1. Data and Methodology:

Data: CIFAR10, MNIST, iyer.
Methodology: CNN, XGBoost, Random Forest

2. MNIST:

a. MNIST on CNN:

Using the standard LeNet5 as the CNN structure. LeNet5 is one of the earliest pre-trained models proposed by Yann LeCun and others in the year 1998, in the research paper Gradient-Based Learning Applied to Document Recognition.

```
    nn.Conv2d(in_channels=1, out_channels=6, kernel_size=5, stride=1),
    nn.Tanh(),
    nn.AvgPool2d(kernel_size=2),
    nn.Conv2d(in_channels=6, out_channels=16, kernel_size=5, stride=1),
    nn.Tanh(),
    nn.AvgPool2d(kernel_size=2),
    nn.Conv2d(in_channels=16, out_channels=120, kernel_size=5, stride=1),
    nn.Tanh()
)

self.classifier = nn.Sequential(
    nn.Linear(in_features=120, out_features=84),
    nn.Tanh(),
    nn.Linear(in_features=84, out_features=10),
)
```

Layer	# filters / neurons	Filter size	Stride	Size of feature map	Activation function
Input	-	-	-	32 X 32 X 1	
Conv 1	6	5 * 5	1	28 X 28 X 6	tanh
Avg. pooling 1		2 * 2	2	14 X 14 X 6	
Conv 2	16	5 * 5	1	10 X 10 X 16	tanh
Avg. pooling 2		2 * 2	2	5 X 5 X 16	
Conv 3	120	5 * 5	1	120	tanh
Fully Connected 1	-	-	-	84	tanh
Fully Connected 2	-	-	-	10	Softmax

- Then we run this structure 5 times, save each model in the "seed" and use cross-validation to find the best model.
- The learning rate is 0.01, and the epoch is 20.

```
for SEED in range(5):
    best_save_path = SAVE_DIR + "MNIST_CNN_Val_SEED_%d_model"%SEED
    print(torch.load(best_save_path)['val_acc'])
```

```
tensor(98.5333, device='cuda:0')
tensor(98.1167, device='cuda:0')
tensor(98.0667, device='cuda:0')
tensor(97.8333, device='cuda:0')
tensor(98.4500, device='cuda:0')
```

- Since the first model has the best performance. We choose it as the test model.

```
print(metrics.accuracy_score(y_vals,y_preds))
print(metrics.f1_score(y_vals,y_preds,average=None))
print(metrics.roc_auc_score(y_vals_onehot,y_outputs,average=None))
```

```
0.9848
[0.98528666 0.99164835 0.98742747 0.98422091 0.98729029 0.98378983
 0.98591549 0.98242188 0.98303342 0.97590361]
[0.99991923 0.99997287 0.999894   0.99987015 0.99992592 0.99984971
 0.99989979 0.99971084 0.99986771 0.99972332]
```

- We use the accuracy score, F1 score, and AUC score to calculate the performance of the LeNet5 model, and the outcome shows it is very suitable for this MNIST dataset since it is a grayscale handwriting image dataset.

b. MNIST on XGBoost:

- Firstly, we used PCA to change these datas to 10-dimensions:
- With the defaults for XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=100, silent=True, objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gamma=0, min_child_weight=1, max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg_alpha=0, reg_lambda=1, scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=None):
- Then we run this structure 5 times, save each model in the "seed" and use cross-validation to find the best model.

```
SEED:0
Accuracy Score: 0.9238333333333333
F1 Score: 0.9237992129953445
ROC AUC Score: 0.9871481481481481
Average score:0.9449
SEED:1
Accuracy Score: 0.9251666666666667
F1 Score: 0.9251686430168448
ROC AUC Score: 0.9865185185185186
Average score:0.9456
SEED:2
Accuracy Score: 0.926
F1 Score: 0.9259002526114295
ROC AUC Score: 0.9864814814814814
Average score:0.9461
SEED:3
Accuracy Score: 0.9173333333333333
F1 Score: 0.9174651460380686
ROC AUC Score: 0.984888888888889
Average score:0.9399
SEED:4
Accuracy Score: 0.9206666666666666
F1 Score: 0.9206766005158774
ROC AUC Score: 0.9869259259259261
Average score:0.9428
```

- Since the third model has the best performance. We choose it as the test model.

```
SEED:2
Accuracy Score: 0.9237
F1 Score: 0.9236704964124014
ROC AUC Score: 0.9957977542440413
```

- XGBoost also did well, but not as well as CNN with LeNet5 overall

c. MNIST on Random Forest

- Firstly, we used PCA to change these data to 10-dimensions:
- With the random_state = 42 for RandomForestClassifier, we run this structure 5 times, save each model in the "seed" and use cross-validation to find the best model.

```
SEED:0
Accuracy Score: 0.9141666666666667
F1 Score: 0.9139101749279563
ROC AUC Score: 0.9853888888888889
Average score:0.9378
SEED:1
Accuracy Score: 0.913
F1 Score: 0.9128148340747895
ROC AUC Score: 0.9839722222222221
Average score:0.9366
SEED:2
Accuracy Score: 0.9153333333333333
F1 Score: 0.915045437593503
ROC AUC Score: 0.9845462962962962
Average score:0.9383
SEED:3
Accuracy Score: 0.9111666666666667
F1 Score: 0.9111354698260534
ROC AUC Score: 0.983824074074074
Average score:0.9354
SEED:4
Accuracy Score: 0.9061666666666667
F1 Score: 0.9060548821152149
ROC AUC Score: 0.9841018518518518
Average score:0.9321
```

- Since the third model has the best performance. We choose it as the test model.

```
SEED:2
Accuracy Score: 0.9132
F1 Score: 0.913101841458489
ROC AUC Score: 0.9940397885978471
```

- Random Forest is OK to MNIST but is not as good as CNN with LeNet5 and XGBoost.

3. Iyer

a. Iyer on CNN:

We define a new CNN structure:

After turning the hyper-parameter, we got the best CNN model:

```
def __init__(self):
    super(CNN, self).__init__()
    self.conv1 = nn.Conv1d(in_channels=1, out_channels=32,kernel_size=5,stride=1)
    self.relu = nn.ReLU()
    self.bn1 = nn.BatchNorm1d(32)
    self.conv2 = nn.Conv1d(in_channels=32, out_channels=128, kernel_size=5, stride=1)
    self.bn2 = nn.BatchNorm1d(128)
    self.fc1 = nn.Linear(4*128,64)
    self.fc2 = nn.Linear(64,10)

def forward(self, x):
    x = self.relu(self.conv1(x))
    x = self.bn1(x)
    x = self.relu(self.conv2(x))
    x = self.bn2(x)
    x = torch.flatten(x, 1)
    x = self.relu(self.fc1(x))
    x = self.fc2(x)
    return x
```

- The first layer is the input layer with a feature map size of 1*12.
- Then the first convolution layer with 32 filters of size 5*5 and stride is 1. The activation function used at this layer is ReLu. The output feature map is 8*32.
- Then we add a Batch_normalize layer to normalize these datas, this layer will not affect the output channel.
- After this comes the second convolution layer with 32 filters of 5*5 and stride 1. Also, the activation function is ReLu. Now the output size is 4*128.
- The second Batch_normalize layer is added the same as the last one.
- The next is a fully connected layer with 64 neurons that result in the output of 64 values.
- The last layer is the output layer with 10 neurons.

Then we run this structure 5 times, and then use cross-validation (k-fold = 5) to find the best model

- It seems that the fourth model is the best one, so we use it to test data.

```
(80, 10)
0.775
0.7768780525030525
0.9680555555555556
(80, 10)
0.825
0.7977682766200208
0.9694444444444444
(80, 10)
0.8375
0.8292512513904338
0.9777777777777779
(80, 10)
0.775
0.7591071428571429
0.9680555555555557
(80, 10)
0.85
0.851358352778498
0.9833333333333334
```

```
print(metrics.accuracy_score(y_te,y_preds))
print(metrics.f1_score(y_te,y_preds,average='weighted'))
print(metrics.roc_auc_score(y_te_onehot,y_outputs,average='samples',multi_class='ovo'))
```

```
0.8809523809523809
0.8811595728137082
0.9867724867724867
```

- According to the accuracy, f1, and AUC score, CNN could analyze the Iyer data set, but it doesn't have very high accuracy.

b. Iyer on XGBoost:

- With the defaults for XGBClassifie, we run this structure 5 times, save each model in the "k_idx" and use cross-validation to find the best model.

```
K_idx:0
Accuracy Score: 1.0
F1 Score: 1.0
ROC AUC Score: 1.0
Average score:1.0000
K_idx:1
Accuracy Score: 1.0
F1 Score: 1.0
ROC AUC Score: 1.0
Average score:1.0000
K_idx:2
Accuracy Score: 1.0
F1 Score: 1.0
ROC AUC Score: 1.0
Average score:1.0000
K_idx:3
Accuracy Score: 1.0
F1 Score: 1.0
ROC AUC Score: 1.0
Average score:1.0000
K_idx:4
Accuracy Score: 0.8375
F1 Score: 0.8439318885448917
ROC AUC Score: 0.9777777777777779
Average score:0.8864
```

```
K_idx:4
Accuracy Score: 0.8333333333333334
F1 Score: 0.8249482280637297
ROC AUC Score: 0.9761904761904762
```

- Since the first four models all performed well, we randomly selected one of them as the test model.

```
K_idx:0
Accuracy Score: 0.8333333333333334
F1 Score: 0.8179749970163505
ROC AUC Score: 0.9708994708994707
```

- According to the accuracy, f1, and AUC score, XGBoost could just barely analyze the Iyer dataset and is worse than CNN.

c. Iyer on Random Forest:

- With the random_state = 42 for RandomForestClassifier, we run this structure 5 times, save each model in the "seed" and use cross-validation to find the best model.

```
K_index:0,Accuracy Score:80.0000%
K_index:1,Accuracy Score:80.0000%
K_index:2,Accuracy Score:75.0000%
K_index:3,Accuracy Score:77.5000%
K_index:4,Accuracy Score:86.2500%
```

- Since the fifth model has the best performance. We choose it as the test model.

```
print('Accuracy Score:', score)
print('F1 Score:', F1_score)
print('ROC AUC Score:', ROC_AUC_score)
```

```
K_idx:4
Accuracy Score: 0.8333333333333334
F1 Score: 0.8249482280637297
ROC AUC Score: 0.9761904761904762
```

- Random Forest has a similar performance on Iyer with XGBoost.

4. CIFAR10:

a. CIFAR10 on CNN:

We try to use Lenet-5 and edit its structure to fit these 32*32*3 images in CIFAR 10.

```
nn.Conv2d(in_channels=3, out_channels=16, kernel_size=5, stride=1),
nn.Tanh(),
nn.AvgPool2d(kernel_size=2),
nn.Conv2d(in_channels=16, out_channels=32, kernel_size=5, stride=1),
nn.Tanh(),
nn.AvgPool2d(kernel_size=2),
nn.Conv2d(in_channels=32, out_channels=120, kernel_size=5, stride=1),
nn.Tanh()
```

- The first layer is the input layer with a feature map size of 32*32*3.

- Then we have the first convolution layer with 16 filters of size 5*5 and stride is 1. The activation function used at this layer is tanh. The output feature map is 28*28*16.

- Next, we have an average pooling layer with filter size 2*2 and stride 1. The resulting feature map is 14*14*16. Since the pooling layer doesn't affect the number of channels.

- After this comes the second convolution layer with 32 filters of 5*5 and stride 1. Also, the activation function is tanh. Now the output size is 10*10*32.

- Again comes the other average pooling layer of 2*2 with stride 1. As a result, the size of the feature map was reduced to 5*5*32.

- The final pooling layer has 120 filters of 5X5 with stride 1 and activation function tanh. Now the output size is 120.

- The next is a fully connected layer with 84 neurons that result in the output of 84 values and the activation function used here is again tanh.

- The last layer is the output layer with 10 neurons and the Softmax function. The Softmax gives the probability that a data point belongs to a particular class. The highest value is then predicted.

- The learning rate is 0.01, and the epoch is 20.

```
for SEED in range(5):
    best_save_path = SAVE_DIR + "CIFAR10_CNN_Val_SEED_%d_model"%SEED
    print(torch.load(best_save_path)['val_acc'])
```

```
tensor(59.2600, device='cuda:0')
tensor(59.2400, device='cuda:0')
tensor(58.7200, device='cuda:0')
tensor(60.7400, device='cuda:0')
tensor(60.1400, device='cuda:0')
```

However, the outcome shows it has a low accuracy on these five models. So we need to turn the hyper-parameter.

We redesign the CNN model as shown below:

The highest accuracy can improve to near 80%

```python
class CNN5(nn.Module):
    def __init__(self):
        super(CNN5, self).__init__()
        self.conv1 = nn.Sequential(
                        nn.Conv2d(3,64,3,1,1),
                        nn.ReLU(),
                        nn.BatchNorm2d(64))
        self.pool =nn.AvgPool2d(kernel_size=2,stride=2
        self.conv2 = nn.Sequential(
                        nn.Conv2d(64,128,3,1,1),
                        nn.ReLU(),
                        nn.BatchNorm2d(128))
        self.conv3 = nn.Sequential(
                        nn.Conv2d(128,128,3,1,1),
                        nn.ReLU(),
                        nn.BatchNorm2d(128))
        self.classifier = nn.Sequential(
                            nn.Linear(128*4*4, 512),
                            nn.ReLU(),
                            nn.BatchNorm1d(512),
                            nn.Dropout(0.2),
                            nn.Linear(512, 512),
                            nn.ReLU(),
                            nn.BatchNorm1d(512),
                            nn.Linear(512,10)
                            )
    def forward(self,x):
        x = self.pool(self.conv1(x))
        x = self.pool(self.conv2(x))
        x = self.pool(self.conv3(x))
        x = x.view((-1,128*4*4))
        return self.classifier(x)
```

```python
print(metrics.accuracy_score(y_vals,y_preds))
print(metrics.f1_score(y_vals,y_preds,average='weighted'))
print(metrics.roc_auc_score(y_vals_onehot,y_outputs,average=None))
```

```
0.782
0.7816714609262266
[0.97953433 0.99224767 0.95125511 0.93657278 0.97004967 0.95776578
 0.98333356 0.98494644 0.98996589 0.990095  ]
```

b. CIFAR10 on XGBoost:

- With the defaults for XGBClassifie, we run this structure 5 times, save each model in the “seed” and use cross-validation to find the best model.

```
SEED:0
Accuracy Score: 0.4574
F1 Score: 0.4542353443341773
ROC AUC Score: 0.8286
Average score:0.5801
SEED:1
Accuracy Score: 0.4516
F1 Score: 0.44935228230665664
ROC AUC Score: 0.8334666666666668
Average score:0.5781
SEED:2
Accuracy Score: 0.4536
F1 Score: 0.451574959849309
ROC AUC Score: 0.8348222222222222
Average score:0.5800
SEED:3
Accuracy Score: 0.4548
F1 Score: 0.4518173426238597
ROC AUC Score: 0.8330888888888889
Average score:0.5799
SEED:4
Accuracy Score: 0.4564
F1 Score: 0.4540931025325998
ROC AUC Score: 0.8329111111111112
Average score:0.5811
```

```
SEED:4
Accuracy Score: 0.4568
F1 Score: 0.4541473383937634
ROC AUC Score: 0.8585634666666667
```

The outcome shows it has a low accuracy on these five models.

c. CIFAR10 on Random Forest

- With the random_state = 42 for RandomForestClassifier, we run this structure 5 times, save each model in the “seed” and use cross-validation to find the best model.

```
SEED:0,Accuracy Score:44.0200%
SEED:1,Accuracy Score:44.2200%
SEED:2,Accuracy Score:44.6800%
SEED:3,Accuracy Score:45.1200%
SEED:4,Accuracy Score:44.8600%
```

The outcome shows it has a low accuracy on these five models.