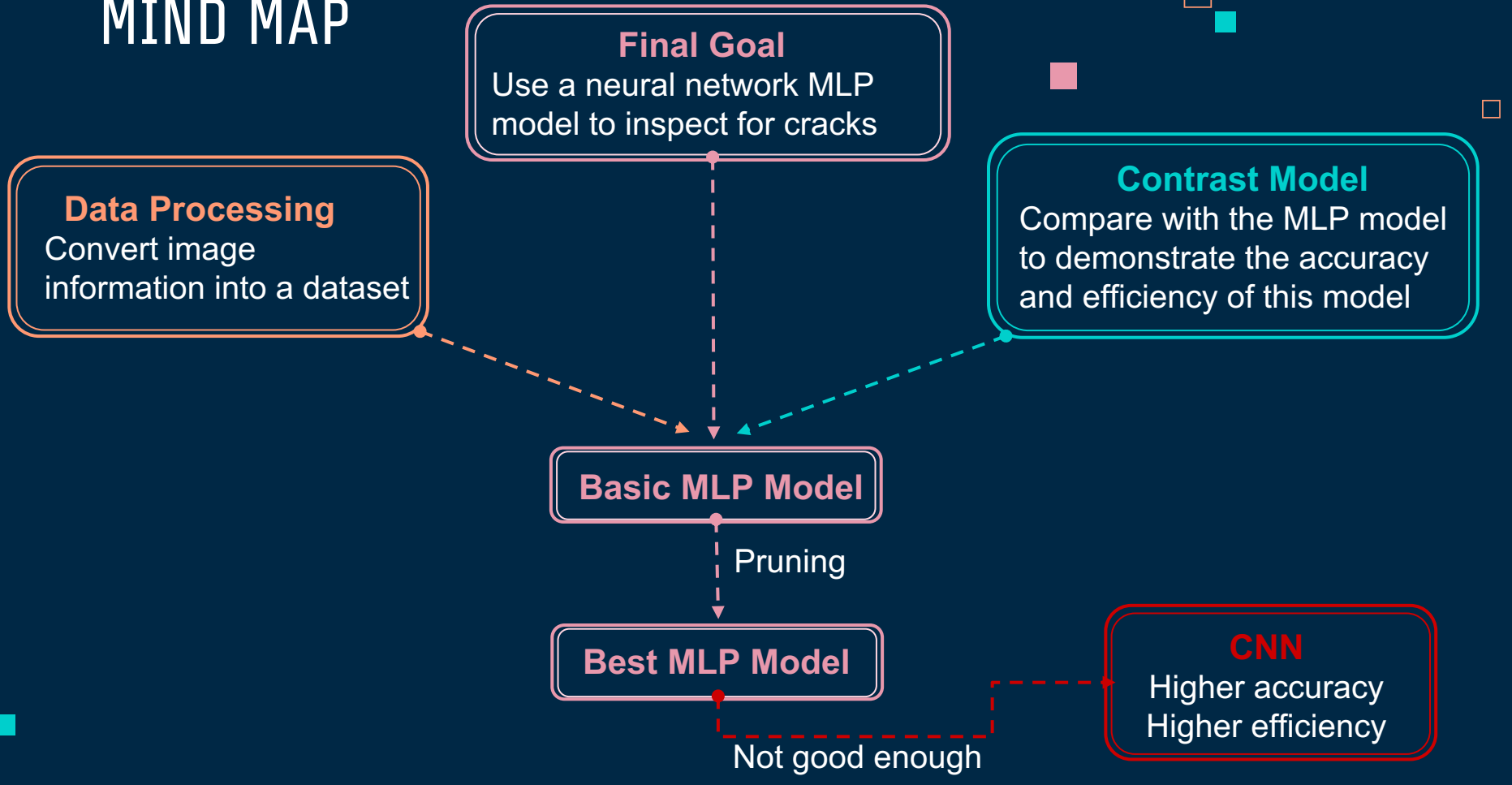
# INFO 6105 FINAL PROJECT

# Visual Crack Inspection and Detection

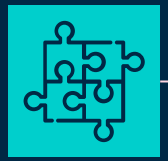
**Made by Haiyan Zhao & Runhan Shi**

# MIND MAP


Final Goal
Use a neural network MLP model to inspect for cracks
Data Processing
Convert image information into a dataset
Contrast Model
Compare with the MLP model to demonstrate the accuracy and efficiency of this model
Basic MLP Model
Pruning
Best MLP Model
Not good enough
CNN
Higher accuracy
Higher efficiency

# Overview

01 Data Processing Tricks



02 Contrast Model

03 Neutral Network

04 Conclusions and Implications

# Data Processing Tricks

01

# Data Processing Tricks

154587

(32107, 227, 227, 3)

**huge dataset**

# Data Processing Tricks (1)

- **'dtype=np.float32'**
- We specify the data type of the array elements as **32-bit** floating point numbers instead of the default 64-bit floating point numbers
- Significantly reduce memory consumption; improve computational efficiency

```
X.shape, y.shape, X.dtype
```

```
((32107, 154587), (32107,), dtype('float32'))
```

```
import numpy as np
import pandas as pd
import os
#from scipy import ndimage #This package contains various functions for mul
import cv2
import scipy
import matplotlib.pyplot as plt
import sys
from tqdm.auto import trange

num_px = 227

def extrcatFeaturesAndLabels(dir, impg_dataset):
  X = np.zeros((len(impg_dataset), num_px*num_px*3), dtype=np.float32)
  y = np.zeros((len(impg_dataset)), dtype=np.float32)

  for i in trange(0,len(impg_dataset)) :
    #Read an image from a file as an array.
    #The different colour bands/channels are stored
    #in the third dimension, such that a
    #grey-image is MxN, an RGB-image MxNx3
    #image = np.array(ndimage.imread(dir + impg_dataset[i]))
    # image = cv2.imread(os.path.join(dir, impg_dataset[i]))
    file_path = os.path.join(dir, impg_dataset[i])
    image = cv2.imread(file_path)
    if image is None:
      print(f"Warning: Unable to read image '{file_path}'. Skipping...")
      continue
    # Resize the image. Size of the output image (height, width)
    image = cv2.resize(image, (num_px, num_px))
```

# Data Processing Tricks (2)

- **np.savez_compressed("data.npz", X = X, y = y)**
- We save multiple NumPy arrays to a single file in **a compressed format**
- Improve I/O efficiency when loading the data back into memory for later use

```
np.savez_compressed("data.npz", X = X, y = y)
```

```
import numpy as np
from sklearn.model_selection import train_test_split

full_data = np.load("data.npz")
X = full_data["X"]
y = full_data["y"]
```

# Data Processing Tricks (3)

- **using trange from the tqdm library**
- Enhance the user experience when running loops by providing a real-time, visually engaging progress bar with estimated completion times

```
from scipy import ndimage
X, y = extrcatFeaturesAndLabels(data_dir, data_imgs)

print('Data set - Rows: %d, columns: %d' % (X.shape[0], X.shape[1]))
print('Label set - Rows: %d' % (y.shape[0]))
print('Number of positive samples: %d' % (np.sum(y == 1)))
```

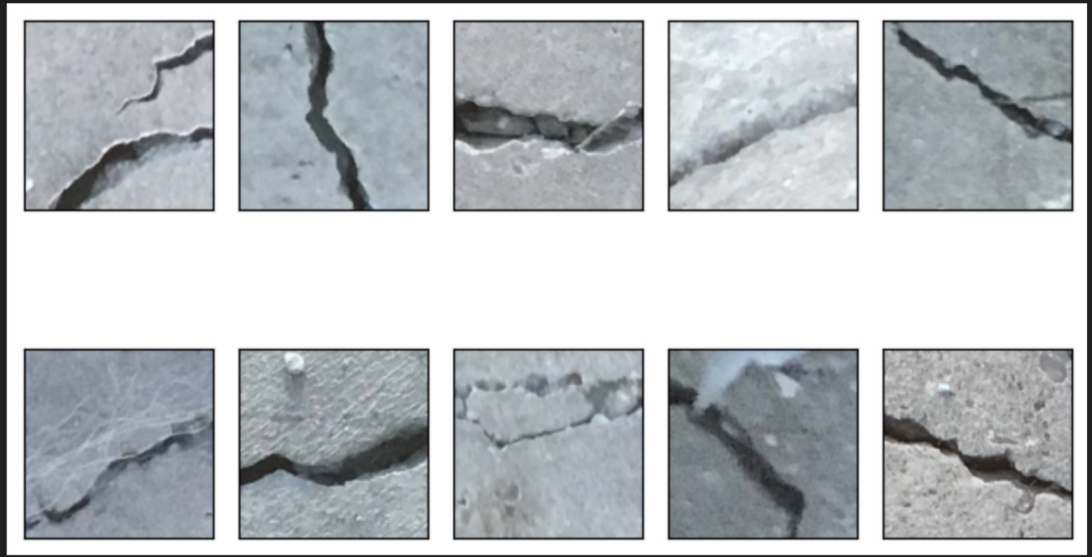
```
100%|██████████| 32107/32107 [22:01<00:00, 24.29it/s]
Data set - Rows: 32107, columns: 154587
```

```
import numpy as np
import pandas as pd
import os
#from scipy import ndimage #This package contains various functions for mul
import cv2
import scipy
import matplotlib.pyplot as plt
import sys
from tqdm.auto import trange

num_px = 227

def extrcatFeaturesAndLabels(dir, impg_dataset):
  X = np.zeros((len(impg_dataset), num_px*num_px*3), dtype=np.float32)
  y = np.zeros((len(impg_dataset)), dtype=np.float32)

  for i in trange(0,len(impg_dataset)) :
    #Read an image from a file as an array.
    #The different colour bands/channels are stored
    #in the third dimension, such that a
    #grey-image is MxN, an RGB-image MxNx3
    #image = np.array(ndimage.imread(dir + impg_dataset[i]))
    # image = cv2.imread(os.path.join(dir, impg_dataset[i]))
    file_path = os.path.join(dir, impg_dataset[i])
    image = cv2.imread(file_path)
    if image is None:
      print(f"Warning: Unable to read image '{file_path}'. Skipping...")
      continue
    # Resize the image. Size of the output image (height, width)
    image = cv2.resize(image, (num_px, num_px))
```

# 10 different images with Cracks

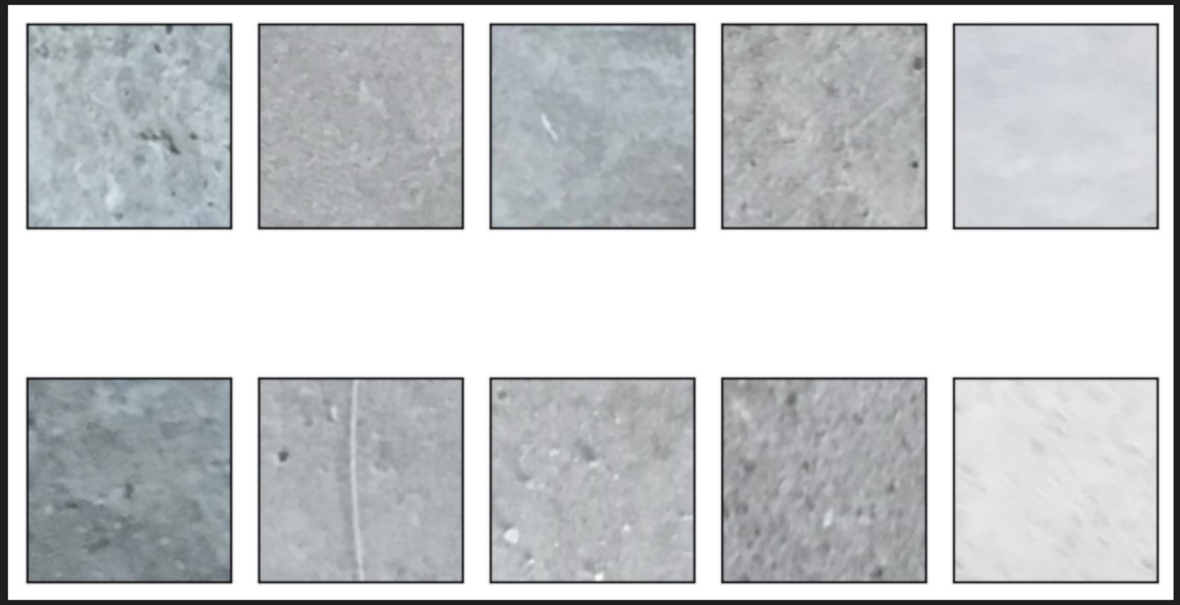
```python
import matplotlib.pyplot as plt

fig, ax = plt.subplots(nrows=2, ncols=5, sharex=True, sharey=True,)
ax = ax.flatten()
for i in range(10):
    img = X_train[y_train == 1][i].reshape(num_px, num_px, 3)
    ax[i].imshow(img)

ax[0].set_xticks([])
ax[0].set_yticks([])
plt.tight_layout()
plt.show()
```

Python



# 10 different images without Cracks

```
fig, ax = plt.subplots(nrows=2, ncols=5, sharex=True, sharey=True,)
ax = ax.flatten()
for i in range(10):
    img = X_train[y_train == 0][i].reshape(num_px, num_px, 3)
    ax[i].imshow(img)

ax[0].set_xticks([])
ax[0].set_yticks([])
plt.tight_layout()
plt.show()
```



# Contrast Model

02

# Logistic Regression

- The model did not successfully reach the optimal solution within the predefined maximum iteration limit.

```python
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression()
logreg.fit(X_train, y_train)
```

✓ 1m 11.9s Python

```
/opt/homebrew/Caskroom/miniforge/base/envs/ds/lib/python3.10/site-packages/sklearn/linear_model/_
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

▾ LogisticRegression
LogisticRegression()

# Logistic Regression

- Increase the number of iterations, change the solver from ‘lbfgs’ to ‘saga’

```
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression(solver='saga', max_iter=1000)
logreg.fit(X_train, y_train)
```

✓ 224m 37.3s

```
LogisticRegression
LogisticRegression(max_iter=1000, solver='saga')
```

```
cofficients = logreg.coef_[0]
intercept = logreg.intercept_[0]

print("Intercept:", intercept)
print(len(cofficients))
print("Cofficients:", cofficients)
```

✓ 0.0s

```
Intercept: 0.8695552
154587
Cofficients: [ 0.00897715  0.00135011 -0.00246307 ...  0.03125    0.03125
  0.03125   ]
```

```
from sklearn.metrics import accuracy_score

train_accuracy = accuracy_score(y_train, y_train_pred)
print('Train accuracy: %.2f%%' %train_accuracy)

test_accuracy = accuracy_score(y_test, y_test_pred)
print('Test accuracy: %.2f%%' %test_accuracy)
```

```
Train accuracy: 0.95%
Test accuracy: 0.91%
```

# AdaBoost

- **n_estimators = 50**

  max_depth = 1

  learning_rate = 1

  algorithm = SAMME.R

  random_state = None

```
from sklearn.ensemble import AdaBoostClassifier


adaBoost = AdaBoostClassifier(n_estimators=50)
model_training(adaBoost, X_train, y_train)
```

✓ 105m 38.0s

```
Find_Optimal_Cutoff(adaBoost, X_test, y_test)

training_accuracy=get_acuuracy(adaBoost, X_train, y_train)
print('Training accuracy: %.2f%%' %training_accuracy)

test_accuracy=get_acuuracy(adaBoost, X_test, y_test)
print('Test accuracy: %.2f%%' %test_accuracy)
```

✓ 43.6s

```
Optimal Cutoff: [0.49195807]
Training accuracy: 93.85%
Test accuracy: 93.20%
```

# AdaBoost

- **n_estimators = 100**

  max_depth = 1

  learning_rate = 1

  algorithm = SAMME.R

  random_state = None

```
from sklearn.ensemble import AdaBoostClassifier


adaBoost = AdaBoostClassifier(n_estimators=100)
model_training(adaBoost, X_train, y_train)
```

✓ 206m 58.9s

```
Find_Optimal_Cutoff(adaBoost, X_test, y_test)

training_accuracy=get_acuuracy(adaBoost, X_train, y_train)
print('Training accuracy: %.2f%%' %training_accuracy)

test_accuracy=get_acuuracy(adaBoost, X_test, y_test)
print('Test accuracy: %.2f%%' %test_accuracy)
```

✓ 1m 12.5s

```
Optimal Cutoff: [0.4951742]
Training accuracy: 96.49%
Test accuracy: 95.41%
```

# Neutral Network

03

# Common values for a Neural Network MLP model

| parameters | commom values | our choice for the basic nn model |
|---|---|---|
| n_hidden | varies from tens to hundreds | 100 |
| L1 | (0.0001, 0.1) | 0.01 |
| L2 | For a large dataset, start with a higher initial value, such as 0.01 or 0 | 0.01 |
| epoch | 100 | 100 |
| eta | 0.01 or 0.001 | 0.01 |
| alpha | 0 or 0.1 | 0 |
| decrease_const | 0 | 0 |
| minibatches | 1, 32, 64, 128 | 32 |

# Basic Neural Network MLP model

```
nn = NeuralNetMLP(n_output=2,
                  n_features=X_train.shape[1],
                  n_hidden=100,
                  l2=0.01,
                  l1=0.01,
                  epochs=100,
                  eta=0.01,
                  alpha=0,
                  decrease_const=0,
                  minibatches=32,
                  shuffle=True,
                  random_state=1)
```

```
nn.fit(X_train, y_train, print_progress=True)
```

```
training_accuracy=get_acuuracy(nn, X_train, y_train)
print('Training accuracy: %.2f%%' %training_accuracy)

test_accuracy=get_acuuracy(nn, X_test, y_test)
print('Test accuracy: %.2f%%' %test_accuracy)
```

```
Training accuracy: 49.97%
Test accuracy: 49.88%
```

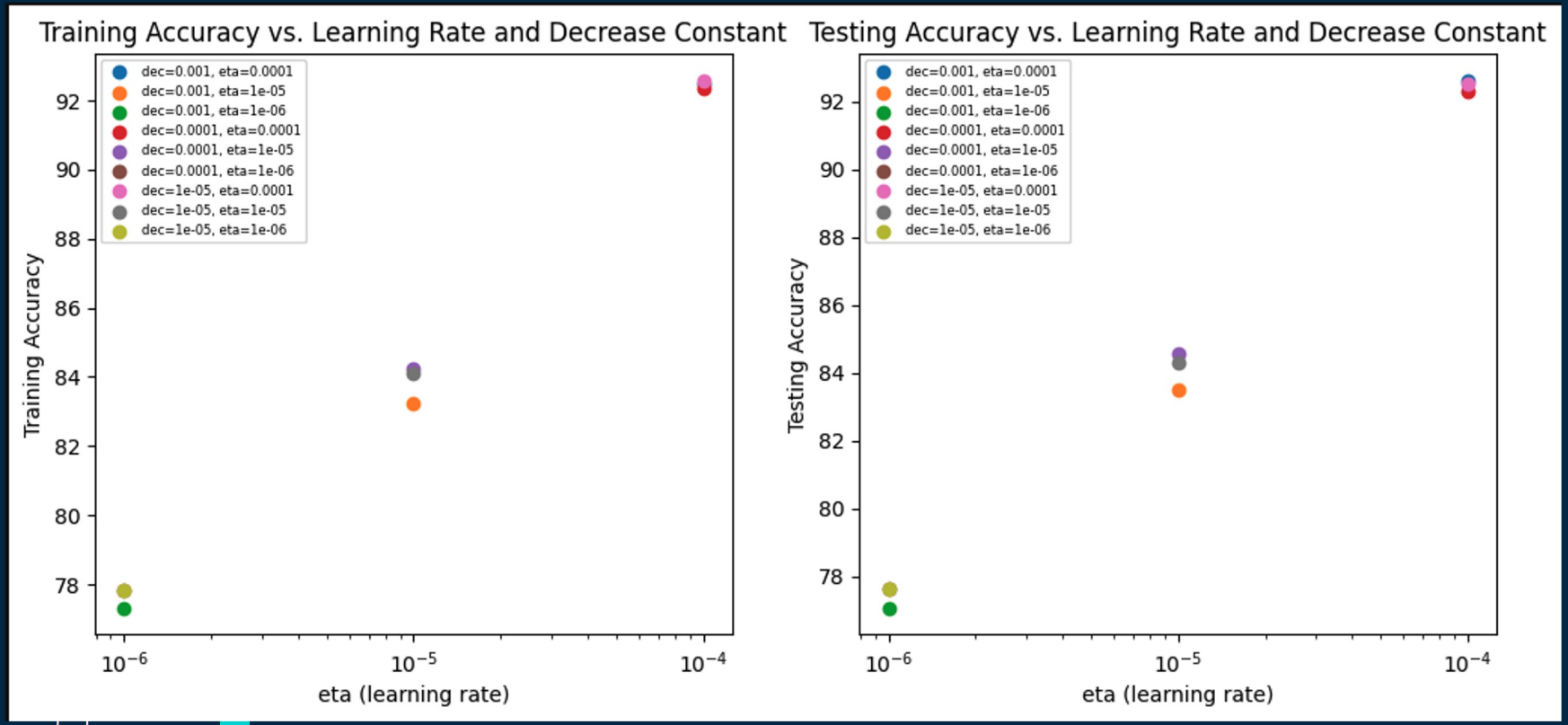
# Pruned Neural Network MLP model

```
decrease_consts = [1e-4, 1e-5]
etas = [1e-3, 5e-4]
n_hiddens = [30, 50, 100]
l1s = [0.0, 0.01, 0.001]
l2s = [0.0, 0.01, 0.001]
minibatch_sizes = [10, 50]
epoch_nums = [50,100]
alphas = [0.01, 0.05]
```

Change values of parameters in Neural Network MLP.

```
decrease_consts = [1e-3, 1e-4, 1e-5]
etas = [1e-4, 1e-5, 1e-6]

results = {}

for decrease_const in decrease_consts:
    for eta in etas:
        nn = NeuralNetMLP(n_output=2,
                          n_features=X_train_new.shape[1],
                          n_hidden = 50,
                          l2=0.02,
                          l1=0.02,
                          epochs = 50,
                          eta=eta,
                          alpha = 0.01,
                          decrease_const=decrease_const,
                          minibatches = 100,
                          shuffle=True,
                          random_state=42)

        nn.fit(X_train_new, y_train_new, print_progress=True)

        training_accuracy = get_acuuracy(nn, X_train_new, y_train_new)
        testing_accuracy = get_acuuracy(nn, X_test_new, y_test_new)

        print(f'decrease_const: {decrease_const}, eta: {eta}')
        print('Training accuracy: %.2f%%' % training_accuracy)
        print('Testing accuracy: %.2f%%' % testing_accuracy)

        results[(decrease_const, eta)] = (training_accuracy, testing_accuracy)
```

# Pruned Neural Network MLP model



# Best Neural Network MLP model

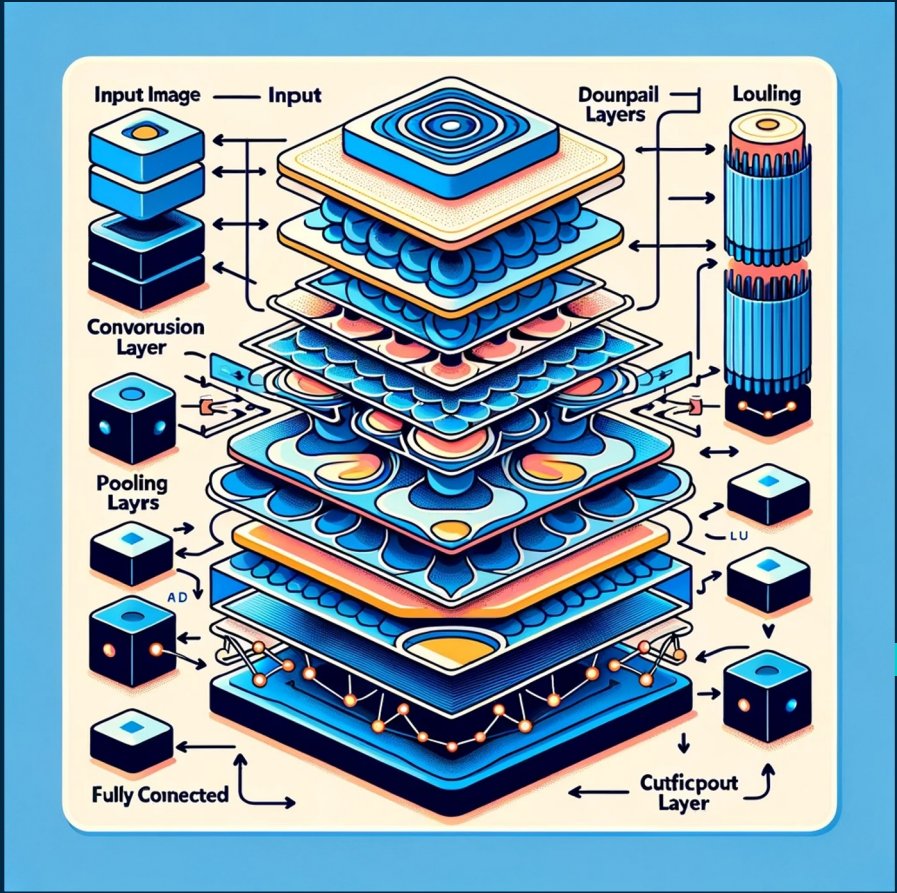
## Best Parameters:

```
nn = NeuralNetMLP(n_output=2,
                  n_features=X_train.shape[1],
                  n_hidden = 100,
                  l2 = 0.02,
                  l1 = 0.02,
                  epochs = 100,
                  eta = 1e-4,
                  alpha = 0.01,
                  decrease_const = 1e-5,
                  minibatches = 100,
                  shuffle = True,
                  random_state = 42 )
```

```
Epoch: 1/500
decrease_const: 0.001, eta: 0.0001
Training accuracy: 92.87%
Testing accuracy: 92.69%
Epoch: 1/500
decrease_const: 0.001, eta: 1e-05
Training accuracy: 86.01%
Testing accuracy: 86.15%
Epoch: 1/500
decrease_const: 0.001, eta: 1e-06
Training accuracy: 79.71%
Testing accuracy: 79.50%
Epoch: 1/500
decrease_const: 0.0001, eta: 0.0001
Training accuracy: 93.81%
Testing accuracy: 93.48%
Epoch: 1/500
decrease_const: 0.0001, eta: 1e-05
Training accuracy: 86.60%
Testing accuracy: 86.70%
Epoch: 1/500
decrease_const: 0.0001, eta: 1e-06
Training accuracy: 81.03%
Testing accuracy: 80.93%
Epoch: 1/500
decrease_const: 1e-05, eta: 0.0001
Training accuracy: 93.80%
Testing accuracy: 93.55%
Epoch: 1/500
decrease_const: 1e-05, eta: 1e-05
Training accuracy: 86.70%
Testing accuracy: 86.80%
Epoch: 50/50
decrease_const: 1e-05, eta: 1e-06
Training accuracy: 81.24%
Testing accuracy: 81.08%
```

# Another Neural Network model

- **Convolutional Neural Network (CNN)**
- This model includes convolutional layers, optional pooling layers, and fully connected layers.
- Convolutional layers effectively handle data with high-dimensional data, like images.
- Reduces the number of parameters and improves computational efficiency through local connections and weight sharing



# Convolutional Neural Network (CNN)

## Model design

```python
# A simple convolutional neural network model
# in_channel (int): Number of input channels (e.g., 3 for RGB images).
class Net(nn.Module):

    def __init__(self, in_channel):
        super(Net, self).__init__()

        # Define the layers
        self.conv1 = nn.Conv2d(in_channel, 32, 3, 1)   # First convolutional layer
        self.conv2 = nn.Conv2d(32, 64, 3, 1)  # Second convolutional layer
        self.dropout1 = nn.Dropout(0.25)  # Dropout layer with 25% drop probability
        self.dropout2 = nn.Dropout(0.5)  # Dropout layer with 50% drop probability
        self.fc1 = nn.Linear(788544, 128)  # First fully connected layer
        self.fc2 = nn.Linear(128, 2)  # Second fully connected layer

    def forward(self, x):
        # Define the forward pass
        x = self.conv1(x)   # Apply the first convolutional layer
        x = F.relu(x)   # Apply ReLU activation function
        x = self.conv2(x)   # Apply the second convolutional layer
        x = F.relu(x)  # Apply ReLU activation function
        x = F.max_pool2d(x, 2)  # Apply max pooling with a 2x2 window
        x = self.dropout1(x)  # Apply the first dropout layer
        x = torch.flatten(x, 1)  # Flatten the tensor for the fully connected layer
        x = self.fc1(x)  # Apply the first fully connected layer
        x = F.relu(x)  # Apply ReLU activation function
        x = self.dropout2(x)  # Apply the second dropout layer
        x = self.fc2(x)  # Apply the second fully connected layer
        output = F.log_softmax(x, dim=1)  # Apply log softmax function
        return output
```

## Model train and test

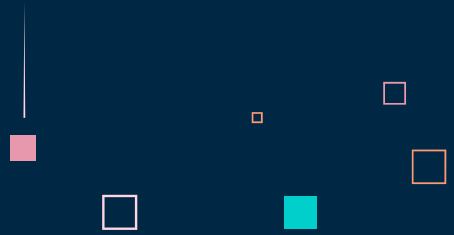
```python
# Train the model
def train(model, device, train_loader, optimizer, epoch, log_interval=10):
    model.train()    # Set the model to training mode
    for batch_idx, (data, target) in enumerate(train_loader):  # Loop over each batch
        data, target = data.to(device), target.to(device)  # Move data to the specified device
        optimizer.zero_grad()  # Clear the gradients
        output = model(data)   # Compute the output
        loss = F.nll_loss(output, target)  # Compute the loss
        loss.backward()  # Compute the gradients
        optimizer.step()  # Update the parameters
        # Print log information
        if batch_idx % log_interval == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                100. * batch_idx / len(train_loader), loss.item()))

# Test the model
def test(model, device, test_loader):
    model.eval()  # Set the model to evaluation mode
    test_loss = 0  # Initialize the test loss
    correct = 0  # Initialize the number of correct predictions
    with torch.no_grad():   # Disable gradient computation
        for data, target in test_loader:   # Loop over each batch in the test data
            data, target = data.to(device), target.to(device)   # Move data to the specified device
            output = model(data)  # Compute the output
            test_loss += F.nll_loss(output, target, reduction='sum').item()  # sum up batch loss
            pred = output.argmax(dim=1, keepdim=True)  # get the index of the max log-probability
            correct += pred.eq(target.view_as(pred)).sum().item()

    test_loss /= len(test_loader.dataset)

    print('\nTest set: Average loss: {:.6f}, Accuracy: {}/{} ({:.3f}%)\n'.format(
        test_loss, correct, len(test_loader.dataset),
        100. * correct / len(test_loader.dataset)))
```

# Convolutional Neural Network (CNN)

```python
batch_size = 32  # Batch size for training
test_batch_size = 32  # Batch size for testing

lr = 1.0  # Initial learning rate
gamma = 0.7  # Learning rate decay factor
epochs = 1  # Number of training epochs
log_interval = 10  # Interval for printing training log
save_model = True  # Flag to determine whether to save the trained model
```

Set related parameters

Run the model



```python
# Initialize the optimizer with Adadelta algorithm and set the learning rate
optimizer = optim.Adadelta(model.parameters(), lr=lr)

# Set up the learning rate scheduler, which adjusts the learning rate after each epoch
scheduler = StepLR(optimizer, step_size=1, gamma=gamma)

# Training and testing loop
for epoch in range(1, epochs + 1):
    # Train the model for one epoch
    train(model, device, train_loader, optimizer, epoch, log_interval)

    # Test the model after each epoch
    test(model, device, dev_loader)

    # Update the learning rate based on the scheduler
    scheduler.step()

    # Save the model's state after each epoch if save_model flag is True
    if save_model:
        torch.save(model.state_dict(), f"data/crack_image_{save_name}_cnn-epoch{epoch}.pt")
```

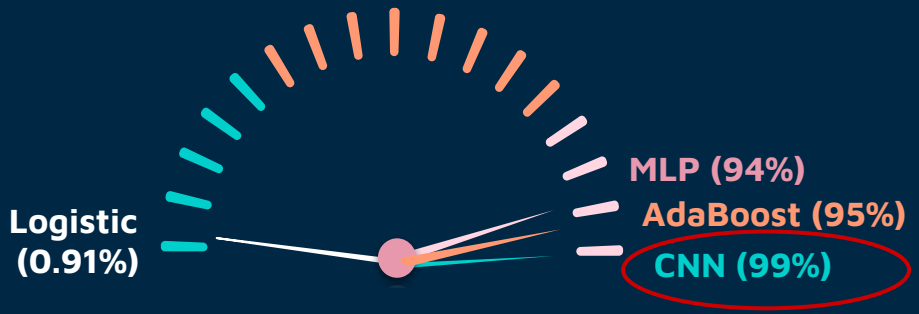
✓ 5m 30.6s

```
Test set: Average loss: 0.0416, Accuracy: 6350/6422 (98.879%)
```

# Conclusions and Implications

04

# Conclusions




Test Accuracy

Execution Time

# Implications

- Fundamental models like boosting algorithms can perform better than expected, delivering satisfactory results comparable to more complex models such as MLPs.
- When the chosen model fails to achieve desired outcomes after many rounds of pruning, stepping out of the existing framework and exploring alternative models, like CNNs, can lead to significant improvements.
- Managing large datasets underscores the importance of employing data processing techniques to enhance work efficiency without compromising the accuracy of the research.

# THANKS