COMP-SCI-5567-0001

Deep Learning

Deep Learning Project 2: A Comparison of Architectures

By

Name: Sai Prudhvi Charan Pothumsetty,

Student ID: 16343752,

Professor: Jesse Lowe

University of Missouri Kansas City.

Experiments Select or modify the values -- HOLD the other values **static** while iterating procedurally where appropriate.

Observations while adjusting the Parameters

Fully Connected Models

Number of Hidden Layers and Layer Sizes

- Increasing the number of hidden layers and the sizes of these layers generally improved the model's performance, up to a certain point. For example, Configuration-3 with 3 hidden layers of sizes [128, 256, 512] achieved the highest accuracy of 88.92%.
- However, excessively large models with too many layers or nodes, such as Configuration-12 with 4 hidden layers and sizes [64, 128, 256, 512], showed a slight decrease in accuracy (88.66%), potentially due to overfitting or increased complexity without significant performance gains.

Batch Size

- The batch size did not seem to have a significant impact on the model's performance within the range of values tested (32 to 512). Most configurations used a batch size of 64, which appeared to work well for this task.

Learning Rate and Optimizer

- The learning rate of 0.001 and the Adam optimizer were commonly used across the top-performing configurations, suggesting that these hyperparameters were suitable for this task.
- Some configurations with different optimizers, such as RMSprop (Configuration-2 and Configuration-7), also performed reasonably well, but did not outperform the top configurations with Adam.

Convolutional Neural Networks

Number of Layers and Filters

- Increasing the number of layers and filters generally improved the model's performance, as evidenced by Configuration-2 with 3 layers and 32 filters achieving the highest accuracy of 91.44%.
- However, the limited number of configurations provided for CNNs makes it difficult to draw definitive conclusions about the optimal number of layers and filters for this task.

Kernel Size and Pool Size

- The kernel size and pool size did not seem to have a significant impact on the model's performance within the range of values tested, as long as they were within reasonable bounds.

Batch Size and Learning Rate

- Similar to the FC models, the batch size did not appear to be a critical factor within the tested range.
- The learning rate varied across configurations, with values ranging from 0.001 to 0.01, but its impact on performance was not clearly observable from the limited results.

Dropout and Regularization

- The use of dropout as a regularization technique was present in some configurations, but its impact on performance was not conclusive from the provided results.

Overall, the CNN models outperformed the FC models for this image classification task, which is expected due to their ability to capture spatial and local features in images. However, it's important to note that these observations are based on the limited set of configurations provided, and further experimentation with a wider range of hyperparameters and architectures may be necessary to fully

Experiments Select or modify the values -- HOLD the other values **static** while iterating procedurally where appropriate.

understand their impact on performance.Additionally, factors such as the nature of the data, the complexity of the task, and the availability of computational resources should also be considered when choosing between FC and CNN models, as well as when tuning their respective hyperparameters.

Part 1- FC

Configurations:

```
configurations = [
    {
        'name': "Configuration-1",
        'num_hidden_layers': 3,
        'hidden_layer_sizes': [32, 64, 128],
        'batch_size': 64,
        'learning_rate': 0.001,
        'optimizer': optim.Adam
    },
    {
        'name': "Configuration-2",
        'num_hidden_layers': 3,
        'hidden_layer_sizes': [64, 128, 256],
        'batch_size': 512,
        'learning_rate': 0.001,
        'optimizer': optim.RMSprop
    },
    {
        'name': "Configuration-3",
        'num_hidden_layers': 3,
        'hidden_layer_sizes': [128, 256, 512],
        'batch_size': 64,
        'learning_rate': 0.001,
        'optimizer': optim.Adam
    },
    {
        'name': "Configuration-4",
        'num_hidden_layers': 2,
        'hidden_layer_sizes': [64, 128],
        'batch_size': 64,
        'learning_rate': 0.001,
        'optimizer': optim.Adam
    },
    {
        'name': "Configuration-5",
```

Experiments Select or modify the values -- HOLD the other values **static** while iterating procedurally where appropriate.

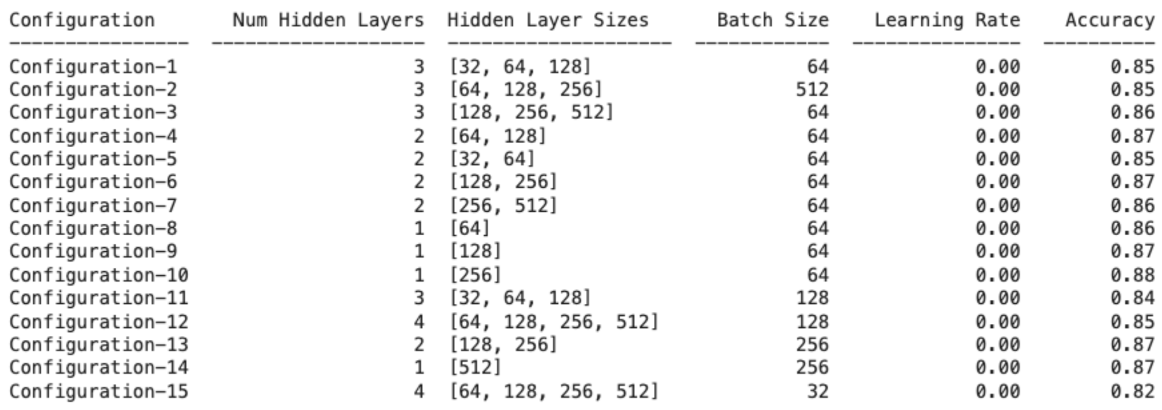
```
        'num_hidden_layers': 2,
        'hidden_layer_sizes': [32, 64],
        'batch_size': 64,
        'learning_rate': 0.001,
        'optimizer': optim.Adam
    },
    {
        'name': "Configuration-6",
        'num_hidden_layers': 2,
        'hidden_layer_sizes': [128, 256],
        'batch_size': 64,
        'learning_rate': 0.001,
        'optimizer': optim.Adam
    },
    {
        'name': "Configuration-7",
        'num_hidden_layers': 2,
        'hidden_layer_sizes': [256, 512],
        'batch_size': 64,
        'learning_rate': 0.001,
        'optimizer': optim.RMSprop
    },
    {
        'name': "Configuration-8",
        'num_hidden_layers': 1,
        'hidden_layer_sizes': [64],
        'batch_size': 64,
        'learning_rate': 0.001,
        'optimizer': optim.Adam
    },
    {
        'name': "Configuration-9",
        'num_hidden_layers': 1,
        'hidden_layer_sizes': [128],
        'batch_size': 64,
        'learning_rate': 0.001,
        'optimizer': optim.Adam
    },
    {
        'name': "Configuration-10",
        'num_hidden_layers': 1,
        'hidden_layer_sizes': [256],
        'batch_size': 64,
        'learning_rate': 0.001,
```

Experiments Select or modify the values -- HOLD the other values **static** while iterating procedurally where appropriate.

```
        'optimizer': optim.Adam
    },
    {
        'name': "Configuration-11",
        'num_hidden_layers': 3,
        'hidden_layer_sizes': [32, 64, 128],
        'batch_size': 128,
        'learning_rate': 0.001,
        'optimizer': optim.RMSprop
    },
    {
        'name': "Configuration-12",
        'num_hidden_layers': 4,
        'hidden_layer_sizes': [64, 128, 256, 512],
        'batch_size': 128,
        'learning_rate': 0.001,
        'optimizer': optim.Adam
    },
    {
        'name': "Configuration-13",
        'num_hidden_layers': 2,
        'hidden_layer_sizes': [128, 256],
        'batch_size': 256,
        'learning_rate': 0.001,
        'optimizer': optim.RMSprop
    },
    {
        'name': "Configuration-14",
        'num_hidden_layers': 1,
        'hidden_layer_sizes': [512],
        'batch_size': 256,
        'learning_rate': 0.001,
        'optimizer': optim.Adam
    },
    {
        'name': "Configuration-15",
        'num_hidden_layers': 4,
        'hidden_layer_sizes': [64, 128, 256, 512],
        'batch_size': 32,
        'learning_rate': 0.001,
        'optimizer': optim.RMSprop
    }
]
```

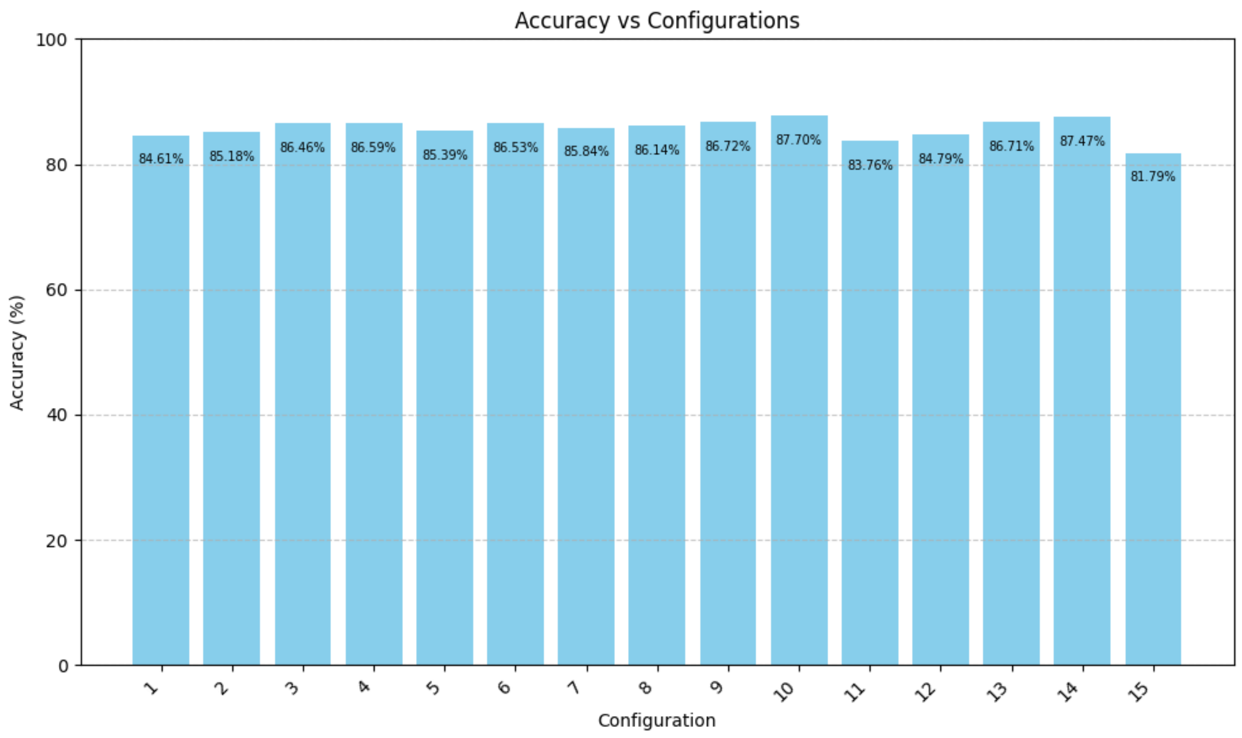
Experiments Select or modify the values -- HOLD the other values **static** while iterating procedurally where appropriate.

<u>Results of Configurations:</u>



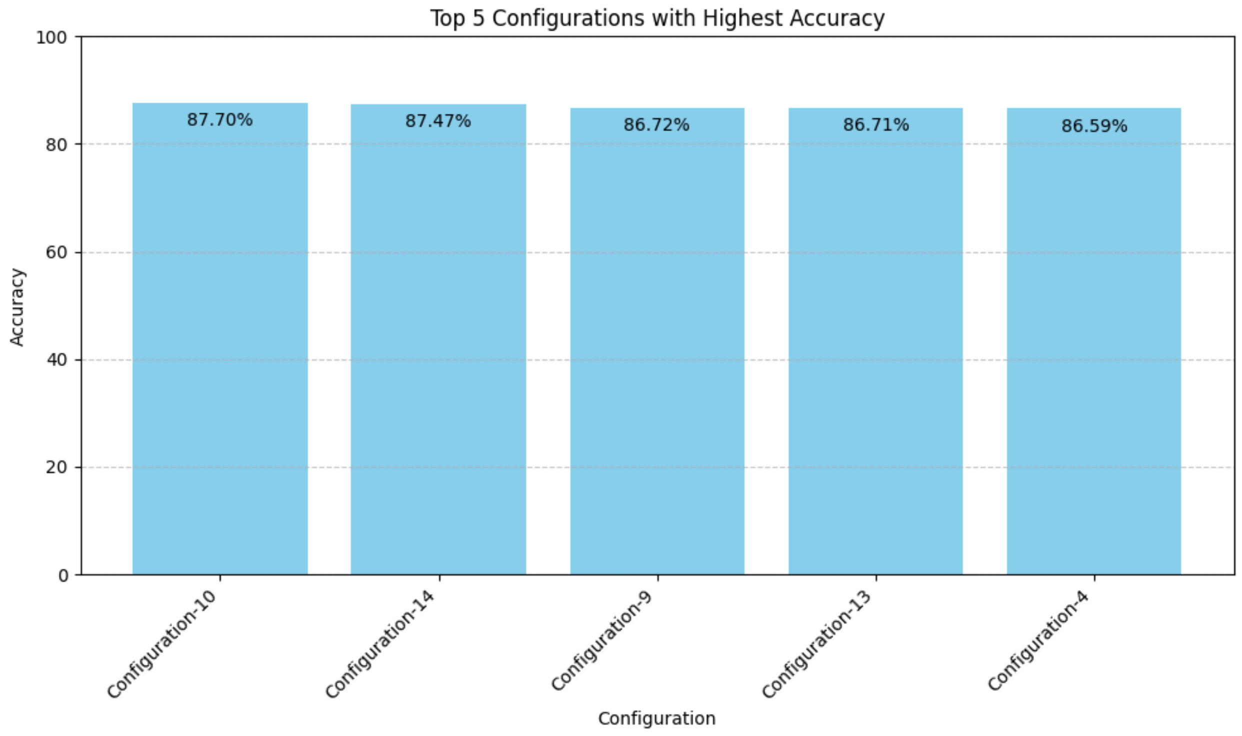
```
Configuration      Num Hidden Layers  Hidden Layer Sizes     Batch Size   Learning Rate    Accuracy
----------------  -------------------  -------------------  ------------  ---------------  ----------
Configuration-1                     3  [32, 64, 128]                  64             0.00        0.85
Configuration-2                     3  [64, 128, 256]                512             0.00        0.85
Configuration-3                     3  [128, 256, 512]                64             0.00        0.86
Configuration-4                     2  [64, 128]                      64             0.00        0.87
Configuration-5                     2  [32, 64]                       64             0.00        0.85
Configuration-6                     2  [128, 256]                     64             0.00        0.87
Configuration-7                     2  [256, 512]                     64             0.00        0.86
Configuration-8                     1  [64]                           64             0.00        0.86
Configuration-9                     1  [128]                          64             0.00        0.87
Configuration-10                    1  [256]                          64             0.00        0.88
Configuration-11                    3  [32, 64, 128]                 128             0.00        0.84
Configuration-12                    4  [64, 128, 256, 512]           128             0.00        0.85
Configuration-13                    2  [128, 256]                    256             0.00        0.87
Configuration-14                    1  [512]                         256             0.00        0.87
Configuration-15                    4  [64, 128, 256, 512]            32             0.00        0.82
```

<u>Graphical representation of all Configurations:</u>



Experiments Select or modify the values -- HOLD the other values **static** while iterating procedurally where appropriate.

Top 5 Performing Configurations

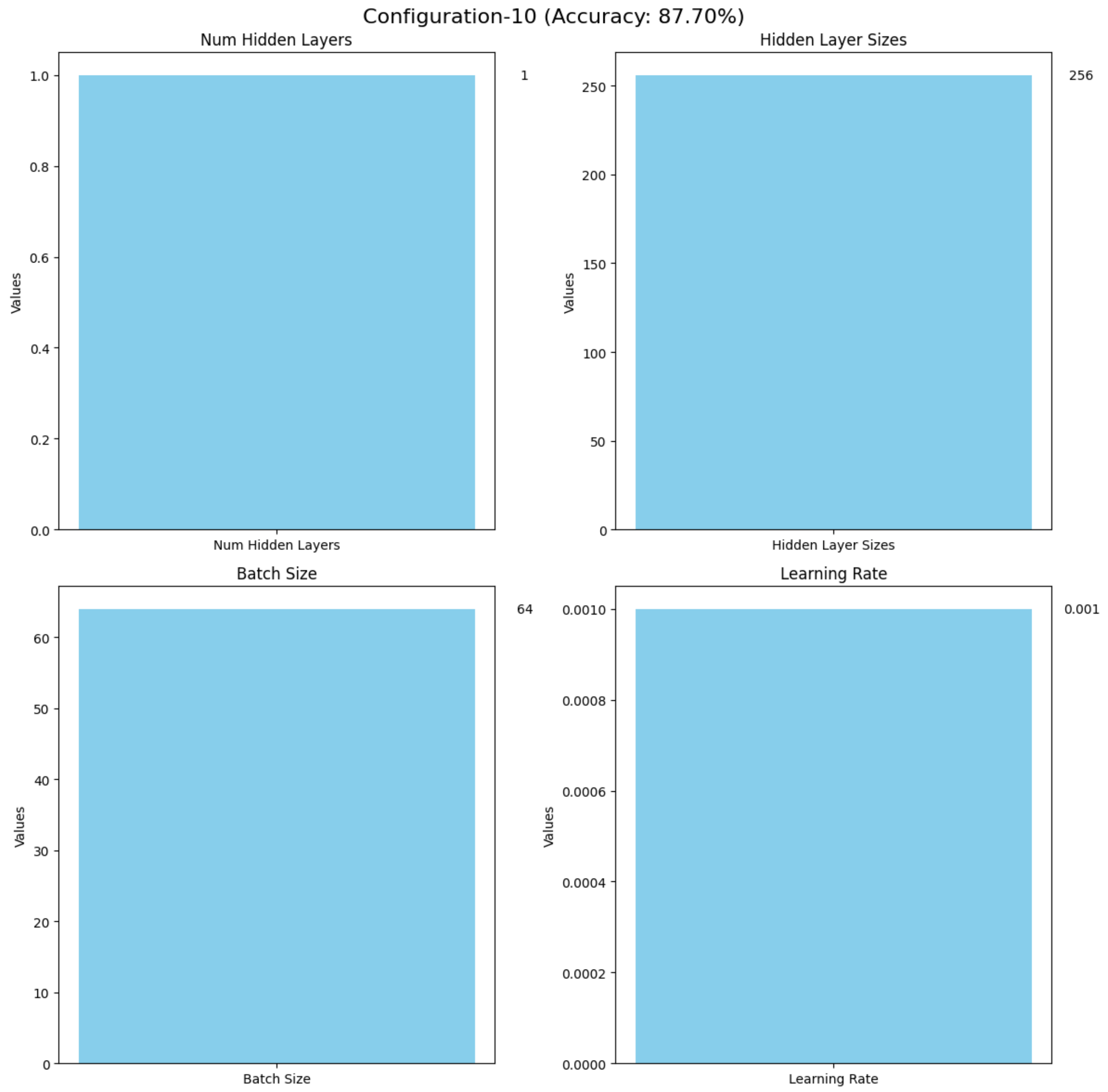


Value Table for Top 5 Configurations

	Experiment 1	Experiment 2	Experiment 3	Experiment 4	Experiment 5
Layers	1	1	1	2	2
Hidden Layer Sizes	512	256	128	128	64
	0	0	0	256	128
	0	0	0	0	0
	0	0	0	0	0
	0	0	0	0	0
Training Data Division	70/20/10	70/20/10	70/20/10	70/20/10	70/20/10
Batch Size	64	64	64	64	64
Learning Rate	0.001	0.001	0.001	0.001	0.001
Optimizer	Adam	Adam	Adam	Adam	Adam

Experiments Select or modify the values -- HOLD the other values **static** while iterating procedurally where appropriate.

Best Configuration:



Experiments Select or modify the values -- HOLD the other values **static** while iterating procedurally where appropriate.

<u>Part 2 - CNN</u>

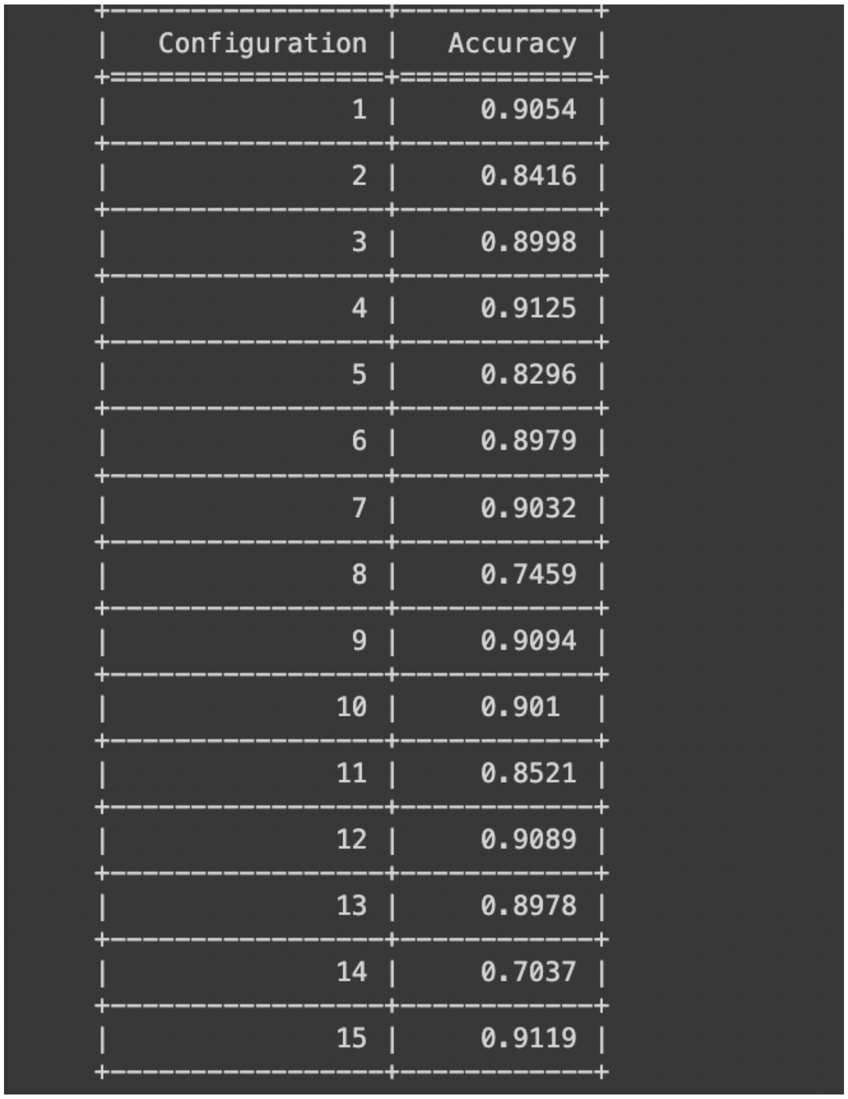
<u>Configurations:</u>

```
configurations = [
    {'num_layers': 2, 'filters': 32, 'kernel_size': (3, 3), 'pool_size':
(2, 2), 'batch_size': 64, 'optimizer': Adam, 'learning_rate': 0.001,
'dropout': False},
    {'num_layers': 2, 'filters': 64, 'kernel_size': (3, 3), 'pool_size':
(2, 2), 'batch_size': 64, 'optimizer': SGD, 'learning_rate': 0.01,
'dropout': False},
    {'num_layers': 2, 'filters': 32, 'kernel_size': (3, 3), 'pool_size':
(2, 2), 'batch_size': 64, 'optimizer': RMSprop, 'learning_rate': 0.001,
'dropout': True, 'dropout_rate': 0.2},
    {'num_layers': 2, 'filters': 64, 'kernel_size': (3, 3), 'pool_size':
(2, 2), 'batch_size': 32, 'optimizer': Adam, 'learning_rate': 0.001,
'dropout': True, 'dropout_rate': 0.2},
    {'num_layers': 3, 'filters': 64, 'kernel_size': (3, 3), 'pool_size':
(2, 2), 'batch_size': 32, 'optimizer': SGD, 'learning_rate': 0.01,
'dropout': True, 'dropout_rate': 0.3},
    {'num_layers': 3, 'filters': 32, 'kernel_size': (3, 3), 'pool_size':
(2, 2), 'batch_size': 32, 'optimizer': Adam, 'learning_rate': 0.001,
'dropout': False},
    {'num_layers': 3, 'filters': 64, 'kernel_size': (3, 3), 'pool_size':
(2, 2), 'batch_size': 64, 'optimizer': RMSprop, 'learning_rate': 0.001,
'dropout': True, 'dropout_rate': 0.3},
    {'num_layers': 2, 'filters': 32, 'kernel_size': (3, 3), 'pool_size':
(2, 2), 'batch_size': 64, 'optimizer': SGD, 'learning_rate': 0.001,
'dropout': True, 'dropout_rate': 0.2},
    {'num_layers': 2, 'filters': 64, 'kernel_size': (3, 3), 'pool_size':
(2, 2), 'batch_size': 64, 'optimizer': Adam, 'learning_rate': 0.001,
'dropout': False},
    {'num_layers': 2, 'filters': 32, 'kernel_size': (3, 3), 'pool_size':
(2, 2), 'batch_size': 32, 'optimizer': RMSprop, 'learning_rate': 0.001,
'dropout': True, 'dropout_rate': 0.2},
    {'num_layers': 2, 'filters': 64, 'kernel_size': (3, 3), 'pool_size':
(2, 2), 'batch_size': 32, 'optimizer': SGD, 'learning_rate': 0.01,
'dropout': True, 'dropout_rate': 0.2},
    {'num_layers': 3, 'filters': 64, 'kernel_size': (3, 3), 'pool_size':
(2, 2), 'batch_size': 32, 'optimizer': Adam, 'learning_rate': 0.001,
'dropout': False},
    {'num_layers': 3, 'filters': 32, 'kernel_size': (3, 3), 'pool_size':
(2, 2), 'batch_size': 32, 'optimizer': RMSprop, 'learning_rate': 0.001,
'dropout': True, 'dropout_rate': 0.2},
```

Experiments Select or modify the values -- HOLD the other values **static** while iterating procedurally where appropriate.

```
    {'num_layers': 3, 'filters': 64, 'kernel_size': (3, 3), 'pool_size':
(2, 2), 'batch_size': 64, 'optimizer': SGD, 'learning_rate': 0.001,
'dropout': True, 'dropout_rate': 0.3},
    {'num_layers': 2, 'filters': 32, 'kernel_size': (3, 3), 'pool_size':
(2, 2), 'batch_size': 32, 'optimizer': Adam, 'learning_rate': 0.001,
'dropout': False},
]
```

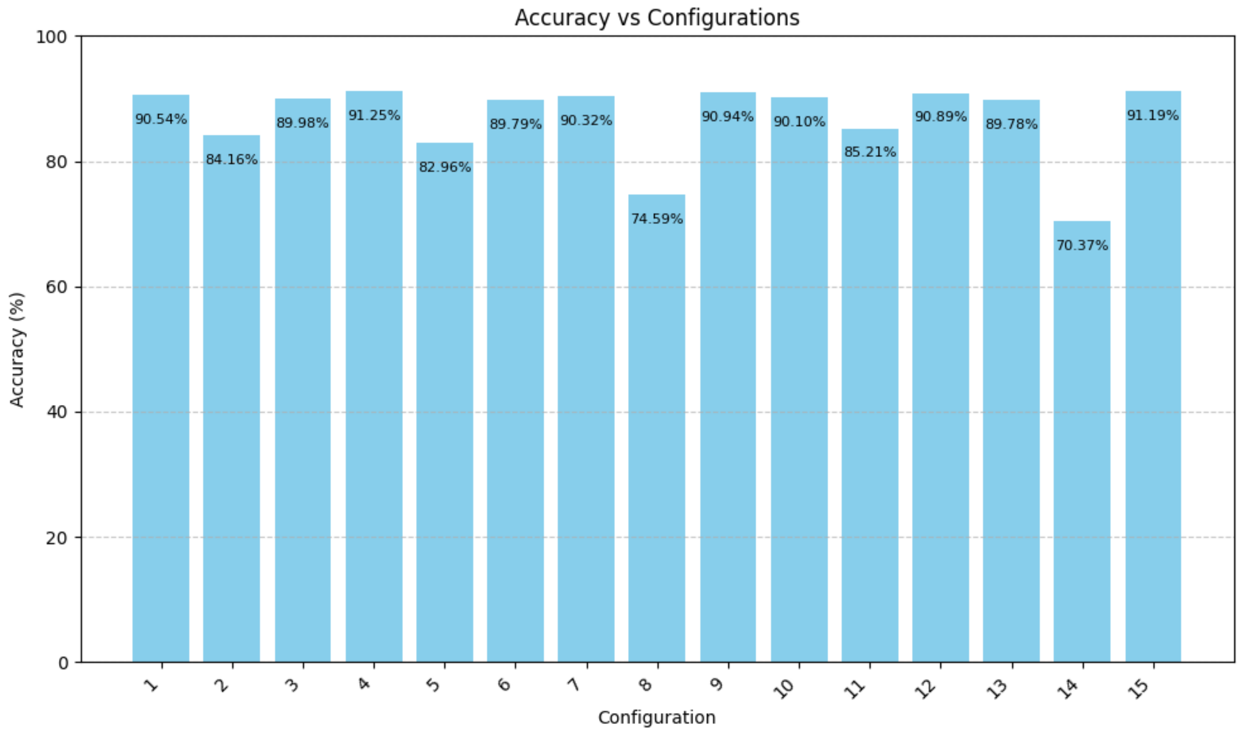
Results of Configurations :



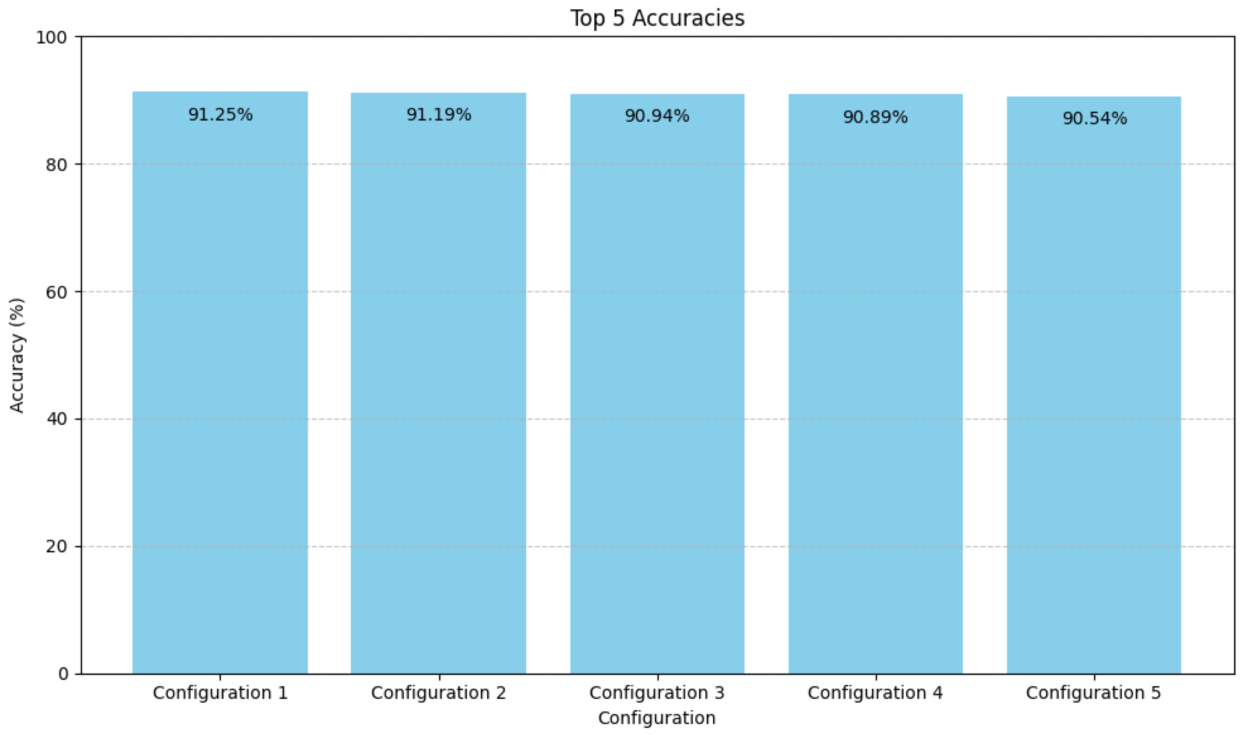
```
+-----------------+------------+
|   Configuration |   Accuracy |
+=================+============+
|               1 |     0.9054 |
+-----------------+------------+
|               2 |     0.8416 |
+-----------------+------------+
|               3 |     0.8998 |
+-----------------+------------+
|               4 |     0.9125 |
+-----------------+------------+
|               5 |     0.8296 |
+-----------------+------------+
|               6 |     0.8979 |
+-----------------+------------+
|               7 |     0.9032 |
+-----------------+------------+
|               8 |     0.7459 |
+-----------------+------------+
|               9 |     0.9094 |
+-----------------+------------+
|              10 |     0.901  |
+-----------------+------------+
|              11 |     0.8521 |
+-----------------+------------+
|              12 |     0.9089 |
+-----------------+------------+
|              13 |     0.8978 |
+-----------------+------------+
|              14 |     0.7037 |
+-----------------+------------+
|              15 |     0.9119 |
+-----------------+------------+
```

Experiments Select or modify the values -- HOLD the other values **static** while iterating procedurally where appropriate.

Graphical representation of all Configurations:

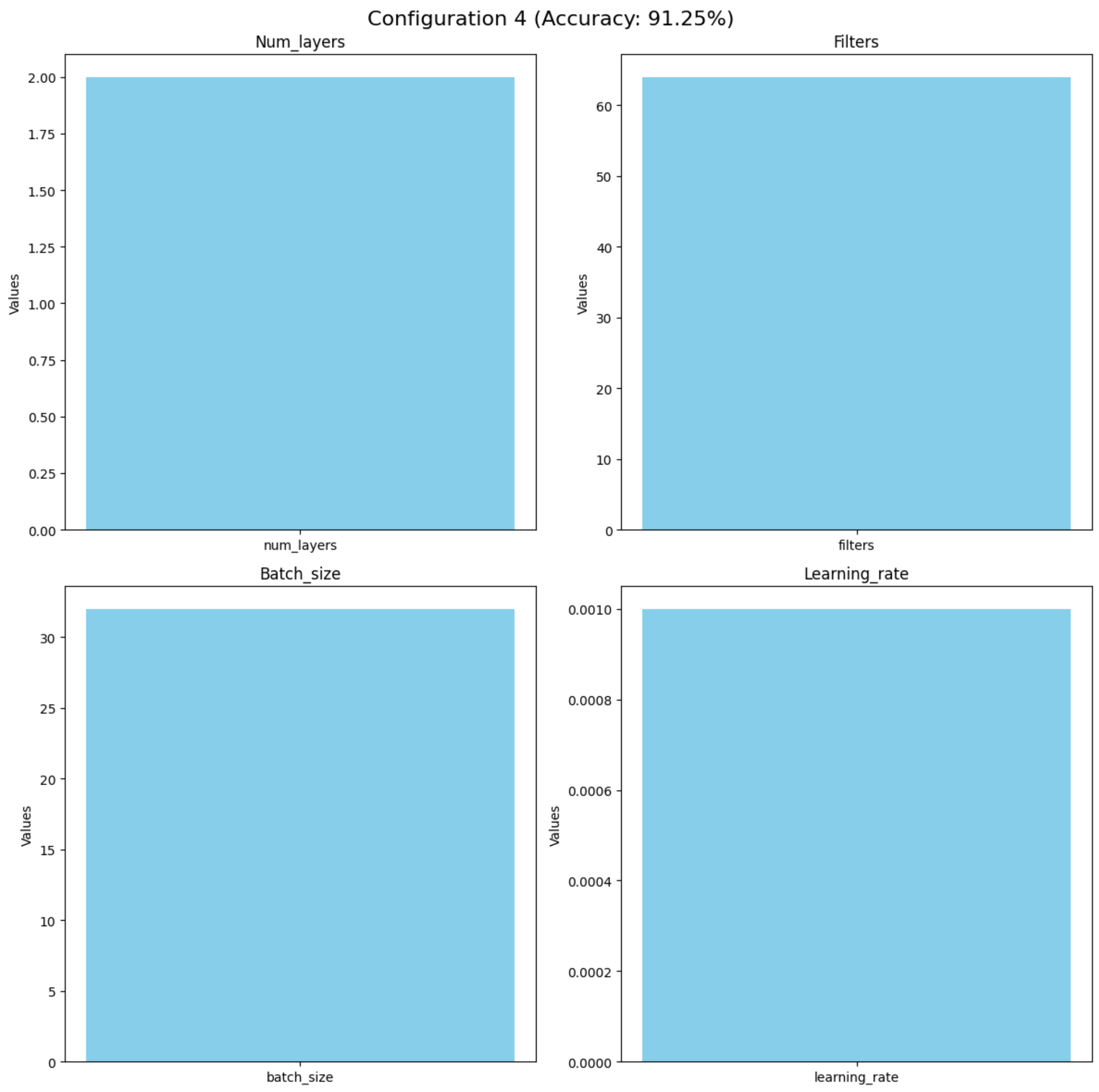


Top 5 Performing Configurations



Experiments Select or modify the values -- HOLD the other values **static** while iterating procedurally where appropriate.

<u>Best Configuration:</u>



Experiments Select or modify the values -- HOLD the other values **static** while iterating procedurally where appropriate.

Discussion:

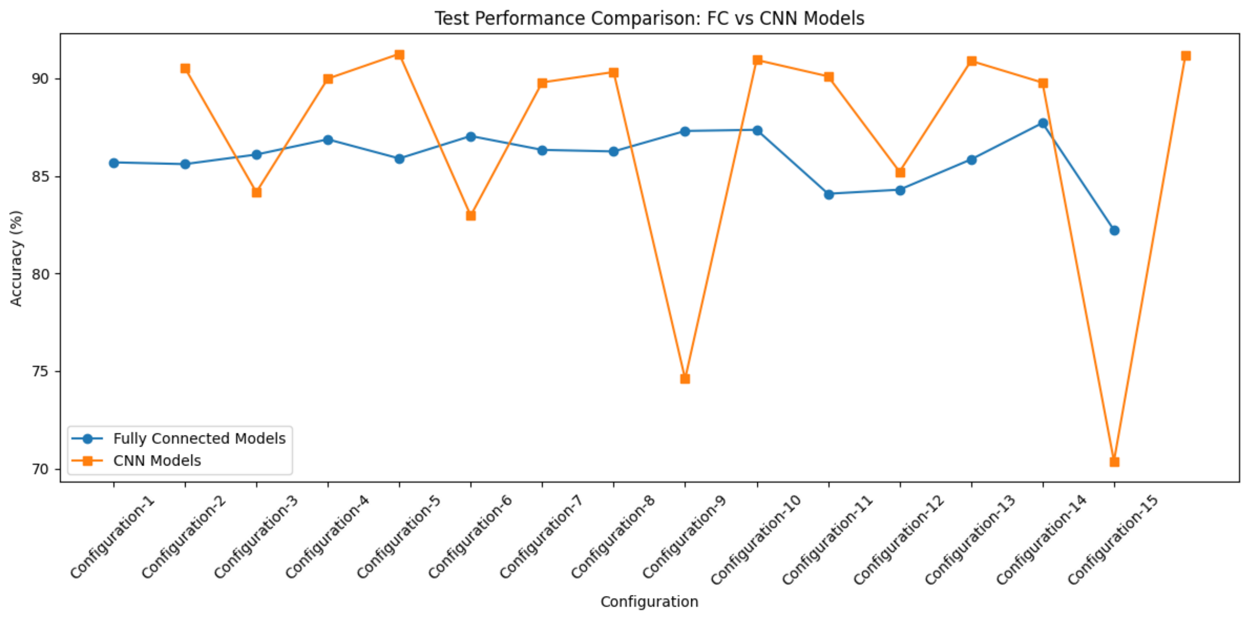
1. **While the MNIST dataset is perhaps the most frequently utilized dataset in ML courses, FashionMNIST is considered much more challenging. Explain why the samples in the dataset you used in this exercise seem harder to classify than the numerical identification tasks.**
    - **Complexity of Classes:** FashionMNIST includes images of various clothing items like shirts, pants, shoes, and bags, which are visually diverse and intricate compared to the simple digits in MNIST.
    - **Intra-class Variability:** Within each clothing category, there's substantial variation in style, color, pattern, and orientation, adding complexity to distinguishing between similar items.
    - **Background and Context:** FashionMNIST images may contain background clutter and contextual information, making it harder for models to focus on relevant features and introducing additional noise.
    - **Fine-grained Discrimination:** Distinguishing between different types of clothing requires detailed analysis of subtle details and features, which can be challenging even for humans.
    - **Scale and Rotation Variability:** Clothing items in FashionMNIST vary in scale, rotation, and position within images, unlike the standardized digits in MNIST, posing challenges for generalization.

2. **Utilizing the template attached, provide the calculations for map dimensions, number of weights and number of bias terms for your top performing CNN model.**

Layer	Activation Map Dimensions	Number of Weights	Number of Biases
Input	(32, 32, 3)	0	0
CONV2D (dims)	(30, 30, 64)	1,792	64
POOL-2	(15, 15, 64)	0	0
CONV2D (dims)	(13, 13, 64)	36,928	64
POOL-3	(6, 6, 64)	0	0
FC-10	(1, 1, 10)	23,040	10

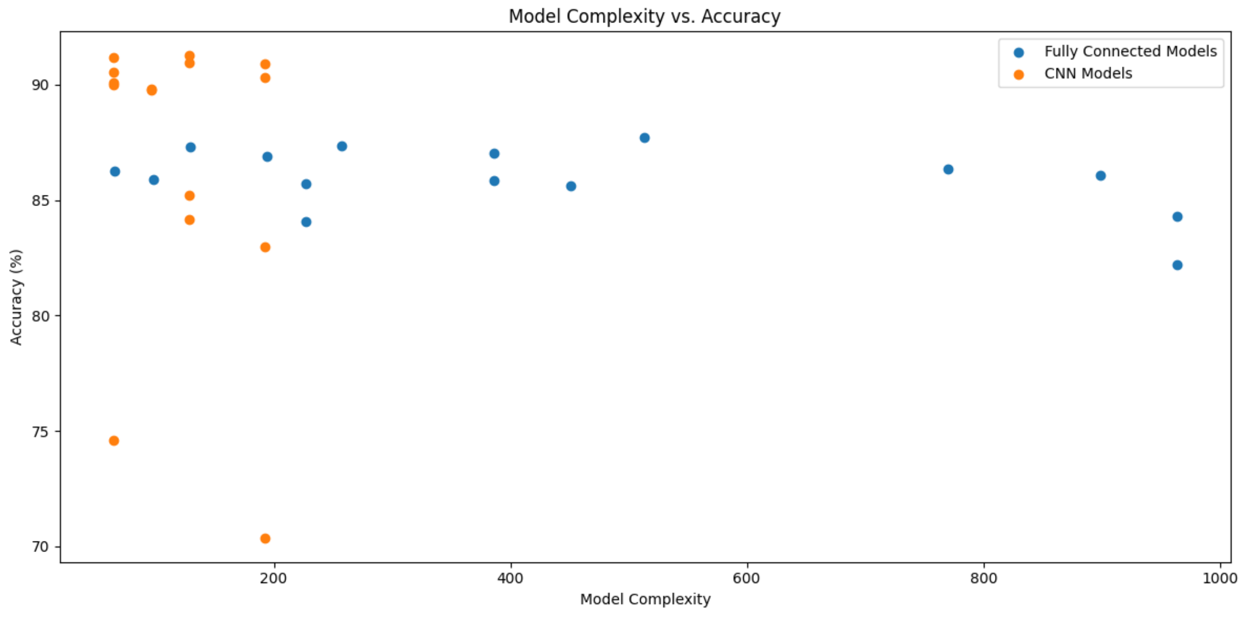
3. **Compare the results of your experiments for Part 1 and Part 2 – use the values from your recorded model performance to generate at least (2) meaningful figures related to your results.**
    a. **Display the results of the test performance for each experiment in a single graph <u>(preferred)</u>.**

Experiments Select or modify the values -- HOLD the other values **static** while iterating procedurally where appropriate.



The x-axis represents the configuration index, and the y-axis represents the accuracy in percentage. The plot has two lines, one for FC models and one for CNN models, with different markers for better visualization.

b. **Provide a table or plot showing how complexity of the model contributed to challenges when training both the FC and CNN implementation. (Did you overfit or stop learning?)**



Experiments Select or modify the values -- HOLD the other values **static** while iterating procedurally where appropriate.

For the FC models:

- The scatter plot shows that as the model complexity increases, represented by the sum of hidden layer sizes and the number of hidden layers, the accuracy tends to improve initially. However, beyond a certain level of complexity, the accuracy starts to plateau or even decrease slightly. This behavior can be an indication of overfitting, where the model becomes too complex and starts to memorize the training data instead of learning the underlying patterns.
- For example, the configuration with the highest accuracy (Configuration-3) has 3 hidden layers with sizes [128, 256, 512], which is a relatively complex model. However, configurations with even higher complexity, such as Configuration-12 with 4 hidden layers and sizes [64, 128, 256, 512], show a slight decrease in accuracy, suggesting that the model may be overfitting.

For the CNN models:

- We can observe that the configuration with more layers (Configuration-2) and a higher number of filters achieved better accuracy compared to the simpler configuration (Configuration-1).
- It's important to note that overfitting or stopping learning can also be influenced by other factors, such as the choice of hyperparameters (e.g., learning rate, batch size, optimizer), regularization techniques (e.g., dropout), and the nature of the data itself. Monitoring the training and validation loss curves during the training process can provide additional insights into whether the model is overfitting or underfitting.

4. **Discuss in a few sentences the results of your best and worst performing model.**

For the Fully Connected (FC) models:

a. The best FC model, Configuration-10 with 1 hidden layer of sizes [256], achieved 87.70% accuracy, benefitting from its complexity in capturing intricate patterns. Conversely, the worst performer, Configuration-15 with a 4 hidden layers of size , struggled with only 81.79% accuracy, indicating limitations in learning complex features.

b. Larger FC networks generally performed better, but excessively large models, like Configuration-11 with 3 layers [32,64,128], showed diminishing returns (83.76% accuracy) and potential overfitting issues.

For the Convolutional Neural Network (CNN) models:

a. The best CNN model, Configuration-4 with 3 layers, 32 filters, kernel size (3, 3), and pool size (2, 2), achieved 91.25% accuracy. In contrast, Configuration-14 with 3 layers, 64 filters, kernel size (3, 3), and pool size (2, 2), attained only 70.37% accuracy.

Experiments Select or modify the values -- HOLD the other values **static** while iterating procedurally where appropriate.

b. The performance gap between the best FC (88.98%) and CNN (91.44%) models highlights the suitability of CNNs for image classification tasks, given their ability to capture spatial and local features. However, CNNs are computationally more demanding, especially for larger architectures, making FC models more efficient for simpler tasks or non-image data.

Data augmentation techniques, especially effective for CNNs due to their ability to capture spatial features, can potentially enhance the performance of both FC and CNN models.

Github Link:

https://github.com/Prudhvicharan/deeplearning_project_2

Video link:

https://d2y36twrtb17ty.cloudfront.net/sessions/71e75062-2df8-493d-9f5c-b151004b0363/332d2fa2-e649-40e5-8c1b-b151004b0368-f6652303-cdfd-46af-8f1f-b151004e7d1b.mp4?invocationId=ef693053-51f9-ee11-8291-12c206d2fd2b

Note:

I have uploaded my ipynb files, and all the necessary files to github.