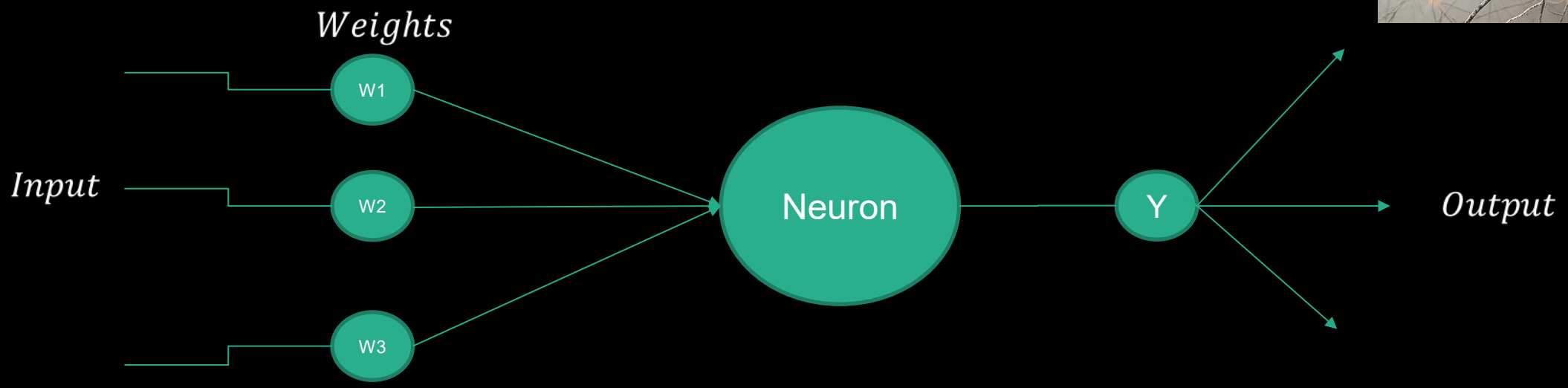
Understanding Neural Network in Geometric Metrics

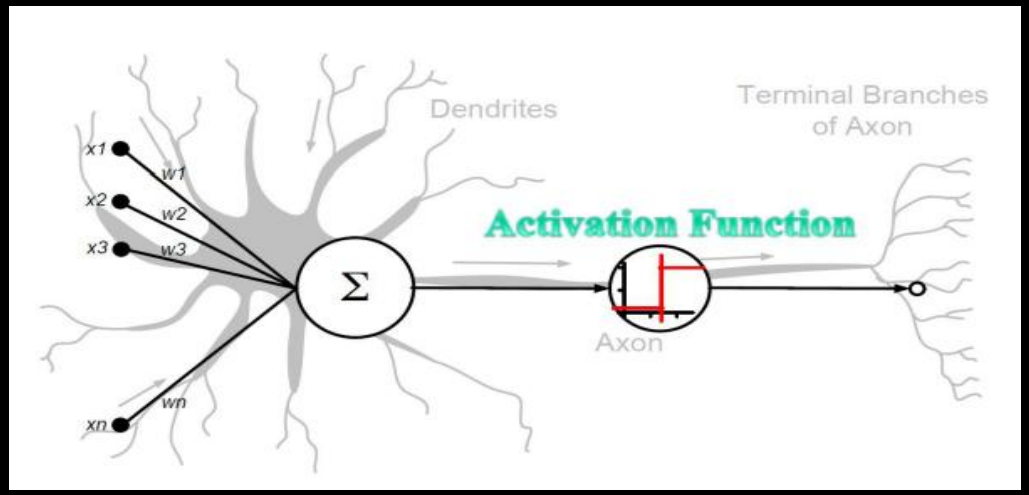
Which geometric properties result in powerful neural networks

Group 7:

Jon Butterfield, Saviz Saei, John Austin Reed

Diagram of a Neuron





Analyzing NN

- Traditional methods mostly rely on performance metrics.
  - Accuracy, Precision, & Recall
- We can’t visually analyze Neural Networks because they are large.
- But Graph Metrics provide information about graphs without having to visualizing them.

Phase 3: Baselines with Benchmarks

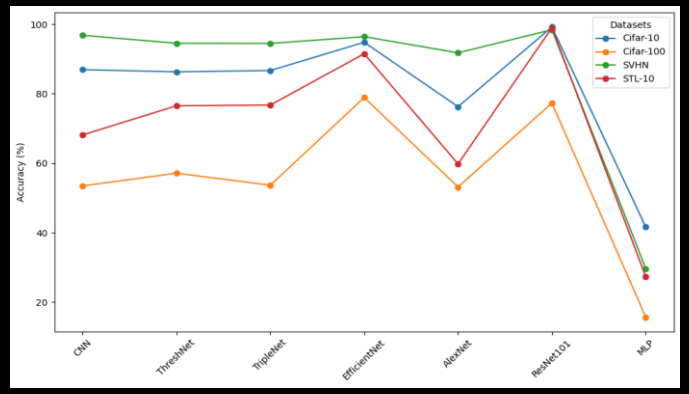
- we use seven models to evaluate their performance on benchmark classification datasets, including Cifar10, Cifar100, SVHN, and STL-10.

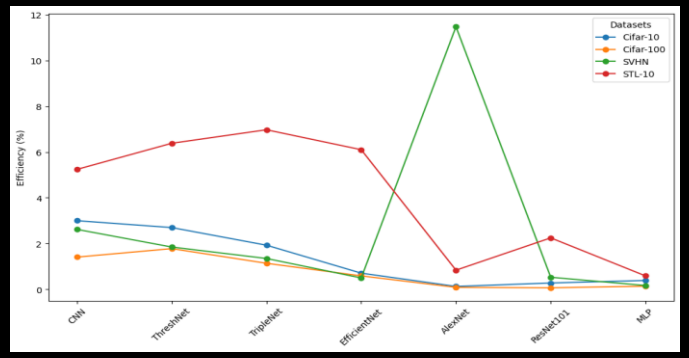
Model/Dataset	Cifar10	Cifar100	SVHN	STL-10
CNN	86.90%	53.46%	96.80%	68.12%
ThreshNet	86.29%	57.11%	94.51%	76.52%
TripleNet	86.66%	53.66%	94.46%	76.72%
EfficientNet	94.78%	78.88%	96.40%	91.49%
AlexNet	76.25%	53.15%	91.76%	59.78%
ResNet101	99.19%	77.28%	98.39%	98.98%
MLP	41.62%	15.7%	29.67%	27.31%

Table 2. Top1 Accuracy Rates

Model/Dataset	Cifar10	Cifar100	SVHN	STL-10
CNN	29s	38s	37s	13s
ThreshNet	32s	32s	51s	12s
TripleNet	45s	47s	70s	11s
EfficientNet	133s	134s	193s	15s
AlexNet	587s	593s	8s	71s
ResNet101	360s	1100s	186s	44s
MLP	105s	106s	168s	46s

Table 3. Total Time Per Epoch



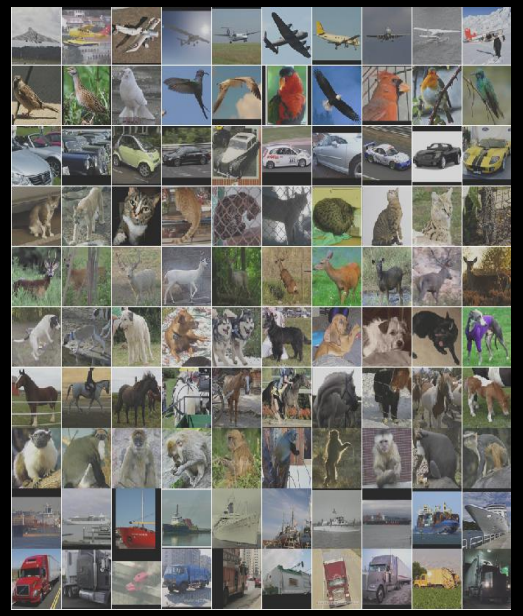


Datasets

CIFAR-10: 10 classes, 32x32 pixels, colors, 60000 images

CIFAR-100: 100 classes, 600 images, Like the CIFAR-10

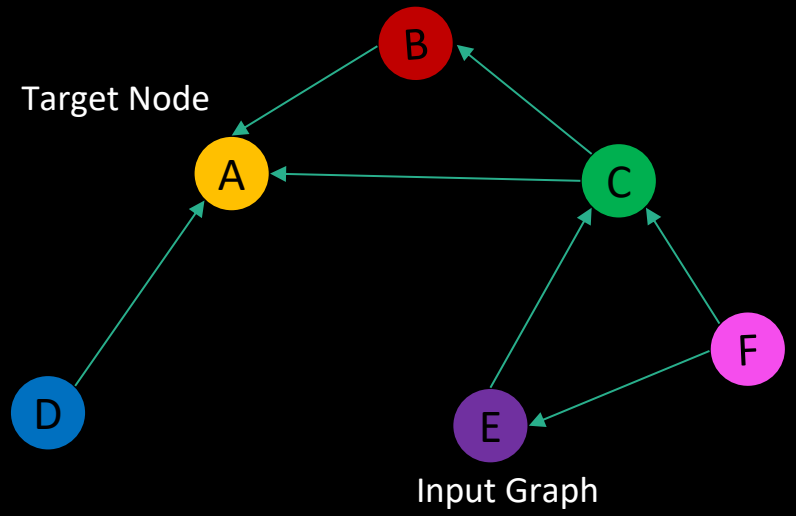
SVHN: 10 classes, 96x96 pixels, color, 100000 images

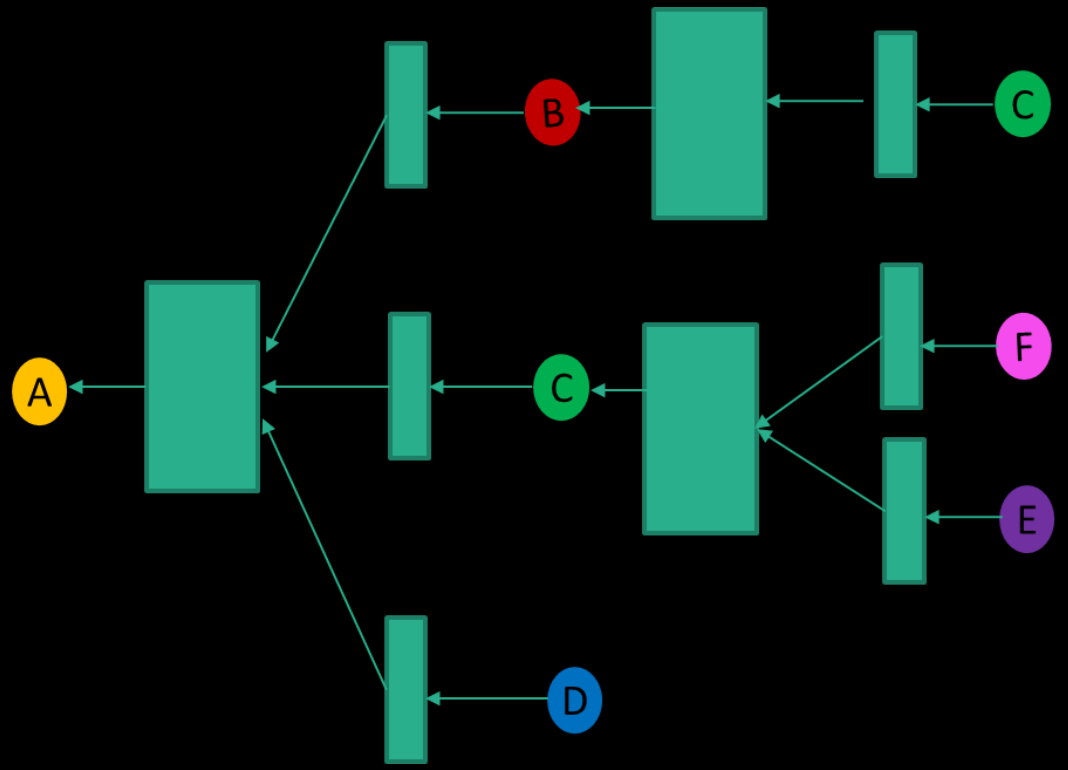


STL10: 10 classes, 96x96 pixels, color, 100000 images

https://www.cs.toronto.edu/~kriz/cifar.html
http://ufldl.stanford.edu/housenumbers/
https://cs.stanford.edu/~acoates/stl10/

Relational Graph





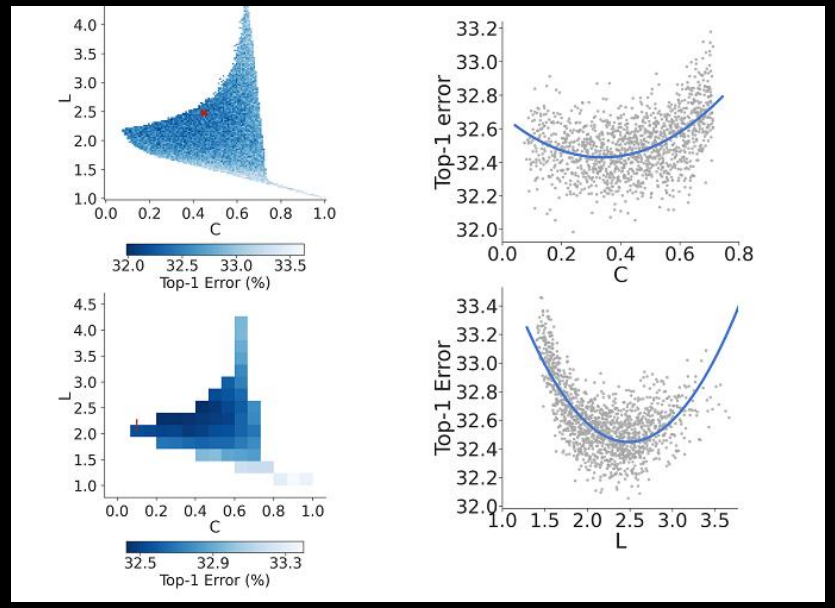
- Nodes → patterns
- Edges → relations

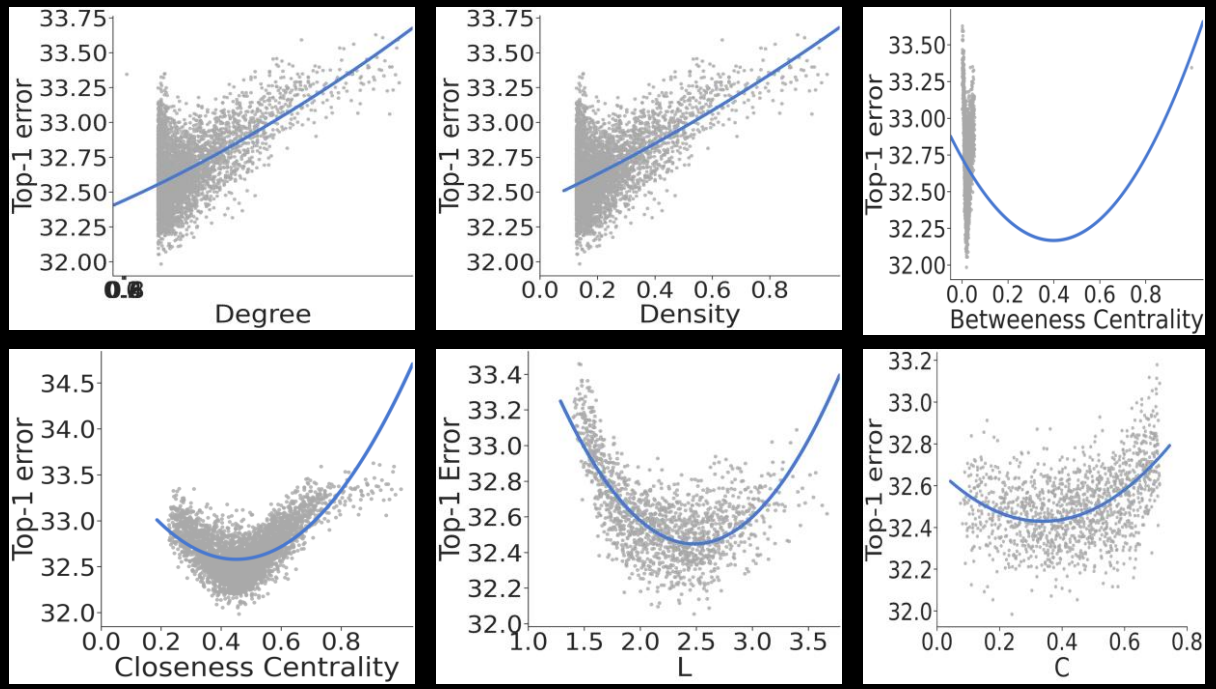
Metrics

Average clustering coefficient	The average number of steps along the shortest paths for all pairs of nodes: $\mathrm{L} = \frac{3 \times \text{number of triangles}}{\text{total number of possible triangles}}$
Average shortest path length	The average number of steps along the shortest paths for all possible pairs of network nodes: $\mathrm{C}(v) = \frac{1}{n\,(n-1)} \sum_{u \neq v \in V} d(u,v)$
Density	The ratio between the edges present in a graph and the maximum number of edges: $D = \frac{2\lvert E \rvert}{n\,(n-1)}$
Betweenness Centrality	Quantifies how often a node acts as a bridge along the shortest path between two other nodes: $\mathrm{B(v)} = \sum_{u \neq v \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}$
Closeness Centrality	Measures how close a node is to all other nodes in the graph: $\mathrm{Cc(v)} = \frac{1}{\sum_{u \neq v \in V} d(u,v)}$

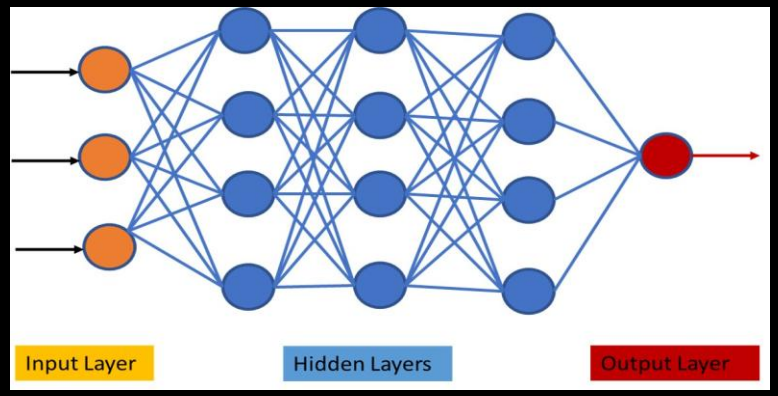
https://networkx.org/documentation

Graph2nn


L
C
Top-1 Error (%)
Top-1 error
Top-1 Error

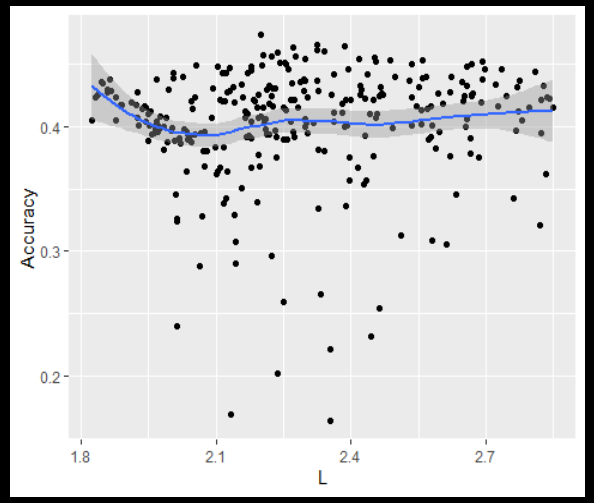

Top-1 error
Degree
Density
Betweeness Centrality
Closeness Centrality
Top-1 Error
L
C

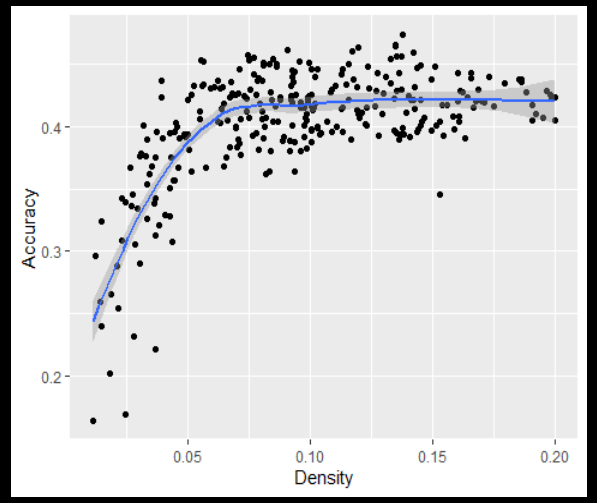
Multilayer Perceptron (MLP)

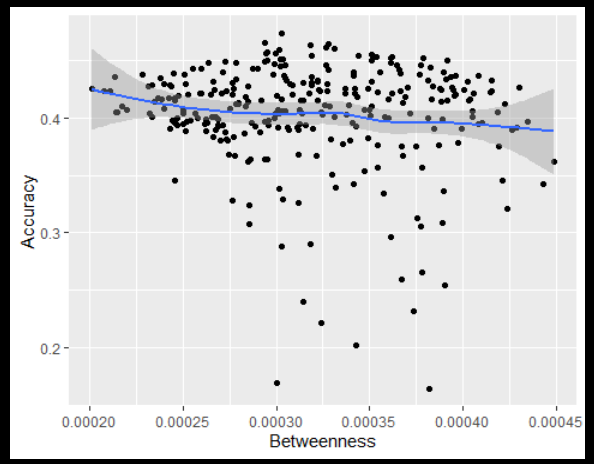


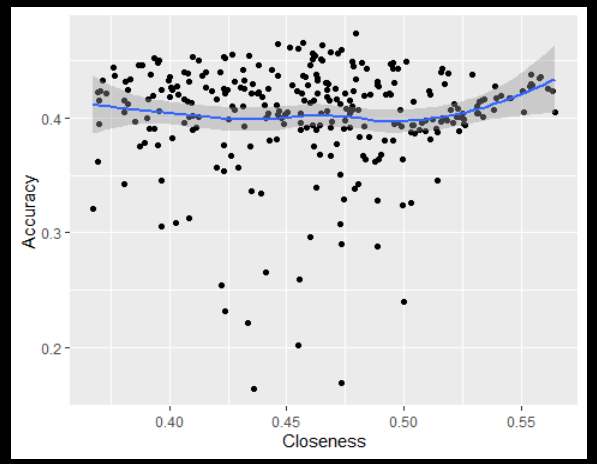

```
while True:
    ex_layers = random.randint(4,7)
    max_layer_nodes = 700 - 50*ex_layers
    min_layer_nodes = 10

    nodes1 = random.randint(min_layer_nodes, max_layer_nodes)
    nodes2 = random.randint(min_layer_nodes, max_layer_nodes)
    nodes3 = random.randint(min_layer_nodes, max_layer_nodes)
    nodes4 = 0
    nodes5 = 0
    nodes6 = 0

    #https://scikit-learn.org/stable/modules/neural_networks_supervised.html
    from sklearn.neural_network import MLPClassifier
    clf = MLPClassifier(solver='adam', alpha=1e-4, hidden_layer_sizes=(nodes1, nodes2, nodes3), random_state=1, max_iter=200, verbose=True)
    if ex_layers == 5:
        nodes4 = random.randint(min_layer_nodes, max_layer_nodes)
        clf = MLPClassifier(solver='adam', alpha=1e-4, hidden_layer_sizes=(nodes1, nodes2, nodes3, nodes4), random_state=1, max_iter=200, verbose=True)
    if ex_layers == 6:
        nodes4 = random.randint(min_layer_nodes, max_layer_nodes)
        nodes5 = random.randint(min_layer_nodes, max_layer_nodes)
        clf = MLPClassifier(solver='adam', alpha=1e-4, hidden_layer_sizes=(nodes1, nodes2, nodes3, nodes4, nodes5), random_state=1, max_iter=200, verbose=True)
    if ex_layers == 7:
        nodes4 = random.randint(min_layer_nodes, max_layer_nodes)
        nodes5 = random.randint(min_layer_nodes, max_layer_nodes)
        nodes6 = random.randint(min_layer_nodes, max_layer_nodes)
        clf = MLPClassifier(solver='adam', alpha=1e-4, hidden_layer_sizes=(nodes1, nodes2, nodes3, nodes4, nodes5, nodes6), random_state=1, max_iter=200, verbose=True)

    print("Nodes: ", nodes1, nodes2, nodes3, nodes4, nodes5, nodes6)
    clf.fit(x_train_f, y_train)
```

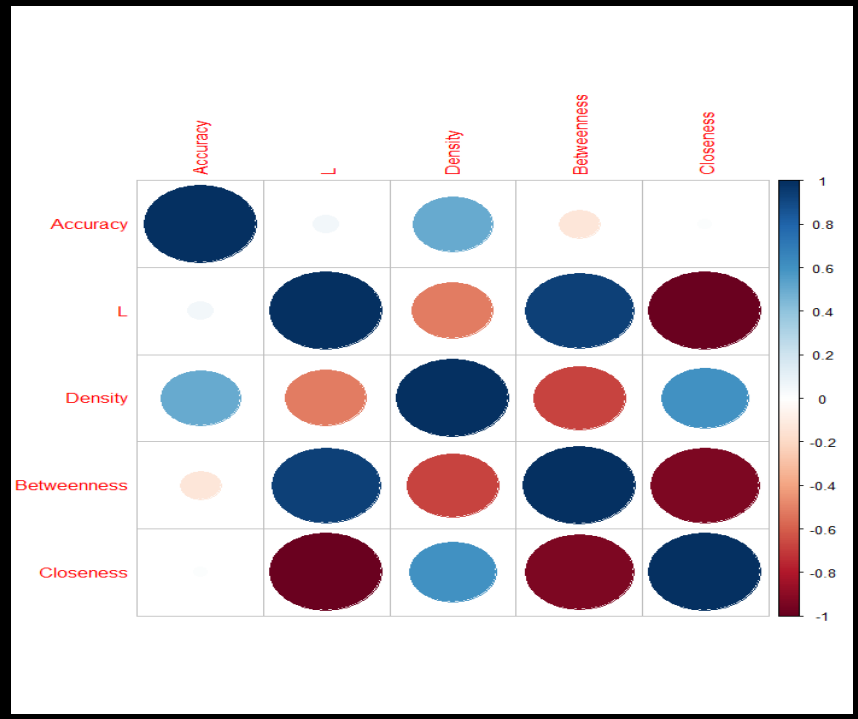
Analysis Metrics



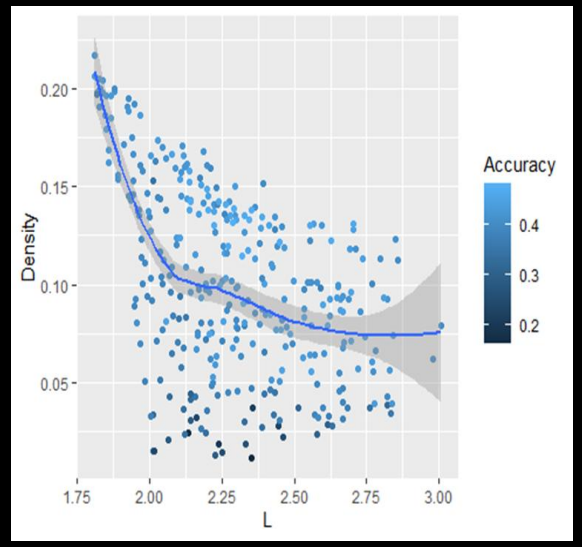


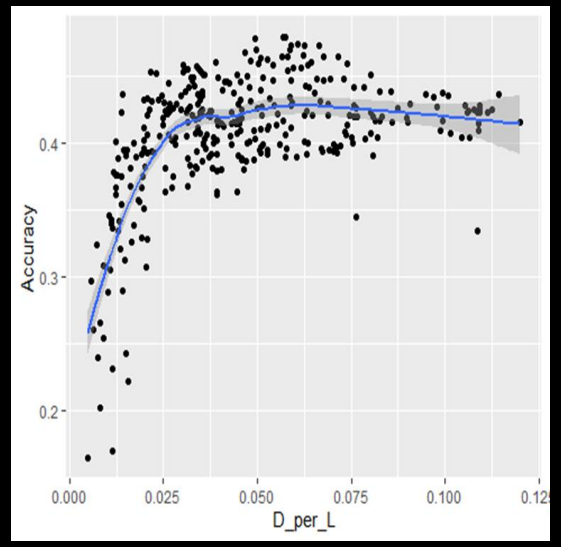


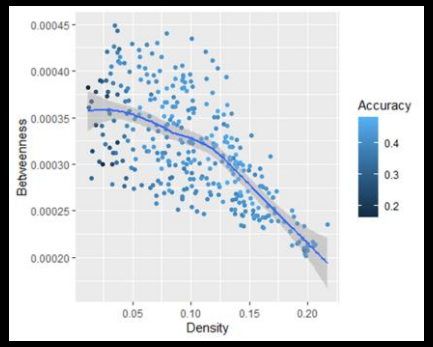


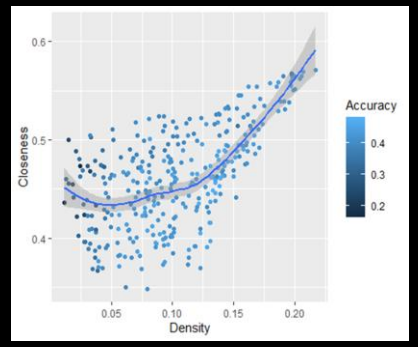


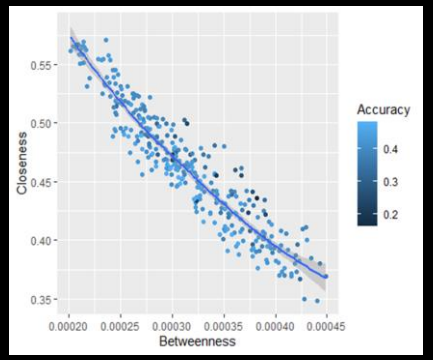
Analysis Metrics

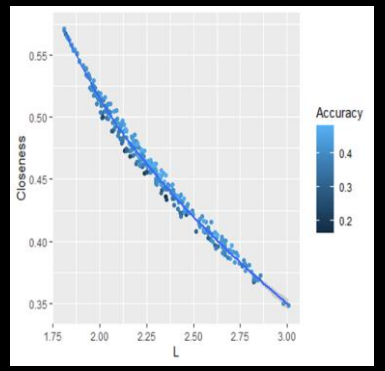


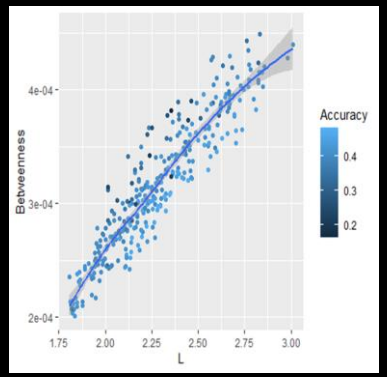












Apply Metrics to Building a Good Model

```
m7 <- lm(Accuracy ~ Layer1 + Layer2 + Layer3 + Layer4 + Layer5 + Layer6, data = acc7)
samples = 500
layers <- sample(4:7, 1, replace=TRUE)
max <- 700 - 50*layers
nums <- sample(10:max, 6*samples, replace=TRUE)
df <- data.frame(matrix(nums, nrow=samples))
p <- predict(m7, df)
```

Layer1	Layer2	Layer3	Layer4	Layer5	Layer6	Pred	Accuracy	L	Density	Betweenness	Closeness
316	329	348	74	52	10	0.4960	0.4606	2.1357	0.1378	0.0003	0.4887
302	253	350	99	44	214	0.4783	0.4704	2.4473	0.1214	0.0003	0.4349
176	327	267	63	61	319	0.4783	0.4587	2.6146	0.0793	0.0004	0.4074

Steps:

1) Use R to generate a linear model
2) Use linear model to predict good classifier architecture
3) Test predicted classifier architecture
4) Go to 1) with more results data (reinforcement learning)

Conclusion

Effectively found a good MLP architecture for classification of Cifar10.

Individual graph metrics are marginally useful.

Using multiple graph metrics provide more insights compared to individual metrics for MLP.

Using random number of layers for MLP to find diverse group of Graph metrics.

Using statistical analysis.

Future Work

Translate more complex Neural Networks into Graphs.

Experiment with ML models to predict NN accuracy from Graph Metrics.

Overcome limitations of MLP.

Thank you