid = train("mo
                                                                                dat
                                                                                dat
```

```
                                                                                num
plotModelPerformance(modelNum = 0, dataSettingNum = 9, loss_hist_train = loss_hist_tra
                        accuracy_hist_train = accuracy_hist_train, accuracy_hist_vali


loss_hist_train, loss_hist_valid, accuracy_hist_train, accuracy_hist_valid = train("mo
                                                                                dat
                                                                                dat
                                                                                num
plotModelPerformance(modelNum = 1, dataSettingNum = 9, loss_hist_train = loss_hist_tra
                        accuracy_hist_train = accuracy_hist_train, accuracy_hist_vali

loss_hist_train, loss_hist_valid, accuracy_hist_train, accuracy_hist_valid = train("mo
                                                                                dat
                                                                                dat
                                                                                num
plotModelPerformance(modelNum = 2, dataSettingNum = 9, loss_hist_train = loss_hist_tra
                        accuracy_hist_train = accuracy_hist_train, accuracy_hist_vali

loss_hist_train, loss_hist_valid, accuracy_hist_train, accuracy_hist_valid = train("mo
                                                                                dat
                                                                                dat
                                                                                num
plotModelPerformance(modelNum = 3, dataSettingNum = 9, loss_hist_train = loss_hist_tra
                        accuracy_hist_train = accuracy_hist_train, accuracy_hist_vali
```








Model3 - Data Setting 9
Loss
0.695
0.694
0.693
0
50
100
150
200
Epoch
train
validation
Accuracy
0.510
0.508
0.506
0.504
0.502
0
50
100
150
200
Epoch
train
validation

In [ ]:

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.nn.init as init
import os
from datetime import datetime

from torch.utils.data import Dataset, DataLoader
```

Question 2

CNN training: Train a Convolutional Neural Network on the simulated data for each of the nine simulation settings. The goal is to use the CNN to predict the cancer status yi based on the simulated images Xi . Additionally, generate a test set of 1000 subjects using the same data generation process and evaluate the CNN's performance in terms of classification accuracy. You are free to build a CNN with arbitrary hyperparameter setting. Conduct at least 10 independent experiments for each setting by generating new datasets each time, and report the hyperparameters for the CNN, the mean and standard deviation of the classification accuracy achieved by your CNN model.

MODEL 2

In [3]:
```
model2 = torch.nn.Sequential()
model2.add_module('conv1', torch.nn.Conv2d(in_channels=1, out_channels=2, kernel_size
model2.add_module('relu1', torch.nn.ReLU())
model2.add_module('pool1', torch.nn.MaxPool2d(kernel_size = 2))


model2.add_module('conv2', torch.nn.Conv2d(in_channels=2, out_channels=4, kernel_size
model2.add_module('relu2', torch.nn.ReLU())
model2.add_module('pool2', torch.nn.MaxPool2d(kernel_size = 2))


model2.add_module('conv3', torch.nn.Conv2d(in_channels=4, out_channels=8, kernel_size
model2.add_module('relu3', torch.nn.ReLU())
model2.add_module('pool3', torch.nn.MaxPool2d(kernel_size = 2))


model2.add_module('Flatten', torch.nn.Flatten())

model2.add_module('fc1', torch.nn.Linear(128, 10))
model2.add_module('relu7', torch.nn.ReLU())
model2.add_module('fc2', torch.nn.Linear(10, 1))

model2.add_module('sigmoid', torch.nn.Sigmoid())
```

In [4]:
```
def simulateData(n, mu_c, mu_n):

    y = np.random.choice([0, 1], size = n, p = [0.5, 0.5])
    m_i = np.random.poisson(lam = mu_c, size = n) * y + np.random.poisson(lam = mu_n,
```

```
    simulated_data = np.zeros([n, 32, 32])
    for i in range(n):
        random_indices = np.random.choice(32 * 32, m_i[i], replace = False)
        row_indices, col_indices = np.unravel_index(random_indices, (32, 32))
        Bi = np.zeros([32, 32])
        Bi[row_indices, col_indices] = 1
        epsilon_i = np.random.normal(loc = 0, scale = np.sqrt(0.04), size = (32, 32))
        simulated_data[i] = Bi + epsilon_i

    return y, simulated_data

class dataSetPytorch(Dataset):
    def __init__(self, x, y):
        self.x = torch.from_numpy(x.reshape([-1, 1, 32, 32])).float()
        self.y = torch.from_numpy(y)
    def __len__(self):
        return len(self.x)
    def __getitem__(self, idx):
        return self.x[idx], self.y[idx]

# Use that to make train and validation data here.
def makeDataLoader(numExperiments = 10):
    n = [200, 500, 1000, 200, 500, 1000, 200, 500, 1000]
    mu_n = [5, 5, 5, 5, 5, 5, 5, 5, 5]
    mu_c = [10, 10, 10, 20, 20, 20, 30, 30, 30]

    dataLoader_experiment_data = []
    for experiment in range(numExperiments):
        dataLoader_settings = []
        for setting in range(9):

            y, simulated_data = simulateData(n = n[setting],
                                             mu_c = mu_c[setting],
                                             mu_n = mu_n[setting])

            datasetSetting = dataSetPytorch(simulated_data, y)
            dataLoader = DataLoader(datasetSetting, batch_size=25, shuffle = True)
            dataLoader_settings.append(dataLoader)

        dataLoader_experiment_data.append(dataLoader_settings)

    return dataLoader_experiment_data

def makeTestLoader(numExperiments = 10):
    n_test = 1000
    mu_n = [5, 5, 5, 5, 5, 5, 5, 5, 5]
    mu_c = [10, 10, 10, 20, 20, 20, 30, 30, 30]

    dataLoader_experiment_data = []
    for experiment in range(numExperiments):
        dataLoader_settings = []
        for setting in range(9):

            y, simulated_data = simulateData(n = n_test,
                                             mu_c = mu_c[setting],
                                             mu_n = mu_n[setting])

            datasetSetting = dataSetPytorch(simulated_data, y)
            dataLoader = DataLoader(datasetSetting, batch_size=25, shuffle = True)
            dataLoader_settings.append(dataLoader)
```

```
        dataLoader_experiment_data.append(dataLoader_settings)

    return dataLoader_experiment_data
```

In [5]:
```
dataLoader_all_experiments_train = makeDataLoader(numExperiments = 10)
dataLoader_all_experiments_val = makeDataLoader(numExperiments = 10)
```

Training the models

In [6]:
```
def reset_weights(model):
    for m in model.modules():
        if isinstance(m, nn.Conv2d) or isinstance(m, nn.Linear):
            init.xavier_uniform_(m.weight)
    return

def saveModel(model, path):
    torch.save(model.state_dict(), path)
```

In [7]:
```
def train(name, model, train_dl, valid_dl, num_epochs = 200):
    # reinitialize weights!
    reset_weights(model)

    loss_fn = torch.nn.BCELoss()
    optimizer = torch.optim.Adam(model.parameters(), lr = 0.001)
    loss_hist_train = [0] * num_epochs
    accuracy_hist_train = [0] * num_epochs
    loss_hist_valid = [0] * num_epochs
    accuracy_hist_valid = [0] * num_epochs

    best_loss = torch.inf

    for epoch in range(num_epochs):
        model.train()

        for x_batch, y_batch in train_dl:
            pred = model(x_batch)[:, 0]
            loss = loss_fn(pred, y_batch.float())
#             print("pred", pred, "observed", y_batch)
            loss.backward()
            optimizer.step()
            optimizer.zero_grad()
            loss_hist_train[epoch] += loss.item() * y_batch.size(0)

            is_correct = ((pred >= 0.5).float() == y_batch).float()
#             print(is_correct)
            accuracy_hist_train[epoch] += is_correct.sum()
        loss_hist_train[epoch] /= len(train_dl.dataset)
        accuracy_hist_train[epoch] /= len(train_dl.dataset)

        model.eval()
        with torch.no_grad():
            for x_batch, y_batch in valid_dl:
                pred = model(x_batch)[:, 0]
                loss = loss_fn(pred, y_batch.float())
                loss_hist_valid[epoch] += loss.item() * y_batch.size(0)
```

```
                is_correct = ((pred >= 0.5).float() == y_batch).float()
                accuracy_hist_valid[epoch] += is_correct.sum()
        loss_hist_valid[epoch] /= len(valid_dl.dataset)
        accuracy_hist_valid[epoch] /= len(valid_dl.dataset)

        if (best_loss > loss_hist_valid[epoch]):
            path = name + "_" + datetime.now().strftime("%d_%m_%Y") + "_epoch_" + str(
            saveModel(model, path)
            best_loss = loss_hist_valid[epoch]

    return loss_hist_train, loss_hist_valid, accuracy_hist_train, accuracy_hist_valid


def plotModelPerformance(modelNum, dataSettingNum, loss_hist_train, loss_hist_valid,
                         accuracy_hist_train, accuracy_hist_valid):
    epochsX = np.arange(len(loss_hist_train)) + 1
    plt.figure(figsize=(10, 2))
    plt.subplot(1, 2, 1)
    plt.plot(epochsX, loss_hist_train, label = "train")
    plt.plot(epochsX, loss_hist_valid, label = "validation")

    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend()

    plt.subplot(1, 2, 2)
    epochsX = np.arange(len(accuracy_hist_train)) + 1
    plt.plot(epochsX, accuracy_hist_train, label = "train")
    plt.plot(epochsX, accuracy_hist_valid, label = "validation")
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.legend()

    plt.suptitle("Model" + str(modelNum) + " - Data Setting " + str(dataSettingNum))
    plt.show()
    return
```

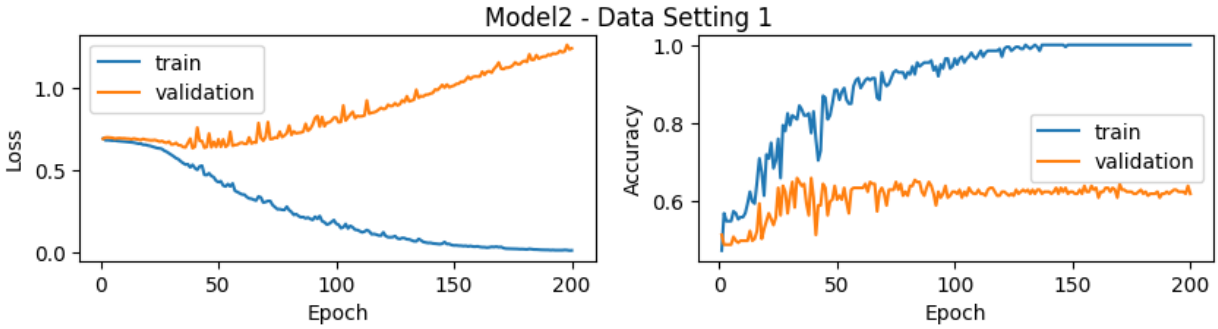
Note: In the print statment its supposed to be print("Experiment:", experiment, "Setting:", setting + 1) but forgot to change it when i ran the long experiment ..

In [9]:

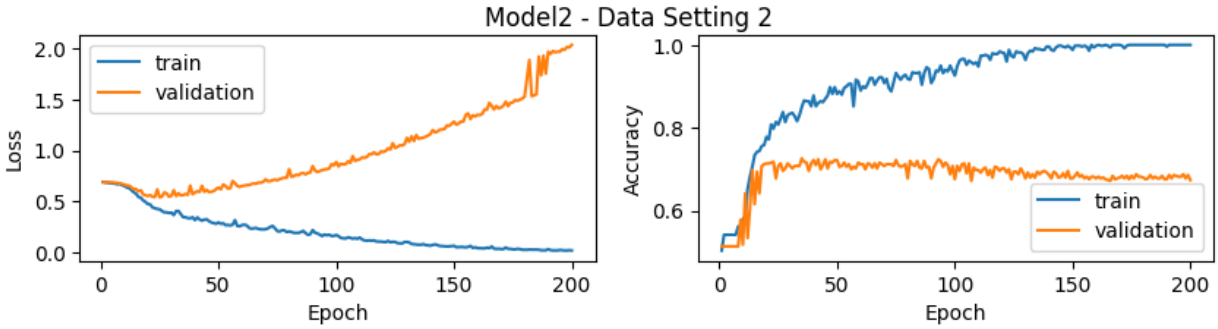
```
numExperiments = 10
numSettings = 9
for experiment in range(numExperiments):
    dataLoader_individual_experiment_train = dataLoader_all_experiments_train[experime
    dataLoader_individual_experiment_val = dataLoader_all_experiments_val[experiment]

    for setting in range(numSettings):
        print("Experiment:", experiment, "Setting:", setting)
        folderPath = 'setting' + str(setting + 1)
        if not os.path.exists(folderPath):
            # Create the folder
            os.makedirs(folderPath)

        n = len(dataLoader_individual_experiment_train[setting].dataset)
        print("N:", n)

        modelName = "./" + folderPath + "/modelSetting" + str(setting + 1) + "_Experim
```

```
        loss_hist_train, loss_hist_valid, accuracy_hist_train, accuracy_hist_valid = t
                                                                    dataLoader_inc
                                                                    dataLoader_inc
                                                                            num
        plotModelPerformance(modelNum = 2, dataSettingNum = setting + 1,
                             loss_hist_train = loss_hist_train, loss_hist_valid = loss_his
                             accuracy_hist_train = accuracy_hist_train, accuracy_hist_vali
```

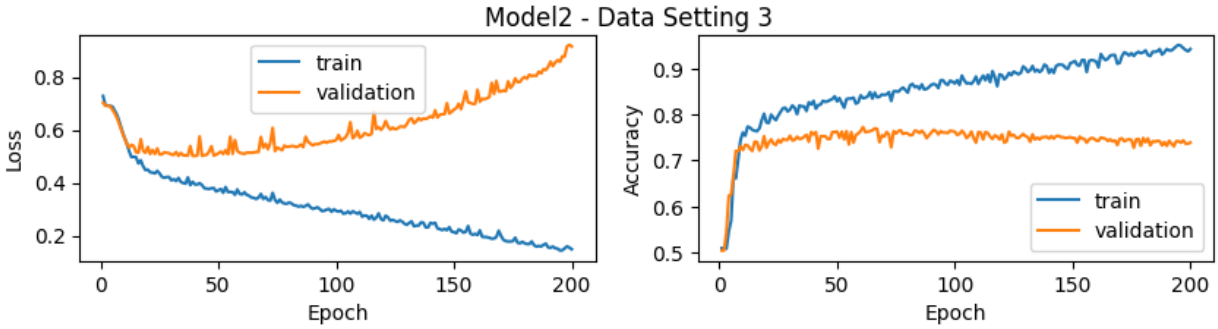
```
Experiment: 0 Setting: 0
N: 200
```



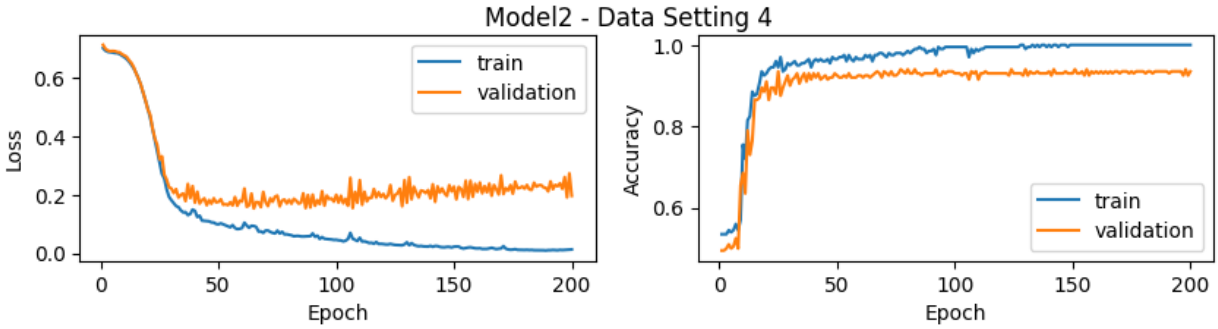

```
Experiment: 0 Setting: 1
N: 500
```



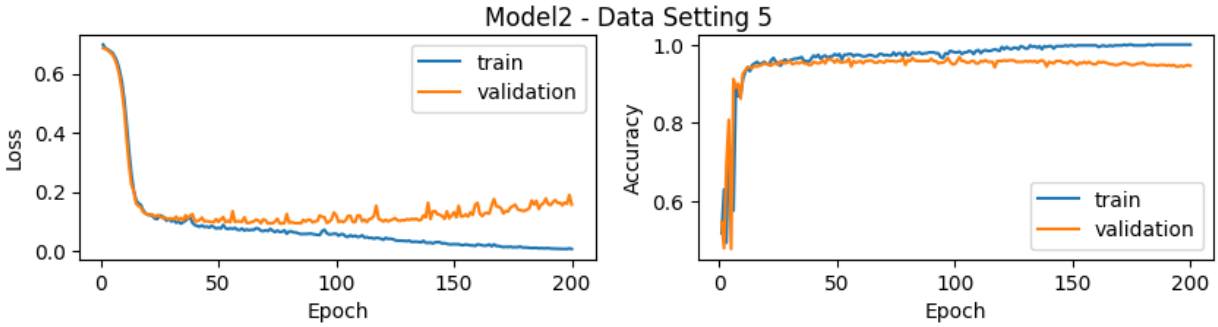

```
Experiment: 0 Setting: 2
N: 1000
```



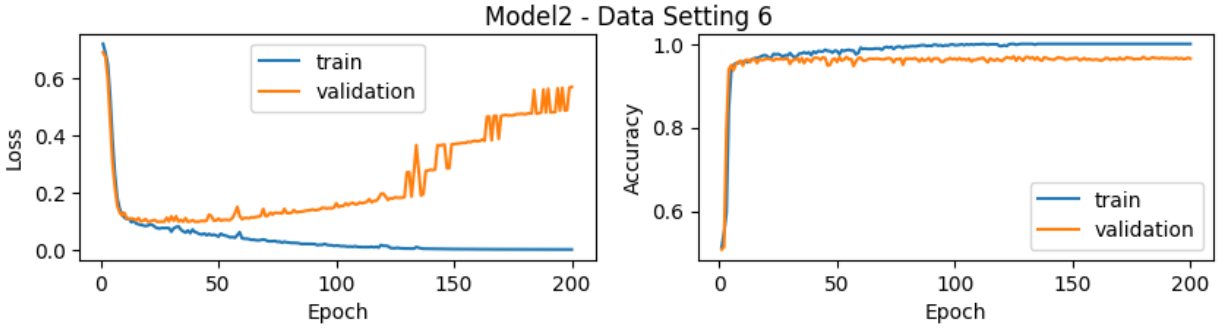

```
Experiment: 0 Setting: 3
N: 200
```



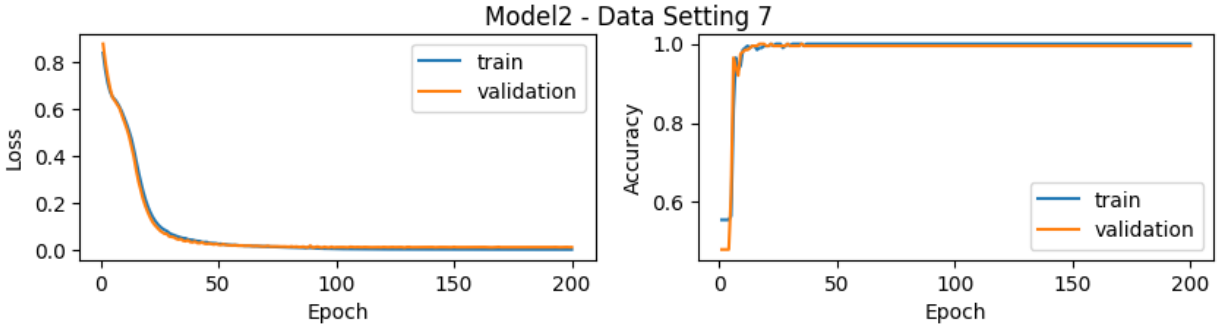
```
Experiment: 0 Setting: 4
N: 500
```



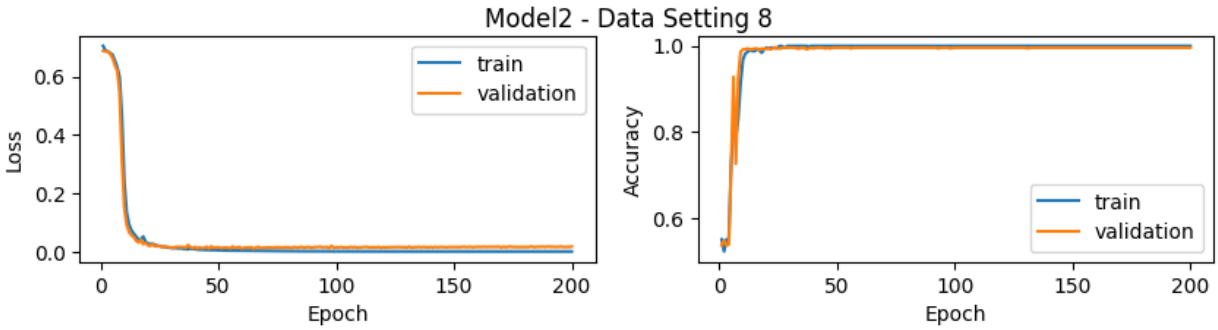

```
Experiment: 0 Setting: 5
N: 1000
```



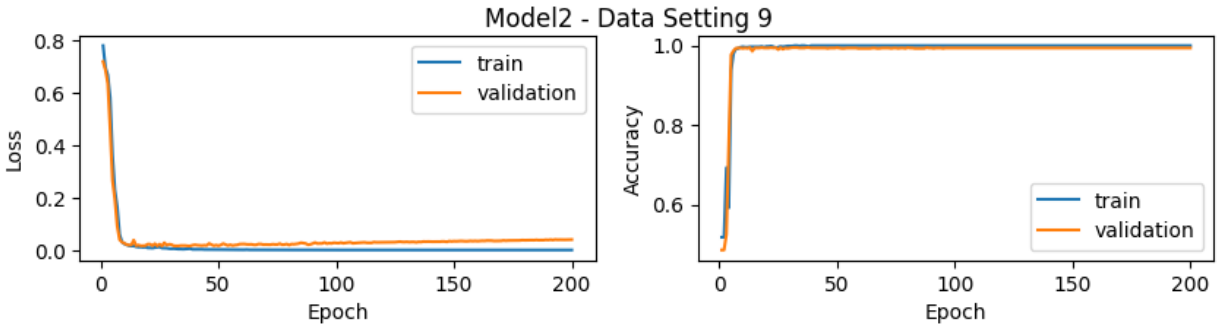

```
Experiment: 0 Setting: 6
N: 200
```



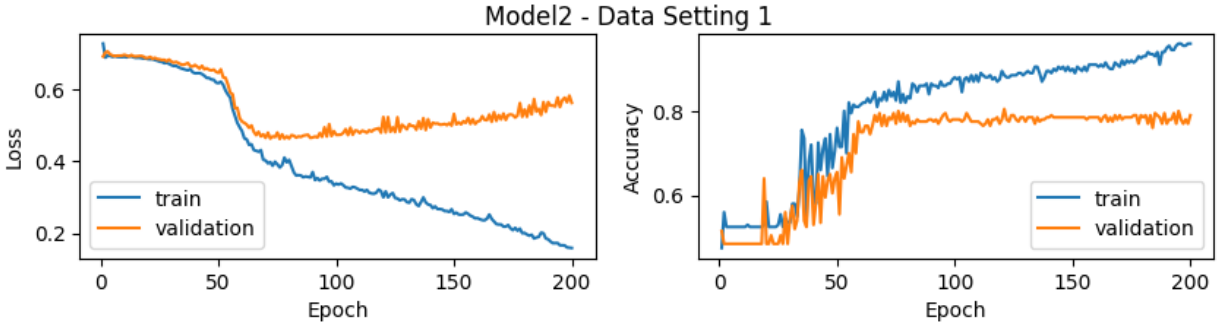

```
Experiment: 0 Setting: 7
N: 500
```



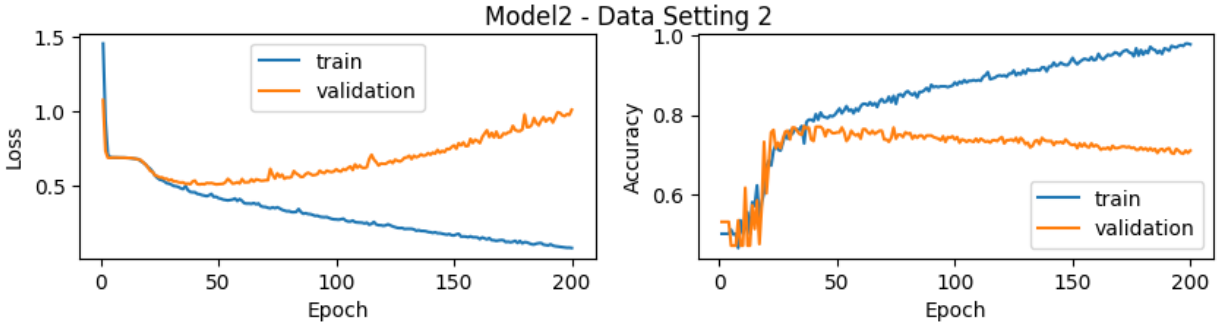

```
Experiment: 0 Setting: 8
N: 1000
```



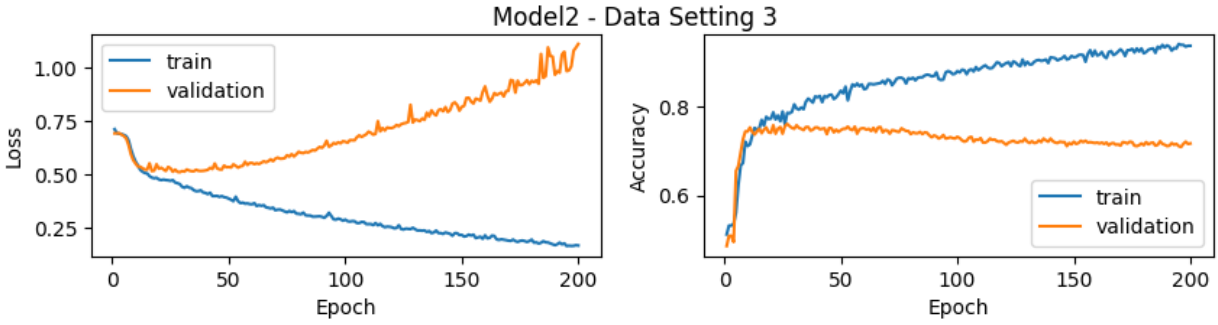

```
Experiment: 1 Setting: 0
N: 200
```



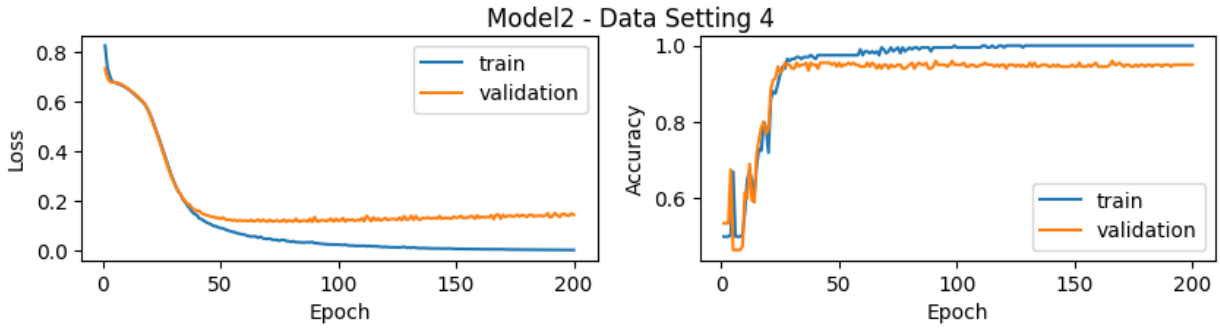

```
Experiment: 1 Setting: 1
N: 500
```



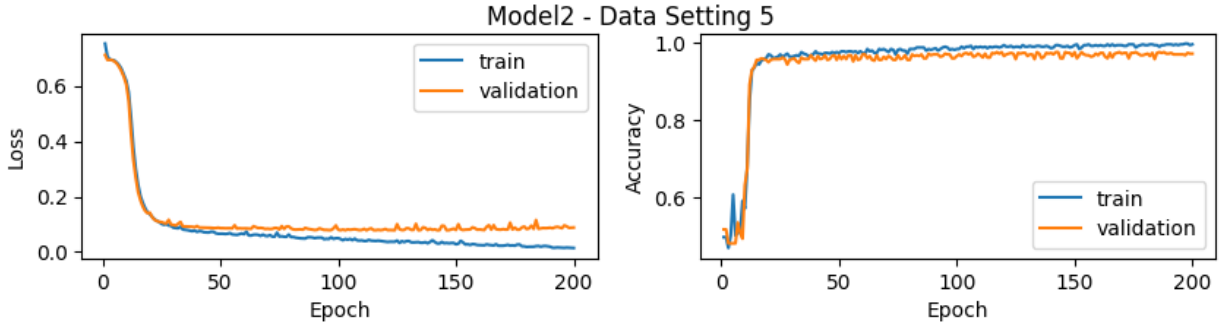

```
Experiment: 1 Setting: 2
N: 1000
```



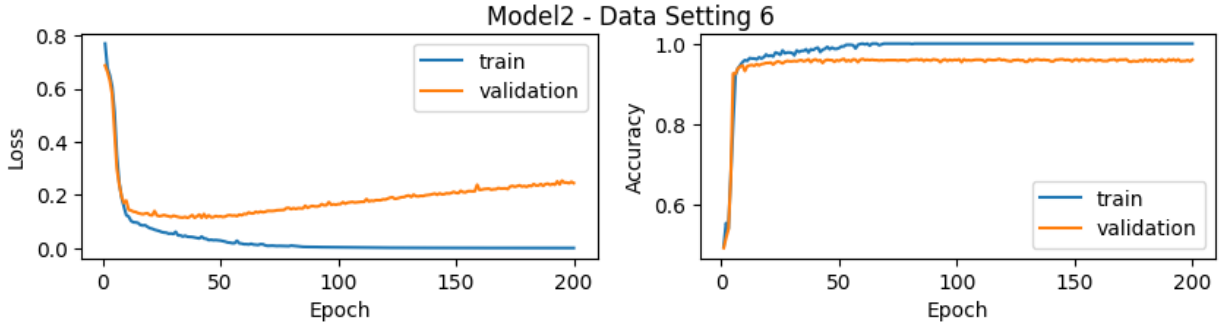

```
Experiment: 1 Setting: 3
N: 200
```



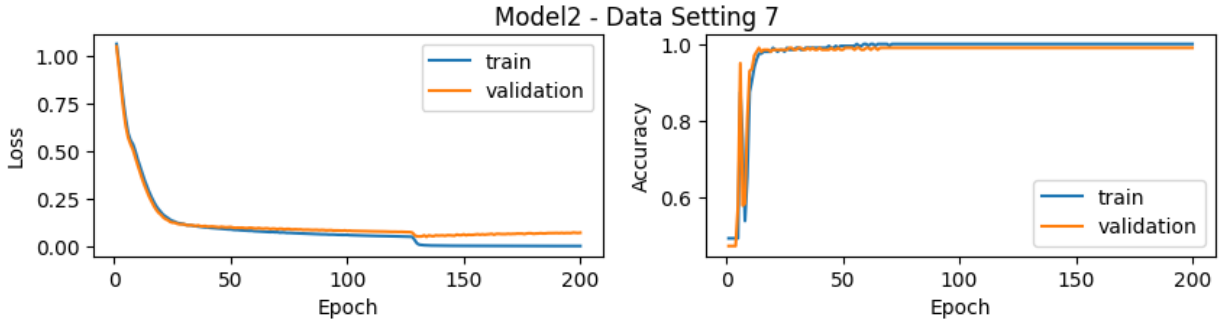

```
Experiment: 1 Setting: 4
N: 500
```



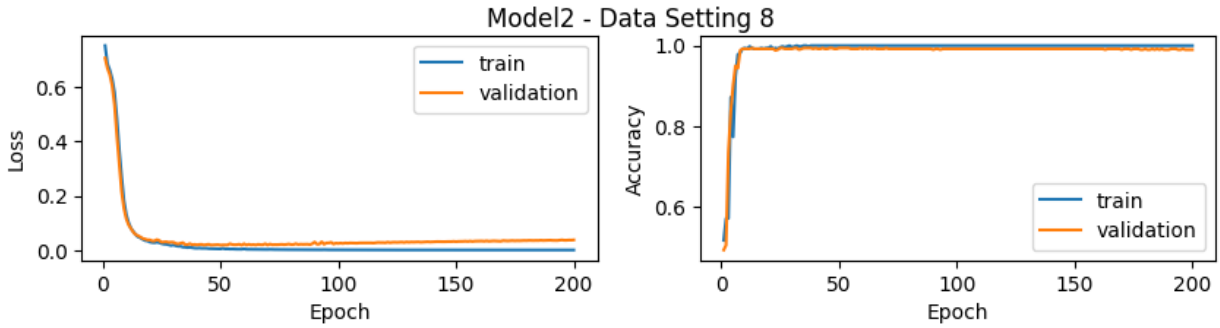

```
Experiment: 1 Setting: 5
N: 1000
```



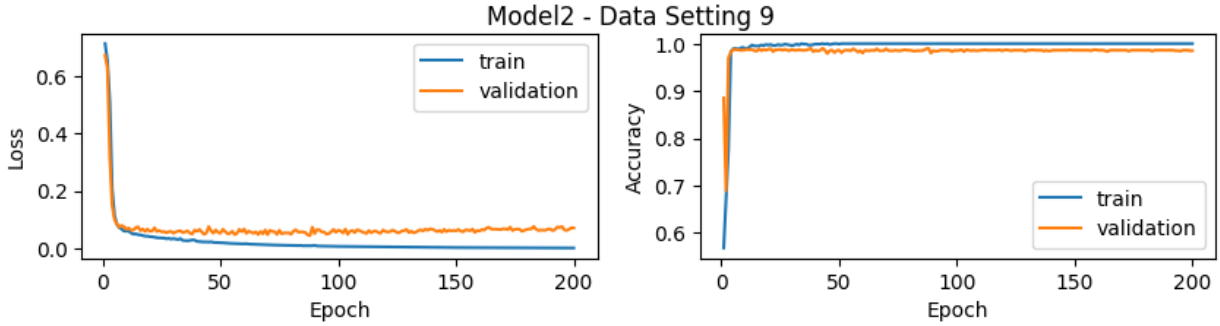

```
Experiment: 1 Setting: 6
N: 200
```



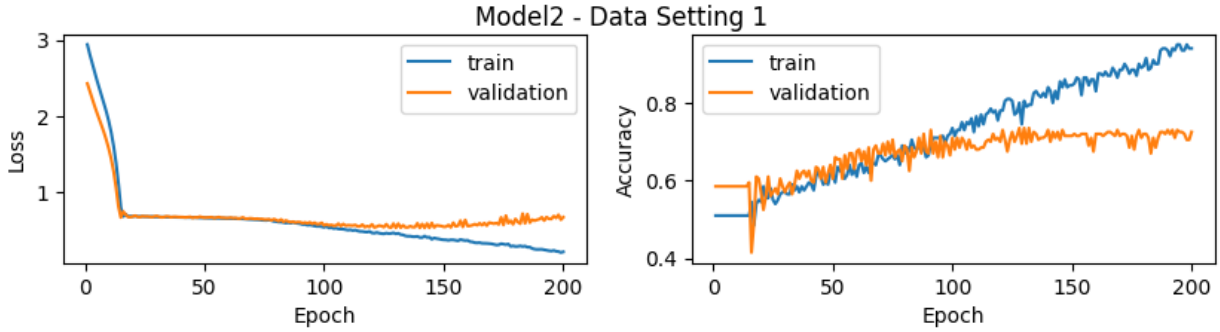

```
Experiment: 1 Setting: 7
N: 500
```



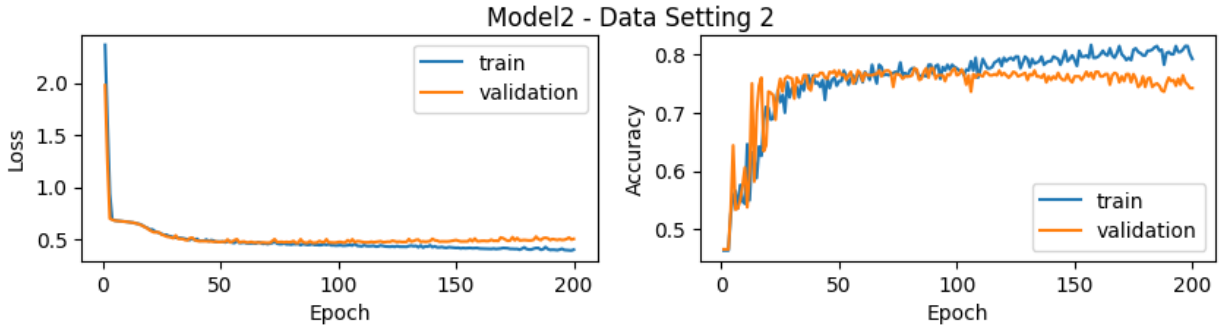

```
Experiment: 1 Setting: 8
N: 1000
```



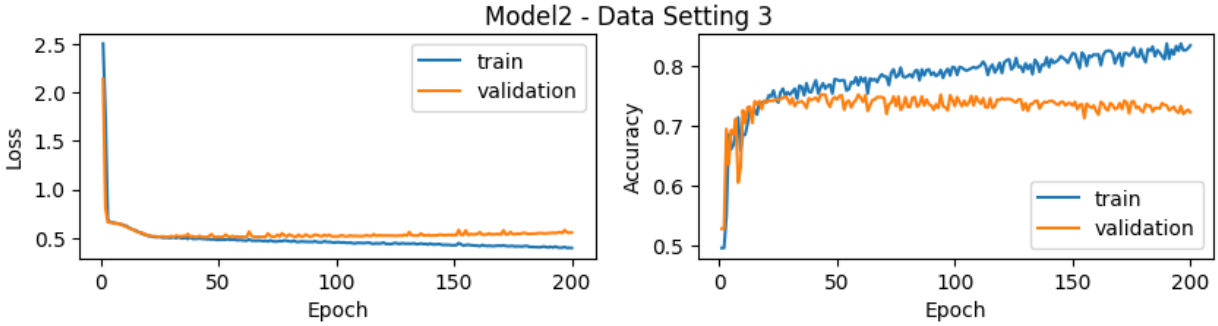

```
Experiment: 2 Setting: 0
N: 200
```



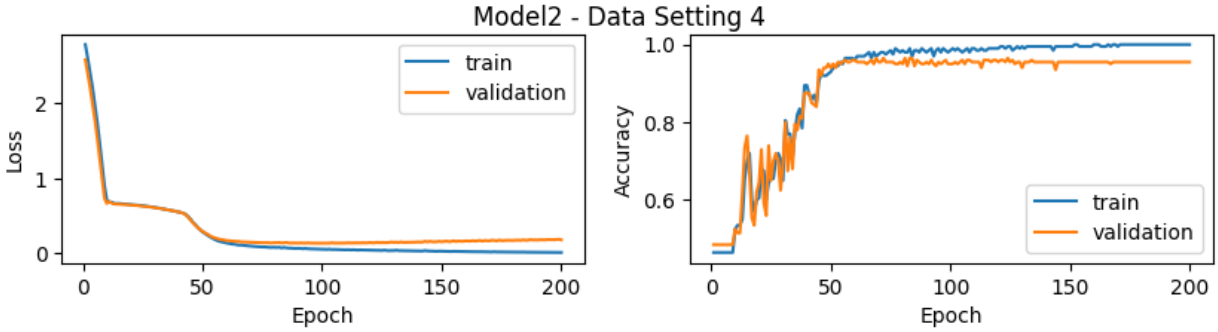

```
Experiment: 2 Setting: 1
N: 500
```



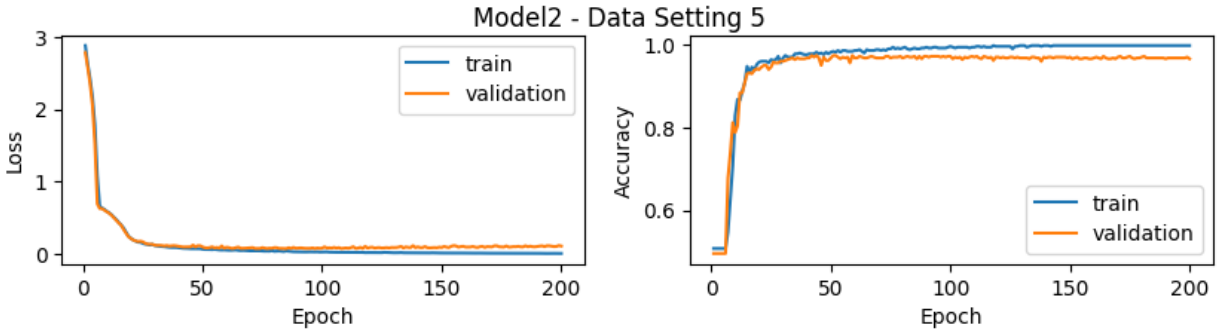

```
Experiment: 2 Setting: 2
N: 1000
```



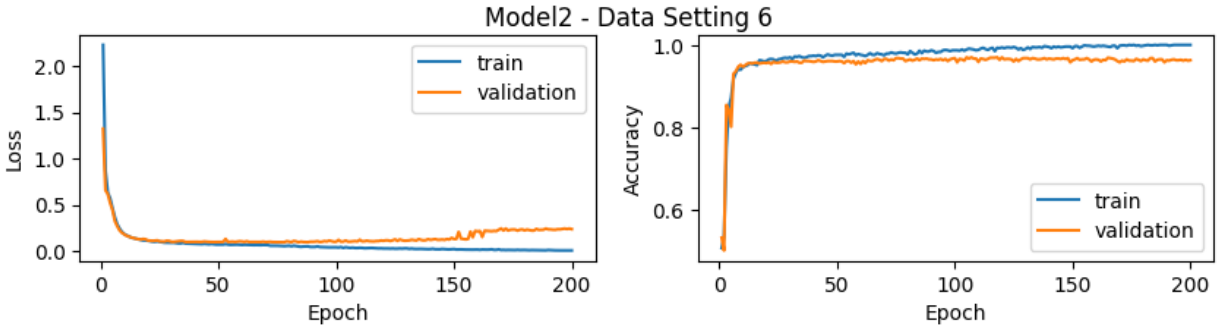

```
Experiment: 2 Setting: 3
N: 200
```



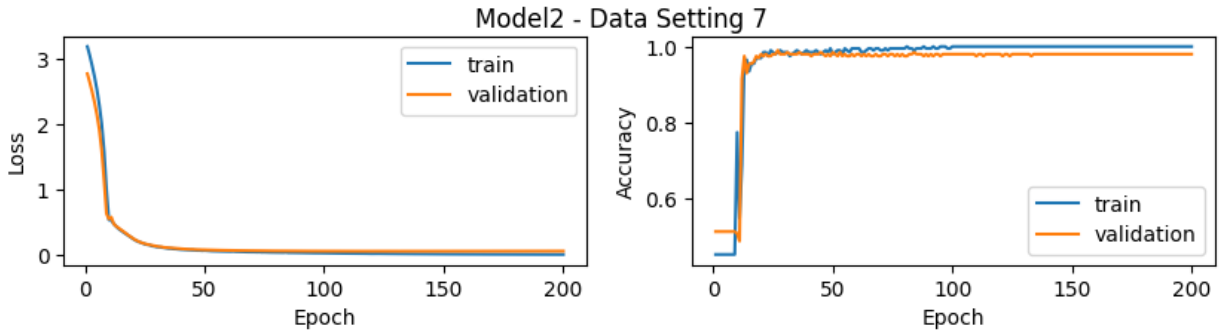

```
Experiment: 2 Setting: 4
N: 500
```



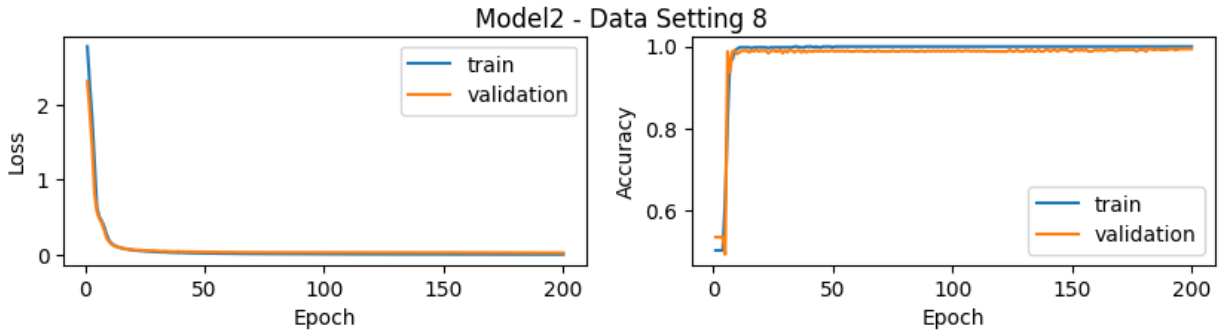

```
Experiment: 2 Setting: 5
N: 1000
```



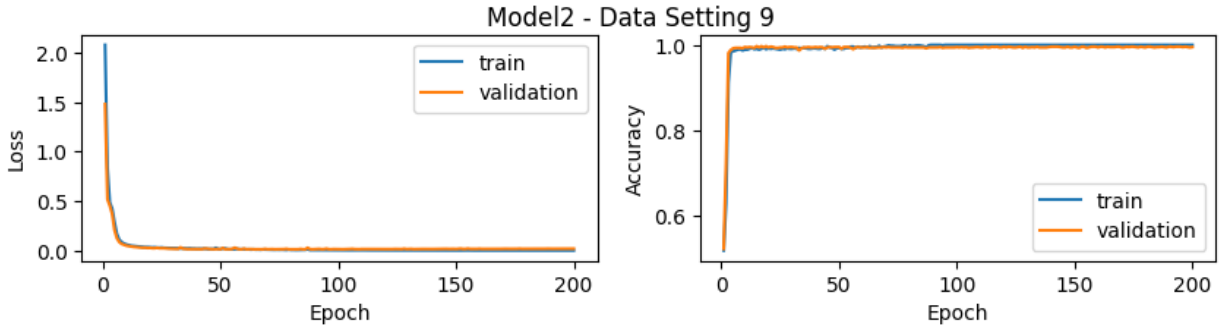

```
Experiment: 2 Setting: 6
N: 200
```



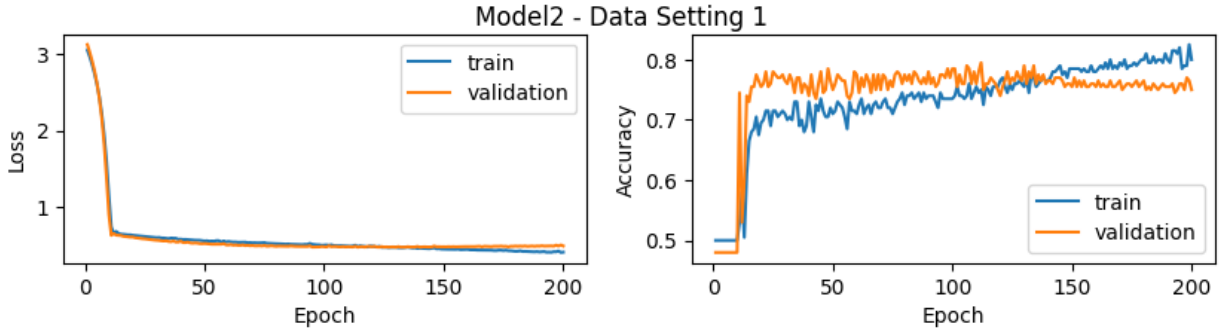

```
Experiment: 2 Setting: 7
N: 500
```



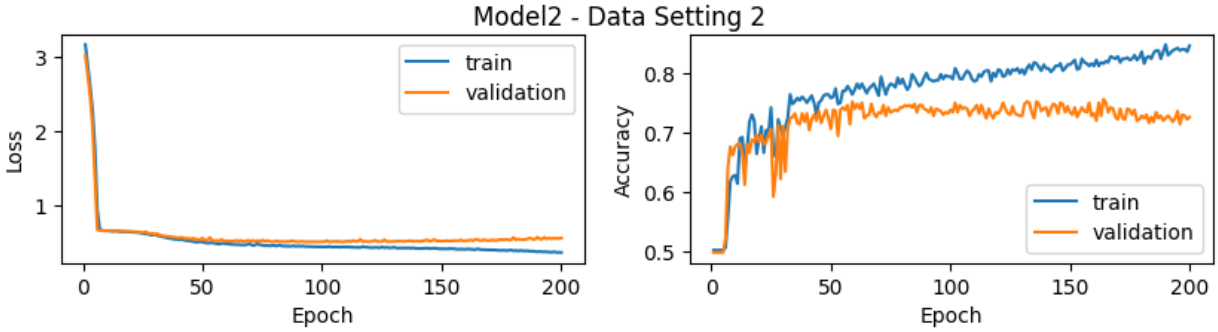

```
Experiment: 2 Setting: 8
N: 1000
```



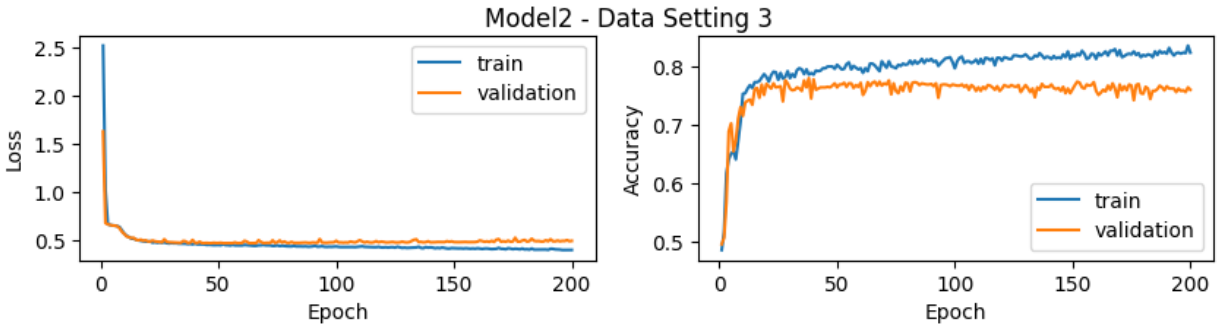

```
Experiment: 3 Setting: 0
N: 200
```



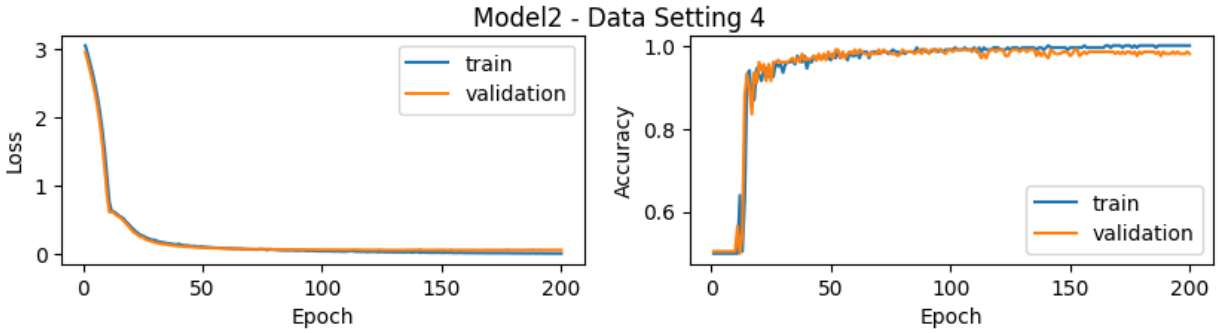

```
Experiment: 3 Setting: 1
N: 500
```



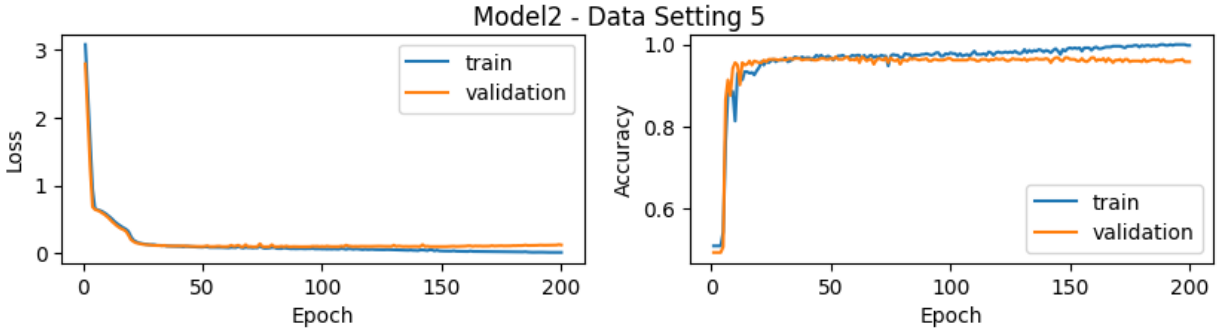

```
Experiment: 3 Setting: 2
N: 1000
```



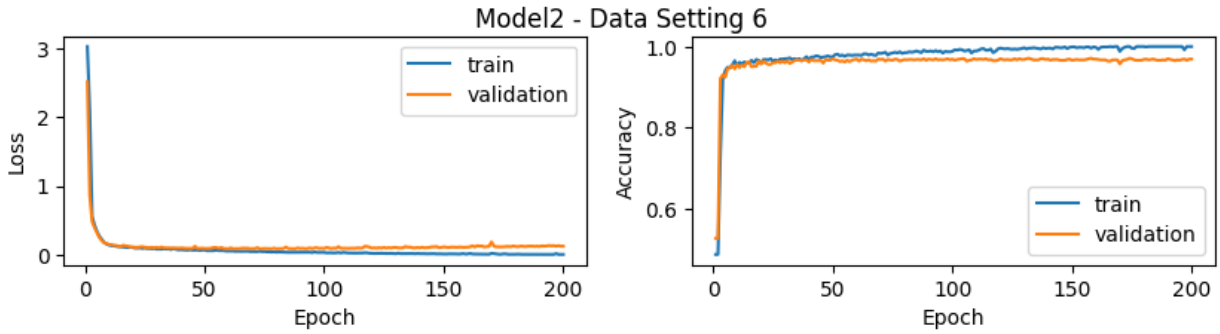

```
Experiment: 3 Setting: 3
N: 200
```



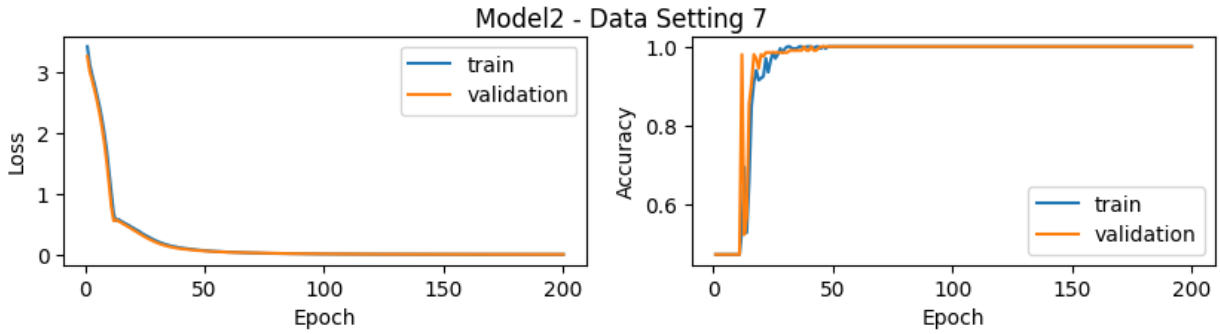

```
Experiment: 3 Setting: 4
N: 500
```



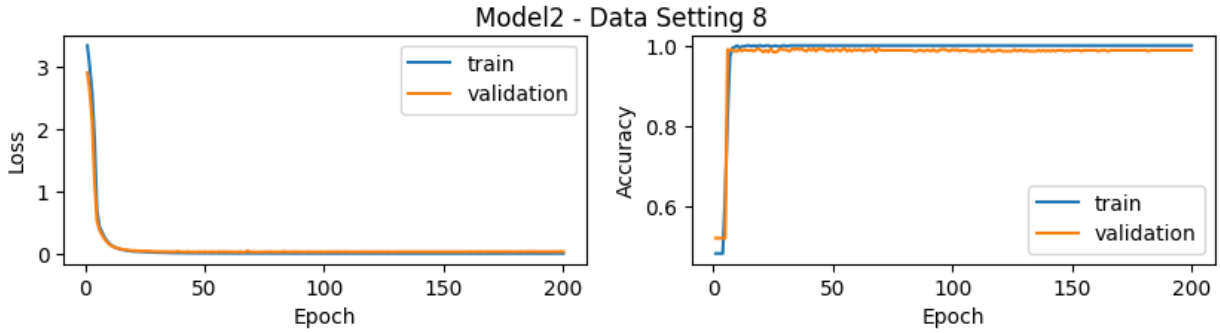

```
Experiment: 3 Setting: 5
N: 1000
```



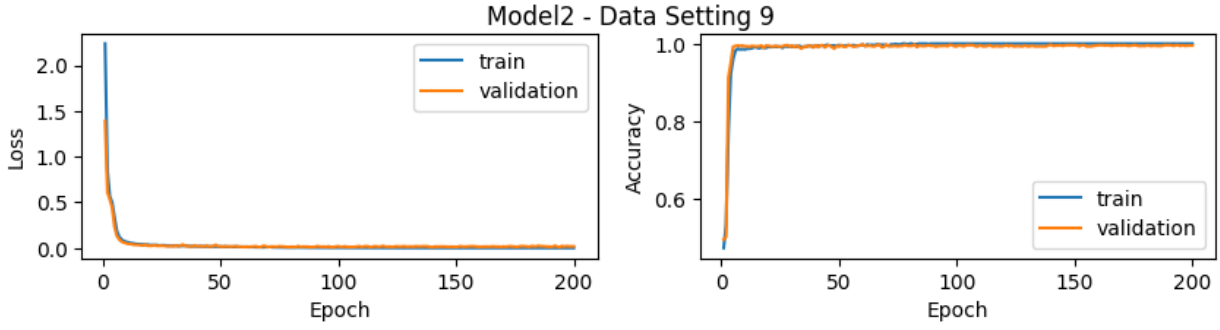

```
Experiment: 3 Setting: 6
N: 200
```



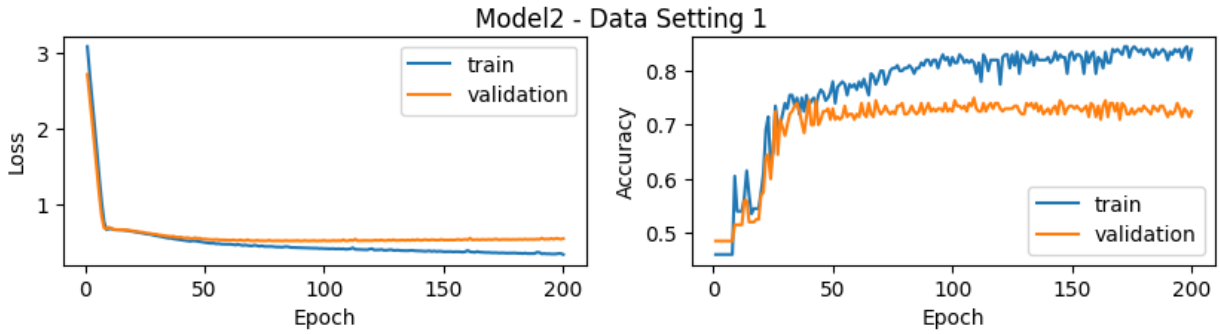

```
Experiment: 3 Setting: 7
N: 500
```



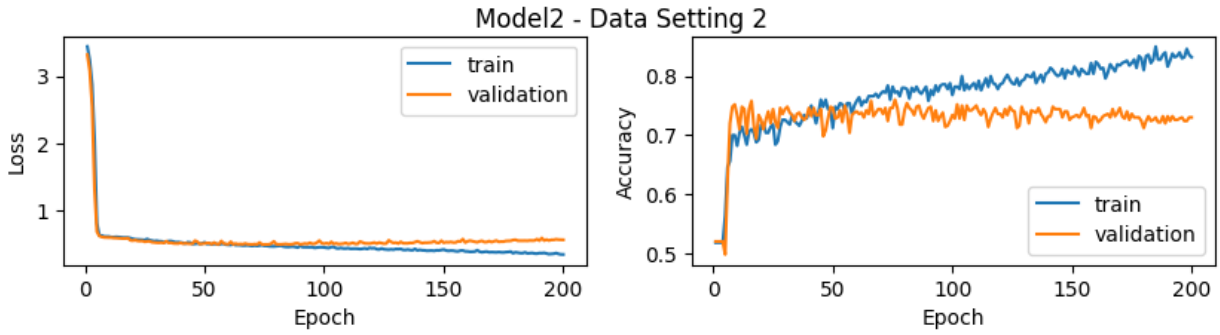

```
Experiment: 3 Setting: 8
N: 1000
```



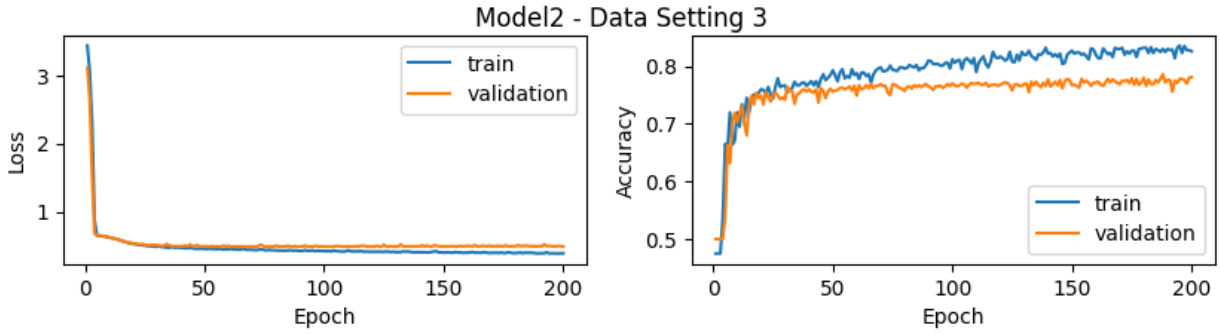

```
Experiment: 4 Setting: 0
N: 200
```



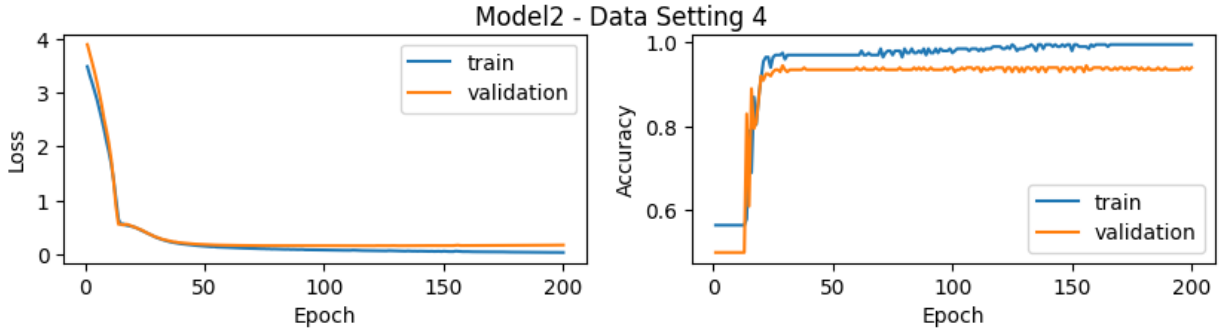

```
Experiment: 4 Setting: 1
N: 500
```



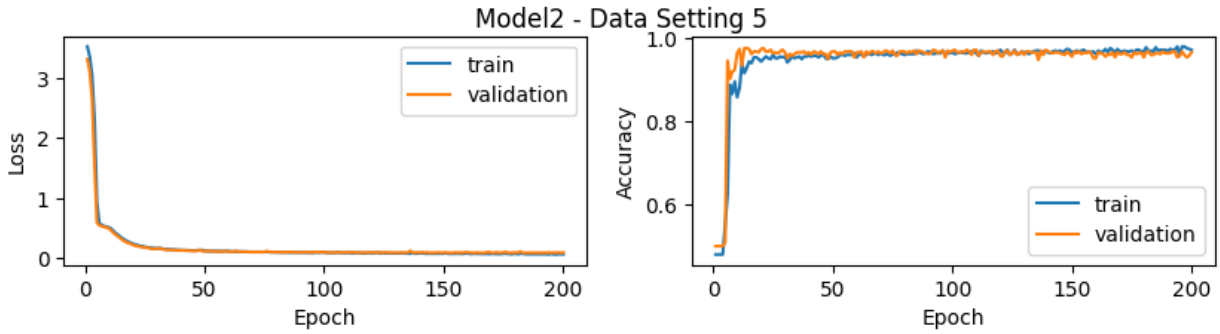

```
Experiment: 4 Setting: 2
N: 1000
```



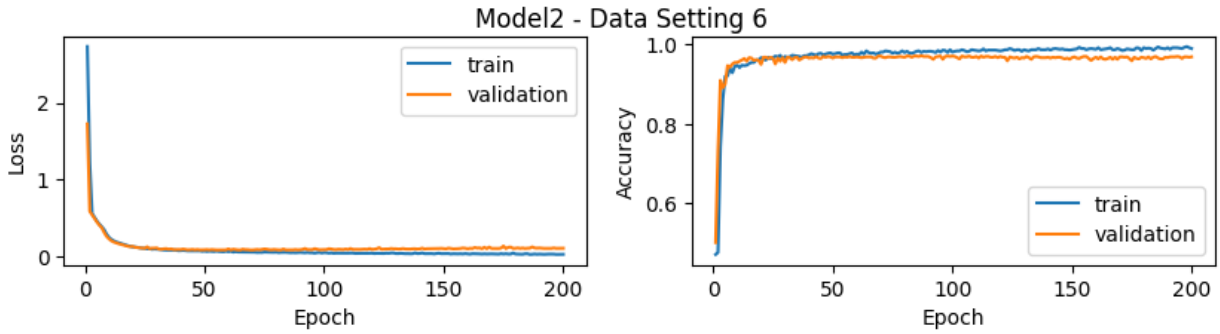

```
Experiment: 4 Setting: 3
N: 200
```



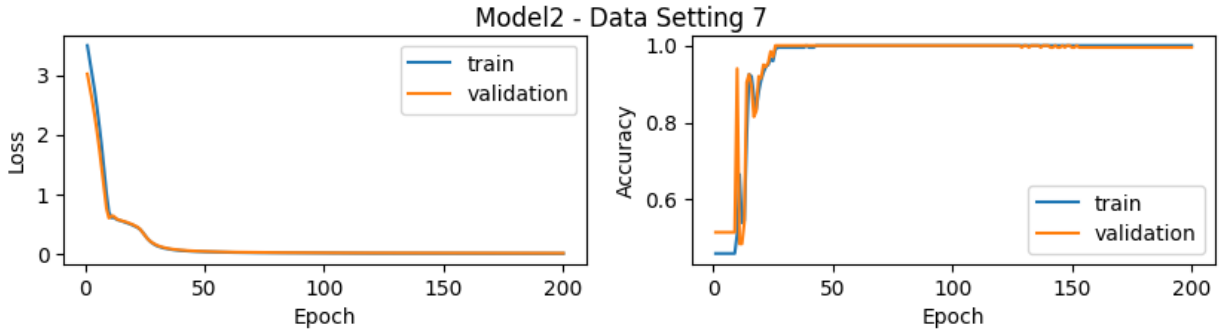

```
Experiment: 4 Setting: 4
N: 500
```



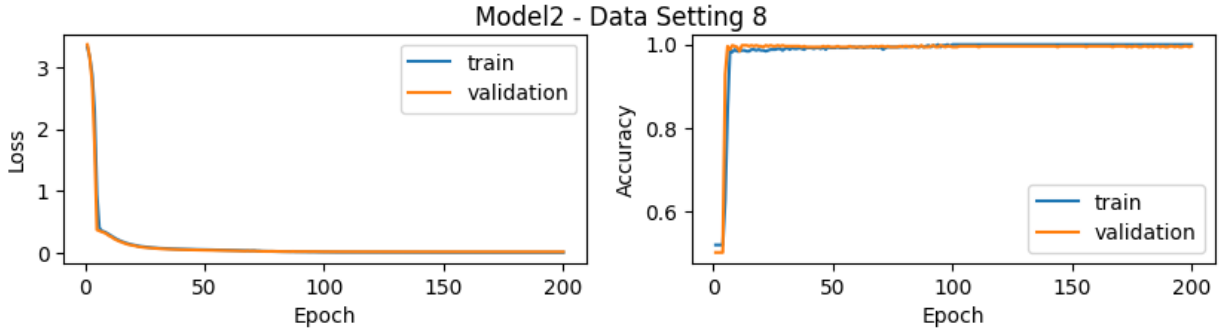

```
Experiment: 4 Setting: 5
N: 1000
```



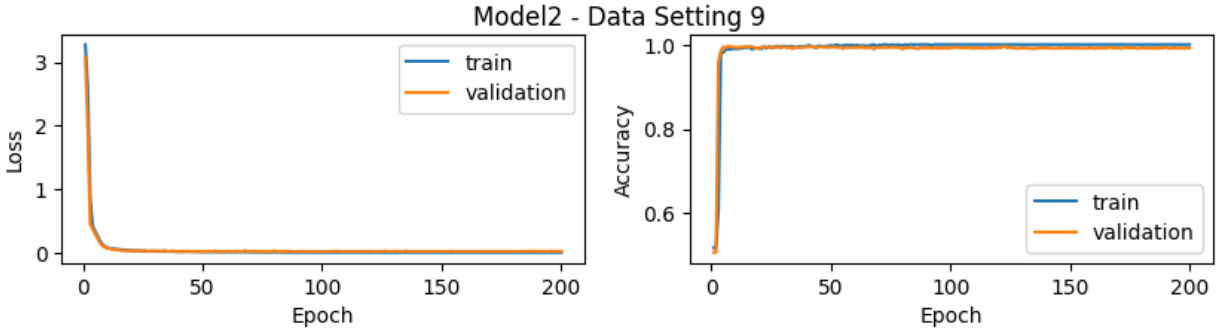

```
Experiment: 4 Setting: 6
N: 200
```



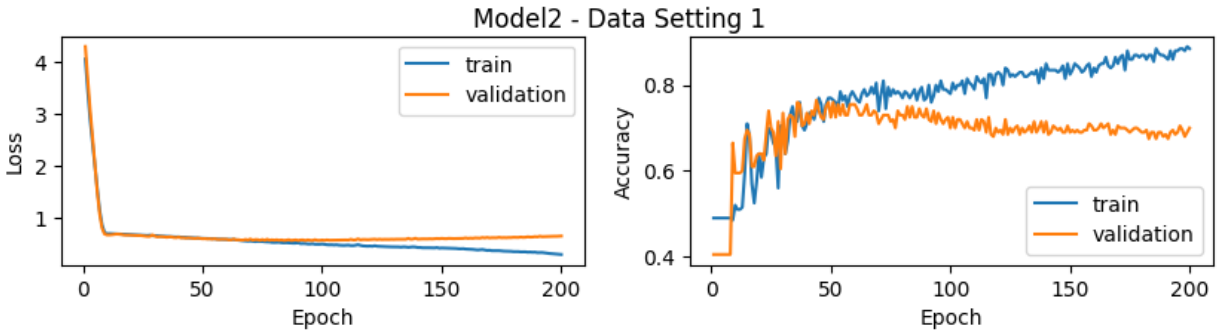

```
Experiment: 4 Setting: 7
N: 500
```



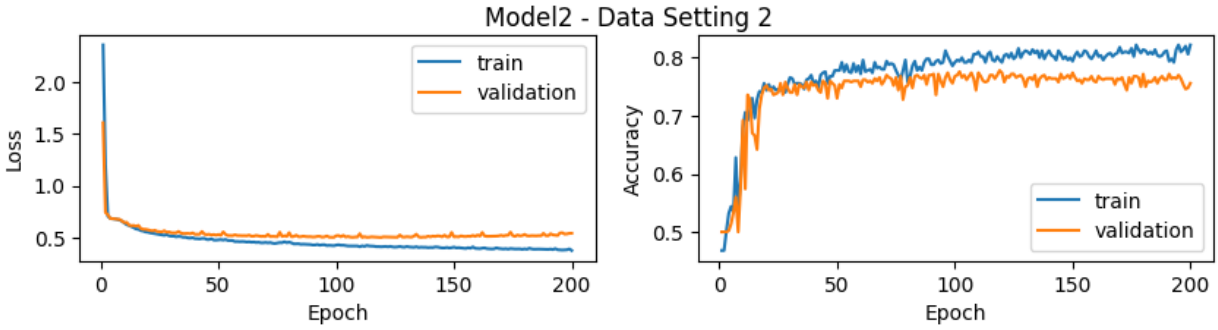

```
Experiment: 4 Setting: 8
N: 1000
```



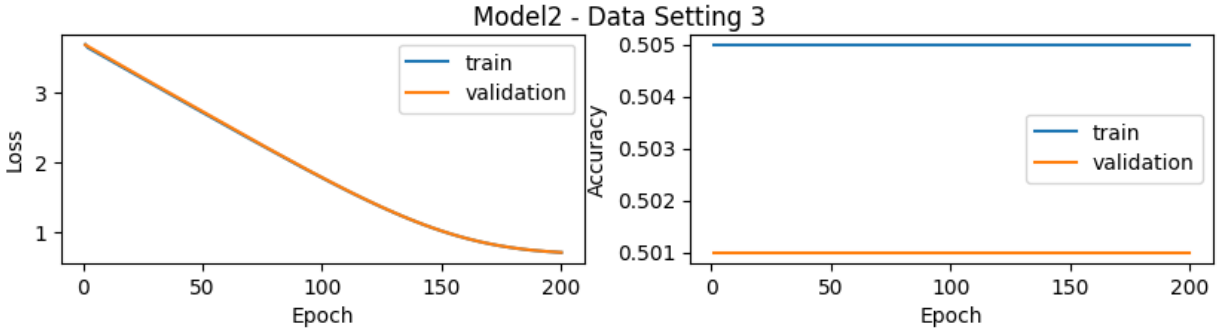

```
Experiment: 5 Setting: 0
N: 200
```



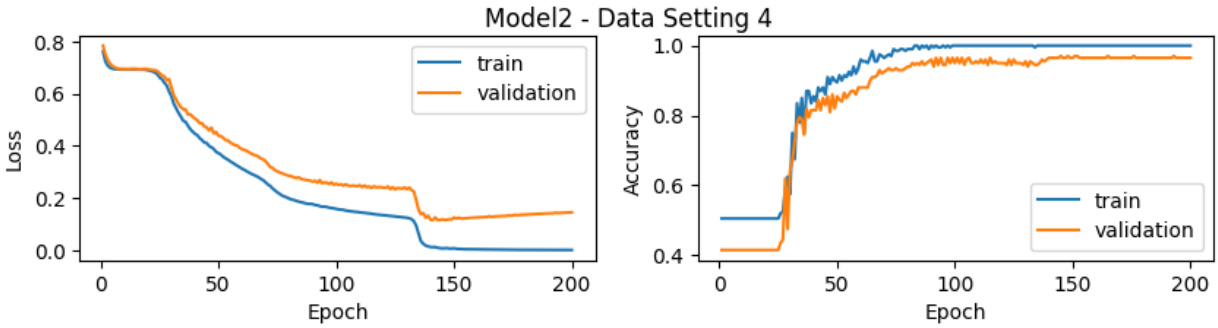

```
Experiment: 5 Setting: 1
N: 500
```



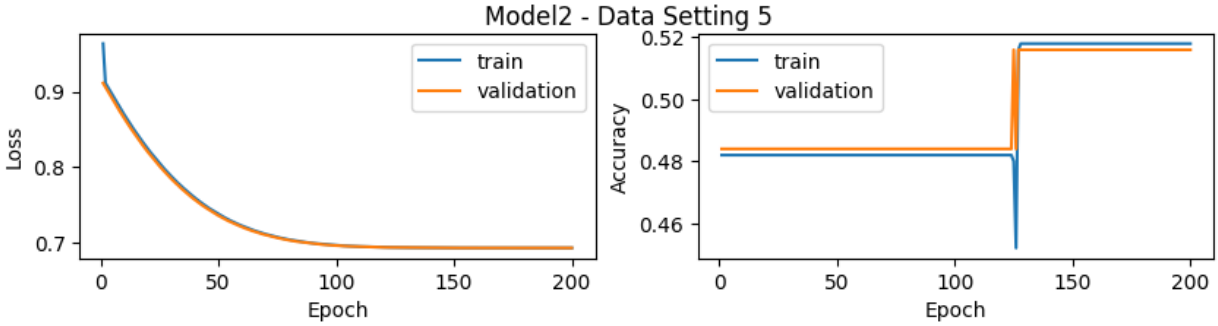

```
Experiment: 5 Setting: 2
N: 1000
```



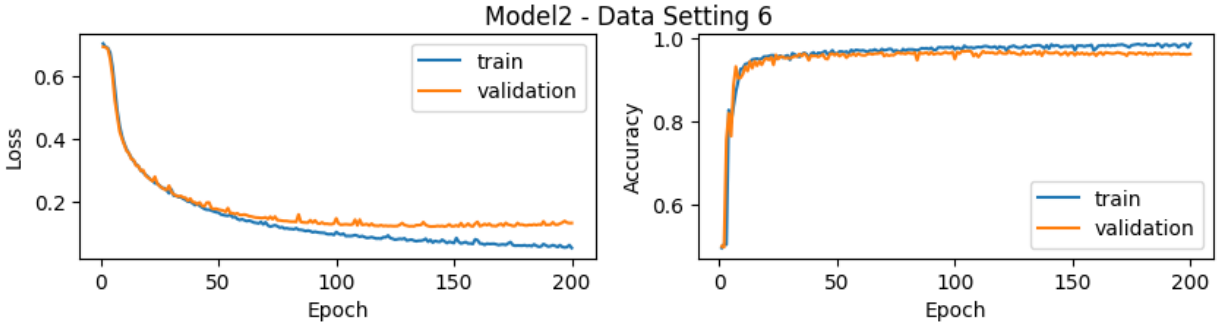

```
Experiment: 5 Setting: 3
N: 200
```



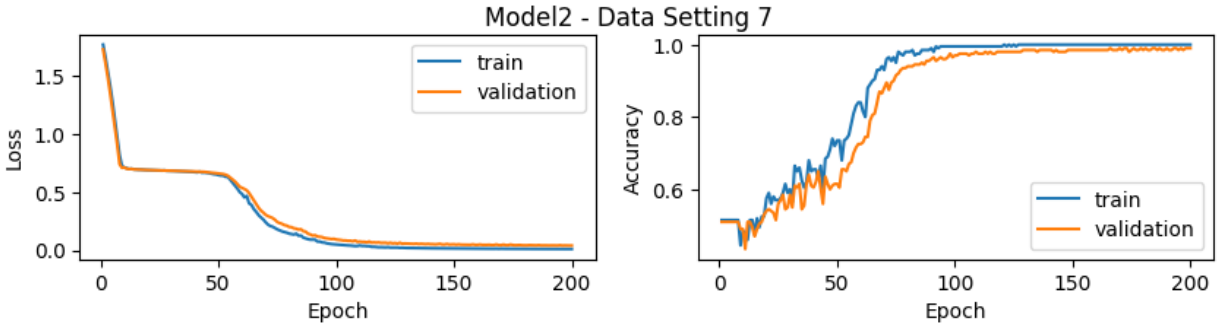

```
Experiment: 5 Setting: 4
N: 500
```



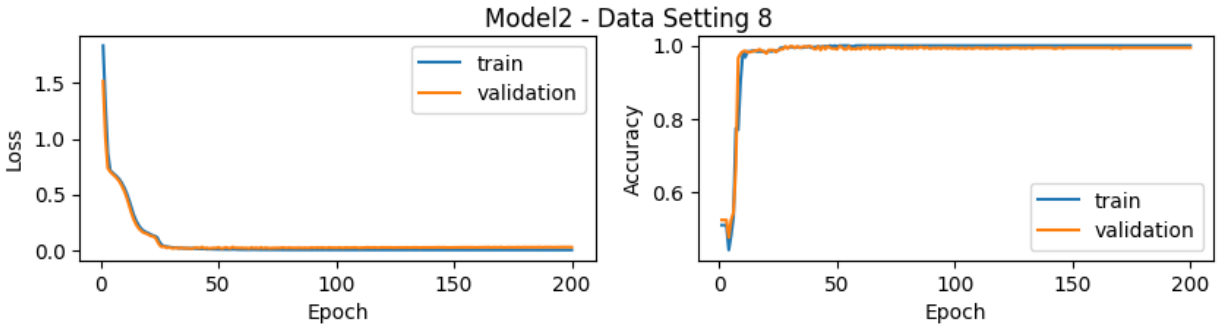

```
Experiment: 5 Setting: 5
N: 1000
```



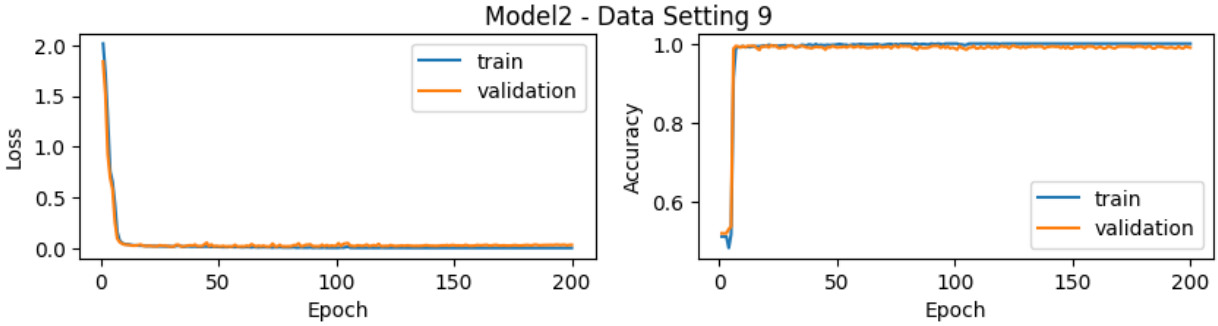

```
Experiment: 5 Setting: 6
N: 200
```



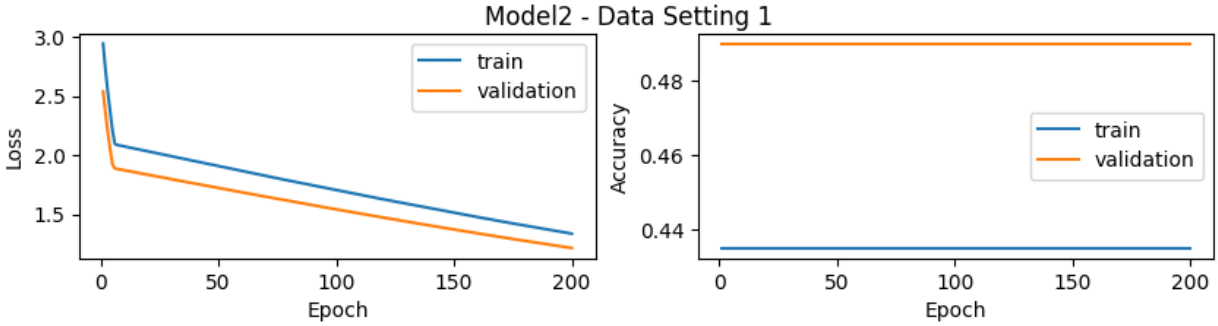

```
Experiment: 5 Setting: 7
N: 500
```



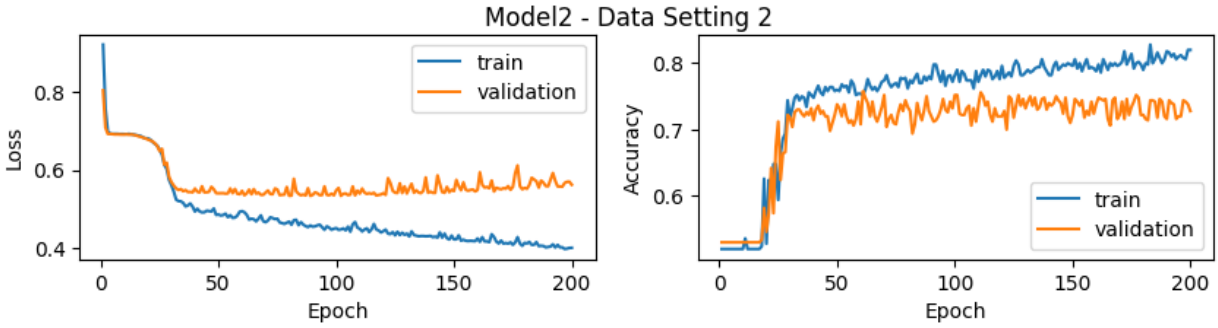

```
Experiment: 5 Setting: 8
N: 1000
```



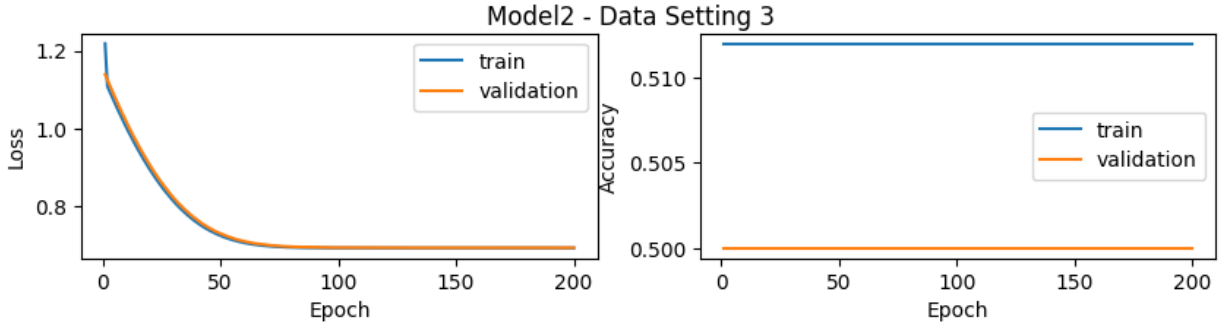

```
Experiment: 6 Setting: 0
N: 200
```



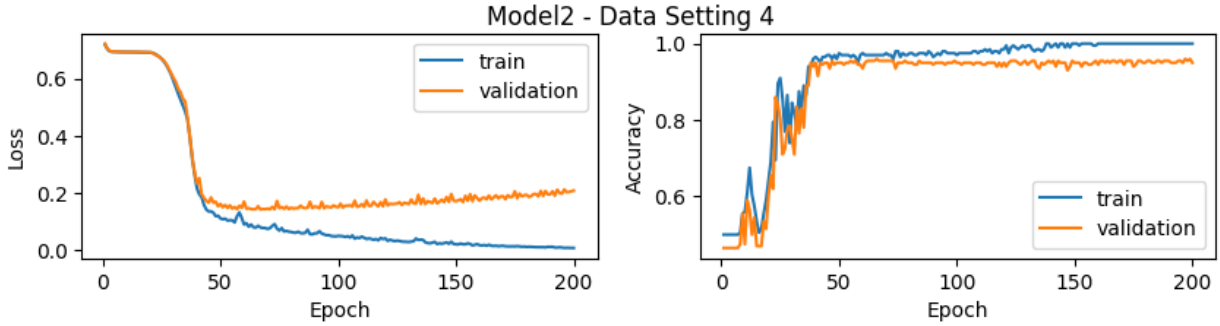

```
Experiment: 6 Setting: 1
N: 500
```



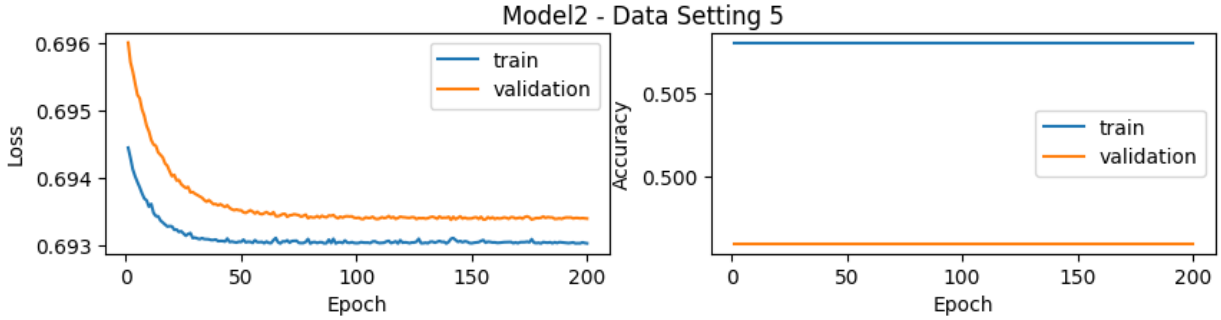

```
Experiment: 6 Setting: 2
N: 1000
```



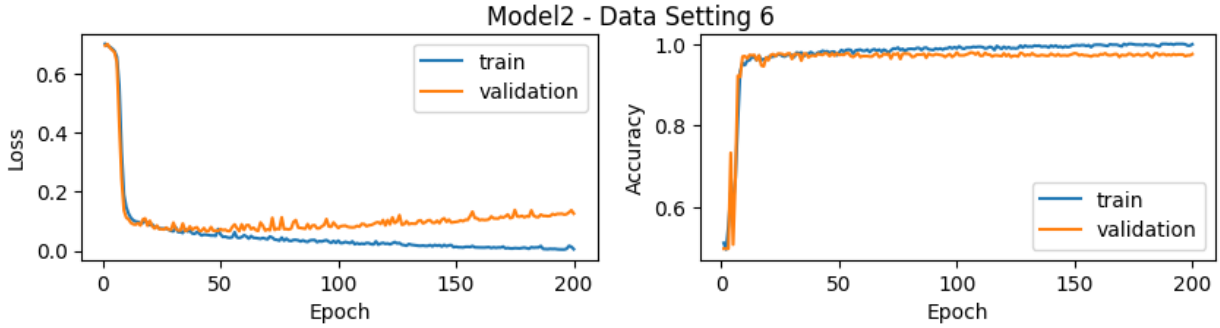

```
Experiment: 6 Setting: 3
N: 200
```



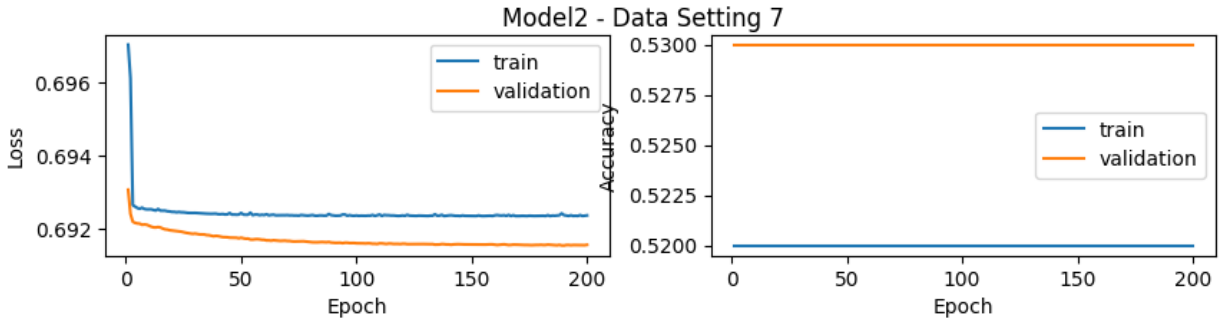

```
Experiment: 6 Setting: 4
N: 500
```



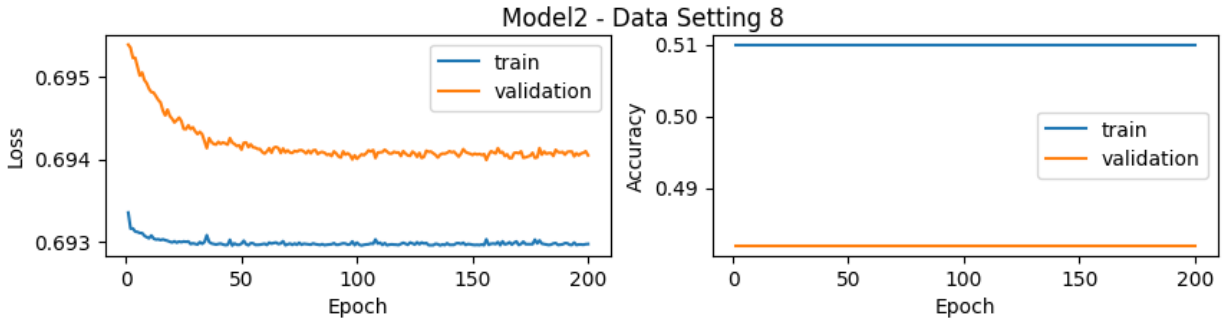

```
Experiment: 6 Setting: 5
N: 1000
```



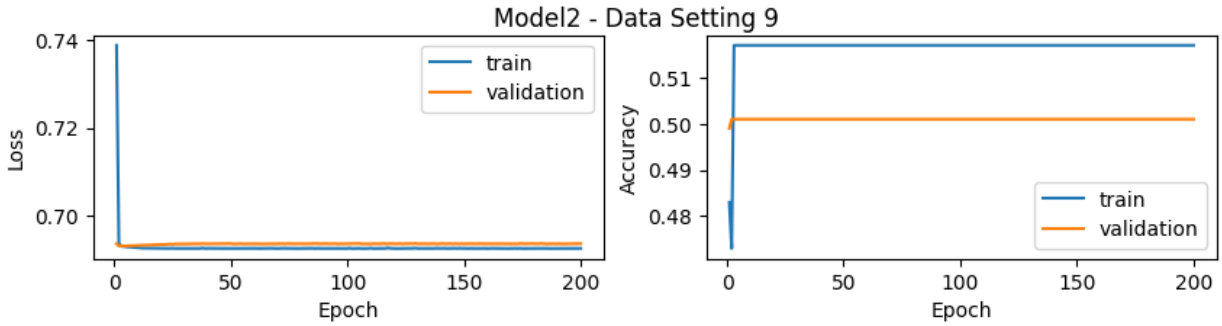

```
Experiment: 6 Setting: 6
N: 200
```



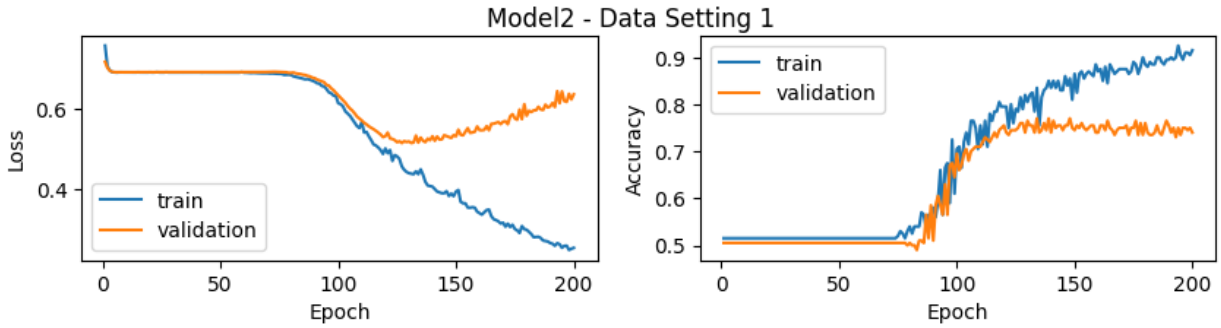

```
Experiment: 6 Setting: 7
N: 500
```



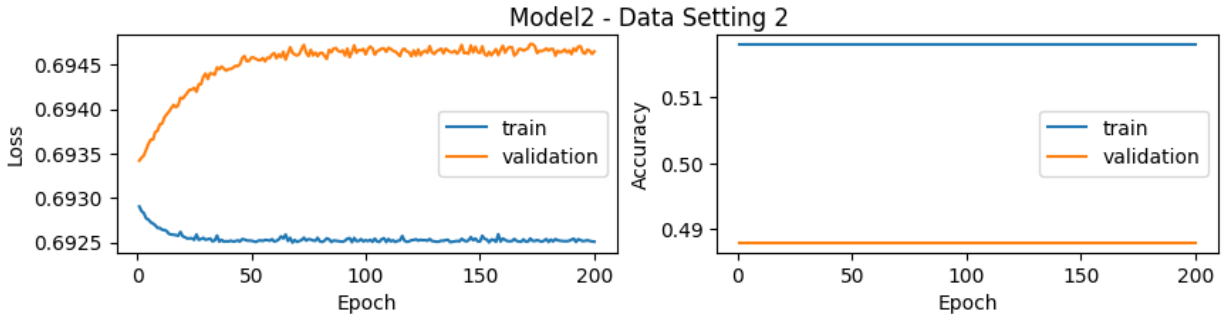

```
Experiment: 6 Setting: 8
N: 1000
```



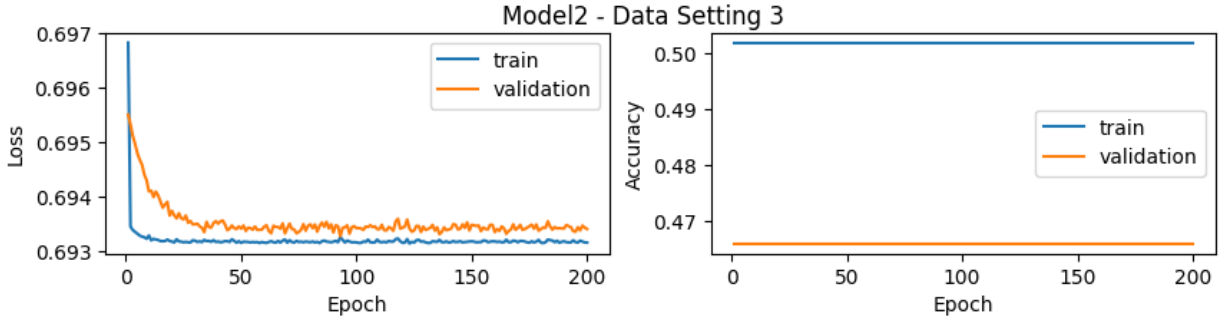

```
Experiment: 7 Setting: 0
N: 200
```



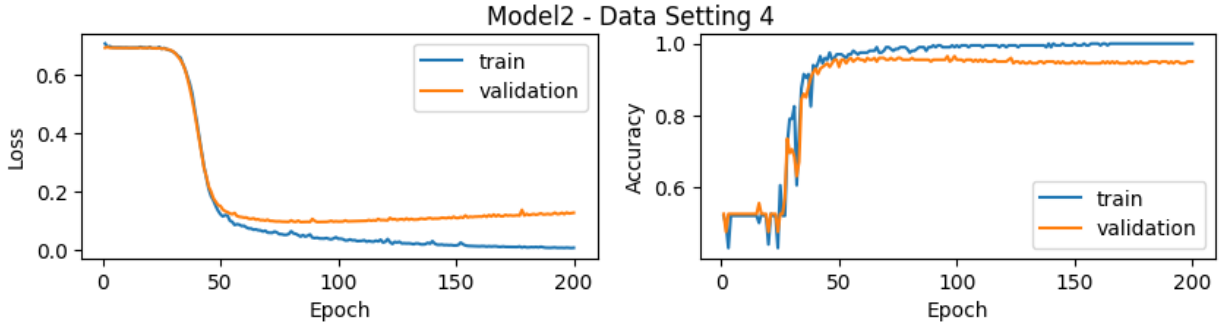

```
Experiment: 7 Setting: 1
N: 500
```



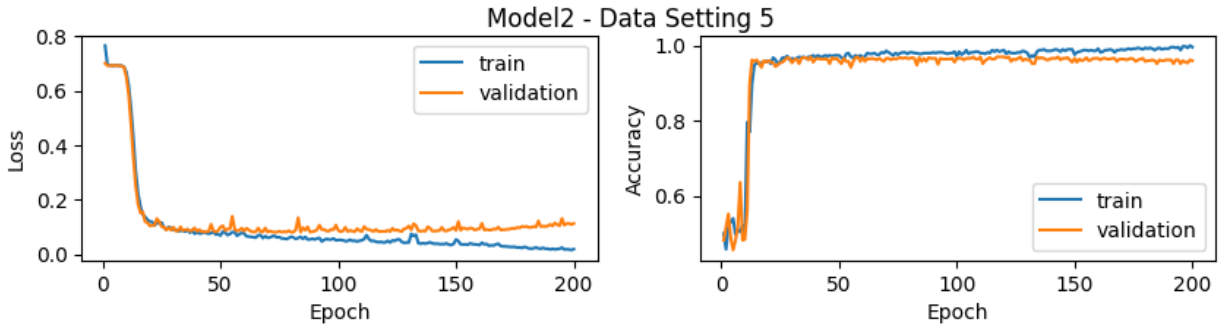

```
Experiment: 7 Setting: 2
N: 1000
```



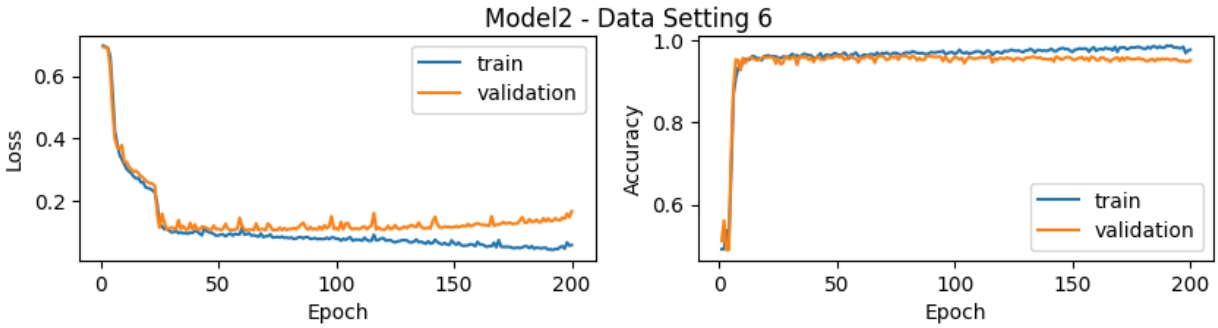

```
Experiment: 7 Setting: 3
N: 200
```



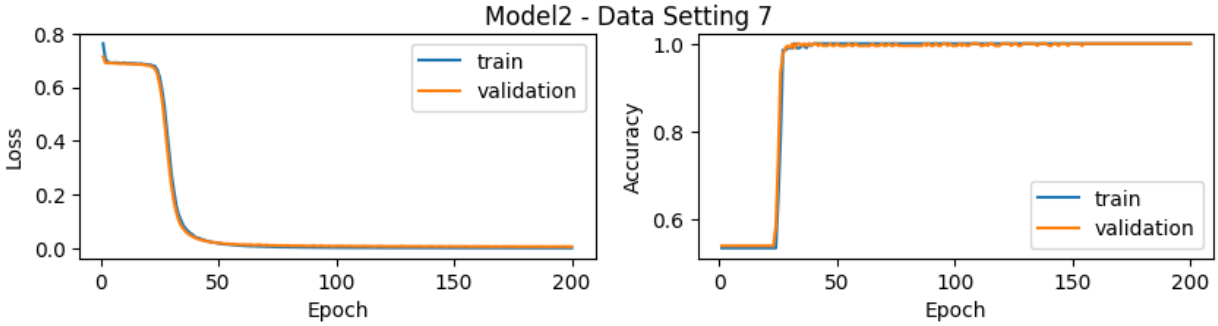

```
Experiment: 7 Setting: 4
N: 500
```



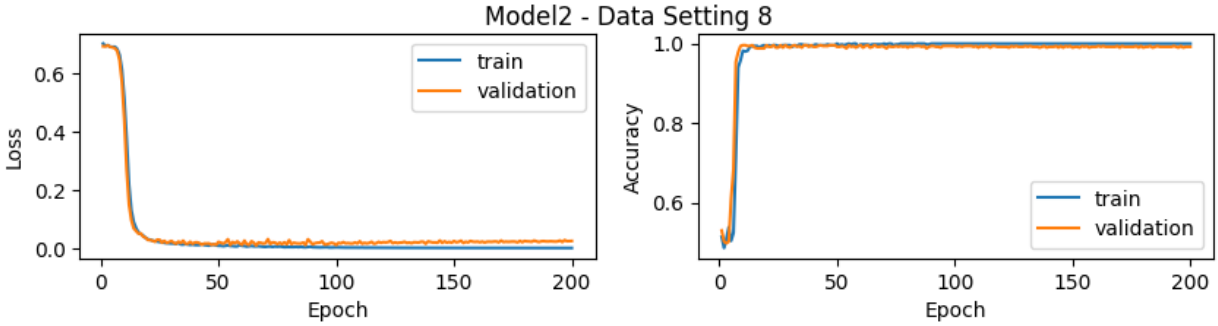

```
Experiment: 7 Setting: 5
N: 1000
```



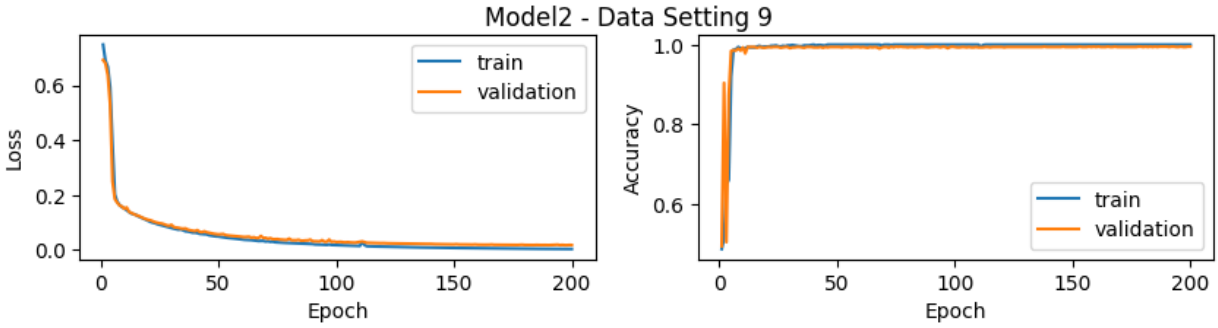

```
Experiment: 7 Setting: 6
N: 200
```



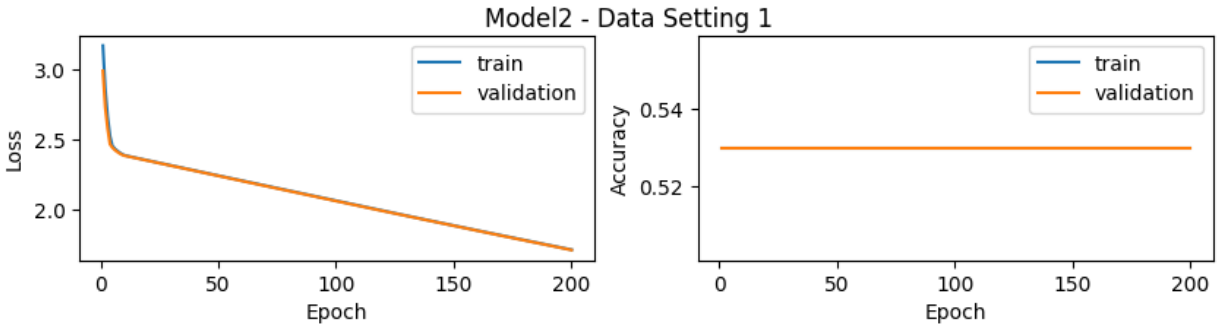

```
Experiment: 7 Setting: 7
N: 500
```



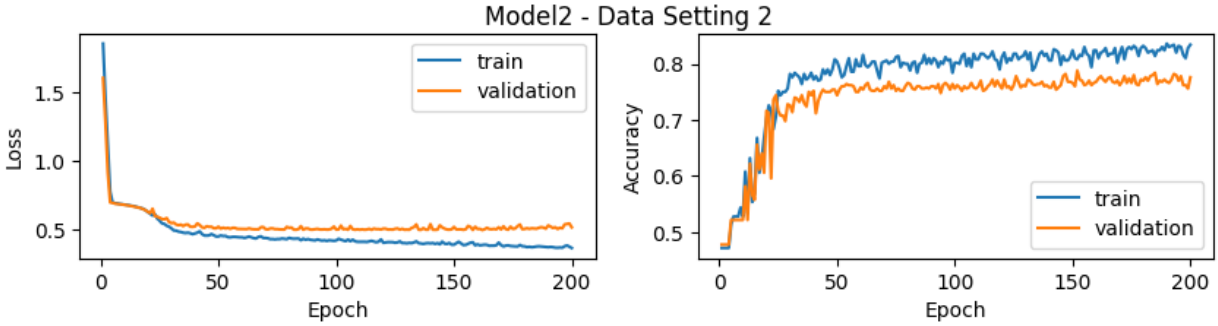

```
Experiment: 7 Setting: 8
N: 1000
```



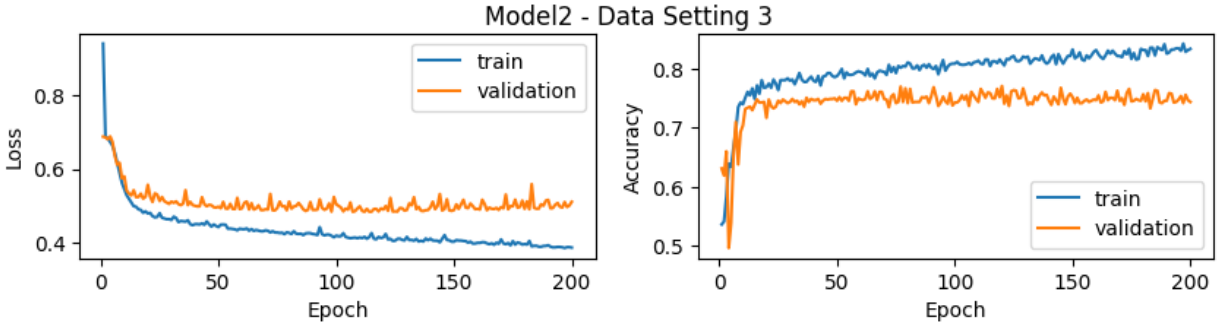

```
Experiment: 8 Setting: 0
N: 200
```



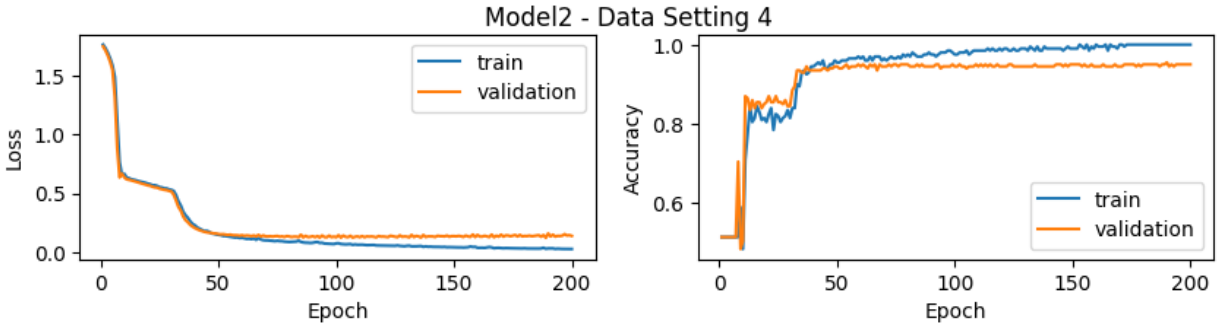

```
Experiment: 8 Setting: 1
N: 500
```



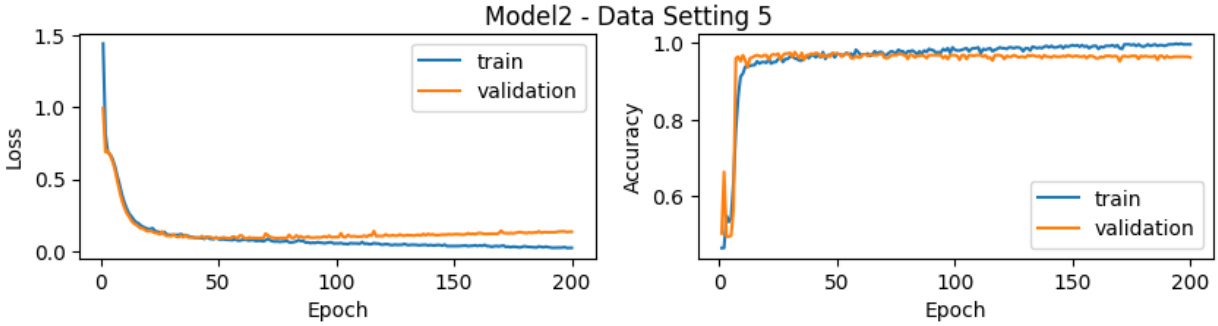

```
Experiment: 8 Setting: 2
N: 1000
```



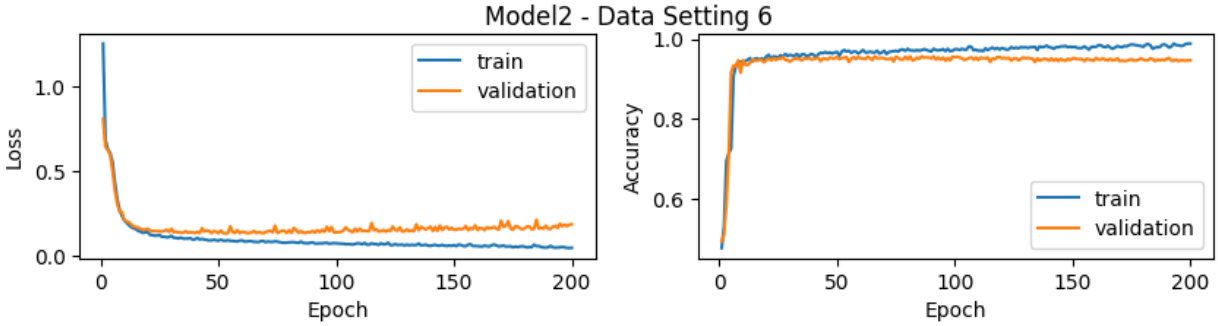

```
Experiment: 8 Setting: 3
N: 200
```



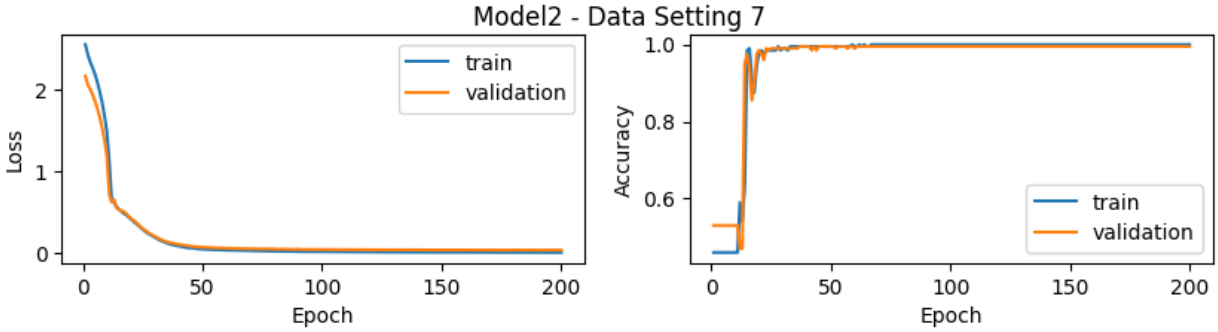

```
Experiment: 8 Setting: 4
N: 500
```



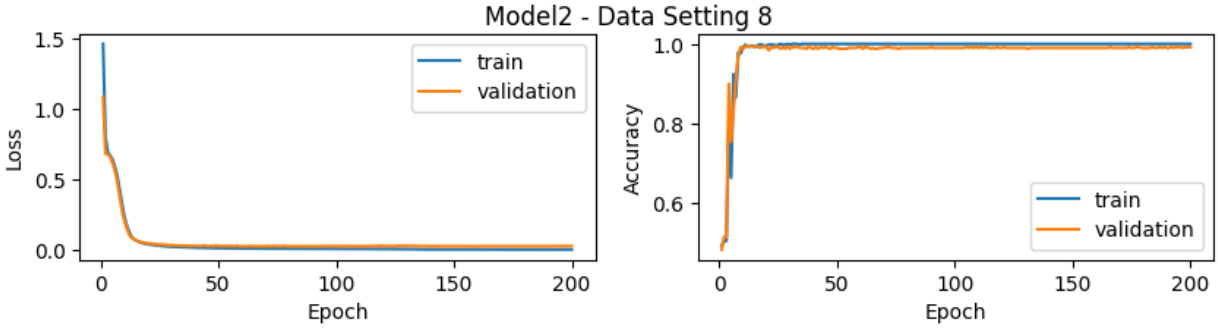

```
Experiment: 8 Setting: 5
N: 1000
```



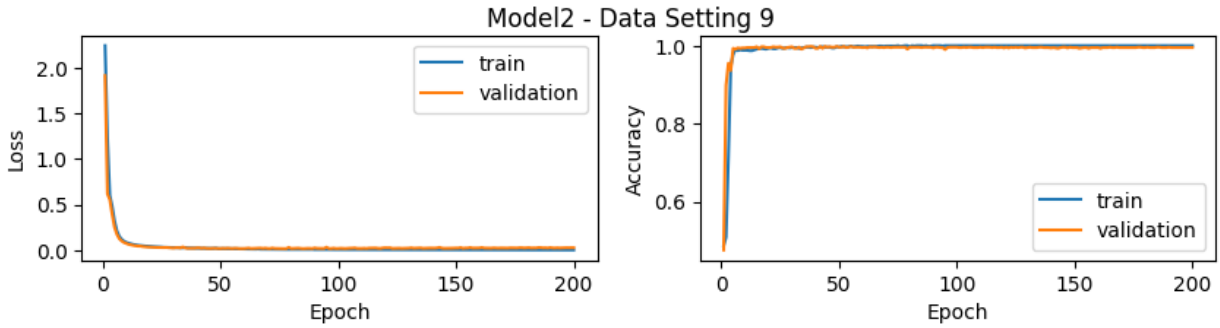

```
Experiment: 8 Setting: 6
N: 200
```



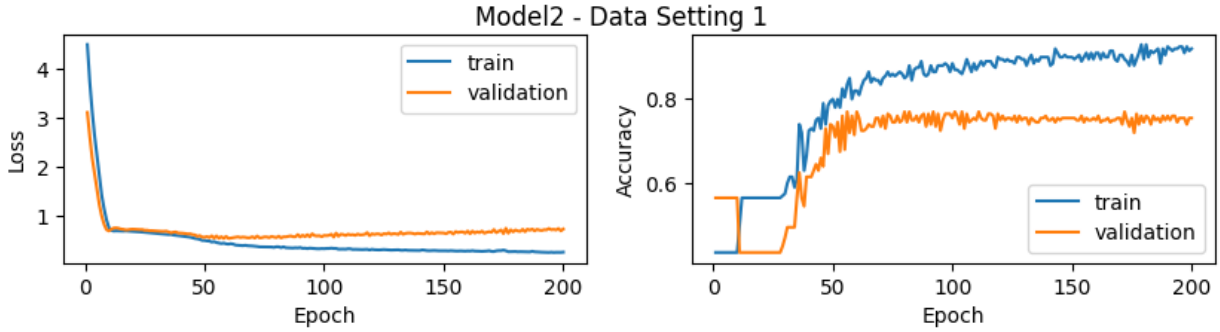

```
Experiment: 8 Setting: 7
N: 500
```



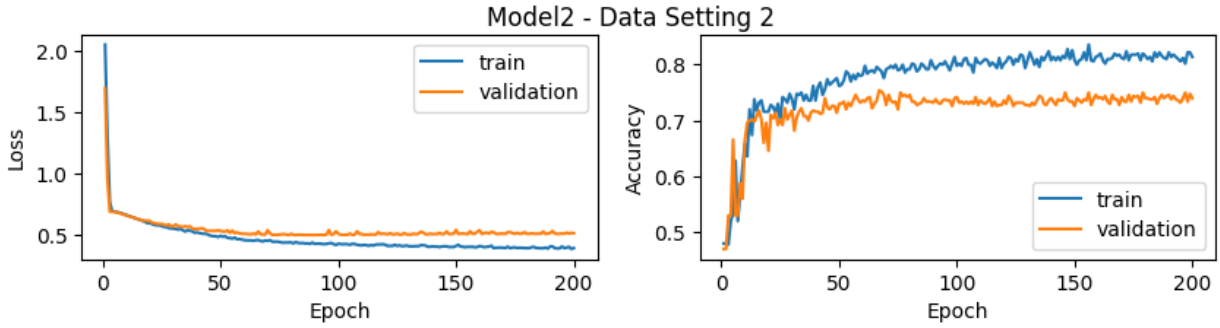

```
Experiment: 8 Setting: 8
N: 1000
```



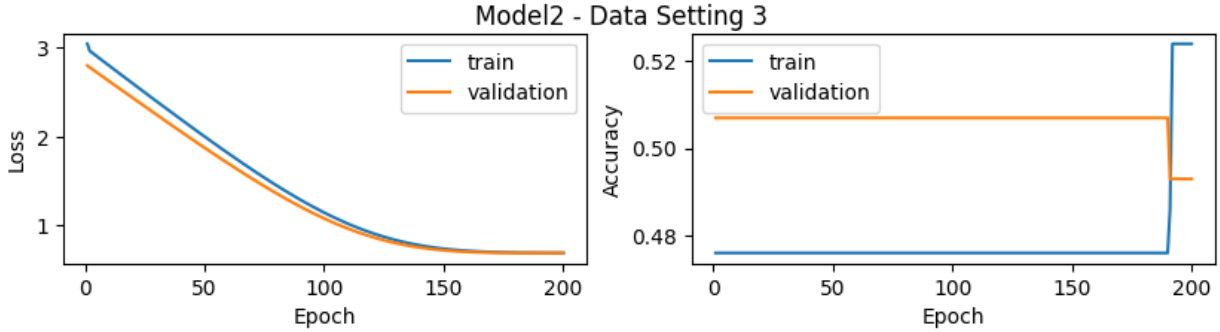

```
Experiment: 9 Setting: 0
N: 200
```



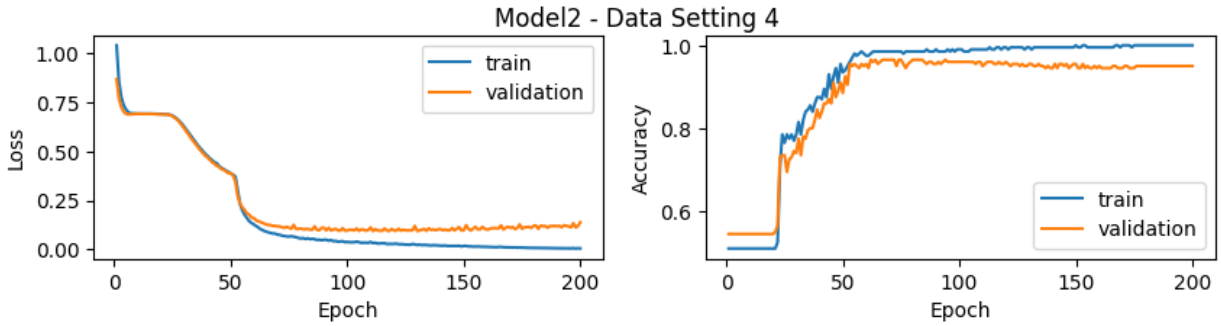

```
Experiment: 9 Setting: 1
N: 500
```



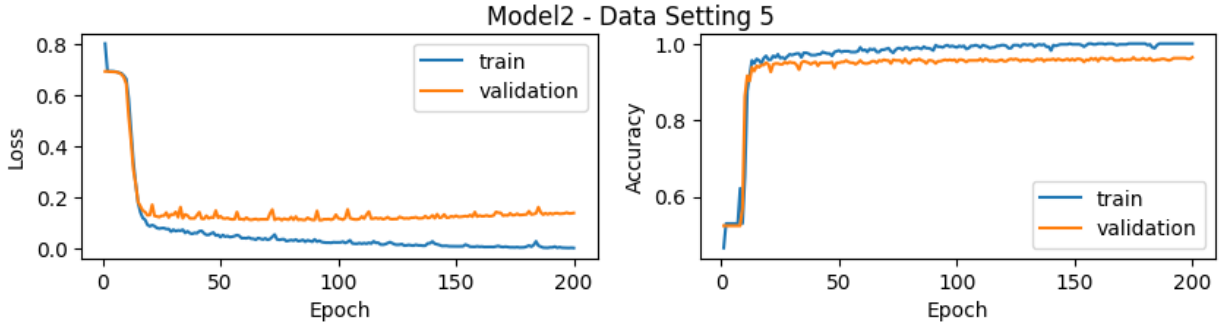

```
Experiment: 9 Setting: 2
N: 1000
```



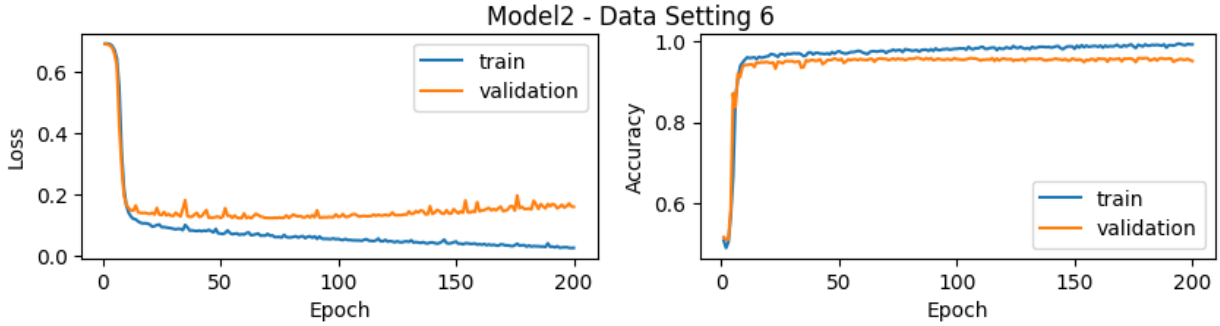

```
Experiment: 9 Setting: 3
N: 200
```



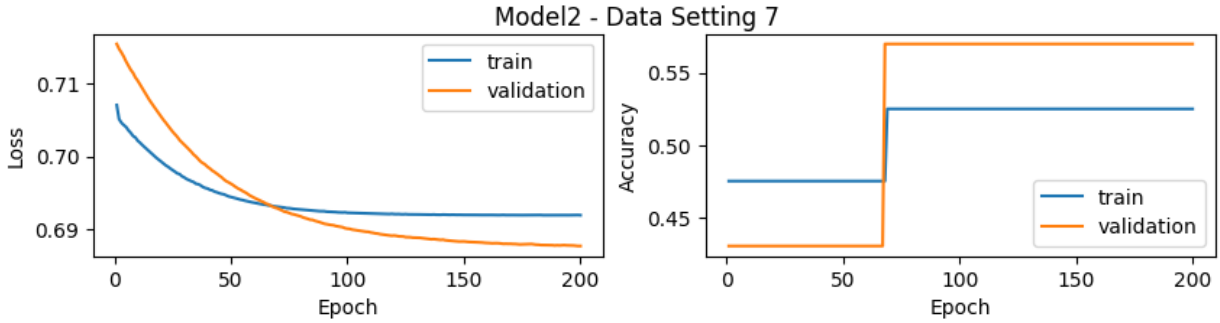
Experiment: 9 Setting: 4
N: 500



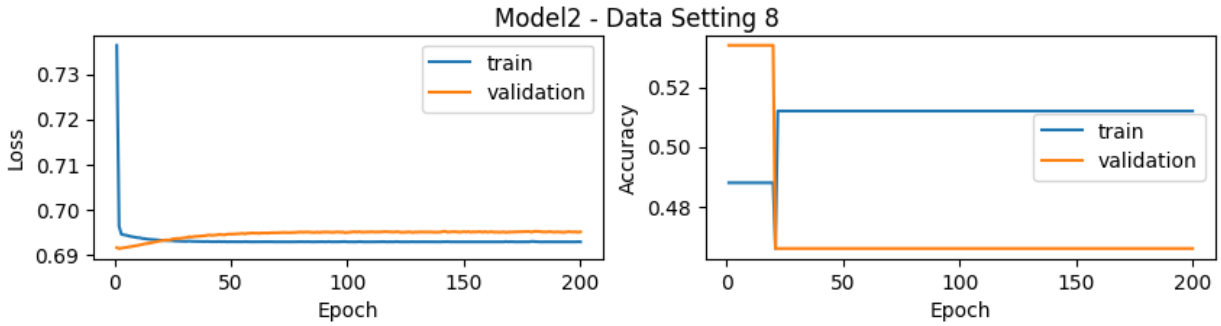
Experiment: 9 Setting: 5
N: 1000



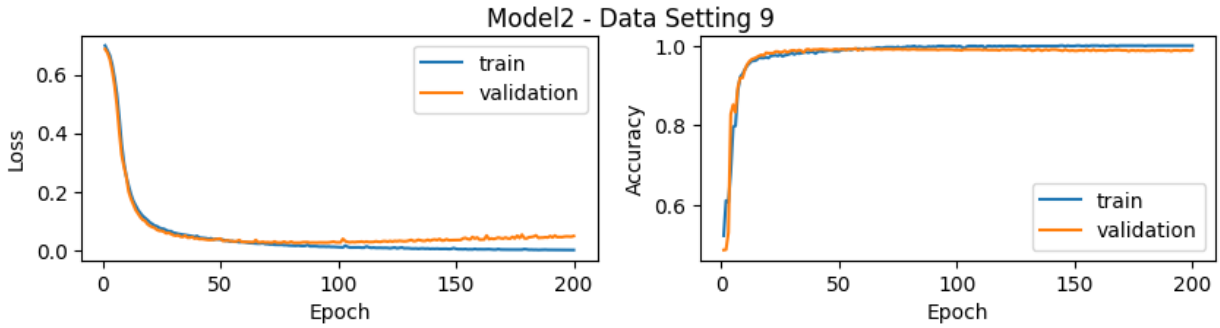
Experiment: 9 Setting: 6
N: 200



Experiment: 9 Setting: 7
N: 500




```
Experiment: 9 Setting: 8
N: 1000
```



In [ ]:

In [62]:
```
import numpy as np
import matplotlib.pyplot as plt
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.nn.init as init
import os, glob
from datetime import datetime
import pandas as pd
from torch.utils.data import Dataset, DataLoader
```

Question 2

CNN training: Train a Convolutional Neural Network on the simulated data for each of the nine simulation settings. The goal is to use the CNN to predict the cancer status yi based on the simulated images Xi . Additionally, generate a test set of 1000 subjects using the same data generation process and evaluate the CNN's performance in terms of classification accuracy. You are free to build a CNN with arbitrary hyperparameter setting. Conduct at least 10 independent experiments for each setting by generating new datasets each time, and report the hyperparameters for the CNN, the mean and standard deviation of the classification accuracy achieved by your CNN model.

MODEL 2

In [46]:
```
model2 = torch.nn.Sequential()
model2.add_module('conv1', torch.nn.Conv2d(in_channels=1, out_channels=2, kernel_size
model2.add_module('relu1', torch.nn.ReLU())
model2.add_module('pool1', torch.nn.MaxPool2d(kernel_size = 2))


model2.add_module('conv2', torch.nn.Conv2d(in_channels=2, out_channels=4, kernel_size
model2.add_module('relu2', torch.nn.ReLU())
model2.add_module('pool2', torch.nn.MaxPool2d(kernel_size = 2))


model2.add_module('conv3', torch.nn.Conv2d(in_channels=4, out_channels=8, kernel_size
model2.add_module('relu3', torch.nn.ReLU())
model2.add_module('pool3', torch.nn.MaxPool2d(kernel_size = 2))


model2.add_module('Flatten', torch.nn.Flatten())

model2.add_module('fc1', torch.nn.Linear(128, 10))
model2.add_module('relu7', torch.nn.ReLU())
model2.add_module('fc2', torch.nn.Linear(10, 1))

model2.add_module('sigmoid', torch.nn.Sigmoid())
```

In [14]:
```
def simulateData(n, mu_c, mu_n):

    y = np.random.choice([0, 1], size = n, p = [0.5, 0.5])
    m_i = np.random.poisson(lam = mu_c, size = n) * y + np.random.poisson(lam = mu_n,
```

```
    simulated_data = np.zeros([n, 32, 32])
    for i in range(n):
        random_indices = np.random.choice(32 * 32, m_i[i], replace = False)
        row_indices, col_indices = np.unravel_index(random_indices, (32, 32))
        Bi = np.zeros([32, 32])
        Bi[row_indices, col_indices] = 1
        epsilon_i = np.random.normal(loc = 0, scale = np.sqrt(0.04), size = (32, 32))
        simulated_data[i] = Bi + epsilon_i

    return y, simulated_data

class dataSetPytorch(Dataset):
    def __init__(self, x, y):
        self.x = torch.from_numpy(x.reshape([-1, 1, 32, 32])).float()
        self.y = torch.from_numpy(y)
    def __len__(self):
        return len(self.x)
    def __getitem__(self, idx):
        return self.x[idx], self.y[idx]

def makeTestLoader(numExperiments = 10):
    n_test = 1000
    mu_n = [5, 5, 5, 5, 5, 5, 5, 5, 5]
    mu_c = [10, 10, 10, 20, 20, 20, 30, 30, 30]

    dataLoader_experiment_data = []
    for experiment in range(numExperiments):
        dataLoader_settings = []
        for setting in range(9):

            y, simulated_data = simulateData(n = n_test,
                                             mu_c = mu_c[setting],
                                             mu_n = mu_n[setting])

            datasetSetting = dataSetPytorch(simulated_data, y)
            dataLoader = DataLoader(datasetSetting, batch_size=25, shuffle = True)
            dataLoader_settings.append(dataLoader)

        dataLoader_experiment_data.append(dataLoader_settings)

    return dataLoader_experiment_data
```

In [7]:
```
dataLoader_all_experiments_test = makeTestLoader(numExperiments = 10)
```

Test the models

In [50]:
```
def modelTest(model, test_dataloader):
    accuracy_test = 0
    model.eval()
    with torch.no_grad():
        for x_batch, y_batch in test_dataloader:
            pred = model(x_batch)[:, 0]
            is_correct = ((pred >= 0.5).float() == y_batch).float()
            accuracy_test += is_correct.sum()
    accuracy_test /= len(test_dataloader.dataset)
    print(f'Test Accuracy: {accuracy_test:4f}')
    return accuracy_test.numpy()
```

Note: In the print statment its supposed to be print("Experiment:", experiment, "Setting:", setting + 1) but forgot to change it when i ran the long experiment ..

In [53]:
```
numExperiments = 10
numSettings = 9
accuracyMatrix = np.zeros([numSettings, numExperiments])
for experiment in range(numExperiments):
    dataLoader_individual_experiment_test = dataLoader_all_experiments_test[experiment

    for setting in range(numSettings):
        print("Setting:", setting + 1, "Experiment:", experiment)
        folderPath = 'setting' + str(setting + 1)

        n = len(dataLoader_individual_experiment_test[setting].dataset)
        print("N:", n)

        targetModelName = "Setting" + str(setting + 1) + "_Experiment" + str(experimer
        print(targetModelName)
        modelPath = os.path.join(".", folderPath)  # Constructing the path using os.pa

        # List all files in the modelPath directory that match the targetModelName pat
        modelString = [file for file in glob.glob(os.path.join(modelPath, f"*{targetMo
        print(matching_files)

        modelWeightParams = torch.load(modelString)
        model2.load_state_dict(modelWeightParams)

        accuracy_test = modelTest(model2, dataLoader_individual_experiment_test[settir
        accuracyMatrix[setting, experiment] = accuracy_test
```

```
Setting: 1 Experiment: 0
N: 1000
Setting1_Experiment0
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.604000
Setting: 2 Experiment: 0
N: 1000
Setting2_Experiment0
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.729000
Setting: 3 Experiment: 0
N: 1000
Setting3_Experiment0
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.754000
Setting: 4 Experiment: 0
N: 1000
Setting4_Experiment0
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.968000
Setting: 5 Experiment: 0
N: 1000
Setting5_Experiment0
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.978000
Setting: 6 Experiment: 0
N: 1000
Setting6_Experiment0
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.957000
Setting: 7 Experiment: 0
N: 1000
Setting7_Experiment0
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.988000
Setting: 8 Experiment: 0
N: 1000
Setting8_Experiment0
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.995000
Setting: 9 Experiment: 0
N: 1000
Setting9_Experiment0
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.992000
Setting: 1 Experiment: 1
N: 1000
Setting1_Experiment1
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.732000
Setting: 2 Experiment: 1
N: 1000
Setting2_Experiment1
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.731000
Setting: 3 Experiment: 1
N: 1000
Setting3_Experiment1
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.798000
```

```
Setting: 4 Experiment: 1
N: 1000
Setting4_Experiment1
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.946000
Setting: 5 Experiment: 1
N: 1000
Setting5_Experiment1
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.967000
Setting: 6 Experiment: 1
N: 1000
Setting6_Experiment1
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.957000
Setting: 7 Experiment: 1
N: 1000
Setting7_Experiment1
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.991000
Setting: 8 Experiment: 1
N: 1000
Setting8_Experiment1
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.988000
Setting: 9 Experiment: 1
N: 1000
Setting9_Experiment1
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.991000
Setting: 1 Experiment: 2
N: 1000
Setting1_Experiment2
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.683000
Setting: 2 Experiment: 2
N: 1000
Setting2_Experiment2
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.762000
Setting: 3 Experiment: 2
N: 1000
Setting3_Experiment2
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.752000
Setting: 4 Experiment: 2
N: 1000
Setting4_Experiment2
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.952000
Setting: 5 Experiment: 2
N: 1000
Setting5_Experiment2
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.965000
Setting: 6 Experiment: 2
N: 1000
Setting6_Experiment2
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.967000
```

```
Setting: 7 Experiment: 2
N: 1000
Setting7_Experiment2
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.989000
Setting: 8 Experiment: 2
N: 1000
Setting8_Experiment2
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.990000
Setting: 9 Experiment: 2
N: 1000
Setting9_Experiment2
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.995000
Setting: 1 Experiment: 3
N: 1000
Setting1_Experiment3
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.732000
Setting: 2 Experiment: 3
N: 1000
Setting2_Experiment3
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.771000
Setting: 3 Experiment: 3
N: 1000
Setting3_Experiment3
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.770000
Setting: 4 Experiment: 3
N: 1000
Setting4_Experiment3
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.942000
Setting: 5 Experiment: 3
N: 1000
Setting5_Experiment3
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.957000
Setting: 6 Experiment: 3
N: 1000
Setting6_Experiment3
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.958000
Setting: 7 Experiment: 3
N: 1000
Setting7_Experiment3
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.995000
Setting: 8 Experiment: 3
N: 1000
Setting8_Experiment3
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.994000
Setting: 9 Experiment: 3
N: 1000
Setting9_Experiment3
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.990000
```

```
Setting: 1 Experiment: 4
N: 1000
Setting1_Experiment4
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.724000
Setting: 2 Experiment: 4
N: 1000
Setting2_Experiment4
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.752000
Setting: 3 Experiment: 4
N: 1000
Setting3_Experiment4
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.792000
Setting: 4 Experiment: 4
N: 1000
Setting4_Experiment4
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.941000
Setting: 5 Experiment: 4
N: 1000
Setting5_Experiment4
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.961000
Setting: 6 Experiment: 4
N: 1000
Setting6_Experiment4
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.969000
Setting: 7 Experiment: 4
N: 1000
Setting7_Experiment4
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.989000
Setting: 8 Experiment: 4
N: 1000
Setting8_Experiment4
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.989000
Setting: 9 Experiment: 4
N: 1000
Setting9_Experiment4
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.994000
Setting: 1 Experiment: 5
N: 1000
Setting1_Experiment5
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.712000
Setting: 2 Experiment: 5
N: 1000
Setting2_Experiment5
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.772000
Setting: 3 Experiment: 5
N: 1000
Setting3_Experiment5
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.499000
```

```
Setting: 4 Experiment: 5
N: 1000
Setting4_Experiment5
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.954000
Setting: 5 Experiment: 5
N: 1000
Setting5_Experiment5
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.497000
Setting: 6 Experiment: 5
N: 1000
Setting6_Experiment5
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.966000
Setting: 7 Experiment: 5
N: 1000
Setting7_Experiment5
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.983000
Setting: 8 Experiment: 5
N: 1000
Setting8_Experiment5
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.992000
Setting: 9 Experiment: 5
N: 1000
Setting9_Experiment5
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.996000
Setting: 1 Experiment: 6
N: 1000
Setting1_Experiment6
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.467000
Setting: 2 Experiment: 6
N: 1000
Setting2_Experiment6
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.756000
Setting: 3 Experiment: 6
N: 1000
Setting3_Experiment6
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.511000
Setting: 4 Experiment: 6
N: 1000
Setting4_Experiment6
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.948000
Setting: 5 Experiment: 6
N: 1000
Setting5_Experiment6
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.515000
Setting: 6 Experiment: 6
N: 1000
Setting6_Experiment6
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.964000
```

```
Setting: 7 Experiment: 6
N: 1000
Setting7_Experiment6
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.471000
Setting: 8 Experiment: 6
N: 1000
Setting8_Experiment6
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.482000
Setting: 9 Experiment: 6
N: 1000
Setting9_Experiment6
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.517000
Setting: 1 Experiment: 7
N: 1000
Setting1_Experiment7
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.714000
Setting: 2 Experiment: 7
N: 1000
Setting2_Experiment7
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.518000
Setting: 3 Experiment: 7
N: 1000
Setting3_Experiment7
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.483000
Setting: 4 Experiment: 7
N: 1000
Setting4_Experiment7
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.951000
Setting: 5 Experiment: 7
N: 1000
Setting5_Experiment7
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.966000
Setting: 6 Experiment: 7
N: 1000
Setting6_Experiment7
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.983000
Setting: 7 Experiment: 7
N: 1000
Setting7_Experiment7
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.999000
Setting: 8 Experiment: 7
N: 1000
Setting8_Experiment7
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.991000
Setting: 9 Experiment: 7
N: 1000
Setting9_Experiment7
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.997000
```

```
Setting: 1 Experiment: 8
N: 1000
Setting1_Experiment8
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.470000
Setting: 2 Experiment: 8
N: 1000
Setting2_Experiment8
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.744000
Setting: 3 Experiment: 8
N: 1000
Setting3_Experiment8
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.752000
Setting: 4 Experiment: 8
N: 1000
Setting4_Experiment8
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.941000
Setting: 5 Experiment: 8
N: 1000
Setting5_Experiment8
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.958000
Setting: 6 Experiment: 8
N: 1000
Setting6_Experiment8
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.964000
Setting: 7 Experiment: 8
N: 1000
Setting7_Experiment8
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.990000
Setting: 8 Experiment: 8
N: 1000
Setting8_Experiment8
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.992000
Setting: 9 Experiment: 8
N: 1000
Setting9_Experiment8
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.996000
Setting: 1 Experiment: 9
N: 1000
Setting1_Experiment9
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.766000
Setting: 2 Experiment: 9
N: 1000
Setting2_Experiment9
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.768000
Setting: 3 Experiment: 9
N: 1000
Setting3_Experiment9
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.538000
```

```
Setting: 4 Experiment: 9
N: 1000
Setting4_Experiment9
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.962000
Setting: 5 Experiment: 9
N: 1000
Setting5_Experiment9
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.954000
Setting: 6 Experiment: 9
N: 1000
Setting6_Experiment9
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.960000
Setting: 7 Experiment: 9
N: 1000
Setting7_Experiment9
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.502000
Setting: 8 Experiment: 9
N: 1000
Setting8_Experiment9
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.510000
Setting: 9 Experiment: 9
N: 1000
Setting9_Experiment9
.\setting9\modelSetting9_Experiment9_23_02_2024_epoch_81.pth
Test Accuracy: 0.989000
```

In [54]:
```
accuracyMatrix
```

Out[54]:
```
array([[0.60399997, 0.73199999, 0.68300003, 0.73199999, 0.72399998,
        0.71200001, 0.46700001, 0.71399999, 0.47      , 0.76599997],
       [0.72899997, 0.73100001, 0.76200002, 0.77100003, 0.75199997,
        0.77200001, 0.75599998, 0.51800001, 0.74400002, 0.76800001],
       [0.75400001, 0.79799998, 0.75199997, 0.76999998, 0.792     ,
        0.49900001, 0.51099998, 0.48300001, 0.75199997, 0.53799999],
       [0.96799999, 0.94599998, 0.95200002, 0.94199997, 0.94099998,
        0.954     , 0.94800001, 0.95099998, 0.94099998, 0.96200001],
       [0.97799999, 0.96700001, 0.96499997, 0.95700002, 0.96100003,
        0.49700001, 0.51499999, 0.96600002, 0.958     , 0.954     ],
       [0.95700002, 0.95700002, 0.96700001, 0.958     , 0.96899998,
        0.96600002, 0.96399999, 0.98299998, 0.96399999, 0.95999998],
       [0.98799998, 0.991     , 0.98900002, 0.995     , 0.98900002,
        0.98299998, 0.47099999, 0.99900001, 0.99000001, 0.50199997],
       [0.995     , 0.98799998, 0.99000001, 0.99400002, 0.98900002,
        0.99199998, 0.48199999, 0.991     , 0.99199998, 0.50999999],
       [0.99199998, 0.991     , 0.995     , 0.99000001, 0.99400002,
        0.99599999, 0.51700002, 0.99699998, 0.99599999, 0.98900002]])
```

In [60]:
```
n = [200, 500, 1000, 200, 500, 1000, 200, 500, 1000]
mu_n = [5, 5, 5, 5, 5, 5, 5, 5, 5]
mu_c = [10, 10, 10, 20, 20, 20, 30, 30, 30]
mean_accuracy = accuracyMatrix.mean(axis = 1)
std_accuracy = accuracyMatrix.std(axis = 1)
```

In [63]:
```
# Creating DataFrame
data = {
```

```
    'n': n,
    'mu_n': mu_n,
    'mu_c': mu_c,
    'mean_accuracy': mean_accuracy,
    'std_accuracy': std_accuracy
}

DataSettignsdf = pd.DataFrame(data)
display(DataSettignsdf)
```

	n	mu_n	mu_c	mean_accuracy	std_accuracy
0	200	5	10	0.6604	0.104083
1	500	5	10	0.7303	0.072259
2	1000	5	10	0.6649	0.129779
3	200	5	20	0.9505	0.008559
4	500	5	20	0.8718	0.183053
5	1000	5	20	0.9645	0.007393
6	200	5	30	0.8897	0.201759
7	500	5	30	0.8923	0.198259
8	1000	5	30	0.9457	0.142924

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.nn.init as init
import os, glob
from datetime import datetime
import pandas as pd
from torch.utils.data import Dataset, DataLoader
from sklearn.model_selection import GridSearchCV
from skorch import NeuralNetClassifier
```

```
C:\Users\Isaac\anaconda3\envs\CS273\lib\site-packages\tqdm\auto.py:21: TqdmWarning: I
Progress not found. Please update jupyter and ipywidgets. See https://ipywidgets.read
thedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

Question 3

CNN tuning: Focus on simulation setting 7 to optimize your CNN for improved classification performance on the test set. Generate an independent validation set of 1000 subjects to assist in tuning. Define a search space for the number of convolution layers, the width of the final fully connected layer, and the optimizer's learning rate. Conduct a grid search over the defined search space to identify the best hyperparameter combination. Report the search space, selected hyperparameters, and the classification accuracy for each combination tested.

- Use model a base model.
  - Remember only using data setting 7. [X]
  - have only one train (n_i), one validation (1000), and one test set (1000) [X]
  - number of convolution layers: 1, 2, 3, 4, 5
  - Readjust model with different final fully connected layers: 8, 16, 32, 64, 128
  - lr = 0.001, .01, .1

In [2]:
```
def simulateData(n, mu_c, mu_n):

    y = np.random.choice([0, 1], size = n, p = [0.5, 0.5])
    m_i = np.random.poisson(lam = mu_c, size = n) * y + np.random.poisson(lam = mu_n,

    simulated_data = np.zeros([n, 32, 32])
    for i in range(n):
        random_indices = np.random.choice(32 * 32, m_i[i], replace = False)
        row_indices, col_indices = np.unravel_index(random_indices, (32, 32))
        Bi = np.zeros([32, 32])
        Bi[row_indices, col_indices] = 1
        epsilon_i = np.random.normal(loc = 0, scale = np.sqrt(0.04), size = (32, 32))
        simulated_data[i] = Bi + epsilon_i

    return y, simulated_data
```

In [3]:
```
y_train, X_train = simulateData(n = 200, mu_c = 5, mu_n = 30)
y_val, X_val = simulateData(n = 1000, mu_c = 5, mu_n = 30)
```

```
y_test, X_test = simulateData(n = 1000, mu_c = 5, mu_n = 30)
```

In [4]:
```
class dataSetPytorch(Dataset):
    def __init__(self, x, y):
        self.x = torch.from_numpy(x.reshape([-1, 1, 32, 32])).float()
        self.y = torch.from_numpy(y)
    def __len__(self):
        return len(self.x)
    def __getitem__(self, idx):
        return self.x[idx], self.y[idx]

def reset_weights(model):
    for m in model.modules():
        if isinstance(m, nn.Conv2d) or isinstance(m, nn.Linear):
            init.xavier_uniform_(m.weight)
    return

def saveModel(model, path):
    torch.save(model.state_dict(), path)
```

In [5]:
```
exampleImg = X_train[0, :, :].reshape([1, 1, 32, 32])
exampleImg = torch.from_numpy(exampleImg).float()
exampleImg.shape
```

Out[5]:
```
torch.Size([1, 1, 32, 32])
```

Model Base

In [6]:
```
# Define the base model
class BaseCNN(torch.nn.Module):
    def __init__(self, num_conv_layers, final_fc_width):
        super(BaseCNN, self).__init__()

        self.num_conv_layers = num_conv_layers

        # Define convolutional layers dynamically based on num_conv_layers
        self.conv_layers = nn.ModuleList()

        in_channels = 1  #  input channels = 1
        out_channels = 2  # Initial output channels
        for i in range(num_conv_layers):
            self.conv_layers.append(nn.Conv2d(in_channels, out_channels, kernel_size=3
            self.conv_layers.append(nn.ReLU())
            self.conv_layers.append(nn.MaxPool2d(kernel_size=2))
            # Update in_channels and out_channels for the next layer
            in_channels = out_channels
            out_channels *= 2  # Double the channels for each subsequent layer

        # Calculate the input size for the fully connected layer
        conv_output_size = self._get_conv_output_size()

        self.flatten = torch.nn.Flatten()
        self.fc = nn.Linear(conv_output_size, final_fc_width)
        self.relu = nn.ReLU()
        self.output_layer = nn.Linear(final_fc_width, 1)
        self.sigmoid = torch.nn.Sigmoid()

    def forward(self, x):
```

```
        # Forward pass through convolutional layers
        for conv_layer in self.conv_layers:
            x = conv_layer(x)

        # Flatten the output from convolutional layers
        x = self.flatten(x)

        # Forward pass through fully connected layers
        x = self.relu(self.fc(x))
        x = self.output_layer(x)
        x = self.sigmoid(x)
        return x

    def _get_conv_output_size(self):
        # Method to calculate the output size of the convolutional layers
        x = torch.rand(1, 1, 32, 32)  # Assuming input size is 28x28
        for conv_layer in self.conv_layers:
            x = conv_layer(x)
        return x.view(1, -1).size(1)

    def reset_weights(self):
        for m in self.modules():
            if isinstance(m, nn.Conv2d) or isinstance(m, nn.Linear):
                init.xavier_uniform_(m.weight)
        return

    def fit(self, X_train, X_val, y_train, y_val, learningRate, num_epochs = 200):

        datasetSetting_train = dataSetPytorch(X_train, y_train)
        train_dl = DataLoader(datasetSetting_train, batch_size=25, shuffle = True)

        datasetSetting_val = dataSetPytorch(X_val, y_val)
        valid_dl = DataLoader(datasetSetting_val, batch_size=25, shuffle = True)

        # reinitialize weights!
        self.reset_weights()

        loss_fn = torch.nn.BCELoss()
        optimizer = torch.optim.Adam(model.parameters(), lr = learningRate)
        loss_hist_train = [0] * num_epochs
        accuracy_hist_train = [0] * num_epochs
        loss_hist_valid = [0] * num_epochs
        accuracy_hist_valid = [0] * num_epochs

        best_loss = torch.inf

        for epoch in range(num_epochs):
            self.train()

            for x_batch, y_batch in train_dl:
                pred = self(x_batch)[:, 0]
                loss = loss_fn(pred, y_batch.float())
                loss.backward()
                optimizer.step()
                optimizer.zero_grad()
                loss_hist_train[epoch] += loss.item() * y_batch.size(0)

                is_correct = ((pred >= 0.5).float() == y_batch).float()
                accuracy_hist_train[epoch] += is_correct.sum()
            loss_hist_train[epoch] /= len(train_dl.dataset)
```

```
            accuracy_hist_train[epoch] /= len(train_dl.dataset)

            self.eval()
            with torch.no_grad():
                for x_batch, y_batch in valid_dl:
                    pred = self(x_batch)[:, 0]
                    loss = loss_fn(pred, y_batch.float())
                    loss_hist_valid[epoch] += loss.item() * y_batch.size(0)

                    is_correct = ((pred >= 0.5).float() == y_batch).float()
                    accuracy_hist_valid[epoch] += is_correct.sum()
            loss_hist_valid[epoch] /= len(valid_dl.dataset)
            accuracy_hist_valid[epoch] /= len(valid_dl.dataset)

            if (best_loss > loss_hist_valid[epoch]):
                saveModel(model, "bestModel.pth")
                best_loss = loss_hist_valid[epoch]

        modelWeightParams = torch.load("bestModel.pth")
        self.load_state_dict(modelWeightParams)

    def predict_score(self, X_test, y_test):
        datasetSetting_test = dataSetPytorch(X_test, y_test)
        test_dl = DataLoader(datasetSetting_test, batch_size=25, shuffle = True)

        # model is done training, now evaluate the test accuracy score here.
        accuracy_test = 0
        self.eval()
        with torch.no_grad():
            for x_batch, y_batch in test_dl:
                pred = self(x_batch)[:, 0]
                is_correct = ((pred >= 0.5).float() == y_batch).float()
                accuracy_test += is_correct.sum()
        accuracy_test /= len(test_dl.dataset)

        return accuracy_test.numpy()
```

Out[6]:
```
array(0.973, dtype=float32)
```

Grid Search.

Im making my own custom one, oculdnt figure it out with sklearn. Im going to continue what i did and select best model based on best validation performance, since i care most about accuracy. Then im going to run the model on the test data set. Im going to attempt all possible combinations here, and I will select the best model with whatever parameters performed best.

Trial run

In [ ]:
```
param_dict = {
    'final_fc_width': 8,
    'num_conv_layers': 1,
    'learningRate': 0.001,
}
```

```
# Create an instance of MyModel
model = BaseCNN(num_conv_layers = param_dict["num_conv_layers"], final_fc_width = para
model.fit(X_train, X_val, y_train, y_val,
          param_dict['learningRate'], num_epochs = 200)
model.predict_score(X_test, y_test)
```

Full send it ma boi

In [10]:

```
import itertools

# Define the hyperparameter grid
param_grid = {
    'final_fc_width': [8, 16, 32, 64],
    'num_conv_layers': [1, 2, 3, 4, 5],
    'learningRate': [0.001, 0.005, 0.01, 0.1],
}

# Generate all combinations of hyperparameters
param_combinations = list(itertools.product(*param_grid.values()))

final_fc_width = []
num_conv_layers = []
learningRate = []
accuracy = []

# Print the combinations
for params in param_combinations:
    print("Running final_fc_width", params[0], "num_conv_layers", params[1], "learning
    # Create an instance of MyModel
    model = BaseCNN(num_conv_layers = params[1], final_fc_width = params[0])
    model.fit(X_train, X_val, y_train, y_val, learningRate = params[2], num_epochs = 2
    accuracy.append(model.predict_score(X_test, y_test))
    final_fc_width.append(params[0])
    num_conv_layers.append(params[1])
    learningRate.append(params[2])
```

```
Running final_fc_width 8 num_conv_layers 1 learningRate 0.001
Running final_fc_width 8 num_conv_layers 1 learningRate 0.005
Running final_fc_width 8 num_conv_layers 1 learningRate 0.01
Running final_fc_width 8 num_conv_layers 1 learningRate 0.1
Running final_fc_width 8 num_conv_layers 2 learningRate 0.001
Running final_fc_width 8 num_conv_layers 2 learningRate 0.005
Running final_fc_width 8 num_conv_layers 2 learningRate 0.01
Running final_fc_width 8 num_conv_layers 2 learningRate 0.1
Running final_fc_width 8 num_conv_layers 3 learningRate 0.001
Running final_fc_width 8 num_conv_layers 3 learningRate 0.005
Running final_fc_width 8 num_conv_layers 3 learningRate 0.01
Running final_fc_width 8 num_conv_layers 3 learningRate 0.1
Running final_fc_width 8 num_conv_layers 4 learningRate 0.001
Running final_fc_width 8 num_conv_layers 4 learningRate 0.005
Running final_fc_width 8 num_conv_layers 4 learningRate 0.01
Running final_fc_width 8 num_conv_layers 4 learningRate 0.1
Running final_fc_width 8 num_conv_layers 5 learningRate 0.001
Running final_fc_width 8 num_conv_layers 5 learningRate 0.005
Running final_fc_width 8 num_conv_layers 5 learningRate 0.01
Running final_fc_width 8 num_conv_layers 5 learningRate 0.1
Running final_fc_width 16 num_conv_layers 1 learningRate 0.001
Running final_fc_width 16 num_conv_layers 1 learningRate 0.005
Running final_fc_width 16 num_conv_layers 1 learningRate 0.01
Running final_fc_width 16 num_conv_layers 1 learningRate 0.1
Running final_fc_width 16 num_conv_layers 2 learningRate 0.001
Running final_fc_width 16 num_conv_layers 2 learningRate 0.005
Running final_fc_width 16 num_conv_layers 2 learningRate 0.01
Running final_fc_width 16 num_conv_layers 2 learningRate 0.1
Running final_fc_width 16 num_conv_layers 3 learningRate 0.001
Running final_fc_width 16 num_conv_layers 3 learningRate 0.005
Running final_fc_width 16 num_conv_layers 3 learningRate 0.01
Running final_fc_width 16 num_conv_layers 3 learningRate 0.1
Running final_fc_width 16 num_conv_layers 4 learningRate 0.001
Running final_fc_width 16 num_conv_layers 4 learningRate 0.005
Running final_fc_width 16 num_conv_layers 4 learningRate 0.01
Running final_fc_width 16 num_conv_layers 4 learningRate 0.1
Running final_fc_width 16 num_conv_layers 5 learningRate 0.001
Running final_fc_width 16 num_conv_layers 5 learningRate 0.005
Running final_fc_width 16 num_conv_layers 5 learningRate 0.01
Running final_fc_width 16 num_conv_layers 5 learningRate 0.1
Running final_fc_width 32 num_conv_layers 1 learningRate 0.001
Running final_fc_width 32 num_conv_layers 1 learningRate 0.005
Running final_fc_width 32 num_conv_layers 1 learningRate 0.01
Running final_fc_width 32 num_conv_layers 1 learningRate 0.1
Running final_fc_width 32 num_conv_layers 2 learningRate 0.001
Running final_fc_width 32 num_conv_layers 2 learningRate 0.005
Running final_fc_width 32 num_conv_layers 2 learningRate 0.01
Running final_fc_width 32 num_conv_layers 2 learningRate 0.1
Running final_fc_width 32 num_conv_layers 3 learningRate 0.001
Running final_fc_width 32 num_conv_layers 3 learningRate 0.005
Running final_fc_width 32 num_conv_layers 3 learningRate 0.01
Running final_fc_width 32 num_conv_layers 3 learningRate 0.1
Running final_fc_width 32 num_conv_layers 4 learningRate 0.001
Running final_fc_width 32 num_conv_layers 4 learningRate 0.005
Running final_fc_width 32 num_conv_layers 4 learningRate 0.01
Running final_fc_width 32 num_conv_layers 4 learningRate 0.1
Running final_fc_width 32 num_conv_layers 5 learningRate 0.001
Running final_fc_width 32 num_conv_layers 5 learningRate 0.005
Running final_fc_width 32 num_conv_layers 5 learningRate 0.01
Running final_fc_width 32 num_conv_layers 5 learningRate 0.1
```

```
Running final_fc_width 64 num_conv_layers 1 learningRate 0.001
Running final_fc_width 64 num_conv_layers 1 learningRate 0.005
Running final_fc_width 64 num_conv_layers 1 learningRate 0.01
Running final_fc_width 64 num_conv_layers 1 learningRate 0.1
Running final_fc_width 64 num_conv_layers 2 learningRate 0.001
Running final_fc_width 64 num_conv_layers 2 learningRate 0.005
Running final_fc_width 64 num_conv_layers 2 learningRate 0.01
Running final_fc_width 64 num_conv_layers 2 learningRate 0.1
Running final_fc_width 64 num_conv_layers 3 learningRate 0.001
Running final_fc_width 64 num_conv_layers 3 learningRate 0.005
Running final_fc_width 64 num_conv_layers 3 learningRate 0.01
Running final_fc_width 64 num_conv_layers 3 learningRate 0.1
Running final_fc_width 64 num_conv_layers 4 learningRate 0.001
Running final_fc_width 64 num_conv_layers 4 learningRate 0.005
Running final_fc_width 64 num_conv_layers 4 learningRate 0.01
Running final_fc_width 64 num_conv_layers 4 learningRate 0.1
Running final_fc_width 64 num_conv_layers 5 learningRate 0.001
Running final_fc_width 64 num_conv_layers 5 learningRate 0.005
Running final_fc_width 64 num_conv_layers 5 learningRate 0.01
Running final_fc_width 64 num_conv_layers 5 learningRate 0.1
```

In [28]:

```
pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)

param_combinations = {
    'final_fc_width': final_fc_width,
    'num_conv_layers': num_conv_layers,
    'learningRate':learningRate,
    'accuracy': accuracy
}

DataSettignsdf = pd.DataFrame(param_combinations)
display(DataSettignsdf)
```

	final_fc_width	num_conv_layers	learningRate	accuracy
0	8	1	0.001	0.947
1	8	1	0.005	0.988
2	8	1	0.010	0.979
3	8	1	0.100	0.97
4	8	2	0.001	0.997
5	8	2	0.005	0.997
6	8	2	0.010	0.993
7	8	2	0.100	0.499
8	8	3	0.001	0.989
9	8	3	0.005	0.995
10	8	3	0.010	0.499
11	8	3	0.100	0.664
12	8	4	0.001	0.993
13	8	4	0.005	0.988
14	8	4	0.010	0.992
15	8	4	0.100	0.499
16	8	5	0.001	0.984
17	8	5	0.005	0.499
18	8	5	0.010	0.994
19	8	5	0.100	0.501
20	16	1	0.001	0.838
21	16	1	0.005	0.941
22	16	1	0.010	0.499
23	16	1	0.100	0.904
24	16	2	0.001	0.997
25	16	2	0.005	0.998
26	16	2	0.010	0.998
27	16	2	0.100	0.499
28	16	3	0.001	0.993
29	16	3	0.005	0.993
30	16	3	0.010	0.995
31	16	3	0.100	0.499
32	16	4	0.001	0.992
33	16	4	0.005	0.985

	final_fc_width	num_conv_layers	learningRate	accuracy
34	16	4	0.010	0.983
35	16	4	0.100	0.499
36	16	5	0.001	0.992
37	16	5	0.005	0.995
38	16	5	0.010	0.995
39	16	5	0.100	0.499
40	32	1	0.001	0.963
41	32	1	0.005	0.986
42	32	1	0.010	0.863
43	32	1	0.100	0.927
44	32	2	0.001	0.995
45	32	2	0.005	0.99
46	32	2	0.010	0.997
47	32	2	0.100	0.972
48	32	3	0.001	0.995
49	32	3	0.005	0.991
50	32	3	0.010	0.994
51	32	3	0.100	0.499
52	32	4	0.001	0.989
53	32	4	0.005	0.992
54	32	4	0.010	0.99
55	32	4	0.100	0.499
56	32	5	0.001	0.98
57	32	5	0.005	0.989
58	32	5	0.010	0.992
59	32	5	0.100	0.499
60	64	1	0.001	0.96
61	64	1	0.005	0.973
62	64	1	0.010	0.931
63	64	1	0.100	0.501
64	64	2	0.001	0.993
65	64	2	0.005	0.995
66	64	2	0.010	0.995
67	64	2	0.100	0.499

	final_fc_width	num_conv_layers	learningRate	accuracy
68	64	3	0.001	0.996
69	64	3	0.005	0.989
70	64	3	0.010	0.995
71	64	3	0.100	0.499
72	64	4	0.001	0.99
73	64	4	0.005	0.993
74	64	4	0.010	0.988
75	64	4	0.100	0.499
76	64	5	0.001	0.991
77	64	5	0.005	0.993
78	64	5	0.010	0.989
79	64	5	0.100	0.499

In [29]:
```
indexMax = np.argmax(DataSettignsdf['accuracy'].values)
DataSettignsdf.iloc[[indexMax]]
```

Out[29]:

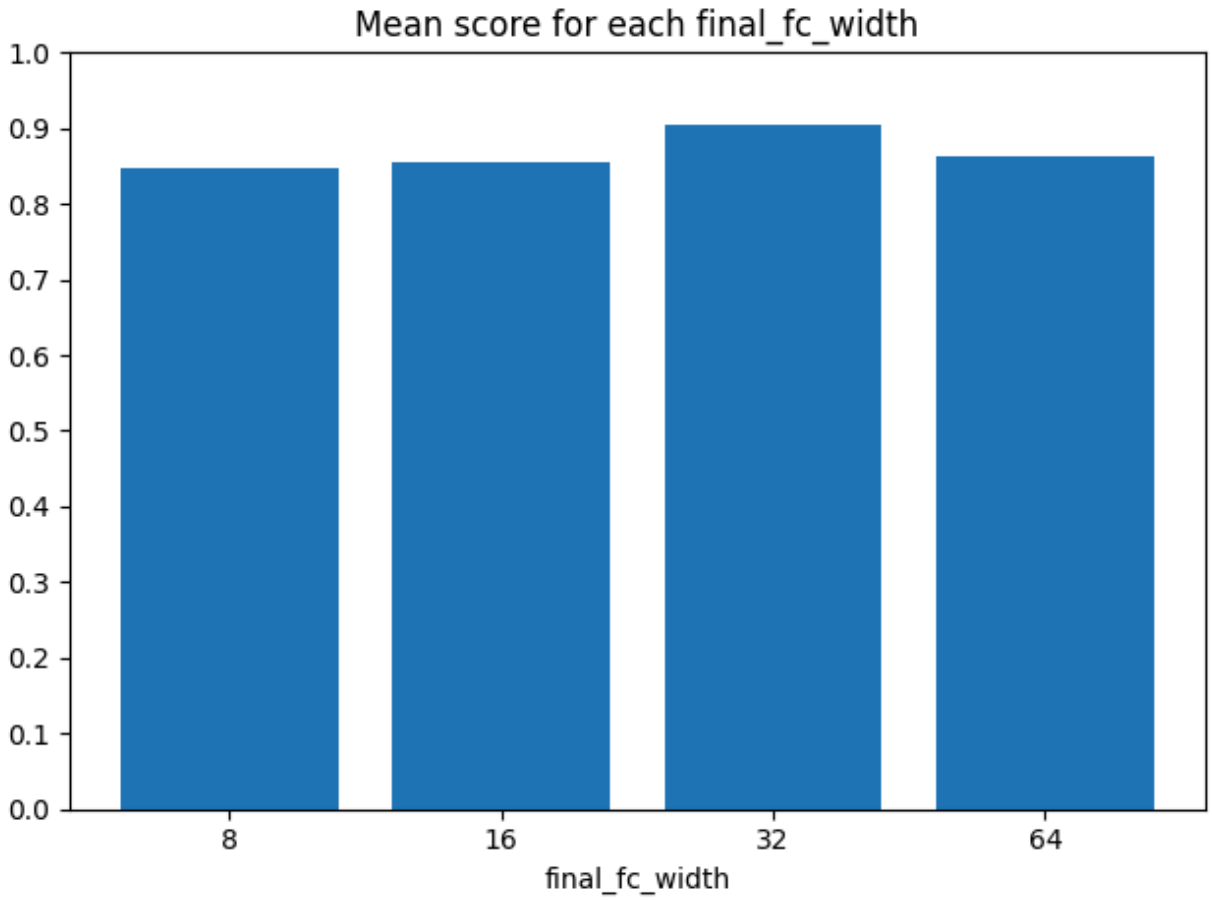
	final_fc_width	num_conv_layers	learningRate	accuracy
25	16	2	0.005	0.998

Lets explore the data now

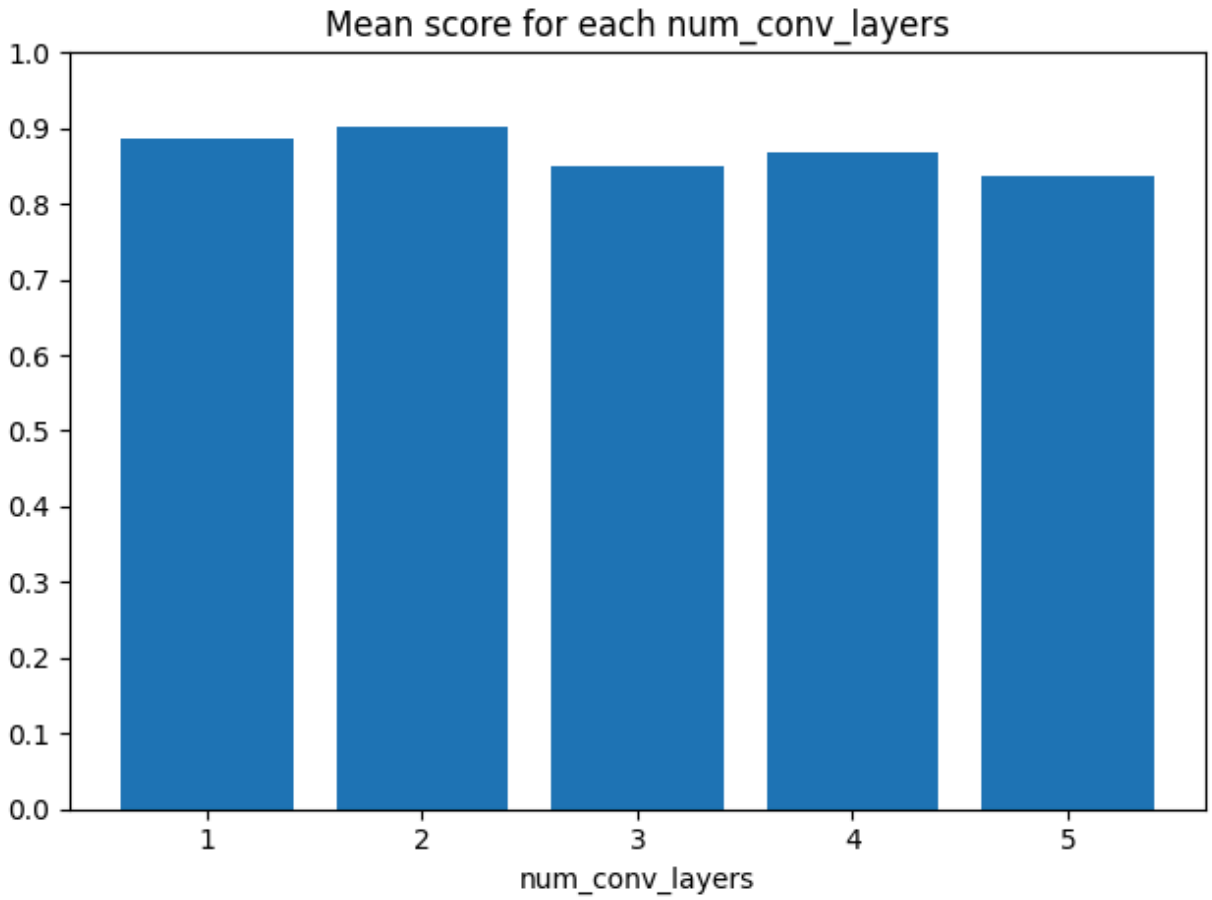
In [34]:
```
param_grid
```

Out[34]:
```
{'final_fc_width': [8, 16, 32, 64],
 'num_conv_layers': [1, 2, 3, 4, 5],
 'learningRate': [0.001, 0.005, 0.01, 0.1]}
```

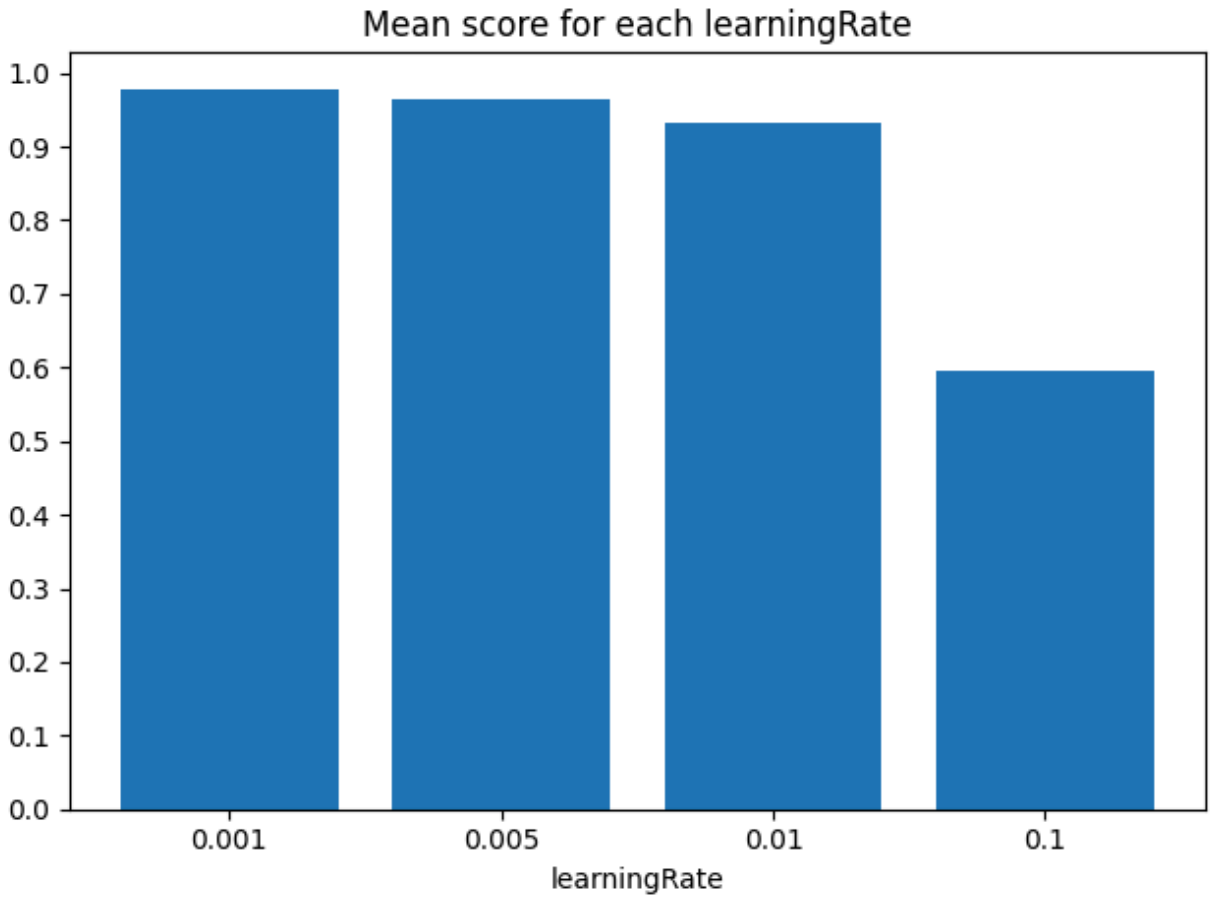
In [56]:
```
score_avg = []
for item in param_grid["final_fc_width"]:
    score_avg.append(DataSettignsdf[DataSettignsdf["final_fc_width"] == item]["accurac

plt.figure()
plt.bar(["8", "16", "32", "64"], score_avg)
ax = plt.gca()
ax.set_yticks([i/10 for i in range(11)])
plt.title("Mean score for each final_fc_width")
plt.xlabel("final_fc_width")
plt.tight_layout()
```



In [55]:
```
score_avg = []
for item in param_grid["num_conv_layers"]:
    score_avg.append(DataSettignsdf[DataSettignsdf["num_conv_layers"] == item]["accura

plt.figure()
plt.bar(["1", "2", "3", "4", "5"], score_avg)
ax = plt.gca()
ax.set_yticks([i/10 for i in range(11)])
plt.title("Mean score for each num_conv_layers")
plt.xlabel("num_conv_layers")
plt.tight_layout()
```



In [57]:
```
score_avg = []
for item in param_grid["learningRate"]:
    score_avg.append(DataSettignsdf[DataSettignsdf["learningRate"] == item]["accuracy"

plt.figure()
plt.bar(["0.001", "0.005", "0.01", "0.1"], score_avg)
ax = plt.gca()
ax.set_yticks([i/10 for i in range(11)])
plt.title("Mean score for each learningRate")
plt.xlabel("learningRate")
plt.tight_layout()
```


Mean score for each learningRate
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.001
0.005
0.01
0.1
learningRate