Machine Learning Final Report

Anushka Idamekorala(anb5km)

CS-6516

Table of Content

Evaluation on six machine learning algorithms

1.Briefing on the workflow

Algorithms

As per as the requirements of the machine learning project we have implemented seven different algorithms on two different data sets:

1. Logistic Regression.
2. Nearest Neighbor.
3. Decision Tree.
4. Support Vector Machine.
5. Random Forest.
6. Boosting.

Metrics

For evaluating the performance of our models, we used following metrices:

1. **Accuracy score:** Model accuracy is a performance metric for machine learning classification models that is defined as the ratio of true positives and true negatives to all positive and negative observations. In other words, accuracy tells us how often our machine learning model will correctly predict an outcome out of the total number of predictions it has made.

2. **Precision score:** The model precision score calculates the percentage of correctly predicted labels that are actually correct. Precision is also referred to as positive predictive value. Precision and recall are used alongside to trade off false positives and false negatives.

3. **Recall score:** The model recall score measures the model's ability to correctly predict positives from a set of positives. This differs from precision, which measures how many positive predictions made by models out of all positive predictions made.

4. **Roc AUC score:** It is the area under the ROC curve, which has False Positive Rate on the x-axis and True Positive Rate on the y-axis at all classification thresholds. The AUC - ROC curve is a performance metric for classification problems at various threshold levels. AUC represents the degree or measure of separability, while ROC is a probability curve. It indicates how well the model can distinguish between classes.

5. **F1 score:** Model F1 score is a function of precision and recall score that represents the model score. F-score is a machine learning model performance metric that gives equal weight to Precision and Recall when measuring accuracy, making it an alternative to Accuracy metrics as it does not require us to know the total number of observations.

We used the following to datasets in our project:

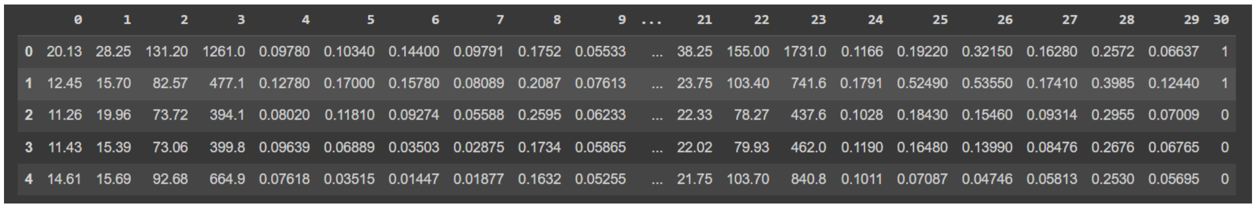
1. Size of Dataset one :569

   Feature count for Dataset one : 30

2. Size of Dataset two: 462

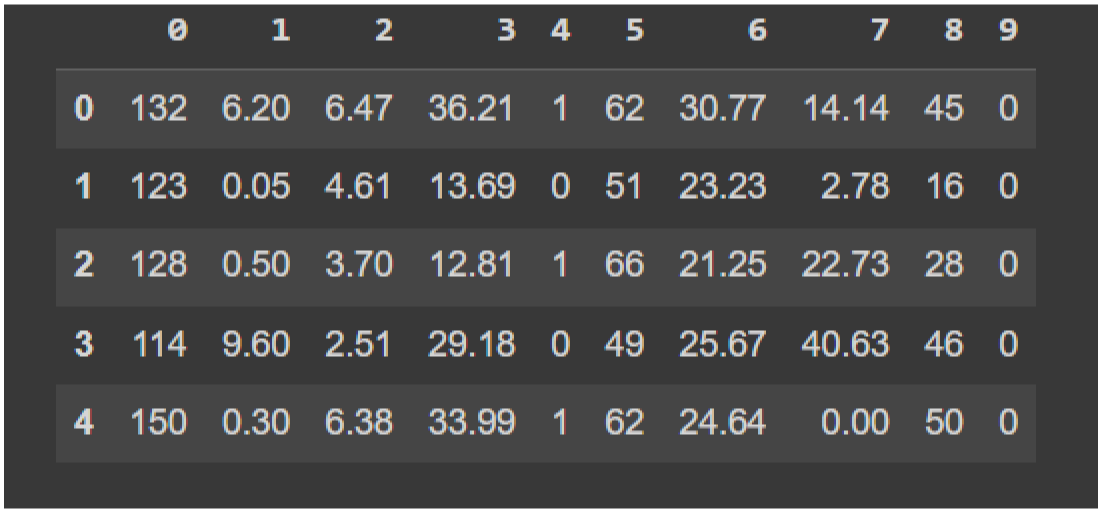
   Feature count of Dataset two: 9

Dataset 1:



	0	1	2	3	4	5	6	7	8	9	...	21	22	23	24	25	26	27	28	29	30
0	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1752	0.05533	...	38.25	155.00	1731.0	0.1166	0.19220	0.32150	0.16280	0.2572	0.06637	1
1	12.45	15.70	82.57	477.1	0.12780	0.17000	0.15780	0.08089	0.2087	0.07613	...	23.75	103.40	741.6	0.1791	0.52490	0.53550	0.17410	0.3985	0.12440	1
2	11.26	19.96	73.72	394.1	0.08020	0.11810	0.09274	0.05588	0.2595	0.06233	...	22.33	78.27	437.6	0.1028	0.18430	0.15460	0.09314	0.2955	0.07009	0
3	11.43	15.39	73.06	399.8	0.09639	0.06889	0.03503	0.02875	0.1734	0.05865	...	22.02	79.93	462.0	0.1190	0.16480	0.13990	0.08476	0.2676	0.06765	0
4	14.61	15.69	92.68	664.9	0.07618	0.03515	0.01447	0.01877	0.1632	0.05255	...	21.75	103.70	840.8	0.1011	0.07087	0.04746	0.05813	0.2530	0.05695	0

Dataset 2:

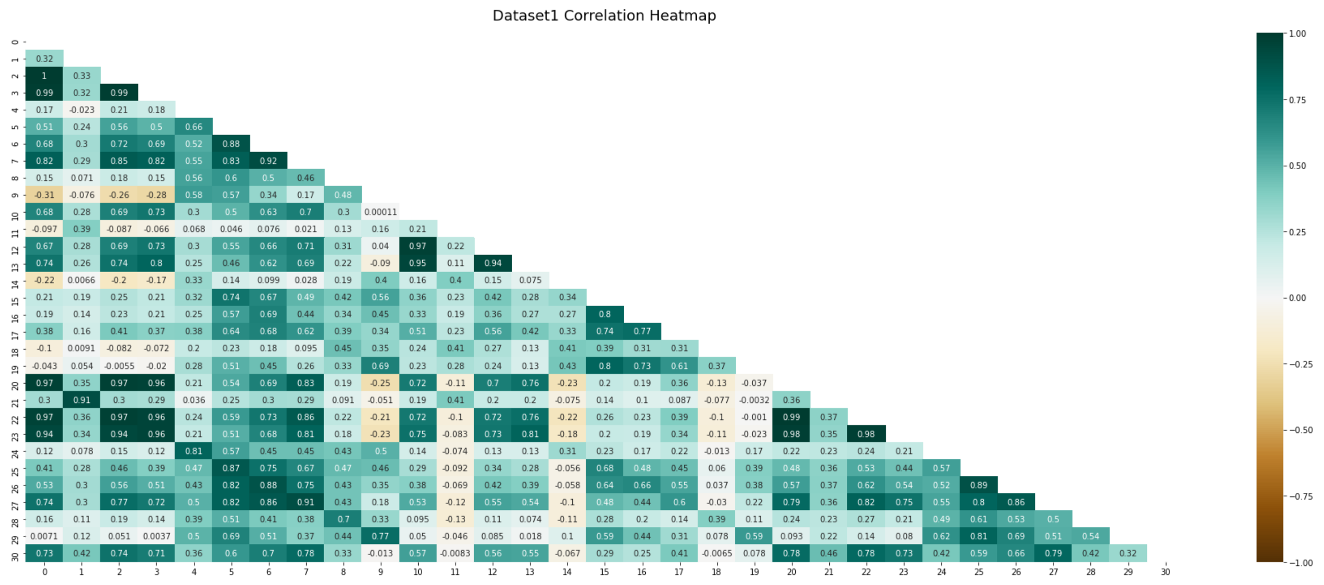


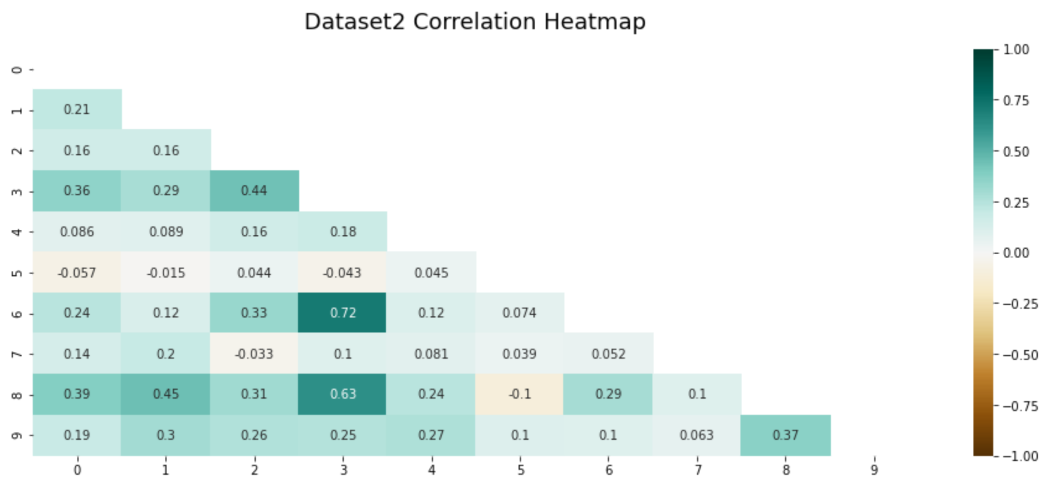
	0	1	2	3	4	5	6	7	8	9
0	132	6.20	6.47	36.21	1	62	30.77	14.14	45	0
1	123	0.05	4.61	13.69	0	51	23.23	2.78	16	0
2	128	0.50	3.70	12.81	1	66	21.25	22.73	28	0
3	114	9.60	2.51	29.18	0	49	25.67	40.63	46	0
4	150	0.30	6.38	33.99	1	62	24.64	0.00	50	0

Preprocessing Data

Normalization is a technique that is frequently used in data preparation for machine learning. Normalization is the process of changing the values of numeric columns in a dataset to use a common scale without distorting differences in ranges of values or losing information.

We applied scaling to a range normalization on both the datasets before training our models,we standardized the dataset in preprocessing by mapping features in to the range of 0 and 1, Also we mapped present and absent data in data set 2 to 0 and 1 to improve the performance and the training stability.
We have generated the heatmaps to evaluate the datasets. Seems dataset1 is highly correlated than dataset 2 where we can expect better prediction accuracies for dataset1.





Hyperparameter Tuning and Cross-validation

We implemented the hyperparameter tuning and the cross validation using GridSearchCV. It is a technique for finding the best parameter values in a grid given a set of parameters. It is, in essence, a cross-validation technique. Both the model and the parameters must be entered. Predictions are made after extracting the best parameter values. GridSearchCV evaluates the model for each combination of the values passed in the dictionary using the Cross-Validation method. As a result of using this function, we can calculate the accuracy/loss for each combination of hyperparameters and select the one with the best performance.

Now let's explore each model in detail:

2. Logistic Regression Classifier

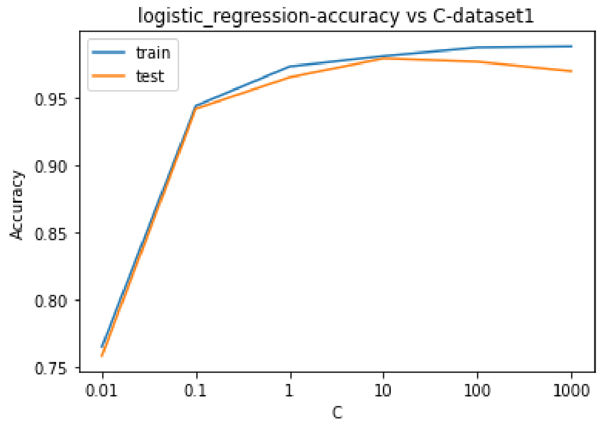
Logistic regression is a statistical analysis method that uses prior observations of a data set to predict a binary outcome, such as yes or no. Supervised learning is demonstrated by logistic regression. A logistic regression model predicts a dependent variable by examining the relationship between one or more existing independent variables. Logistic regression is also referred to as logit regression, binary logit regression, or binary logistic regression. It is widely used in a variety of fields, particularly medical and social science research.
We trained our Logistic Regression model on two different datasets, we used GridSearchCV for the hyperparameter tuning and 10 - fold cross validation for getting the ideal parameters yielding the best performance.

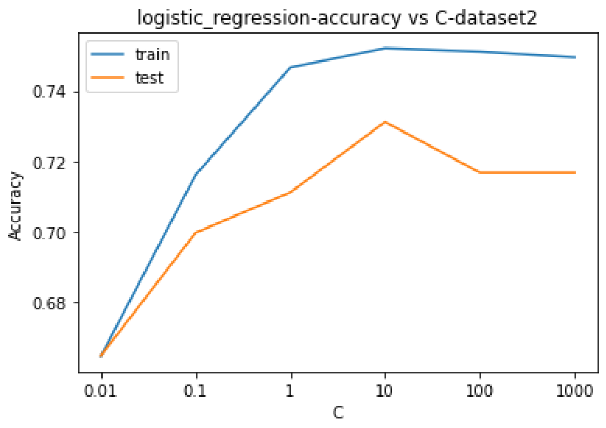
Hyperparameters:

C: Is the **inverse of regularization strength**. Smaller values specify stronger regularization.

C values used for Cross validation: [0.01, 0.1, 1, 10, 100, 1000]

Comparison of C for dataset 1 and dataset 2:



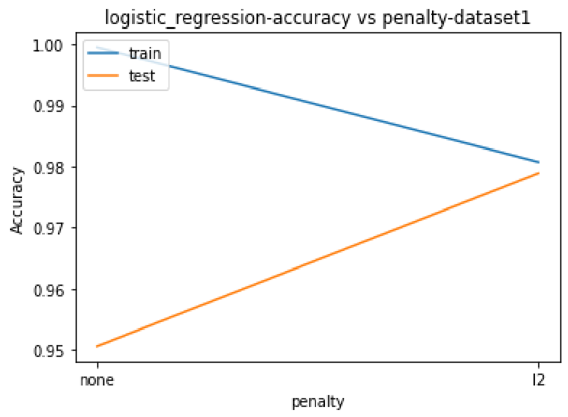


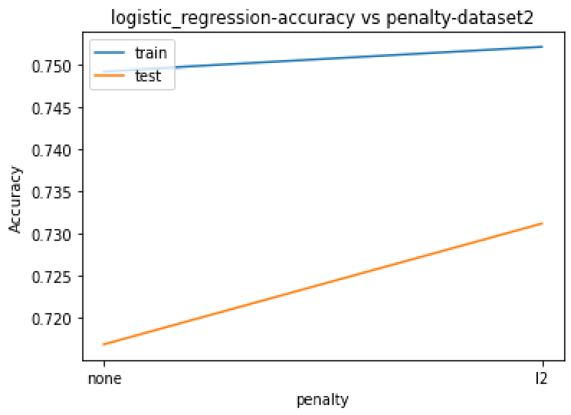
Best parameter for Dataset 1: C=10
Best parameter for Dataset 2: C=10

penalty: Penalized logistic regression imposes a penalty to the logistic model for having too many variables. This results in shrinking the coefficients of the less contributive variables toward zero.

penalty values used for Cross validation: [none, l2]

Comparison of penalty for dataset 1 and dataset 2:





Best parameter for Dataset 1: penalty=l2
Best parameter for Dataset 2: penalty=l2

Performance Metrics

Performance Metrics	**Dataset 1**	**Dataset 2**
Validation Accuracy	0.9788	0.7311
Test Accuracy	0.9580	0.7241
Precision score	0.9500	0.7065
Recall score	0.9583	0.6982
Roc AUC score	0.9961	0.7891
F1 score	0.9400	0.6190

Best parameter for Dataset 1: [c= 10, penalty=l2]
Best parameter for Dataset 2: [c= 10, penalty=l2]

Trade-off Analysis

Hyperparameter C: C decides the regularization factor, regularization introduces simplicity to the model reducing the variance and increasing bias. When C increases regularization decreases therefore bias decreases and variance increases. Therefore as seen in the charts when C is very low both training and test accuracies are low due to underfitting and when C is very high though the train accuracy is high test accuracy is low due to overfitting. This is quite obvious in the above graphs.

Hyperparameter penalty: penalty decides whether to add a regularization or not. Through the above graphs we can see that through the penalty we can reduce the overfitting. That is because penalties add a bias while reducing the variance of the model.

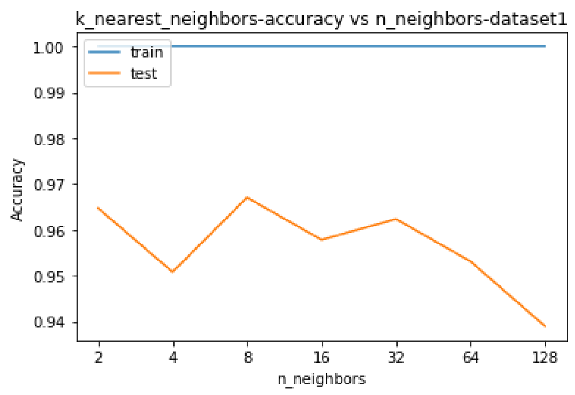
3. K-nearest Neighbors Classifier

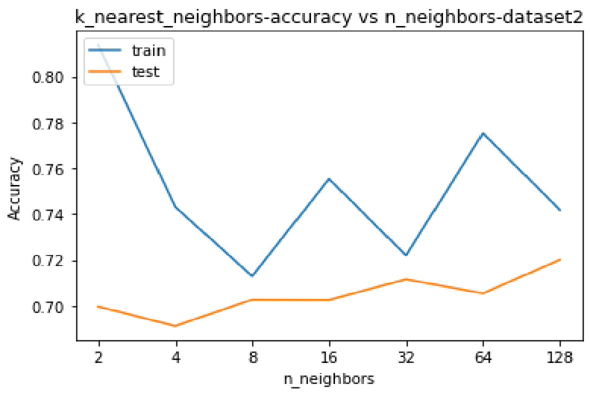
K Nearest Neighbour is a simple algorithm that stores all available cases and classifies new data or cases using a similarity measure. It is typically used to classify a data point based on the classification of its neighbors. KNN algorithms select a number k as the nearest Neighbor to the data point to be classified. If k is 5, it will look for the 5 nearest Neighbors to that data point. Example: Assume we have an image of a creature that resembles a cat or a dog and we want to know whether it is a cat or a dog. The k-nearest neighbors algorithm is known as "lazy" because it does no training when given training data. At training time, all it does is store the entire data set without performing any calculations. The K-nearest neighbors (KNN) algorithm is a type of supervised machine learning (ML) algorithm that can be used for both classification and regression predictive problems. However, it is primarily used in industry for classification and prediction problems. The main benefits of the K nearest neighbor algorithm include the elimination of the need to make assumptions about data - or to make additional assumptions, tune several parameters, or build a model. This is especially important in the case of nonlinear data. On the contrary its main disadvantages are that it is quite computationally inefficient and its difficult to pick the “correct” value of K.
We trained our KNN model on two different datasets, we used GridSearchCV for the hyperparameter tuning and 10 - fold cross validation for getting the ideal parameters yielding the best performance.

Hyperparameter

n-neighbors: Number of neighbors used in the algorithm.
N_neighbour values used for cross validation = [2,4,8,16,32,64,128]

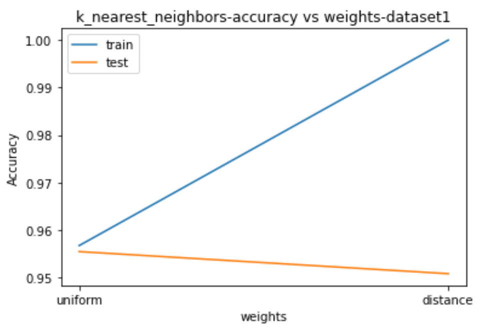
Comparison of n-neighbors for dataset 1 and dataset 2:

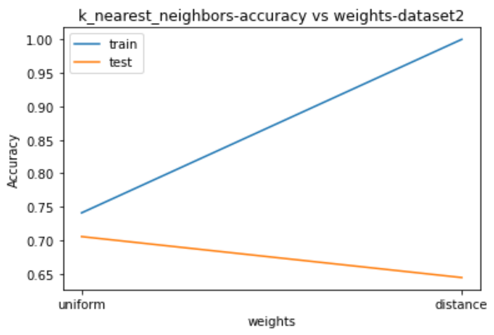




Best Parameters for dataset 1: N_neighbour = 4
Best Parameters for dataset 2: N_neighbour = 32

weights: It signifies how weight should be distributed between neighbor values.
weights values that are cross validated = [uniform,distance]
Comparison of Weights for dataset 1 and 2:



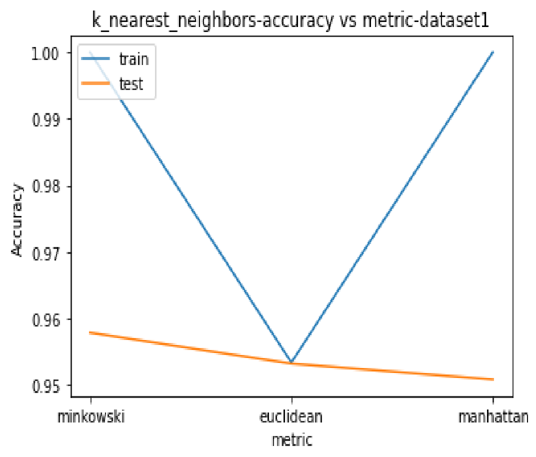


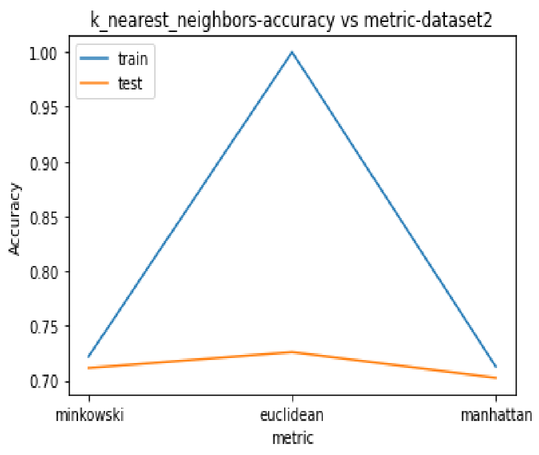
Best Parameters for dataset 1: weights = distance
Best Parameters for dataset 2: weights = uniform

metrics: It is a measure of the true straight line distance between two points in Euclidean space.

metics values used for cross validation = [minkowski ,euclidean , manhattan]

Comparison of metrics for dataset 1 and dataset 2:





Best parameter for dataset 1: Metric = manhattan
Best parameter for dataset 2: Metric = minkowski

Performance Metrics:

Performance Metrics	**Dataset 1**	**Dataset 2**
Validation Accuracy	0.9741	0.7347
Test Accuracy	0.9510	0.7155
Precision score	0.9438	0.7452
Recall score	0.9481	0.6426
Roc AUC score	1.0000	0.7824
F1 score	0.9292	0.4761

Best Parameters for dataset 1: [n_neighbour = 4, weights = distance, metric = manhattan]
Best Parameters for dataset 2: [n_neighbour= 32, weights= uniform, metric= minkowski]

Trade-off Analysis

Hyperparameter n-neighbors: When the k (number of neighbors) increases model complexity increases causing increase in variance and decrease in bias. When the number of neighbors is too low, the model is underfitting and when that is too high, the model is overfitting. This can be obviously seen in the above graphs.

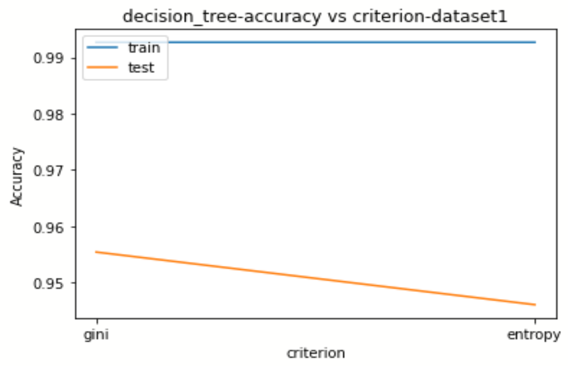
4. Decision Tree Classifier

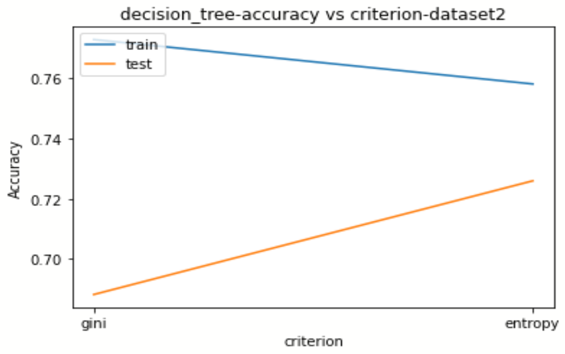
A decision tree is a type of supervised machine learning that is used to categorize or predict based on the answers to a previous set of questions. The model is supervised learning in the sense that it is trained and tested on data that contains the desired categorization. It is a graphical representation of all possible decisions based on specific conditions. We try to form a condition on the features at each step or node of a decision tree used for classification to separate all the labels or classes contained in the dataset to the greatest extent possible. The best example is when we buy something from an online shopping portal and receive several recommendations based on what we are looking for.Decision trees are commonly used in operations research, specifically in decision analysis, to aid in the identification of the most likely strategy to achieve a goal. The Decision Tree algorithm is used to solve classification and regression problems. The main disadvantage of Decision Tree is that it generally leads to data overfitting.
We trained our Decision Tree Classifier model on two different datasets, we used GridSearchCV for the hyperparameter tuning and 10 - fold cross validation for getting the ideal parameters yielding the best performance.

Hyperparameters

criterion : Function to measure the quality of the split. Supported criterias are gini impurity and information gain
Criterion values used for cross validation : [gini,entropy]

Comparison of criterion for dataset 1 and dataset 2:



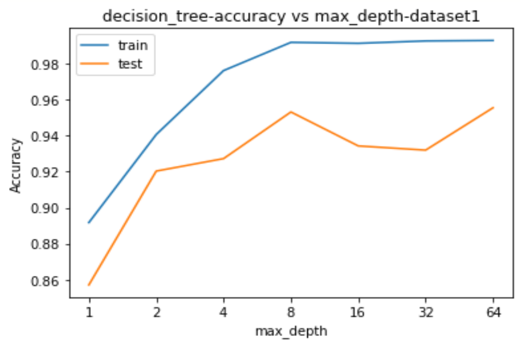


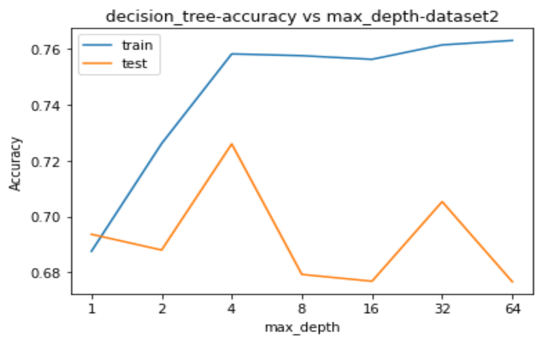
Best Parameters for Dataset 1: criterion : gini
Best Parameters for Dataset 2: criterion : entropy

max-depth : The maximum depth of the tree. If not given then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples(= 2).

Max-depth values used for cross validation :[1,2,4,8,16,32,64]

Comparison of Max-depth for dataset 1 and dataset 2:





Best Parameters for Dataset 1: max_depth : 64
Best Parameters for Dataset 2: max_depth : 4

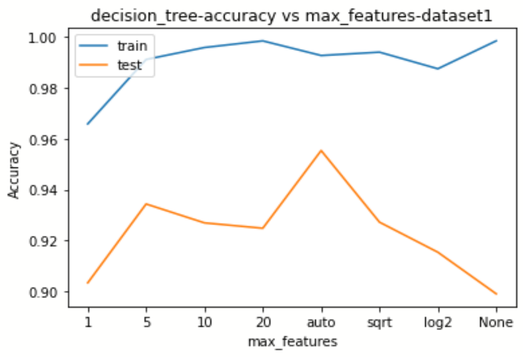
max_features: The number of features to consider when looking for the best split:

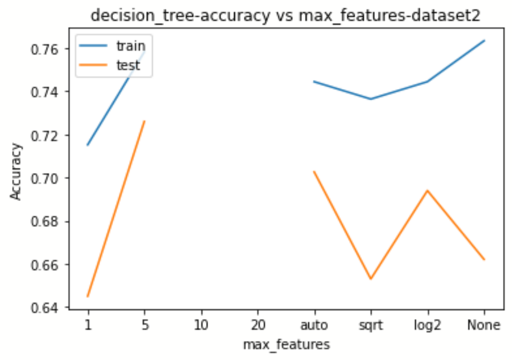
- If "auto", then `max_features=sqrt(n_features)`.
- If "sqrt", then `max_features=sqrt(n_features)`.
- If "log2", then `max_features=log2(n_features)`.
- If None, then `max_features=n_features`.

Note: the search for a split does not stop until at least one valid partition of the node samples is found, even if it requires to effectively inspect more than ``max_features`` features.

max_features values used for cross validation:[1,5,10,20,"auto","sqrt","log2",None]

Comparison of max_features for dataset 1 and dataset 2:





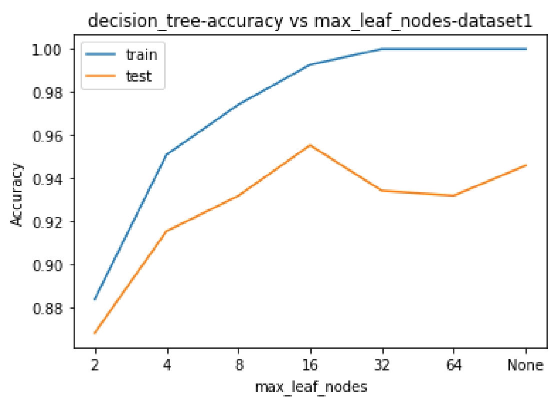
Best Parameters for Dataset 1: max_features : auto

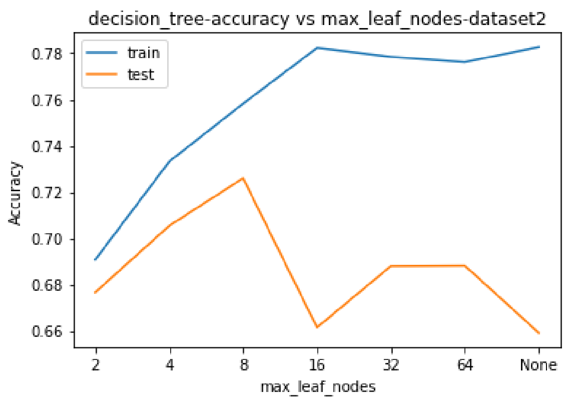
Best Parameters for Dataset 2: max_features : 5

max_leaf_nodes: Grow a tree with ``max_leaf_nodes`` in best-first fashion. Best nodes are defined as relative reduction in impurity. If None then unlimited number of leaf nodes.

max_leaf_nodes values used for cross validation: [1,2,4,8,16,32,64,None]

Comparison of max_leaf_nodes for dataset 1 and dataset 2:





Best Parameters for Dataset 1: max_leaf_nodes: 16

Best Parameters for Dataset 2: max_leaf_nodes: 8

Performance Metrics:

Performance Metrics	**Dataset 1**	**Dataset 2**
Validation Accuracy	0.9553	0.7259
Test Accuracy	0.9510	0.6379
Precision score	0.9409	0.6137
Recall score	0.9529	0.5448
Roc AUC score	0.9896	0.7984
F1 score	0.9306	0.2500

Best Parameters for Dataset 1:
[criterion : gini ,max_depth : 64, max_features : auto, max_leaf_nodes: 16]

Best Parameters for Dataset 2:
[criterion : entropy ,max_depth : 4, max_features : 5, max_leaf_nodes: 8]

Trade-off Analysis

Hyperparameter max-depth : Decision-trees are models with low-bias and high variance. When the **max-depth** is very low, the model is less in complexity, therefore has a high bias and a low variance. Hence the model is under-fitting when the decision-tree is too shallow. But when the max-depth of the decision tree is increased, model complexity increases while increasing the variance and decreasing the bias. Due to that if the decision tree is too deep (max-depth is very high) the model becomes over-fitting. This fluctuation is transparent in the above max-depth vs accuracy graph.

Hyperparameter max-leaf-nodes : The max-leaf-nodes are also responsible for increasing the model complexity. So an increase of that parameter increases the variance and reduces the bias, where too much can cause overfitting. Same as that, reducing this too much can cause underfitting. Therefore parameters should be chosen appropriately. In the above max-leaf-nodes graphs, this phenomena can be seen obviously.

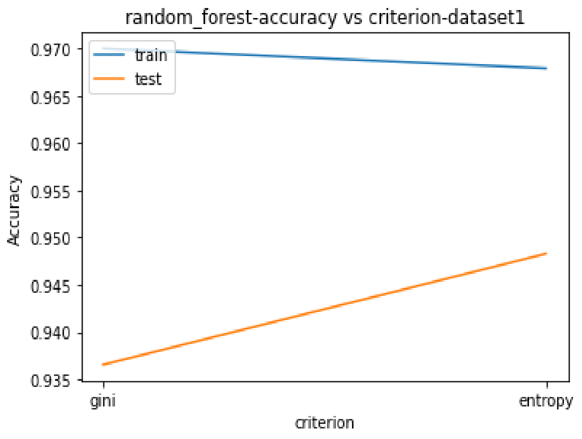
5. Random Forest Classifier

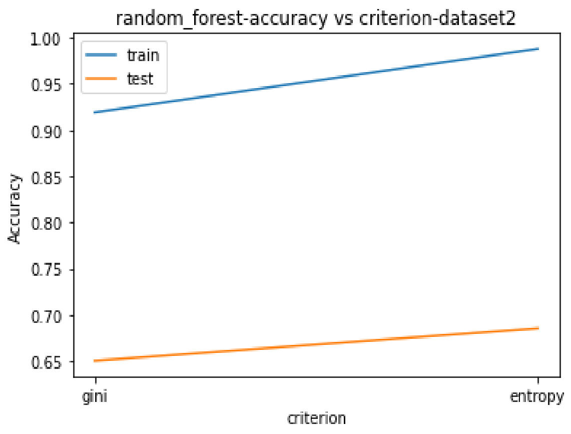
Random Forest is a supervised machine learning algorithm that grows and combines multiple decision trees to form a "forest." It is applicable to both classification and regression problems. When building each individual tree, it employs bagging and feature randomness in an attempt to create an uncorrelated forest of trees whose prediction by committee is more accurate than that of any individual tree. When splitting a node, it looks for the best feature among a random subset of features rather than the most important feature. As a result, there is a greater variety, which leads to a better model. Random forests, like other machine-learning techniques, use training data to learn to make predictions. Overfitting is one of the drawbacks of learning with a single tree. Single trees have a tendency to overlearn the training data, resulting in poor prediction performance on unseen data.
We trained our Random forest classifier model on two different datasets, we used GridSearchCV for the hyperparameter tuning and 10 - fold cross validation for getting the ideal parameters yielding the best performance.

Hyperparameters

criterion : Function to measure the quality of the split. Supported criterias are gini impurity and information gain.

Criterion values used for cross validation:[gini,entropy]
Comparison of Criterion for dataset 1 and dataset 2:

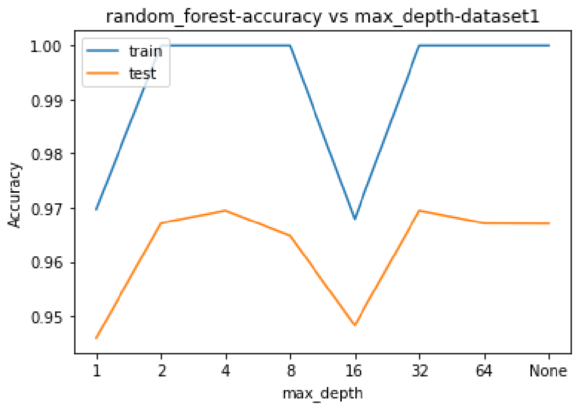


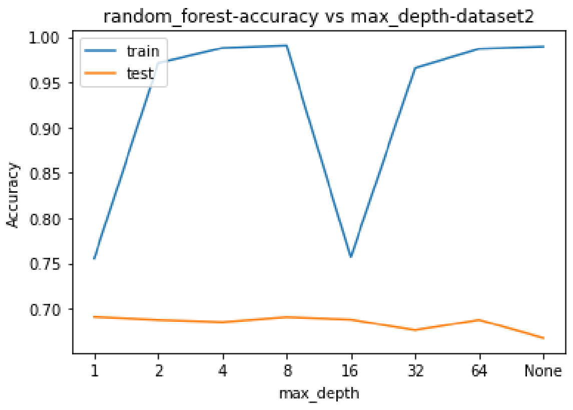


Best Parameters for Dataset 1: criterion = entropy
Best Parameters for Dataset 2: criterion = entropy

max-depth : The maximum depth of the tree. If not given then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples(= 2).

max_depth values used for cross validation: [2, 4, 8, 16,32,64,128,None]
Comparison of max_depth for dataset 1 and dataset 2:



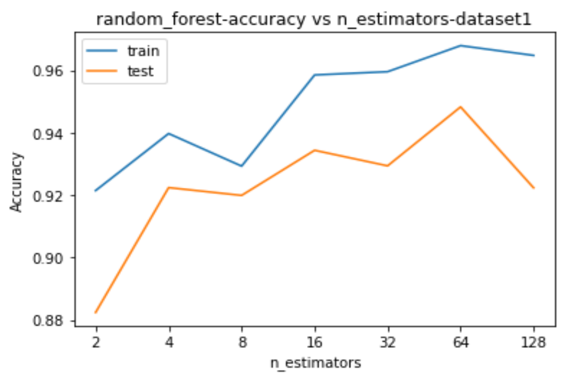


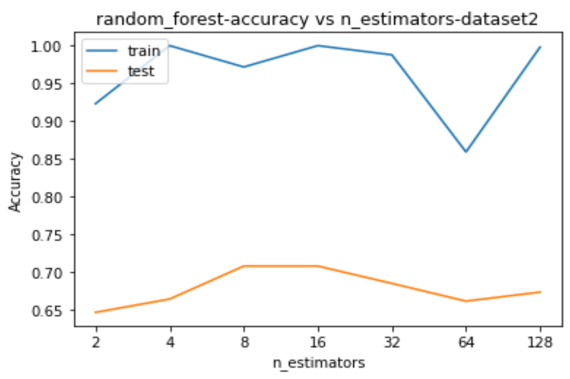
Best Parameters for Dataset 1: max_depth = 16
Best Parameters for Dataset 2: max_depth = 4

n-estimators: The number of trees in the forest

n-estimators values used for cross validation:[2,4,8,16,32, 64, 128]

Comparison of n-estimators for dataset 1 and dataset 2:

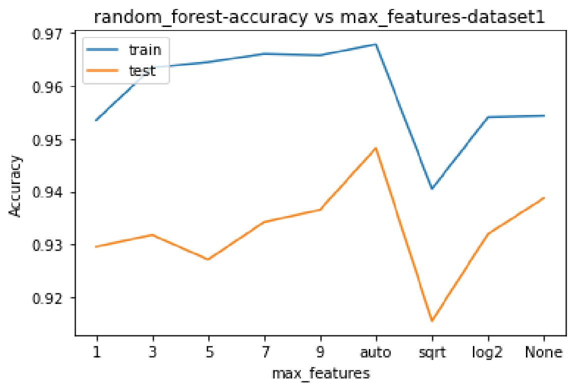


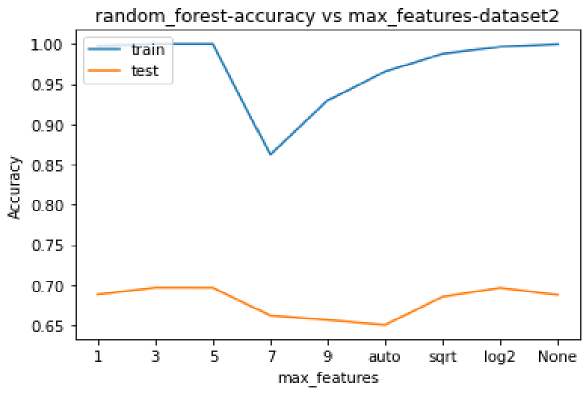


Best Parameters for Dataset 1: n-estimators = 64
Best Parameters for Dataset 2: n-estimators = 32

max-features:
max-features values used for cross validation:[1,3,5,7,9,"auto","sqrt","log2",None]

Comparison of max-features for dataset 1 and dataset 2:





Best Parameters for Dataset 1: max_features : auto
Best Parameters for Dataset 2: max_features : sqrt

Performance Metrics:

Performance Metrics	**Dataset 1**	**Dataset 2**
Validation Accuracy	0.9718	0.7228
Test Accuracy	0.9510	0.7068
Precision score	0.9476	0.6937

Recall score	0.9432	0.6578
Roc AUC score	1.0000	0.9154
F1 score	0.9278	0.5405

Best Parameters for Dataset 1:
[criterion: entropy, max_depth: 16, max_features: auto, n_estimators: 64]

Best Parameters for Dataset 2:
[criterion: entropy, max_depth: 4, max_features: sqrt, n_estimators: 32]

Trade-off Analysis

Hyperparameter max-depth: As same as in the decision tree, When the **max-depth** is very low, the learners less in complexity, therefore have a high bias and a low variance. Hence the model is under-fitting when the trees in the forest are too shallow. But when the max-depth of the trees is increased, model complexity increases while increasing the variance and decreasing the bias. Due to that if the trees in the random-forest are too deep (max-depth is very high) the model becomes over-fitting. This can be seen in the above max-depth graphs.

Hyperparameter number of estimators: Considering the **number of estimators**. Random forests aggregate the decisions of multiple less-correlated trees so the model has less variance. So when the number of estimators are low, the model has a higher bias and less variance so prone for over-fitting. When that number increases , bias is also increased and variance is reduced. Where too much can cause under-fitting.

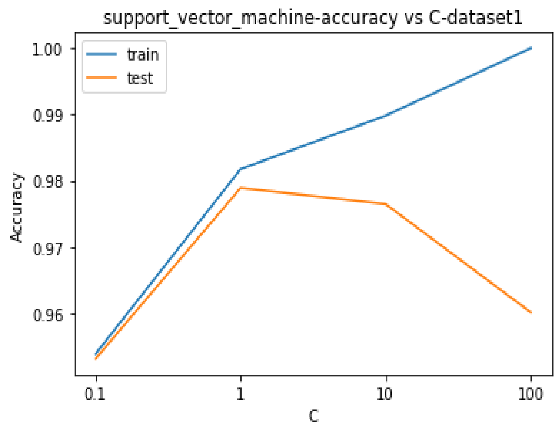
6. Support Vector Machine

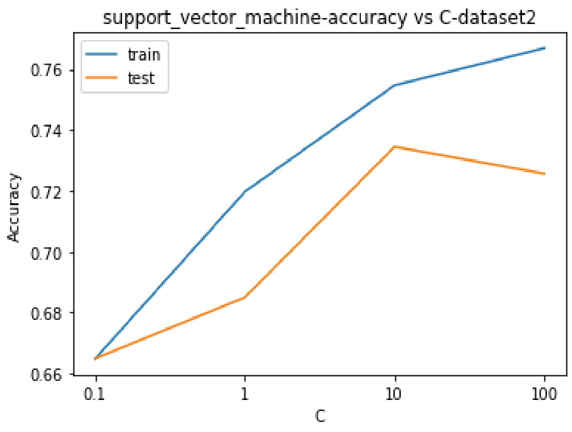
A support vector machine (SVM) is a deep learning algorithm that uses supervised learning to classify or predict data groups. The following are the benefits of support vector machines: Effective in high-dimensional environments. When the number of dimensions exceeds the number of samples, the method remains effective. The algorithm is called SVM because the separating hyperplane is supported (defined) by the vectors (data points) closest to the margin. SVM has been shown to perform well in a wide range of real-world learning problems and is widely regarded as one of the best "out-of-the-box" classifiers.

Hyperparameters

C : Regularization parameter. The strength of the regularization is inversely proportional to C.

C values used for cross-validation: C: [0.1, 1, 10, 100]
Comparison of C for dataset 1 and 2:

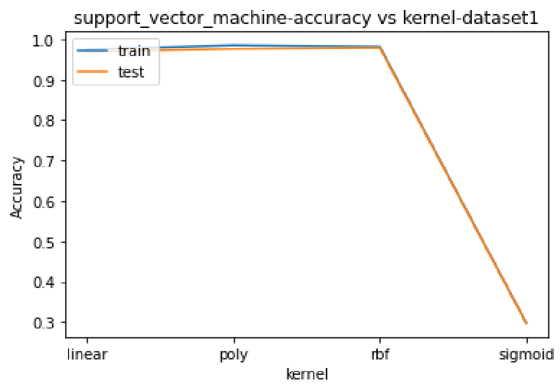


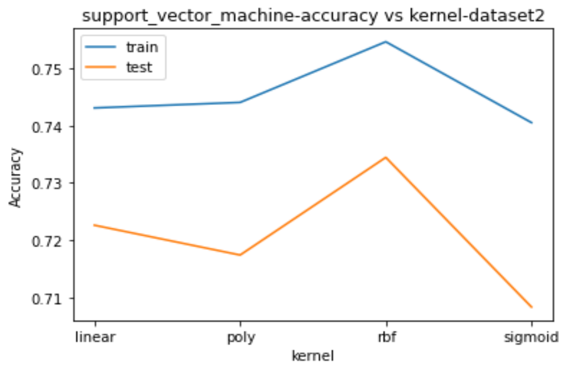


Best Parameters for dataset 1: 1
Best Parameters for dataset 2: 10

Kernel : Specifies the kernel type to be used in the algorithm.

Kernel values used for cross-validation: kernel : [linear, poly, rbf, sigmoid]
Comparison of kernel for dataset 1 and 2:

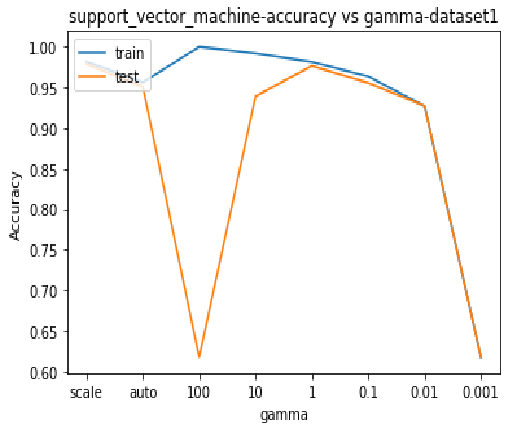


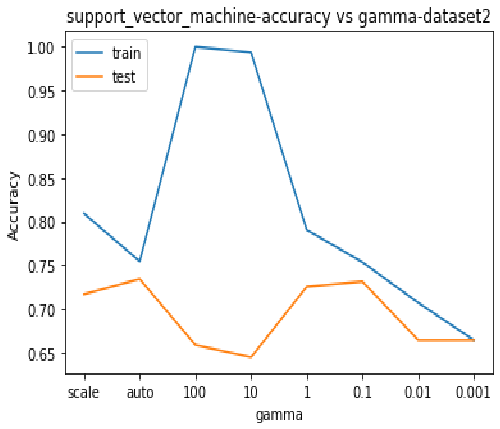


Best Parameters for dataset 1: rbf
Best Parameters for dataset 2: rbf

gamma : Kernel coefficient for non-linear kernels

gamma values used for cross-validation: gamma : [scale, auto,100,10,1, 0.1, 0.01, 0.001]
Comparison of gamma for dataset 1 and 2:





Best Parameters for dataset 1: scale
Best Parameters for dataset 2: auto

Performance Metrics:

Performance Metrics	**Dataset 1**	**Dataset 2**
Validation Accuracy	0.9789	0.7344
Test Accuracy	0.9790	0.6982
Precision score	0.9746	0.6814
Recall score	0.9791	0.6508
F1 score	0.9696	0.5333

Best Parameters for Dataset 1: [C: 1, gamma: scale, 'kernel: rbf]

Best Parameters for Dataset 2: [C: 10, gamma: auto, kernel': rbf]

Trade-off Analysis

Hyperparameter Gamma: Gamma is the kernel coefficient of non-linear kernels. It decides how much of a wide influence a support vector has. Lower the gamma the wider the spread of influence. So when gamma is very low the model is more generalized due to that bias is very high and variance is very low. As of that when gamma is very low the model becomes under-fitting (both test and train accuracies are low). But when gamma increases, bias decreases and variance decreases. And gamma is very high, bias is very low and variance is very high causing over-fitting (train accuracy is very high but test accuracy is very low). This is obvious in the plot figures of gamma.

Hyperparameter C: C decides the regularization factor, regularization introduces simplicity to the model reducing the variance and increasing bias. When C increases regularization decreases therefore bias decreases and variance increases. Therefore as seen in the charts when C is very low both training and test accuracies are low due to underfitting and when C is very high though the train accuracy is high test accuracy is low due to overfitting. The previous graphs on C elaborate this trade-off

Hyperparameter Kernel: Kernel decides the linearity of the algorithm. When the kernels non-linearity grows the model complexity increases as well. Therefore rather than a linear kernel non-linear kernel has a less bias and a higher variance. However when non-linearity grows models become more prone to overfitting.

7. AdaBoost Classifier

A random sample of data is chosen, fitted with a model, and then trained sequentially—that is, each model attempts to compensate for the shortcomings of its predecessor. Each iteration combines the weak rules from each individual classifier to form a single, strong prediction rule. Boosting is a machine learning algorithm that aids in the reduction of variance and bias in a machine learning ensemble. Boosting can also help learning algorithms improve their model predictions. AdaBoost assigns the same weight to each dataset at first. The weights of the data points are then automatically adjusted after each decision tree. It gives incorrectly classified items more weight in order to correct them for the next round.It repeats the process until the residual error, or the difference between the actual and predicted values, is less than a certain level.
We trained our AdaBoost classifier model on two different datasets, we used GridSearchCV for the hyperparameter tuning and 10 - fold cross validation for getting the ideal parameters yielding the best performance.

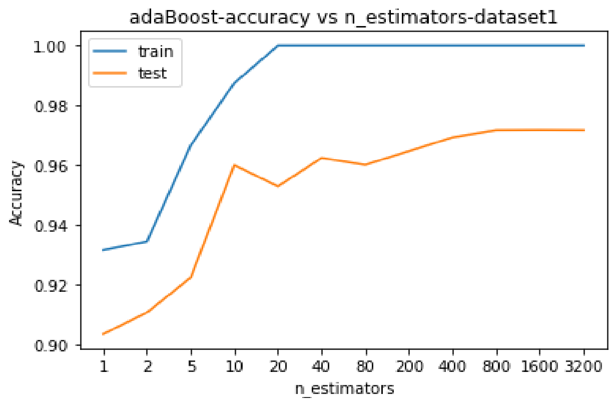
Hyperparameters

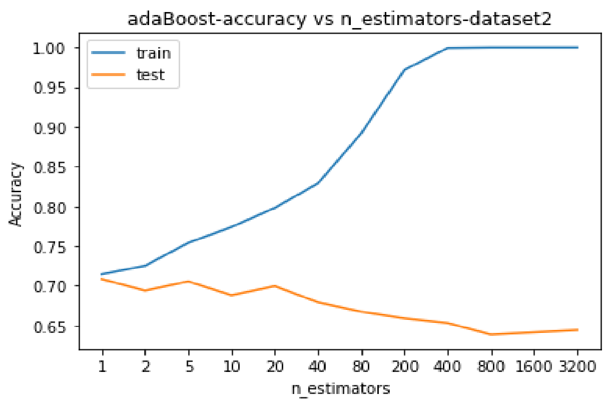
n-estimators: The maximum number of estimators at which boosting is terminated.

In case of perfect fit, the learning procedure is stopped early.

n-estimators values used for cross validation:[1,2,5,10,20,40,80,200,400,800,1600,3200]

Comparison of n-estimators for dataset 1 and dataset 2:





Best Parameters for Dataset 1: n-estimators = 800
Best Parameters for Dataset 2: n-estimators = 1

Performance Metrics:

Performance Metrics	**Dataset 1**	**Dataset 2**
Validation Accuracy	0.9717	0.7083
Test Accuracy	0.9860	0.6293
Precision score	0.9900	0.6359
Recall score	0.9946	0.6439
Roc AUC score	1.0000	0.6822
F1 score	0.9898	0.5904

Trade-off Analysis

Hyperparameter number of estimators: Estimators are weak learners with high bias and low variance. Boosting ensembles them to approach the required accuracy. Therefore when the **number of estimators** are low, the bias is high and variance is low causing under-fitting. When this factor is increased bias is reduced and variance is increased. Hence when n-estimators are too high, the model becomes over-fitting. This can be obviously seen in the dataset 1 number of estimator fluctuation graphs.

8. Trade-off Summary

Algorithm	Hyperparameter	How does increasing the parameter affect?		When parameter is too low	When parameter is too high
		Bias	Variance		
Logistic Regression	C [Inverse of regularization]	Decreases	Increases	underfitting	overfitting
	penality	Increases	Decreases	overfitting	underfitting
KNN	n-neighbors	Decreases	Increases	underfitting	overfitting
Decision Tree	max-depth	Decreases	Increases	underfitting	overfitting
	max-leaf-nodes	Decreases	Increases	underfitting	overfitting
Random-Forest	max_depth	Decreases	Increases	underfitting	overfitting
	n-estimators	Increases	Decreases	overfitting	underfitting
AdaBoost	n-estimators	Decreases	Increases	underfitting	overfitting
SVM	C	Increases	Decreases	overfitting	underfitting
	Non-Linearity of Kernel	Decreases	Increases	underfitting	overfitting
	gamma	Decreases	Increases	underfitting	overfitting

Overall best performance:

Adaboost has given the best performance for the dataset 1 - with n-estimators = 800

Logistic regression has given the best performance for the dataset 2 - with C = 10 and l2 penalty

Experimenting MNIST on Neural Networks

1. Briefing on the Experiment

MNIST Data Set

Experiment Plan

This work is to find out the best performing neural network for the MNIST dataset. Each MNIST image is 28x28, so 784 features.
Training data size = 50000
Validation data size = 10000
Test data size = 10000

Pre-processing

Data has been loaded through mnist_loader.py and converted to categorical data.

Experimental Setting

The neural network experiment has been done using the keras library and the complete experiment has been done on a single note-book for the ease of execution.

Stages of the Experiment is can be elaborate as follows

- Comparing CNN layer with Fully connected layer
- Experimenting with Different number of Convolutional layers
- Experimenting with Different Map sizes
- Experimenting with Different Dense Layer sizes
- Experimenting with Different kernel sizes
- Experimenting with Different dropout probabilities
- Experimenting with Different Optimizers
- Experimenting with weight initializers
- Experimenting with bias initializers
- Experimenting with Different Learning rates

The metrics used to evaluate and compare the model strengths can be pointed out as follows

- Model train accuracy
- Model validation accuracy
- Model training time
- Model inference time
- Model training loss
- Model validation loss
- Accuracy fluctuation with the epoch

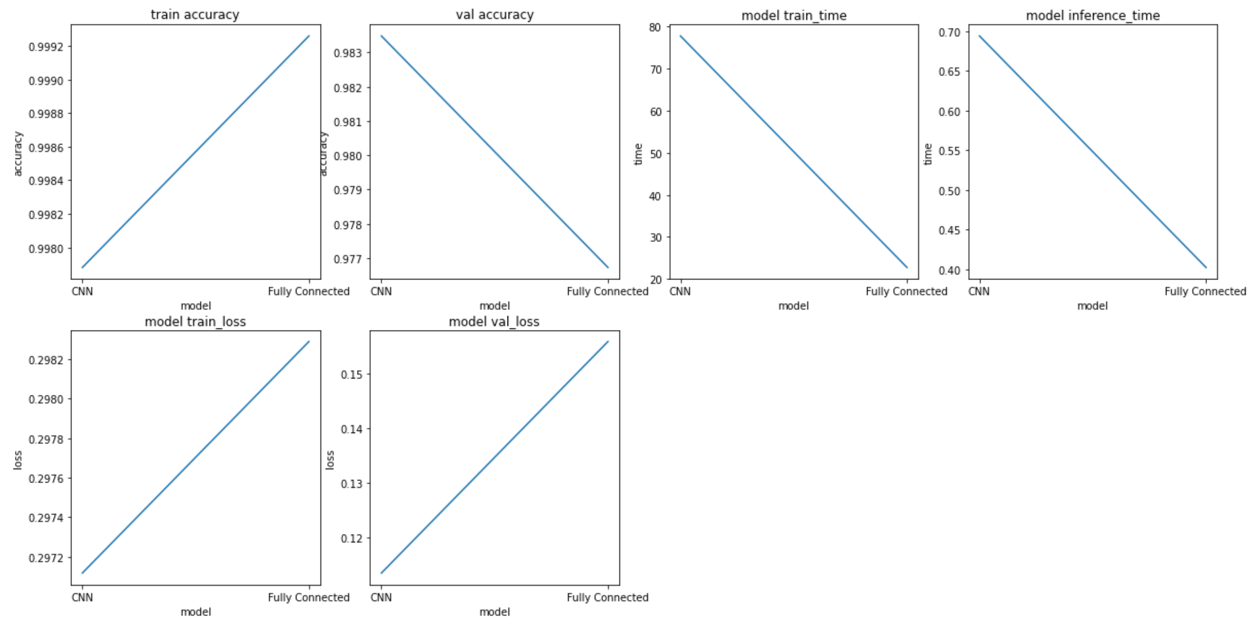
Work Flow of the Experiment

1. To check each hyperparameter a model has been created and put it into an array.

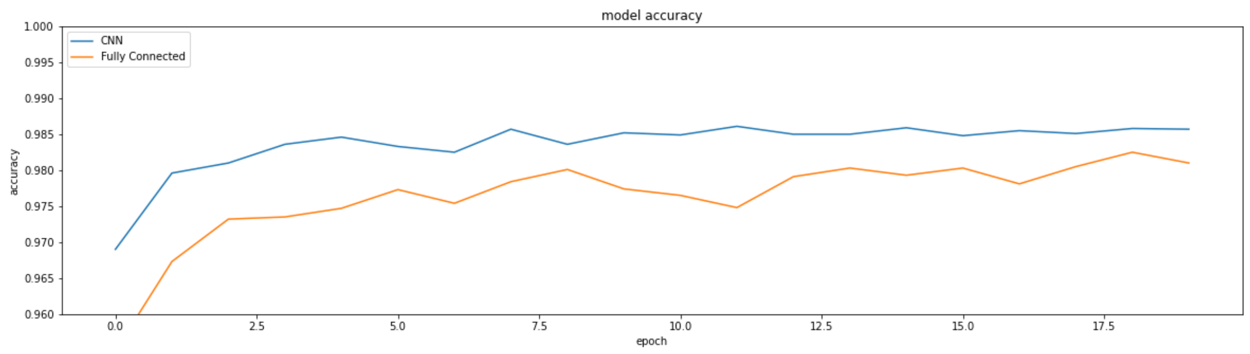
2. Then models has been trained with training dataset and evaluation data is used to evaluate
3. Using the history data plots are created and models have been compared in terms of various metrics.
4. Best hyper-parameter has been chosen in each experiment related to the neural network
5. And the experiments have been continued with chosen parameters from previous experiments.
6. At the end, performances or both the model with just two fully connected layers and model with the hyper-parameters chosen from experiments are compared against each other.

2.Comparing CNN layer with Fully connected layer

First experiment is done to check which type of layer is better for the neural network. We did a comparison between a Fully connected layer vs a Convolutional layer
Note- Convolution layer performs well for image processing as it is able to extract patch wise data effectively as only a neighborhood of data is used for the output for the layer.



Train Accuracy	Validation Accuracy
Conv. Layer-0.99824	0.98610
Fully-0.99980	0.98250



Convolution layer holds higher validation accuracy than the Fully connected, moreover the train and validation accuracies proves that Fully connected layer is overfitted.So it is preferred that having a Convolution layer is better than fully connected layer for image processing purpose.

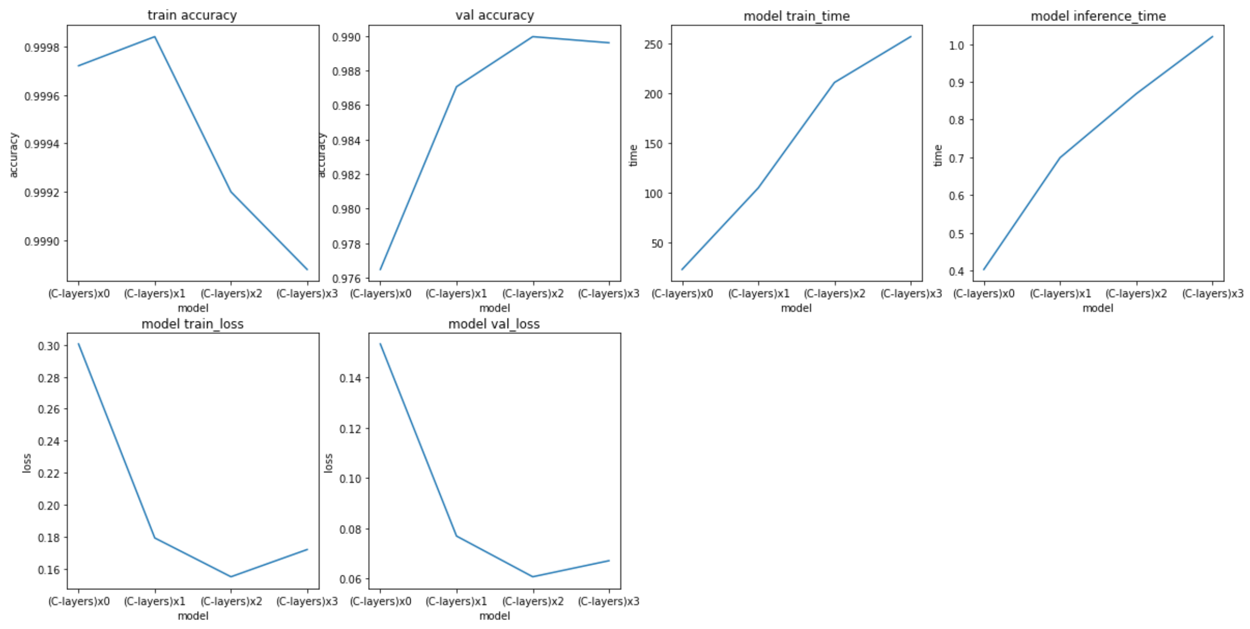
Decision - Use a convolutional layer.

Analysis:

Fully connected layers don't see any order of the inputs. But in images pixels are given as inputs and they are connected to each other. However, convolutional layers take the advantage of local spatial coherence of the images as it considers patches of adjacent pixels. Therefore the convolutional layer performs better.

3. Experimenting with Different number of Convolutional layers

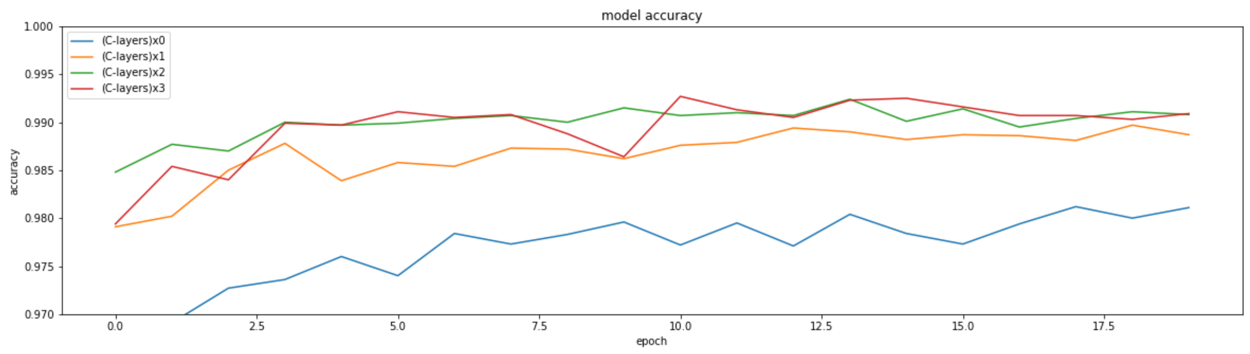
This experiment is done to decide the optimum number of Convolutional layers in the neural network Increasing the number of layers, reducing the bias and increasing the variance which enlarges the possibility of overfitting.



C-layers	Train_Acc	Val_Acc
0	0.99990	0.98140
1	0.99966	0.99010
2	0.99914	0.99200
3	0.99916	0.99260

Though a model with 3 layers has the highest validation accuracy the training and inference time is the highest as well due to model complexity. We decided to continue with two hidden convolution layers which significantly increase the validation accuracy.

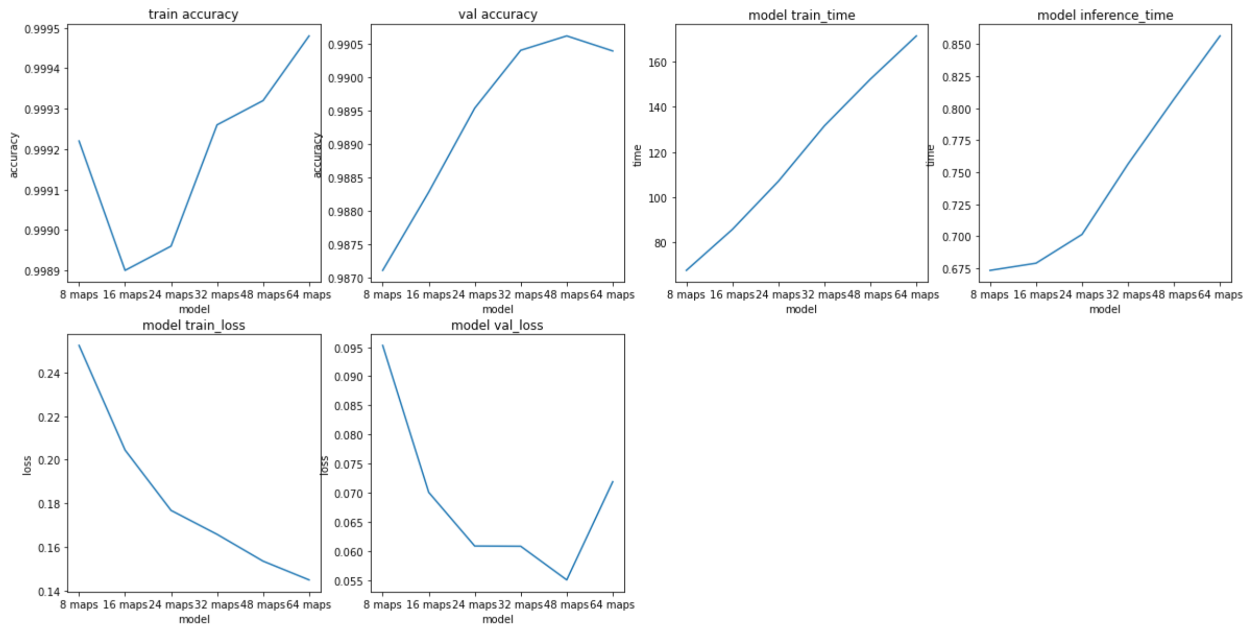
Decision - Use two hidden Convolutional Layers



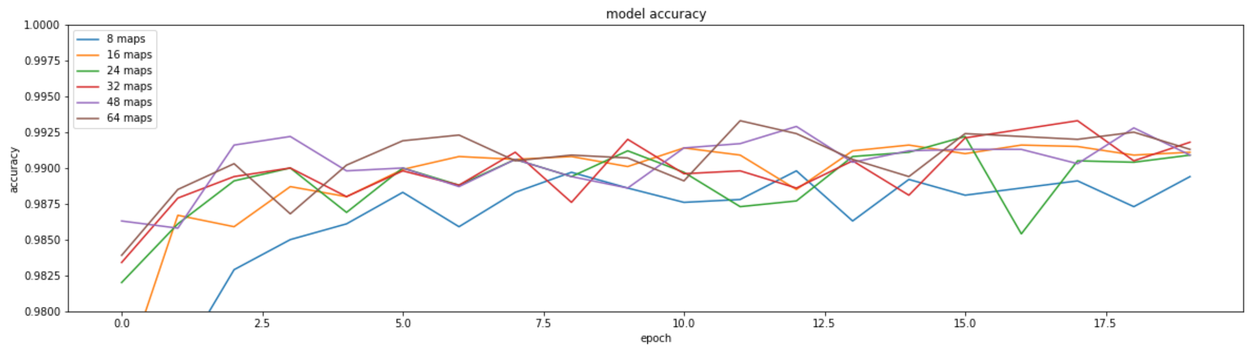
Analysis:
Increasing the number of convolutional layers increases the model complexity. Where resulting decrease in the bias and increase in the variance. Hence when layer number is too low under-fitting happens and if layer number is too high over-fitting happens. This is quite obvious in above train_loss and validation_loss graphs.

4. Experimenting with Different Map sizes

This experiment is done to decide the optimum number of map size of convolutional layers in the neural network. Increasing the number of map sizes reduces the bias and increase the variance which enlarges the possibility of overfitting.



Map Size	Train_Acc	Val_Acc
8	0.99788	0.98980
16	0.99944	0.99160
24	0.99916	0.99220
32	0.99922	0.99330
48	0.99906	0.99290
64	0.99922	0.99330



Considering the results we can finalize that the best map size is 32 for the convolutional layer.

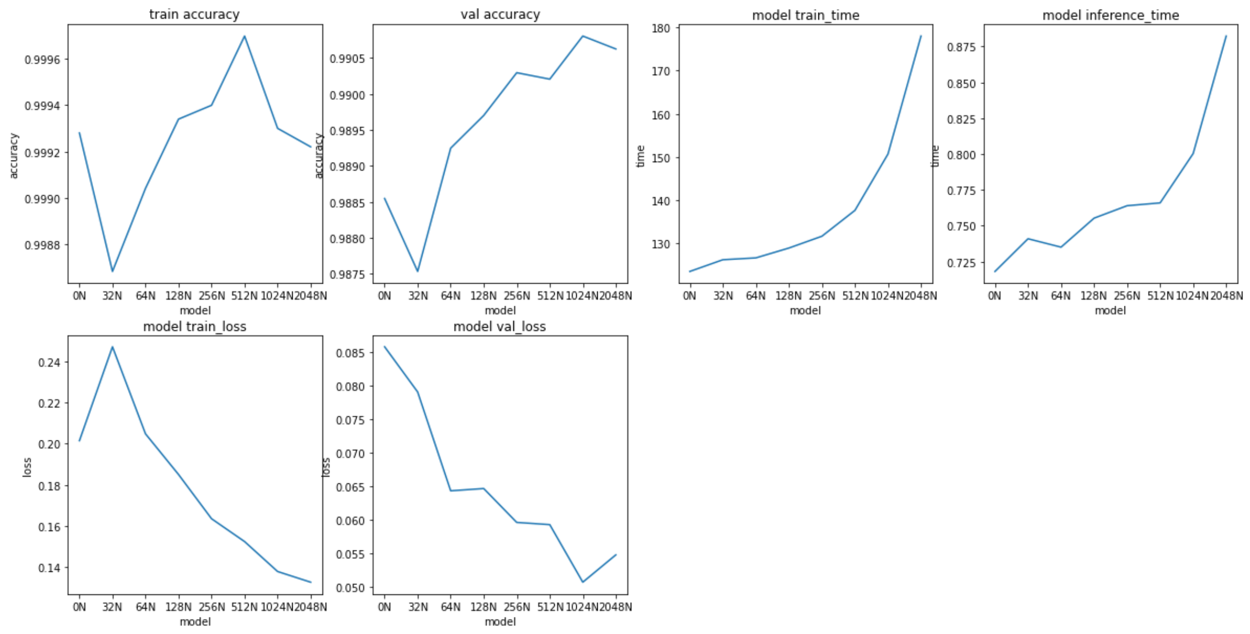
Decision - Use 32 as the map size for the convolutional layers

Analysis:

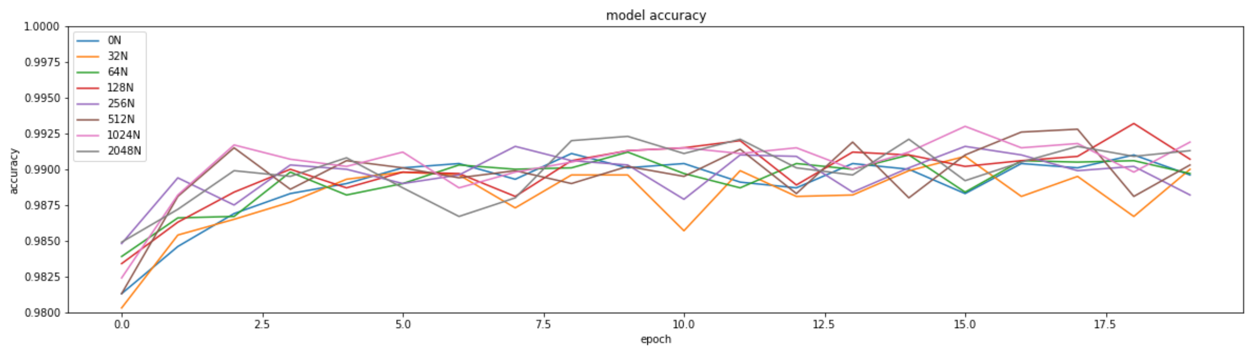
When the map size increases the complexity of the model increases. Due to this bias decreases and the variance increases. Therefore when the map size is very low, the model tends to underfit and when the map size is very high the map tends to overfit. This is quite obvious in the above graphs. Therefore the map size should be chosen appropriately.

5. Experimenting with Different Dense Layer sizes

This experiment is done to decide whether to add a dense layer prior to the softmax layer, if so what should be the output dimension of it. Increasing the number of dimensions, reduce the bias and increase the variance which enlarge the possibility of overfitting.



DenseLayer	Train_Acc	Val_Acc
0	0.99922	0.99110
32	0.99868	0.99090
64	0.99946	0.99120
128	0.99918	0.99320
256	0.99928	0.99160
512	0.99988	0.99280
1024	0.99912	0.99300



Ran the above test several times but we got best accuracy from 128 and 256 from different occasions. Therefore considering the complexity factor as well decided to go with 128.

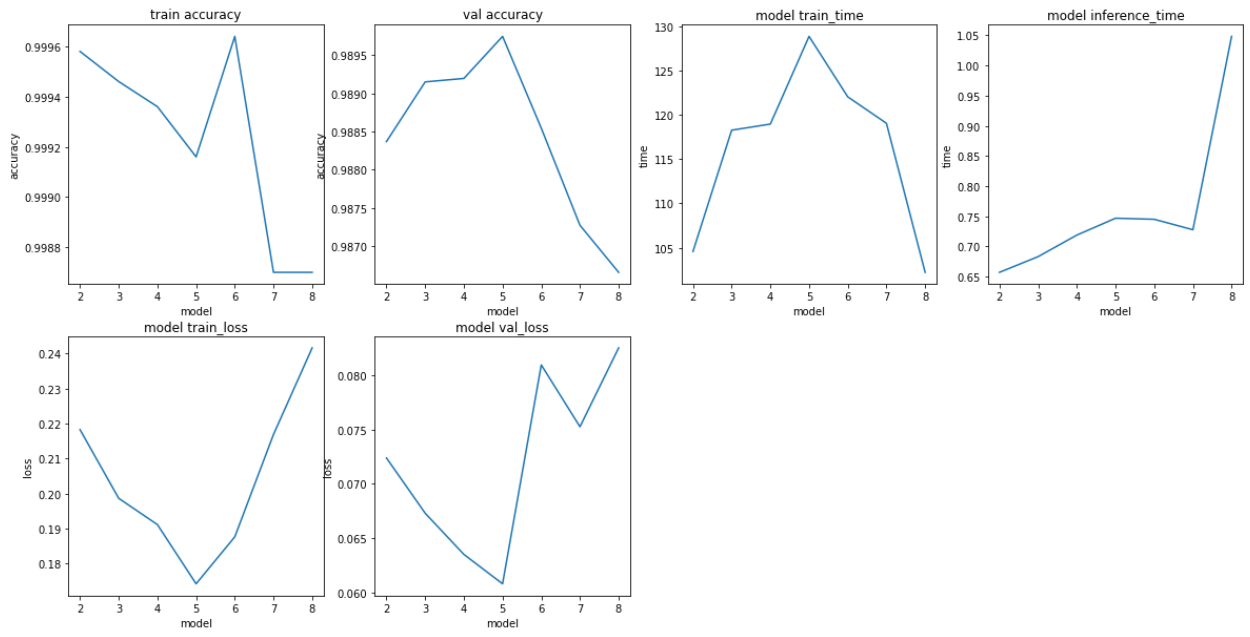
Decision - Use 256 as the dimension of the dense layer.

Analysis:

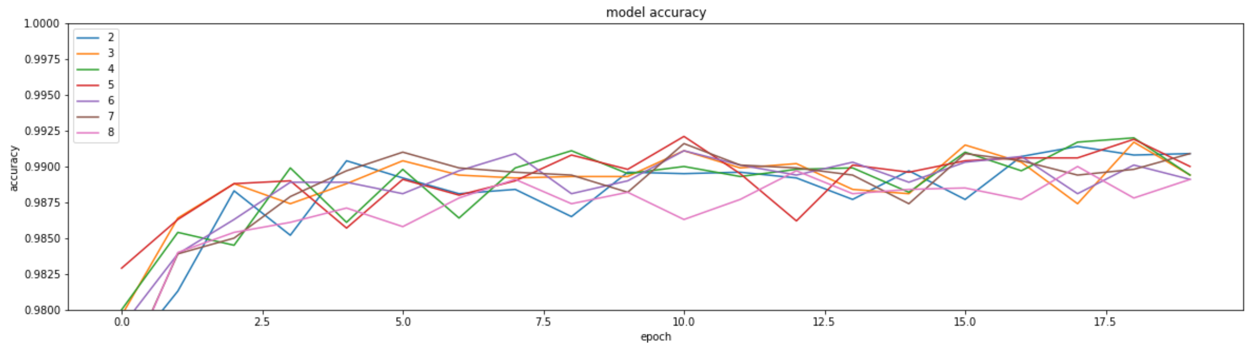
Increasing the dense layer size increases the model complexity. Due to this bias decreases and the variance increases. Therefore when the map size is very low, the model tends to underfit and when the map size is very high the map tends to overfit. This is quite obvious in the above graphs. Therefore the dense layer size should be chosen appropriately.

6. Experimenting with Different kernel sizes

This experiment is done to decide the kernel size of the convolutional layer. This value is a hyperparameter of the CNN.



Kernel size	Train accuracy	Validation Accuracy
2	0.99914	0.99140
3	0.99936	0.99170
4	0.99902	0.99200
5	0.99950	0.99210
6	0.99888	0.99110
7	0.99844	0.99160
8	0.99834	0.99000



Considering the results we can see that the kernel with size 5 has the highest accuracy.

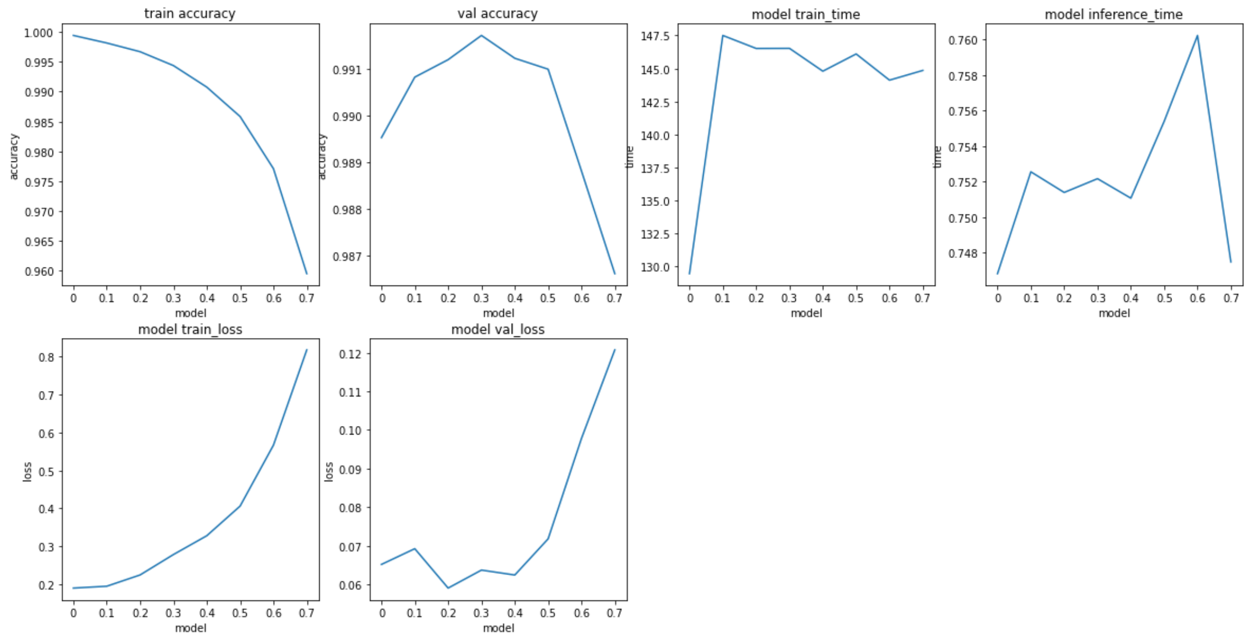
Decision - Use kernel size 5

Analysis:

The kernel size decides how much of a patch is considered for the input of the next layer's input. When kernel size is too large , it considers more data where output is more generalized causing high bias and lower variance resulting in underfitting. But when kernel size is too small this causes overfitting due to lower bias and higher variance. This can be seen in the above graphs. So kernel size must be chosen appropriately.

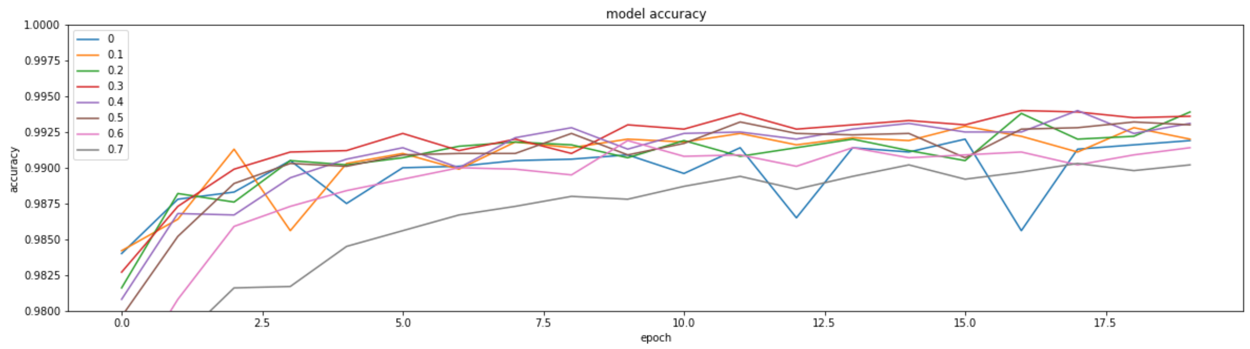
7. Experimenting with Different dropout probabilities

Dropout is done to reduce the complexity of the neural network and avoid overfitting. When the drop out probability is increased the bias of the model increases and the variance decreases.



Drop. Prob.	Train Accuracy	Validation Accuracy
0	0.99946	0.99200

0.1	0.99800	0.99290
0.2	0.99638	0.99390
0.3	0.99434	0.99400
0.4	0.99098	0.99400
0.5	0.98602	0.99320
0.6	0.97702	0.99190
0.7	0.95874	0.99030



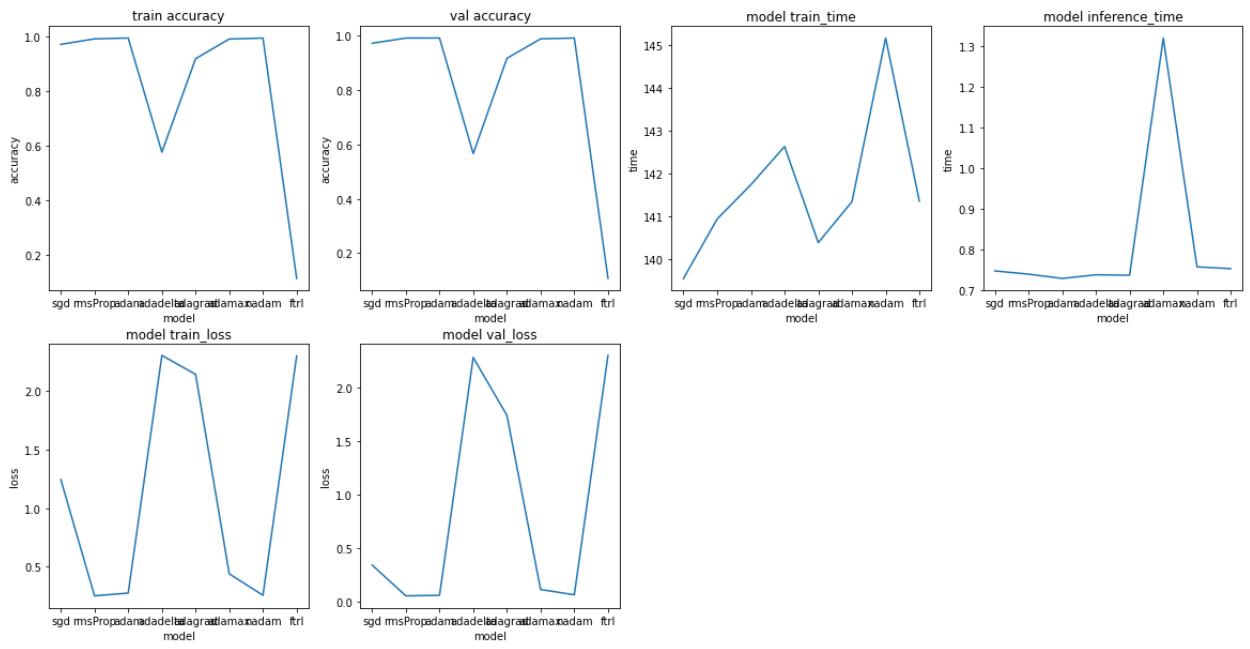
Decision - Use 0.3 as the dropout probability

Analysis:

Dropout is introduced to reduce the complexity of the neural network and avoid overfitting. When the dropout probability increases the model complexity decreases, so it increases the bias and reduces the variance. When the dropout probability is very low the model has low bias, high variance so is overfitting. But when dropout probability is very high, bias is too high, variance is too low causing under-fitting. This can obviously seen in the above graphs. Therefore dropout probability should be chosen appropriately.

8. Experimenting with Different Optimizers

Optimizer is the technique used to update the weights and bias to achieve the convergence point. Choosing the optimal technique is important to that. Therefore we are cross checking with each of the methods and comparing the performance.



Optimizer	Train Acc	Val. Acc.
sgd	0.97178	0.98480
rmsProp	0.99112	0.99320
adam	0.99456	0.99410
adadelta	0.61578	0.82040
adagrad	0.91884	0.95790

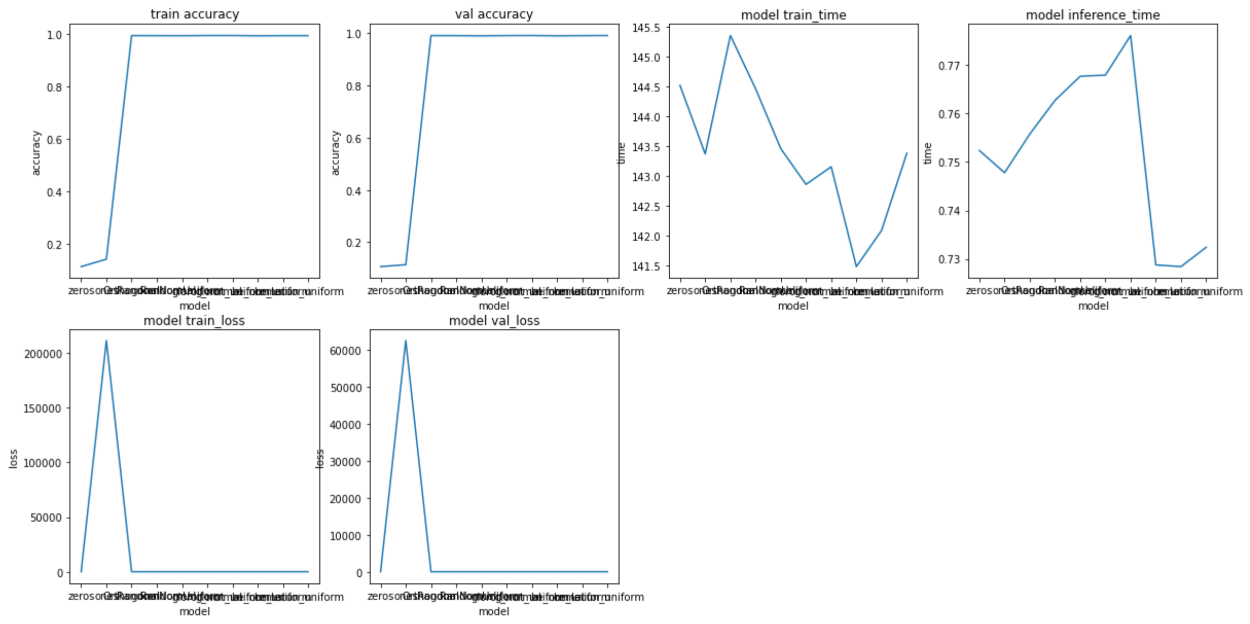
adamax	0.99084	0.99260
nadam	0.99430	0.99370
ftrl	0.11356	0.10640

Decision : Considering all the optimizers ADAM is chosen

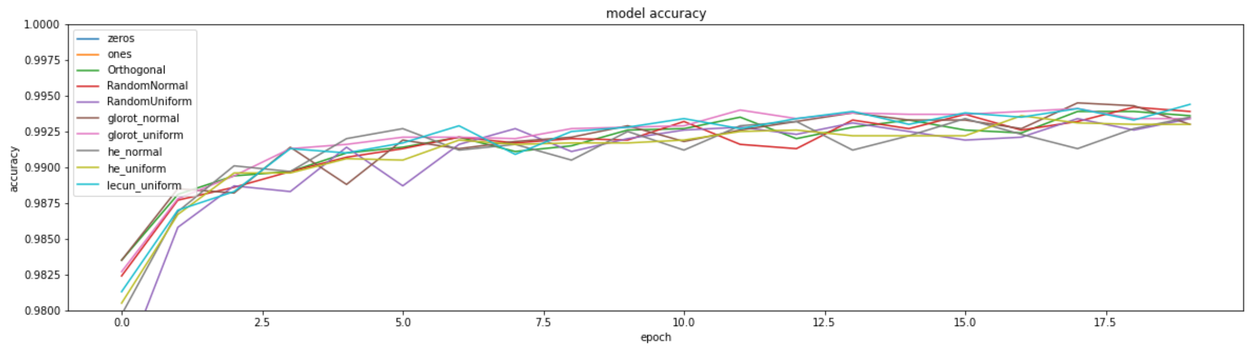
Analysis:
FTRL optimizer is performing extremely bad and ADADELTA is bad for the mnist dataset as well. The ADAGRAD optimizer is not good for the dataset and SGD is slightly bad. Compared to previous optimizers other optimizers perform quite well.

9. Experimenting with Weight Initializer

Weight initializers have an impact on the final accuracy and the convergence time. We analyze each initializer and decide the best one.



Initializer	Train acc.	Test_acc
zeros	0.11356	0.10640
ones	0.14096	0.14650
Orthogonal	0.99410	0.99320
RandomNormal	0.99402	0.99370
RandomUniform	0.99388	0.99380
glorot_normal	0.99456	0.99480
glorot_uniform	0.99402	0.99420
he_normal	0.99372	0.99340
he_uniform	0.99378	0.99370
lecun_uniform	0.99404	0.99380



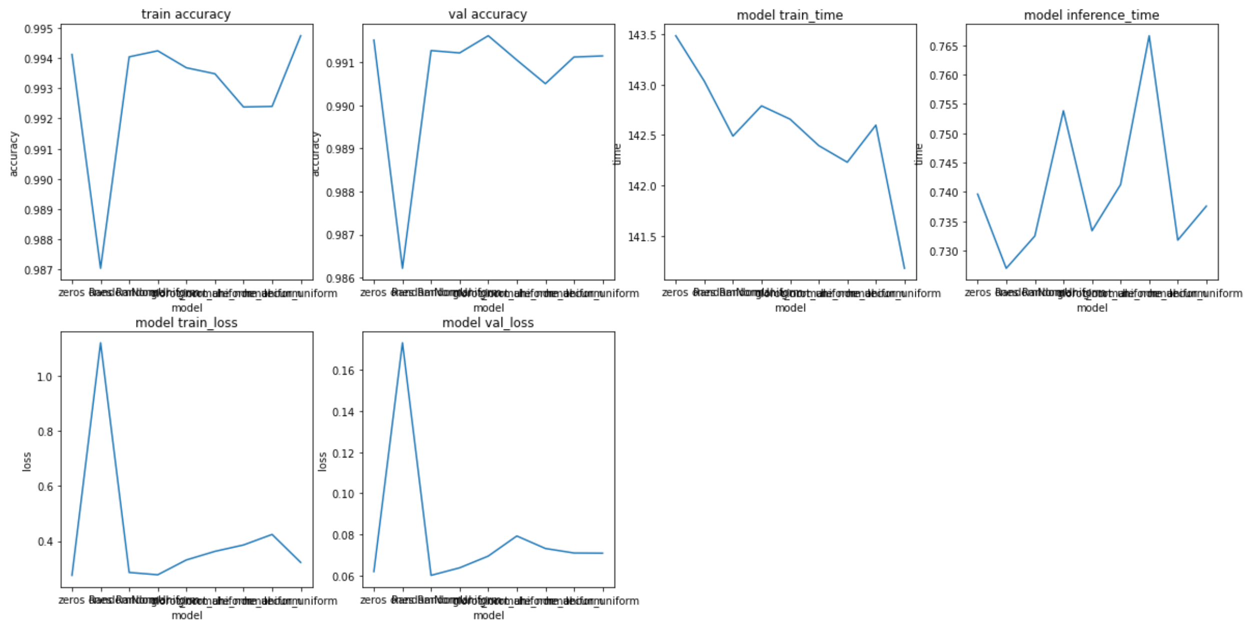
Decision : As the weight initializer glorot_normal is chosen

Analysis:

There are many weight initializers introduced in the keras. Though most of them work quite well both **zeros** and **ones** initializer work extremely bad as initializers. This is because neural networks tend to get stuck in local minima, so using zeros and ones cause local minimas. Also, if the neurons start with the same weights, then all the neurons will follow the same gradient, and will always end up doing the same thing as one another.

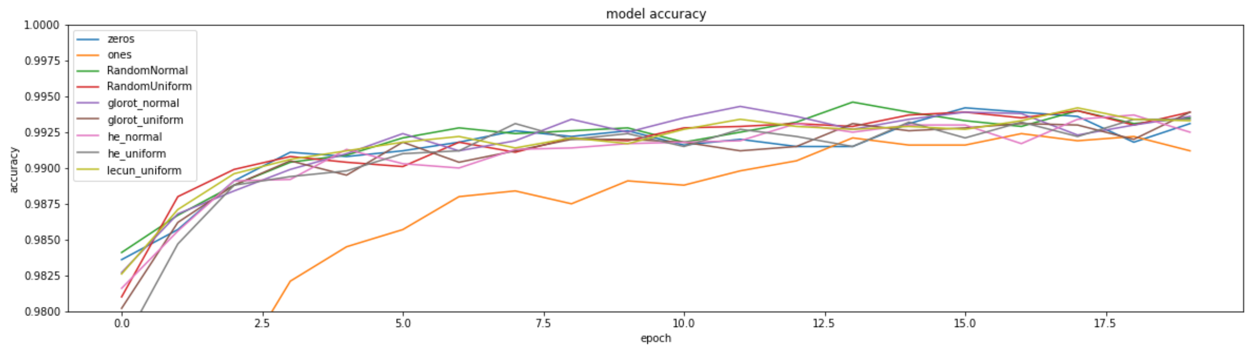
10. Experimenting with Bias initializer

Bias initializers have an impact on the final accuracy and the convergence time. We analys each initializer and decide the best one.



Initializer	Train acc.	Test_acc
zeros	0.99460	0.99400
ones	0.98908	0.99240
RandomNormal	0.99392	0.99390
RandomUniform	0.99480	0.99340
glorot_normal	0.99410	0.99390

glorot_uniform	0.99350	0.99340
he_normal	0.99272	0.99340
he_uniform	0.99222	0.99440
lecun_uniform	0.99326	0.99340

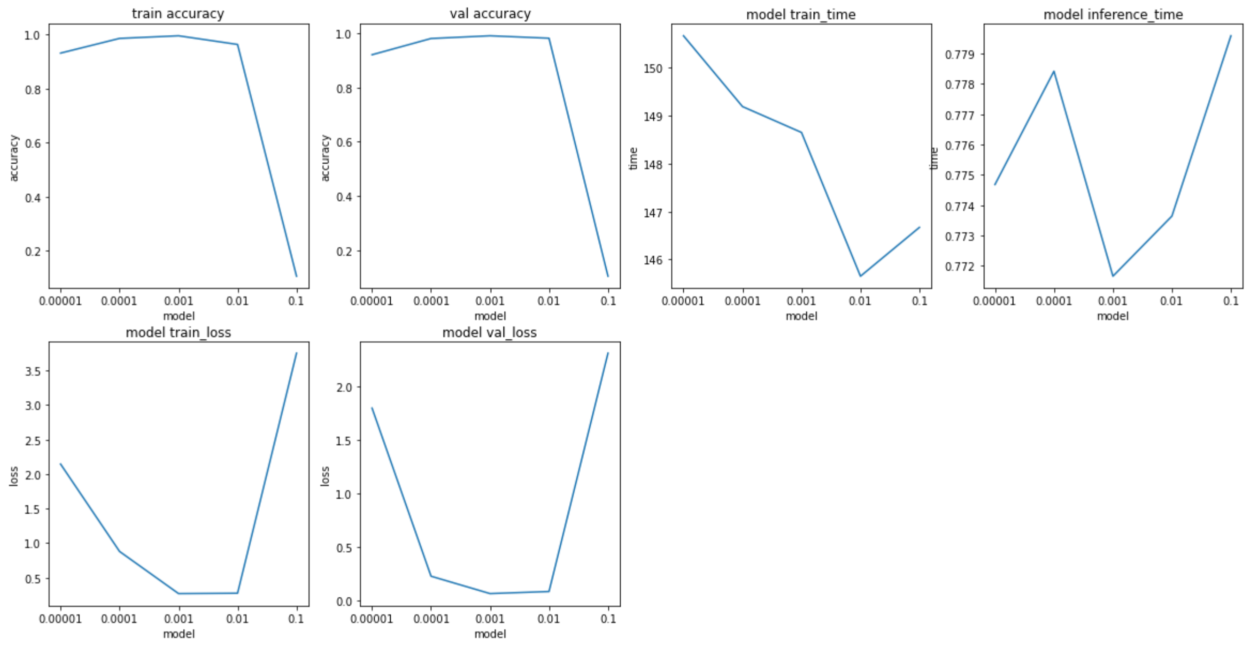


Decision : As the bias initializer zeros is chosen

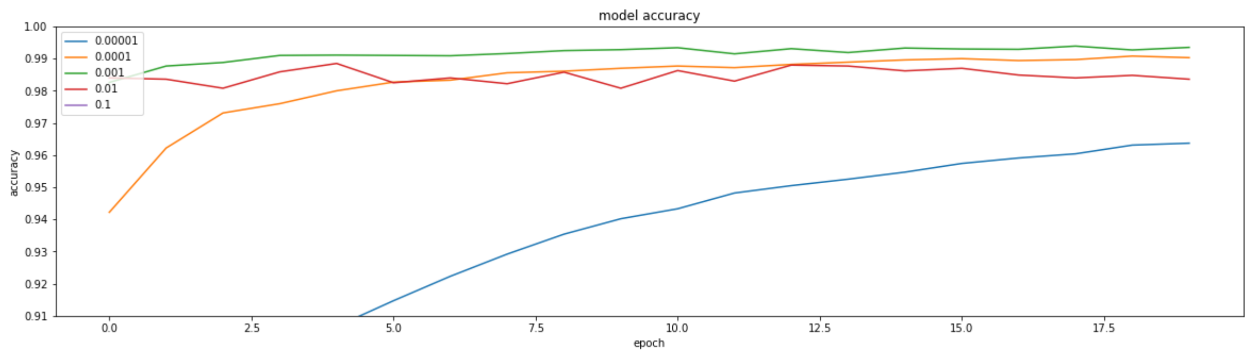
Analysis:
All the bias initializers work quite well for the MNIST dataset. While **zero** and **he-uniform** performs the best comparative to the others, the **ones** initializer gives the worst performance and is significantly poor.

11. Experimenting with Different Learning rates

Learning rate decide the convergence speed of the model. We experimented with different Learning rates to find out the best performing learning rate.



Learning rate	Train accuracy	Validation accuracy
0.00001	0.93664	0.96300
0.0001	0.98540	0.99040
0.001	0.99442	0.99400
0.01	0.95974	0.98570
0.1	0.10702	0.10900



Decision : As the learning rate, 0.001 is chosen

Analysis:
Learning rate decides the step size of the convergence. If the learning rate is too low the model convergence time becomes very high. But if the learning rate is too high the minima could be missed while the convergence. This can be seen in the above . Therefore the learning rate should be

12. Trade-Off Summary

Hyperparameter	How does increasing the parameter affect?		When parameter is too low	When parameter is too high
	Bias	Variance		
No. of Convolutional layers	Decreases	Increases	underfitting	overfitting
Map size of conv. layers.	Decreases	Increases	underfitting	overfitting
Dimension if dense layer	Decreases	Increases	underfitting	overfitting
Drop out probability	Increases	Decreases	overfitting	underfitting
Kernel size	Increases	Decreases	overfitting	underfitting

13. Deciding the Parameters

Considering all the above experiments best performance is given by following model
Use Convolutional layer = True
Number of convolutional layers = 2
Is using dense layer before softmax =True
Map size of convolutional layer = 32
The size of the Dimension of dense layer = 256
Drop out probability = 0.3
The size of the Convolutional layer kernel = 5
Weight Initializer = glorot_normal
Bias Initializer = zeros
Optimizer = Adam
Learning rate = 0.001
Note: The validation accuracy as well as the order of the performance changed in different times. Therefore the above parameters have been selected after several runs and may not be giving best performance when executing above code

To summarize the effectiveness of the experiment we have compared a simple fully connected two hidden layer neural network with the neural network obtained through the experiments.

Metric	Preliminary- Neural Network	Experimentally Obtained NN
Training accuracy	0.9984	0.9951
Validation accuracy	0.9769	0.9936
Test Accuracy	0.9776	0.9932
Precision Score	0.9774	0.9932
Recall Score	0.9776	0.9932

As it is seen in the data though the preliminary neural network is overfitting, Experimentally Obtained NN is perfectly fitting. And could been able to gain an accuracy gain of **3.4x** times.

Obtained Validation accuracy is = 0.9936

Obtained Test accuracy is = 0.9932