# ECE 460J Data Science Lab 4 Report:

Authors: William Wooten, Aksheeth Ilamparithi, Laith Altarabishi

Department of Electrical and Computer Engineering, The University of Texas at Austin

## Problem 0:

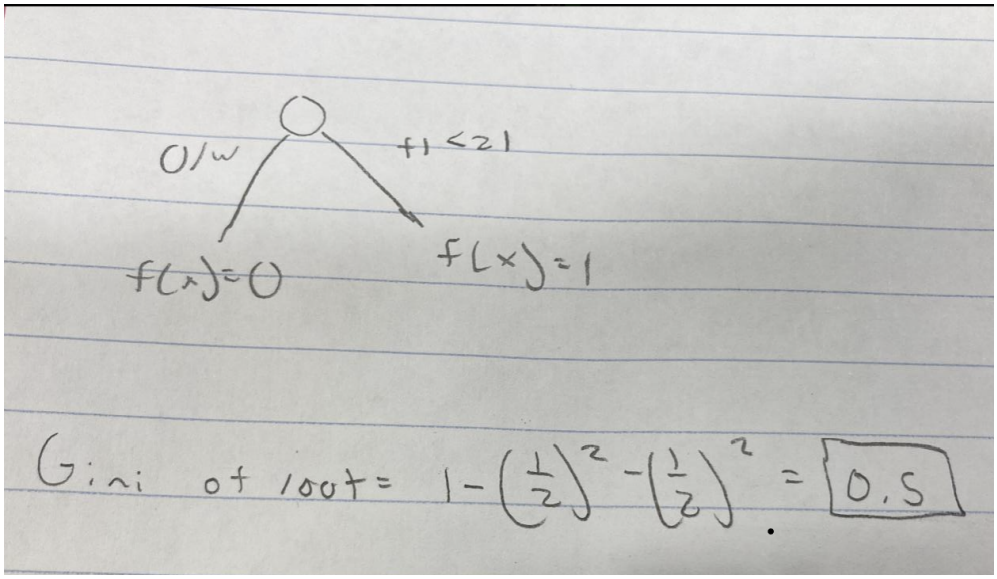
For problem 0, we first compute the probabilities for logistic regression for advertisement clicking:

```
import numpy as np
w = [1, -30, 3]
X = [[20, 0, 0],[23, 1, 1]]
y = [1, 0]


# Turns score x into a probability
def sigmoid(x):
  return 1 / (1 + np.exp(-x))

# Computes the likelihood of success for each customer
def binary_log_reg(w, x):
    # Make x into a vertical col
    x = np.array(x).transpose()
    w = np.array(w)
    score = np.matmul(w, x)
    return sigmoid(score)


print("The probability the first customer clicks according to the model is: ",
binary_log_reg(w, X[0]))
print("The probability the second customer clicks according to the model is: ",
binary_log_reg(w, X[1]))
```

Then we compute the cross entropy loss for the regression:

```
# Compute the cross entropy
y =  [binary_log_reg(w, X[0]), binary_log_reg(w, X[1])]
actual = [1, 0]
c_entropy = 0
for i in range(len(y)):
  c_entropy += (actual[i] * np.log2(y[i]))
print("The cross entropy loss of the regression is ", - c_entropy)
```

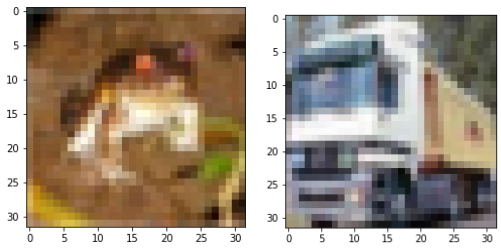
Then we design a decision stump that split on the first feature, and computed the gini impurity.



## Problem 1:

To start problem one we imported the CIFAR-10 dataset and printed out some of the images

```
from sklearn.datasets import fetch_openml
from matplotlib import pyplot
import numpy as np

dataset, target = fetch_openml(name = 'CIFAR_10_Small', as_frame = False, return_X_y= True)


#dataset.feature_names
from matplotlib import pyplot as plt
from sklearn.model_selection import train_test_split
#dataset[0].shape
#dataset.data['a0'].to_numpy()
R = dataset[0][0:1024].reshape(32,32)/255.0
G = dataset[0][1024:2048].reshape(32,32)/255.0
B = dataset[0][2048:].reshape(32,32)/255.0

img = np.dstack((R,G,B))

plt.imshow(img)

R = dataset[1][0:1024].reshape(32,32)/255.0
G = dataset[1][1024:2048].reshape(32,32)/255.0
B = dataset[1][2048:].reshape(32,32)/255.0

img = np.dstack((R,G,B))
plt.imshow(img)
```



This was the output!

We did a ¾ -¼ training split, and performed multiclass logistic regression.

```
X_train, X_test, y_train, y_test = train_test_split(
    dataset, target, stratify=target, random_state=0, train_size=15000, test_size=5000
)
```

```
best_score = 0
#coeff = [0.001, 0.01, 0.1, 1, 10, 100, 1000]
model = LogisticRegression(multi_class="multinomial", solver = "saga")
grid_params = {'penalty': ['l1','l2', 'elasticnet'], 'C': [0.001, 0.01,0.1,1, 10]} # consider all
penalties and find the best learning rate amongst these orders of magnitude
             #adjust tolerance
grid_search = GridSearchCV(estimator=model, param_grid=grid_params, scoring='accuracy')
grid_search.fit(X_train,y_train)

print("tuned hyperparameters : ",grid_search.best_params_)
print("accuracy :",grid_search.best_score_)
```

```
from sklearn.linear_model import LogisticRegressionCV, LogisticRegression
from sklearn.metrics import log_loss
tuned_model = LogisticRegression(multi_class="multinomial", solver = "saga", C = 1, penalty = 'l2')
tuned_model.fit(X_train, y_train)

predictions_prob = tuned_model.predict_proba(X_test)
print(log_loss(y_test, predictions_prob))
print("accuracy :", tuned_model.score(X_train, y_train))
acc = cross_val_score(tuned_model, dataset, target) #USE INSTEAD OF CUSTOM RSME FUNCTION
acc
```

With the following outputs:

```
array([0.3555 , 0.352  , 0.37025, 0.37625, 0.35725])
```

```
predictions_prob = tuned_model.predict_proba(X_test)
print('The test loss is ' + str(log_loss(y_test, predictions_prob)))
predictions_prob2 = tuned_model.predict_proba(X_train)
print('The training loss is ' + str(log_loss(y_train, predictions_prob2)))
```

```
The test loss is 1.9888273427662462
The training loss is 1.2861548259953623
```

We computed the above for the losses.

## Problem 2

Like in problem 1, we used the fetch openml command to import the MNIST dataset.

```
from sklearn.datasets import fetch_openml
from matplotlib import pyplot
import numpy as np

dataset, target = fetch_openml(name = 'mnist_784', as_frame = False, return_X_y= True)
```

Then we split the test and training sets, and ran multiclass logistic regression.

```
from matplotlib import pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score, GridSearchCV
from sklearn.preprocessing import StandardScaler

trans = StandardScaler()
dataset = trans.fit_transform(dataset) #normalize data to help with convergence

X_train, X_test, y_train, y_test = train_test_split(
    dataset, target, stratify=target, random_state=0, train_size=int(len(dataset)*6/7),
test_size=int(len(dataset)/7))

# model = LogisticRegression(multi_class="multinomial", solver = "saga")
# grid_params = {'penalty': ['l1','l2', 'elasticnet'], 'C': [0.01,0.1,1]} #C use more granular
values
# grid_search = GridSearchCV(estimator=model, param_grid=grid_params, scoring='accuracy',
#                        cv=6)
# grid_search.fit(X_train,y_train)

# print("tuned hyperparameters : ",grid_search.best_params_)
# print("accuracy :",grid_search.best_score_)

from sklearn.model_selection import RandomizedSearchCV

#more optimization attempts
model = LogisticRegression(multi_class="multinomial", solver = "saga", penalty = 'l2', max_iter =
1000, C = .895, n_jobs = -1)
# model_params = {'C': [float(x) for x in np.linspace(0.8, 1.0)], 'tol': [float(x) for x in
np.linspace(1e-6, 1e-4)]} #C use more granular values
# random_search = RandomizedSearchCV(model, model_params, n_iter = 25, scoring='accuracy')
model.fit(X_train,y_train)

# print("tuned hyperparameters : ",random_search.best_params_)
print("accuracy :", model.score(X_train, y_train))
```

With the following output:

```
accuracy : 0.9376166666666667
```

we had the following results found from GridSearch for hyperparameters
tuned hyperparameters : {'C': 1, 'penalty': 'l2'} accuracy : 0.9171833333333334
Results found from manual search + random search for hyperparameters

tuned hyperparameters : "multi_class="multinomial", solver = "saga", penalty = 'l2', max_iter = 1000, C = .895" accuracy : 0.9376333333333333

Here we computed the training and the test loss:

```
from sklearn.metrics import log_loss
# predictions = tuned_model.predict(X_test)
predictions_prob = model.predict_proba(X_test)
print(log_loss(y_test, predictions_prob))
print(log_loss(y_train, model.predict_proba(X_train)))
```

```
0.2859308493055544
0.22525112343899228
```

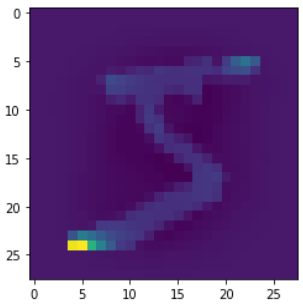
Here were the training and test losses and code to compute them:

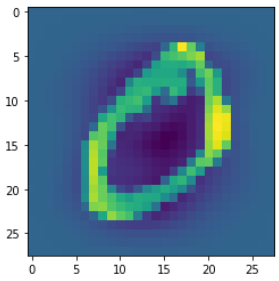
```
#hyperparameter tuning with manual inspection + randomCV
model2 = LogisticRegression(multi_class="multinomial", solver = "saga", penalty = 'l1', max_iter
= 1000, C = 1.1, tol = 0.003)
model2.fit(X_train, y_train)

print("accuracy :", model2.score(X_train, y_train)) #0.9293333333333333 tol = .003 c= 1
```

```
accuracy : 0.9293333333333333
```

Lastly we visualized the coefficients:

```
coefs = model2.coef_
indexes = [i for i in range(10)]
for index in indexes:
    img = img_gen(dataset, index)
    plt.imshow(img)
    plt.figure()
```




...

## Problem 3:

For problem 3 we once again examine the MNIST dataset. We created a random forest modela dn computed its accuracy.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RandomizedSearchCV

#find best hyperparams here
# forest_params = {'n_estimators': [int(x) for x in np.linspace(650, 750)], 'max_depth' : [int(x) for x in np.linspace(1,5,3)]}
forest_model = RandomForestClassifier(n_estimators = 709, max_depth = 25)
# random_search = RandomizedSearchCV(forest_model, forest_params, n_iter = 5,cv = 3, scoring='accuracy')
forest_model.fit(X_train, y_train)

print("accuracy :", forest_model.score(X_test, y_test)) #accuracy : 0.9726 estimators = 300
```

With output:

accuracy : 0.9719

---

Then we computed the cross validation score, and relative accuracy

```
acc = cross_val_score(forest_model, dataset, target)
acc
```

```
array([0.97035714, 0.96871429, 0.967     , 0.96728571, 0.97371429])
```

```
sum(acc)/len(acc)
```

```
0.9694142857142858
```

We found that our best random forest model had the following best hyperparameters: n_estimators = 709, max_depth = 25 Best score: 0.9694142857142858 LR vs RF: RF was more accurate with 96.9% accuracy vs 93.7%

Second we created a gradient boosting model to do the same.

```
from catboost import CatBoostClassifier
```

```
accuracy : 0.973
```

```
acc = cross_val_score(forest_model, dataset, target)
sum(acc)/len(acc)
```

```
0.9696857142857143
```

```
cat_model.get_all_params()
```

```
{'nan_mode': 'Min',
 'eval_metric': 'MultiClass',
 'iterations': 1000,
 'sampling_frequency': 'PerTree',
 'leaf_estimation_method': 'Newton',
 'grow_policy': 'SymmetricTree',
 'penalties_coefficient': 1,
 'boosting_type': 'Plain',
 'model_shrink_mode': 'Constant',
 'feature_border_type': 'GreedyLogSum',
 'bayesian_matrix_reg': 0.10000000149011612,
 'eval_fraction': 0,
 'force_unit_auto_pair_weights': False,
 'l2_leaf_reg': 3,
 'random_strength': 1,
 'rsm': 1,
 'boost_from_average': False,
 'model_size_reg': 0.5,
 'pool_metainfo_options': {'tags': {}},
 'use_best_model': False,
 'class_names': ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9'],
 'random_seed': 0,
 'depth': 6,
 'posterior_sampling': False,
 'border_count': 254,
 'bagging_temperature': 1,
 'classes_count': 0,
 'auto_class_weights': 'None',
 'sparse_features_conflict_fraction': 0,
 'leaf_estimation_backtracking': 'AnyImprovement',
 'best_model_min_trees': 1,
 'model_shrink_rate': 0,
 'min_data_in_leaf': 1,
 'loss_function': 'MultiClass',
 'learning_rate': 0.0975010022521019,
 'score_function': 'Cosine',
 'task_type': 'CPU',
 'leaf_estimation_iterations': 1,
 'bootstrap_type': 'Bayesian',
 'max_leaves': 64}
```

The above output represent the optimal hyperparameters for our gradient boosting model.

## Problem 4:

We first send a random forest classifier on Cifar 10.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RandomizedSearchCV

forest_model = RandomForestClassifier(n_estimators = 700, max_depth = 30)
# random_search = RandomizedSearchCV(forest_model, forest_params, cv = 3, scoring='accuracy')
```

```
forest_model.fit(X_train, y_train)

print("accuracy :", forest_m
```

## accuracy : .623

```
acc = cross_val_score(forest_model, dataset, target)
sum(acc)/len(acc)
```

### .612

We found through experimentation that n_estimators and max_Depths were the most impactful hyperparameters, so we manually messed around with those values to find something that fit.

### Next, we used gradient boosting to do the same

```
from catboost import CatBoostClassifier

cat_params = {'eta': np.linspace(.1,1.5), 'max_depth': [int(x) for x in np.linspace(0,100,20)]}
cat_model = CatBoostClassifier()
# random_search = RandomizedSearchCV(cat_model, cat_params, n_iter = 5,cv = 3,
scoring='accuracy')
cat_model.fit(X_train, y_train)

print("accuracy :", cat_model.score(X_test, y_test))
```

```
cat_model.get_all_params()
```

```
acc = cross_val_score(forest_model, dataset, target)
sum(acc)/len(acc)
```

## .568

## Problem 5:

After running through the PyTorch and CNN tutorial, we did the following to construct a CNN and compute accuracy scores for the MNIST dataset.

First, we created datasets and dataloaders to make evaluation of the data easier during the forward steps and backpropogations.

```
# Dataset and Dataloader make the algorithm much simpler
from torch.utils.data import Dataset
from torchvision import datasets
from torchvision.transforms import ToTensor
import matplotlib.pyplot as plt

training_data = datasets.MNIST(
    root = 'data',
    train = True,
    download = True,
    transform = ToTensor()
)

test_data = datasets.MNIST(
    root = "data",
    train = False,
    download = True,
    transform = ToTensor()
)
```

```
# data loader helps partition the dataset into batches

from torch.utils.data import DataLoader

train_dataloader = DataLoader(training_data, batch_size = 100, shuffle = True)
test_dataloader = DataLoader(test_data, batch_size = 100, shuffle = True)
```

Then we created a neural network class from the nn.module framework, and defined a forward step, and initialized the convolutional layers.

```
# Two layer neural net using nn.Module
import torch.nn as nn
import torch.nn.functional as F
class Model(nn.Module):
  def __init__(self):
    super().__init__()

    self.conv1 = nn.Sequential(
            nn.Conv2d(
                in_channels=1,
                out_channels=16,
```

```
                kernel_size=5,
                stride=1,
                padding=2,
            ),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2),
        )
    self.conv2 = nn.Sequential(
            nn.Conv2d(16, 32, 5, 1, 2),
            nn.ReLU(),
            nn.MaxPool2d(2),
        )
    # fully connected layer, output 10 classes
    self.out = nn.Linear(32 * 7 * 7, 10)
  def forward(self, x):
    x = F.relu(self.conv1(x)) # perform the convolution, then the activation function on the
layer
    x = F.relu(self.conv2(x))

    # Flatten the out
    x= x.view(x.size(0), -1)
    output = self.out(x)
    return output
```

Next, we defined a train and a test function.

```
def train(dataloader, model, loss_func, optimizer):
  size = len(dataloader.dataset)
  model.train()

  # Train the model on every batch; perform forward then backward
  for batch, (X, y) in enumerate(dataloader):
    #X, y = X.to(device), y.to(device) # ??

    # Compute the prediction error
    pred = model(X)
    loss = loss_func(pred, y)

    # Clear the previous gradients
    optimizer.zero_grad()

    # computes graidents
    loss.backward()

    # Applies gradients
    optimizer.step()

    # Every 100 batches, print the loss
    if batch % 100 == 0:
        loss, current_batch = loss.item(), batch
```

```
# Determine the accuracy of the neural net
def test(dataloader, model, loss_func):
  size = len(dataloader.dataset)
  num_batches = len(dataloader)
  model.eval()
  test_loss, correct = 0, 0
  with torch.no_grad():
    for X, y in dataloader:
      pred = model(X)
      test_loss += loss_func(pred, y).item() # Add the loss amount
      correct += (pred.argmax(1) == y).type(torch.float).sum().item()

  test_loss /= num_batches
  correct /= size
  print("The test error: ")
  print("\t Accuracy: ", 100 * correct)
  print("\t Avg Loss: ", test_loss)
```

Lastly, we trained the network and computed its accuracy:

```
model = Model()
loss_func = nn.CrossEntropyLoss() # Loss function

optimizer2 = optim.Adam(model.parameters(), lr = 0.001)
for i in range(10):
  train(train_dataloader, model, loss_func, optimizer2)
```

```
For the batch 0 the loss is 2.242325782775879
For the batch 10000 the loss is 2.211134672164917
For the batch 20000 the loss is 2.1764798164367676
For the batch 30000 the loss is 2.141780138015747
For the batch 40000 the loss is 2.1108481884002686
For the batch 50000 the loss is 2.1034765243530273
For the batch 0 the loss is 2.034909725189209
For the batch 10000 the loss is 1.9596937894821167
For the batch 20000 the loss is 1.778715968132019
For the batch 30000 the loss is 1.659938931465149
For the batch 40000 the loss is 1.5504835844039917
For the batch 50000 the loss is 1.342168927192688
For the batch 0 the loss is 1.2801090478897095
For the batch 10000 the loss is 1.109608769416809
For the batch 20000 the loss is 0.9837875366210938
For the batch 30000 the loss is 0.8924985527992249
…
For the batch 0 the loss is 0.232575461268425
For the batch 10000 the loss is 0.1689080446958542
For the batch 20000 the loss is 0.27062997221946716
For the batch 30000 the loss is 0.287861168384552
For the batch 40000 the loss is 0.31047645211121979
For the batch 50000 the loss is 0.24618534743785858
```

```
test(test_dataloader,·model,·loss_func)
```

The test error:

Accuracy: 98.95

Avg Loss: 0.030895500187034484

Avg Loss: 0.030895500187034484

---

## Problem 6:

## First we imported the CIFAR 10 as dataset and dataloaders

## Ran out of time formatting; please see the attach .ipynb if insufficient.

```
# Load Cifar10 as datasets
from torch.utils.data import Dataset
from torchvision import datasets
from torchvision.transforms import ToTensor
import matplotlib.pyplot as plt

training_data = datasets.CIFAR10(
    root = 'data',
    train = True,
    download = True,
    transform = ToTensor()
)

test_data = datasets.CIFAR10(
    root = "data",
    train = False,
    download = True,
    transform = ToTensor()
)
```

## Then we created several different models, with different layers and convolutional filters

```
# Create dataloaders from them

from torch.utils.data import DataLoader

train_dataloader = DataLoader(training_data, batch_size = 100, shuffle = True)
```

```
test_dataloader = DataLoader(test_data, batch_size = 100, shuffle = True)
# Create a basic neural net to compute results

import torch.nn as nn
import torch.nn.functional as F

class CNN_1(nn.Module):
  def __init__(self):
    super().__init__()
    self.conv1 = nn.Conv2d(3, 6, 3)
    self.pool = nn.MaxPool2d(2, 2)
    self.conv2 = nn.Conv2d(6, 16, 3)
    self.l1 = nn.Linear(16 * 5 * 5, 120)
    self.l2 = nn.Linear(120, 84)
    self.l3 = nn.Linear(84, 10)
    # fully connected layer, output 10 classes

  def forward(self, x):
    x = self.pool(F.relu(self.conv1(x))) # perform the convolution, then the
activation function on the layer
    x = self.pool(F.relu(self.conv2(x)))
    x = torch.flatten(x,1)
    x = F.relu(self.l1(x))
    x = F.relu(self.l2(x))
    x = self.l3(x)
    # Flatten the out
    return x
```

```
# Create a basic neural net to compute results

import torch.nn as nn
import torch.nn.functional as F

class CNN_1(nn.Module):
  def __init__(self):
    super().__init__()
    self.conv1 = nn.Conv2d(3, 6, 3)
    self.pool = nn.MaxPool2d(2, 2)
    self.conv2 = nn.Conv2d(6, 16, 3)
    self.l1 = nn.Linear(16 * 5 * 5, 120)
    self.l2 = nn.Linear(120, 84)
```

```
    self.l3 = nn.Linear(84, 10)
    # fully connected layer, output 10 classes

  def forward(self, x):
    x = self.pool(F.relu(self.conv1(x))) # perform the convolution, then the
activation function on the layer
    x = self.pool(F.relu(self.conv2(x)))
    x = torch.flatten(x,1)
    x = F.relu(self.l1(x))
    x = F.relu(self.l2(x))
    x = self.l3(x)
    # Flatten the out
    return x
```

```
class CNN_2(nn.Module):
  def __init__(self):
    super().__init__()
    self.conv1 = nn.Sequential(
        nn.Conv2d(in_channels=3, out_channels=16, kernel_size=3, stride=1,
padding=1),
        nn.ReLU(),
        nn.MaxPool2d(2)
    )
    self.conv2 = nn.Sequential(
        nn.Conv2d(in_channels=16, out_channels=32, kernel_size=3, stride=1,
padding=1),
        nn.ReLU(),
        nn.MaxPool2d(2)
    )
    self.out = nn.Linear(32 * 8 * 8, 10)

  def forward(self, x):
    x = F.relu(self.conv1(x)) # perform the convolution
    x = F.relu(self.conv2(x))
    x= x.view(x.size(0), -1)
    output = self.out(x)
    # Flatten the out
    return output
```

```
class CNN_3(nn.Module):
  def __init__(self):
    super().__init__()
    self.conv1 = nn.Sequential(
```

```
        nn.Conv2d(in_channels=3, out_channels=16, kernel_size=3, stride=1,
padding=1),
        nn.ReLU(),
        nn.MaxPool2d(2)
    )
    self.conv2 = nn.Sequential(
        nn.Conv2d(in_channels=16, out_channels=32, kernel_size=3,
stride=1, padding=1),
        nn.ReLU(),
        nn.MaxPool2d(2)
    )
    self.conv3 = nn.Sequential(
        nn.Conv2d(in_channels = 32, out_channels = 64, kernel_size = 3,
stride = 1, padding = 1),
        nn.ReLU(),
        nn.MaxPool2d(2)
        )
    self.out = nn.Linear(64 * 4 * 4, 10)

  def forward(self, x):
    x = F.relu(self.conv1(x)) # perform the convolution
    x = F.relu(self.conv2(x))
    x = F.relu(self.conv3(x))
    x= x.view(x.size(0), -1)
    output = self.out(x)
    # Flatten the out
    return output
```

```
# Determine the accuracy of the neural net
import torch
def test(dataloader, model, loss_func):
  size = len(dataloader.dataset)
  num_batches = len(dataloader)
  model.eval()
  test_loss, correct = 0, 0
  with torch.no_grad():
    for X, y in dataloader:
      pred = model(X)
      test_loss += loss_func(pred, y).item() # Add the loss amount
      correct += (pred.argmax(1) == y).type(torch.float).sum().item()
```

```
  test_loss /= num_batches
  correct /= size
  print("The test error: ")
  print("\t Accuracy: ", 100 * correct)
  print("\t Avg Loss: ", test_loss)
```

```
# create models
model1 = CNN_1()

# create an optimizer, ADAM first
optimizer = torch.optim.Adam(model1.parameters(), lr = .001)

# create a loss function
loss_func = nn.CrossEntropyLoss()

# Train the nn
epochs = 10

for epoch in range(epochs):
  train(train_dataloader, model1, loss_func, optimizer)
```

```
# Test results for model 1, using adam optimizer
test(test_dataloader, model1, loss_func)
```

```
The test error:
       Accuracy:  55.730000000000004
       Avg Loss:  1.2278882354497909
```

```
#train and test model 2
model2 = CNN_2() #cnn 2 is a deeper extension of cnn1 with an additional
convolutional layer
optimizer2 = torch.optim.Adam(model2.parameters(), lr = .001)
```

```
for epoch in range(epochs):
  train(train_dataloader, model2, loss_func, optimizer)
```

```
test(test_dataloader, model2, loss_func)
```

```
lr = [.0001, .001, .01]
momentums = [.7,.8,.9]

outputs = []
for l in lr:
  # create an optimizer, ADAM first
  optimizer = torch.optim.Adam(model1.parameters(), lr = l)
  for epoch in range(epochs):
    train(train_dataloader, model1, loss_func, optimizer)
  test(test_dataloader, model1, loss_func)
```

```
The test error:
       Accuracy:  59.550000000000004
       Avg Loss:  1.1599075984954834
The test error:
       Accuracy:  59.61
       Avg Loss:  1.1560079091787339
The test error:
       Accuracy:  54.779999999999994
       Avg Loss:  1.2966955476999282
```

```
for m in momentums:
  # create an optimizer, ADAM first
  optimizer = torch.optim.SGD(model1.parameters(), lr=0.01, momentum=m)
  # optimizer = torch.optim.Adam(model1.parameters(), lr = .001, momentum
= m)
  for epoch in range(epochs):
    train(train_dataloader, model1, loss_func, optimizer)
```

```
  test(test_dataloader, model1, loss_func)
```

```
The test error:
        Accuracy:  57.85
        Avg Loss:  1.232039583325386
The test error:
        Accuracy:  57.97
        Avg Loss:  1.243187518119812
The test error:
        Accuracy:  56.54
        Avg Loss:  1.3101933073997498
```

---

Again, momentum does effect our accuracy greatly. Too much/little momentum will give us poor accuracy.