Practical No 1.

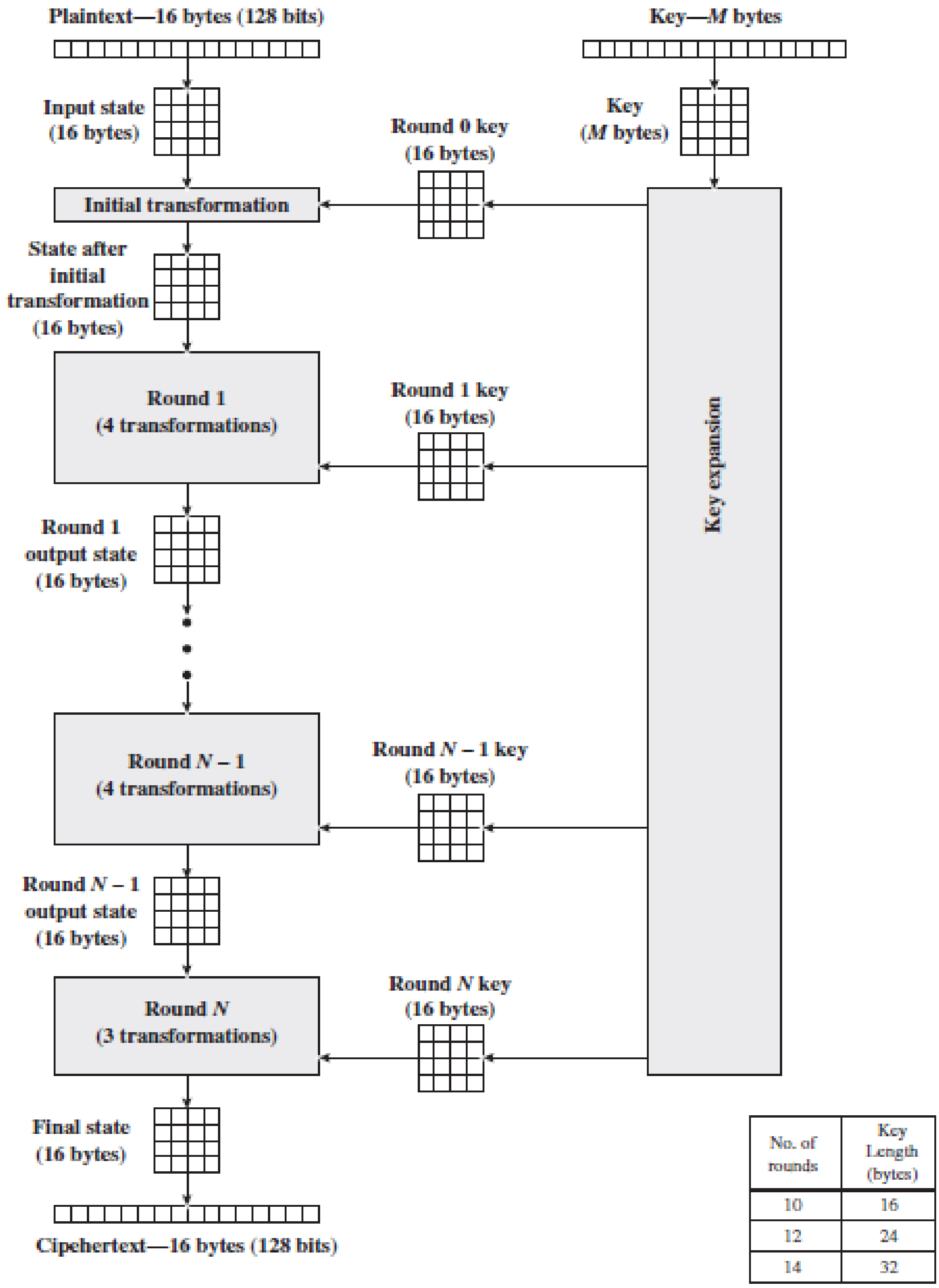
Build the program where user-entered text is encrypted using caeser cipher algorithm

Program:

```
letters=['a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v','w','x','y','z']
playing=''
org=''
result=[]
def encrypt(str,n):
    for x in str:
        if x ==" ":
            converted = ' '
            result.append(converted)
        else:
            converted = (letters.index(x) + n)%26
            result.append(letters[converted])
    final =' '.join(result)
    print(final)
    return final
def decrypt(str,n):
    back =[]
    for x in str:
        if x ==" ":
            original =' '
            back.append(original)
        else:
            original = (letters.index(x)-n)%26
            back.append(letters[original])
            org =''.join(back)
    print(org)
str=input("Enetr The String To Be Encrypted : ")
n=int(input("Eneter the Key : "))
final=encrypt(str,n)
print("Let's Decrypt The Text")
decrypt(final,n)
```

Output:

Enetr The String To Be Encrypted : government college of engineering
Eneter the Key : 3
j r y h u q p h q w   f r o o h j h   r i   h q j l q h h u l q j
Let's Decrypt The Text
g o v e r n m e n t   c o l l e g e   o f   e n g i n e e r i n g



No. of rounds	Key Length (bytes)
10	16
12	24
14	32

Figure 1: AES Encryption structure

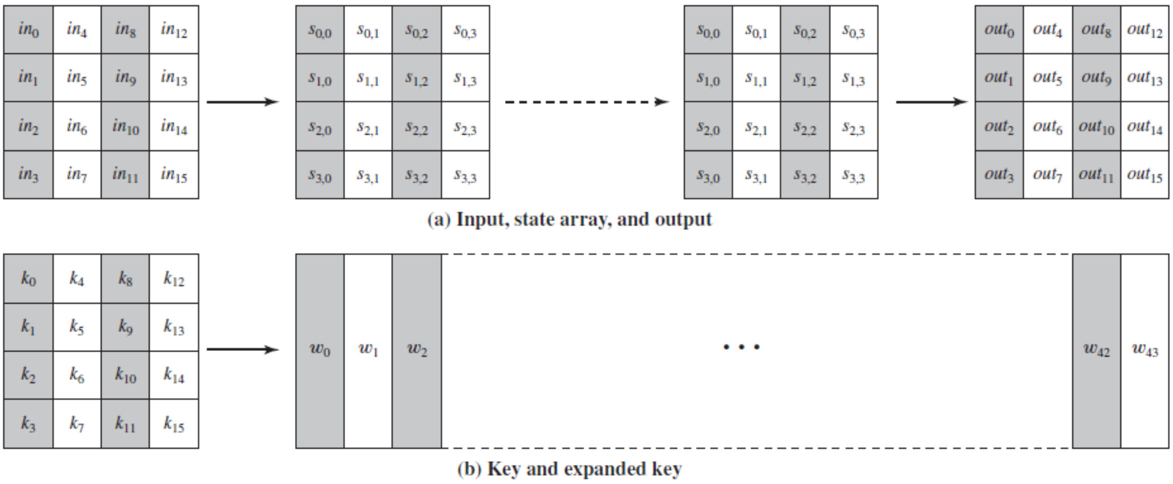


Figure 2: AES Key expansion

Table 1: AES Parameters

Key Size (words/bytes/bits)	4/16/128	6/24/192	8/32/256
Plaintext Block Size (words/bytes/bits)	4/16/128	4/16/128	4/16/128
Number of Rounds	10	12	14
Round Key Size (words/bytes/bits)	4/16/128	4/16/128	4/16/128
Expanded Key Size (words/bytes)	44/176	52/208	60/240

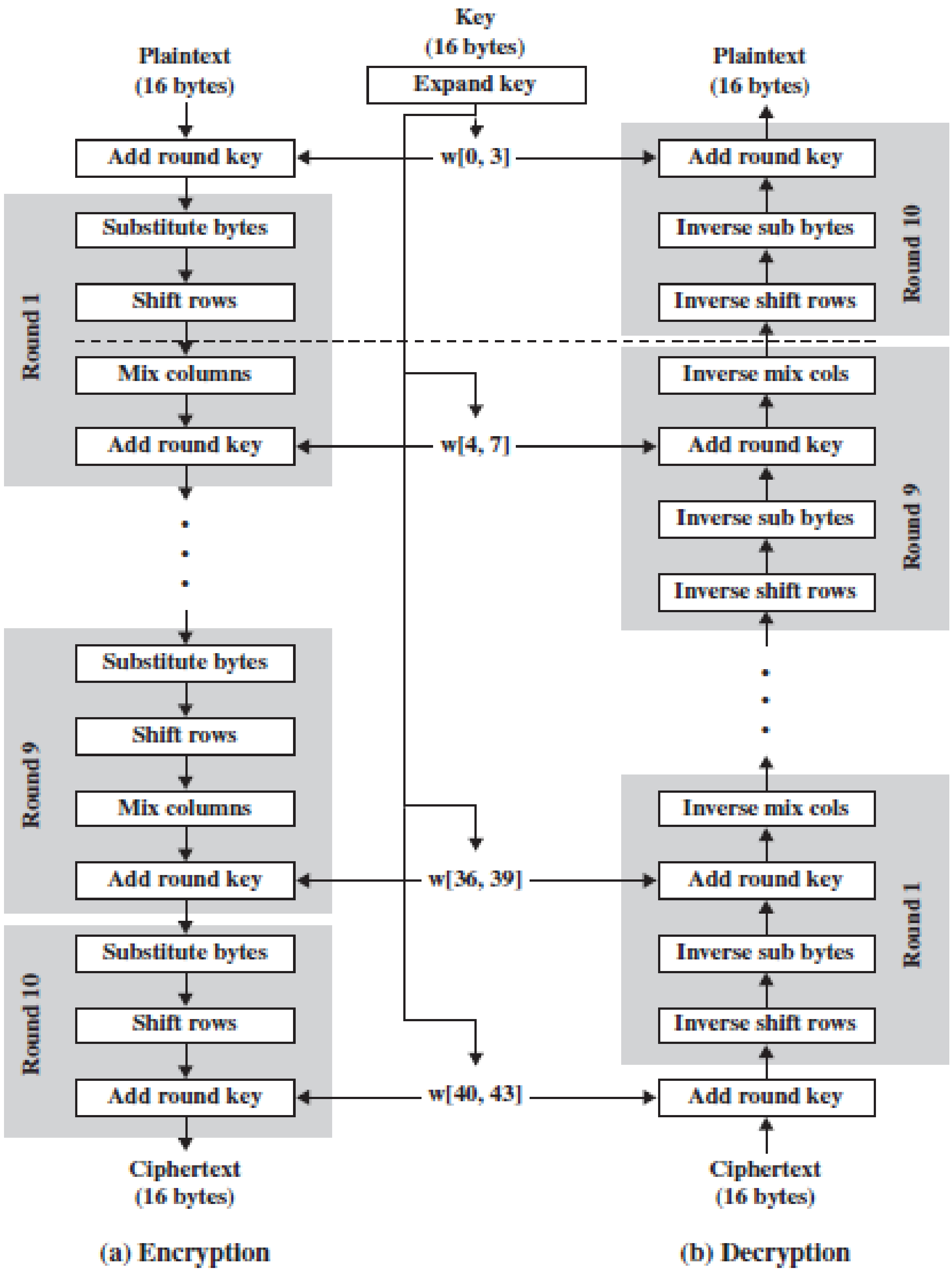


Figure 3: AES Encryption and Decryption

Figure 3 shows the AES cipher in more detail, indicating the sequence of transformations in each round and showing the corresponding decryption function.

Practical No 2.

Implement Advanced Encryption Standard to encrypt and decrypt data

Program:

```
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad, unpad
import binascii

# Replace these with your actual key and data
key = b'my_secret_key_16'
data = b'Hello, this is a test message for encryption!'

# Encryption
cipher = AES.new(key, AES.MODE_CBC)
cipher_text = cipher.encrypt(pad(data, AES.block_size))
iv = cipher.iv

# Print encrypted message
print("Encrypted ciphertext:", binascii.hexlify(cipher_text))

# Decryption
decrypt_cipher = AES.new(key, AES.MODE_CBC, iv)
plain_text = unpad(decrypt_cipher.decrypt(cipher_text), AES.block_size)

print("Decrypted plaintext:", plain_text.decode('utf-8'))
```

Output:

Encrypted ciphertext:
b'd60e9efb1aef189a7439a9aeea3a00fcf3a60e504641fafa80a6f3cdec1d86290a2cdb99d4cd5086611684501284faef'
Decrypted plaintext: Hello, this is a test message for encryption!

Practical No 3.

Implement RSA Algorithum to encrypt and decrypt data

Program:

```
# Python for RSA asymmetric cryptographic algorithm.
# For demonstration, values are
# relatively small compared to practical application
import math
def gcd(a, h):
    temp = 0
    while(1):
        temp = a % h
        if (temp == 0):
            return h
        a = h
        h = temp


p = 3
q = 7
n = p*q
e = 2
phi = (p-1)*(q-1)

while (e < phi):

    # e must be co-prime to phi and
    # smaller than phi.
    if(gcd(e, phi) == 1):
        break
    else:
        e = e+1

# Private key (d stands for decrypt)
# choosing d such that it satisfies
# d*e = 1 + k * totient

k = 2
d = (1 + (k*phi))/e
```

```
# Message to be encrypted
msg = 12.0

print("Message data = ", msg)

# Encryption c = (msg ^ e) % n
c = pow(msg, e)
c = math.fmod(c, n)
print("Encrypted data = ", c)

# Decryption m = (c ^ d) % n
m = pow(c, d)
m = math.fmod(m, n)
print("Original Message Sent = ", m)
```

Output:

```
Message data =  12.0
Encrypted data =  3.0
Original Message Sent =  12.0
```

Practical No 4.
Implement Advanced Encryption Standard to encrypt and decrypt data

Program:

```
import hashlib

# prints all available algorithms
print ("The available algorithms are : ", end ="")
print (hashlib.algorithms_guaranteed)

# SHA hash algorithms.

import hashlib

# initializing string
str = "Government college of engineering"

# encoding "Government college of engineering" using encode()
# then sending to SHA256()
result = hashlib.sha256(str.encode())

# printing the equivalent hexadecimal value.
print("The hexadecimal equivalent of SHA256 is : ")
print(result.hexdigest())

print ("\r")

# initializing string
str = "Government college of engineering"

# encoding Government college of engineering using encode()
# then sending to SHA384()
result = hashlib.sha384(str.encode())

# printing the equivalent hexadecimal value.
print("The hexadecimal equivalent of SHA384 is : ")
print(result.hexdigest())

print ("\r")

# initializing string
str = "Government college of engineering"

# encoding Government college of engineering using encode()
# then sending to SHA224()
result = hashlib.sha224(str.encode())
```

```
# printing the equivalent hexadecimal value.
print("The hexadecimal equivalent of SHA224 is : ")
print(result.hexdigest())

print ("\r")

# initializing string
str = "Government college of engineering"

# encoding Government college of engineering using encode()
# then sending to SHA512()
result = hashlib.sha512(str.encode())

# printing the equivalent hexadecimal value.
print("The hexadecimal equivalent of SHA512 is : ")
print(result.hexdigest())

print ("\r")

# initializing string
str = "Government college of engineering"

# encoding Government college of engineering using encode()
# then sending to SHA1()
result = hashlib.sha1(str.encode())

# printing the equivalent hexadecimal value.
print("The hexadecimal equivalent of SHA1 is : ")
print(result.hexdigest())
```

Output:

```
The available algorithms are : {'sha256', 'sha224', 'blake2b', 'sha3_224', 'md5', 'sha3_512', 'shake_256', 'sha3_384',
'sha512', 'shake_128', 'sha1', 'sha384', 'blake2s', 'sha3_256'}
The hexadecimal equivalent of SHA256 is :
da9d962faa88d2f5d4407585e6efc35f23a825c2df282f2d72f147fdb2da2b06

The hexadecimal equivalent of SHA384 is :
8f5f84e5192ac5c6ec5ca692b6de76938da3c39509b0ae2de214ccb74f85aa9af557e40229e57503b49c2497a50b553a

The hexadecimal equivalent of SHA224 is :
72b9e0a839bc11dd52f382432552c7cacd89d1f77c3f279033cc0984

The hexadecimal equivalent of SHA512 is :
c9a88b672e4d06d13ade87d7b9d9ec027a2614e046acfa591eada9ffea0f03b9756ca92b365f5cb7aec42e42f9290504e5a606
bfb422cd48a830fd10f32460bb

The hexadecimal equivalent of SHA1 is :
96140c62dbdbf1f964c1dcfc4927ceaa75972f33
```

Practical No 5.

Implement Advanced Encryption Standard to encrypt and decrypt data

Program:

```
from ecdsa import SigningKey

# Generate a new private key
private_key = SigningKey.generate()

# Sign a message
message = b"creating a digital signature"
signature = private_key.sign(message)

print("Signature:", signature)
```

Output:

Signature:
b'x\xa8\xf3\x88\xc5*m\x92\x8f.\x95\x89=\xaf\x8f\xeb.\xbcx\xbf\x9fS\xd5
\xe6\x02C[\x84\x00\xf4J2\xbb)M\t\\\xa46x\x06Fb\xaa\x897 \x9e'

Practical No 6.

Generate Digital Signature and verify it using DSA/RSA/ECC

Program:

```
from Crypto.PublicKey import DSA
from Crypto.Signature import DSS
from Crypto.Hash import SHA256

 # Create a new DSA key
key = DSA.generate(2048)
f = open("public_key.pem", "w")
f.write(key.publickey().export_key())
f.close()
# Sign a message
message = b"Hello"
hash_obj = SHA256.new(message)
signer = DSS.new(key, 'fips-186-3')
signature = signer.sign(hash_obj)
# Load the public key
f = open("public_key.pem", "r")
hash_obj = SHA256.new(message)
pub_key = DSA.import_key(f.read())
verifier = DSS.new(pub_key, 'fips-186-3')

 # Verify the authenticity of the message
try:
   verifier.verify(hash_obj, signature)
   print("The message is authentic.")
except ValueError:
   print("The message is not authentic.")
```

Output:

The message is authentic.

Practical No 7.

Write a program to check strength of password

Program:

```
def checkPassword(password):
    upperChars, lowerChars, specialChars, digits, length = 0, 0, 0, 0, 0
    length = len(password)

    if (length < 6):
        print("Password must be at least 6 characters long!\n")
    else:
        for i in range(0, length):
            if (password[i].isupper()):
                upperChars += 1
            elif (password[i].islower()):
                lowerChars += 1
            elif (password[i].isdigit()):
                digits += 1
            else:
                specialChars += 1

    if (upperChars != 0 and lowerChars != 0 and digits != 0 and specialChars != 0):
        if (length >= 10):
            print("The strength of password is strong.\n")
        else:
            print("The strength of password is medium.\n")
    else:
        if (upperChars == 0):
            print("Password must contain at least one uppercase character!\n")
        if (lowerChars == 0):
            print("Password must contain at least one lowercase character!\n")
        if (specialChars == 0):
            print("Password must contain at least one special character!\n")
        if (digits == 0):
            print("Password must contain at least one digit!\n")


password = input("Please enter password: ")
checkPassword(password)
```

Output:

Please enter password: MCA@Geca2023
The strength of password is strong.

Practical No 8.

Implement a program in MATLAB for classification using supervised learning algorithm

Program:

Classification.py

```
%% classification
close all;
clear all;
clc;


% load simpleclass_dataset
%[x, t] = simplefit_dataset;
% [x,t] = cancer_dataset;
[x,t] = iris_dataset;

net = patternnet(10);
% net = feedforwardnet(10);

% train
[net, tr] = train(net, x, t);
view(net)

% estimate the targets
y = net(x);
classes = vec2ind(y)
perf = perform(net, y, t);
perf_test = perform(net, t(tr.testInd),net(x(:,tr.testInd)))

plotconfusion(t(:,tr.testInd), (net(x(:,tr.testInd ...
   ))))
```

Cluster.py

```
% Data clustering  problem
%x = simplecluster_dataset;
clc;
close all;
clear all;

x = iris_dataset;

net = selforgmap([8 8]);
net = train(net,x);
```

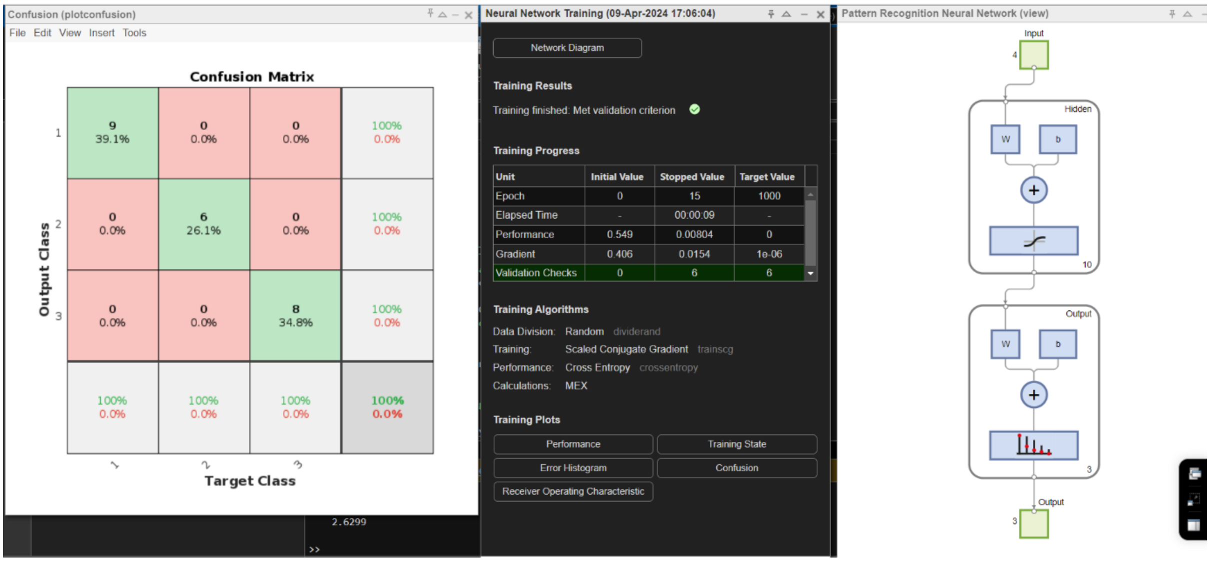
```
view(net)
y = net(x);
classes = vec2ind(y);
```

Funcfit.py

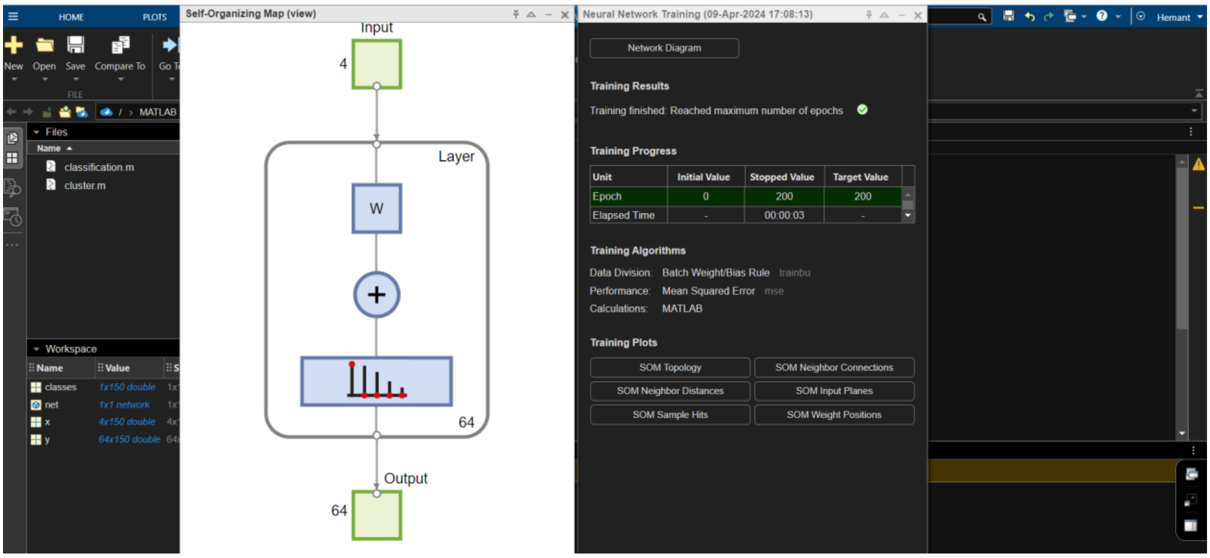
```
%  Estimation of body fat using function fitting
load bodyfat_dataset;


x = bodyfatInputs;
t = bodyfatTargets;

% Choose a Training Function
% For a list of all training functions type: help nntrain
% 'trainlm' is usually fastest.
% 'trainbr' takes longer but may be better for challenging problems.
% 'trainscg' uses less memory. Suitable in low memory situations.
trainFcn = 'trainlm';  % Levenberg-Marquardt backpropagation.

% Create a Fitting Network
hiddenLayerSize = 10;
net = fitnet(hiddenLayerSize,trainFcn);

% Setup Division of Data for Training, Validation, Testing
net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 15/100;

% Train the Network
[net,tr] = train(net,x,t);

% Test the Network
y = net(x);
e = gsubtract(t,y);
performance = perform(net,t,y)

% View the Network
view(net)

% Plots
% Uncomment these lines to enable various plots.
%figure, plotperform(tr)
%figure, plottrainstate(tr)
%figure, ploterrhist(e)
%figure, plotregression(t,y)
%figure, plotfit(net,x,t)
```

Output:

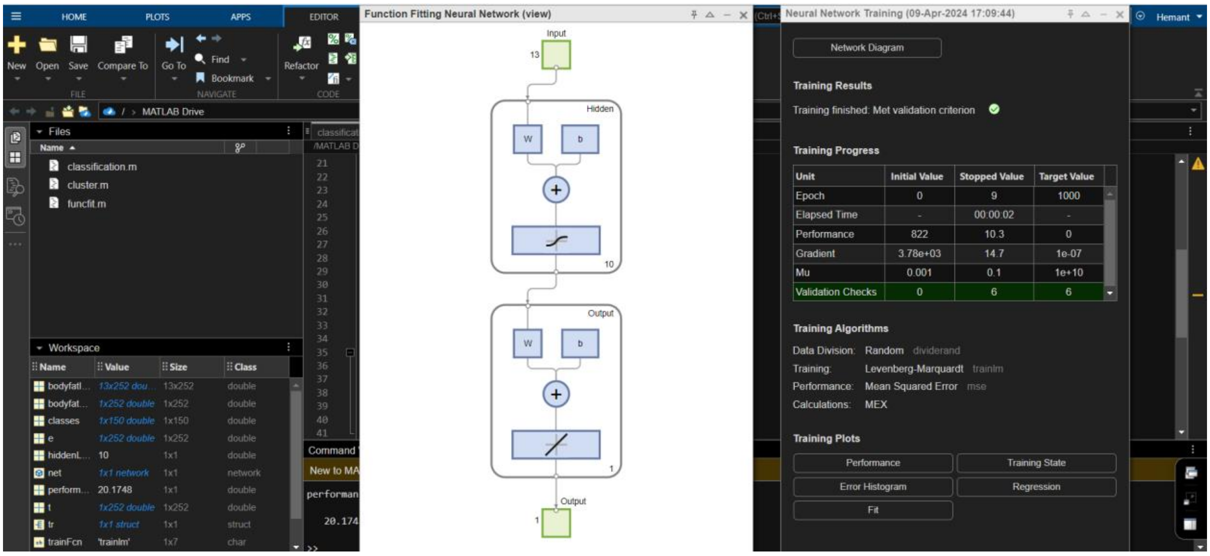
Classification.m



Cluster.m



Funcfit.m



Function Fitting Neural Network (view)
Neural Network Training (09-Apr-2024 17:09:44)
Network Diagram
Training Results
Training finished: Met validation criterion
Training Progress
Unit | Initial Value | Stopped Value | Target Value
Epoch | 0 | 9 | 1000
Elapsed Time | - | 00:00:02 | -
Performance | 822 | 10.3 | 0
Gradient | 3.78e+03 | 14.7 | 1e-07
Mu | 0.001 | 0.1 | 1e+10
Validation Checks | 0 | 6 | 6
Training Algorithms
Data Division: Random dividerand
Training: Levenberg-Marquardt trainlm
Performance: Mean Squared Error mse
Calculations: MEX
Training Plots
Performance
Training State
Error Histogram
Regression
Fit