Assignment 1

Problem Statement

i) Generate a random 2-D matrix having 200 rows i.e. (200,1)

ii) Fix each row number with seed values=10

iii) Formulate a linear regression line (model) having the following hypothesis: y=10x+7

iv) Find the mean & standard deviation of row elements.

v) Fit the linear regression "regression model" and train with row elements.

vi) Predict "y_predicted" values using the "regression_model"

vii) Find/Evaluate the "root mean square" error between the actual level i.e. "y" and the predicted level "y_predicted".

viii) Find the slope

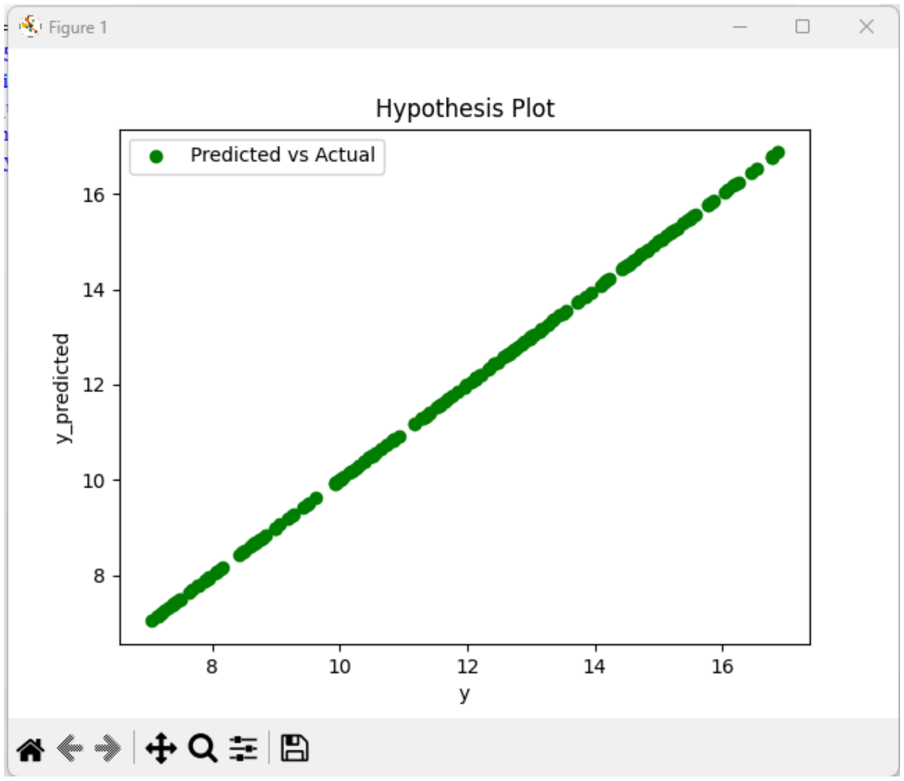
ix) Find the intercept of the hypothesis.

Program Code:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error as mse

# Set the random seed for reproducibility
np.random.seed(10)

# Generate the dataset
x = np.random.rand(200, 1)
y = 10 * x + 7

# Initialize the model
model = LinearRegression()

# Calculate mean and standard deviation
mean = x.mean()
std = x.std()
print("Mean:", mean)
print("Standard Deviation:", std)

# Fit the model
model.fit(x, y)

# Predict y values
y_predicted = model.predict(x)

# Calculate RMSE
rmse = np.sqrt(mse(y, y_predicted))
print("Root Mean Squared Error:", rmse)
```

```
# Print model parameters
print("Intercept of the hypothesis:", model.intercept_[0])
print("Slope of the hypothesis:", model.coef_[0][0])

# Plot the original vs. predicted values
plt.scatter(y, y_predicted, color='g', label='Predicted vs Actual')
plt.xlabel('y')
plt.ylabel('y_predicted')
plt.title('Hypothesis Plot')
plt.legend()
plt.show()
```

Output



```
Mean: 0.47205441448840870
Standard Deviation: 0.27470993318476294
Root Mean Squared Error: 2.4900696205201077e-15
Intercept of the hypothesis: 7.000000000000004
Slope of the hypothesis: 9.999999999999995
```

Assignment 2

Problem Statement

Download the following customer dataset from the below link:
Data Set: https://www.kaggle.com/shwetabh123/mall-customers
This dataset gives the data of Income and money spent by the customers visiting a Shopping Mall. The data set contains Customer ID, Gender, Age, Annual Income, and Spending Score. Therefore, as a mall owner, you need to find the group of people who are the profitable customers for the mall owner. Apply at least two clustering algorithms (based on Spending Score) to find the group of customers.

a. Apply Data pre-processing (Label Encoding, Data Transformation….) techniques if necessary.
b. Perform data preparation (Train-Test Split)
c. Apply Machine Learning Algorithm
d. Evaluate Model.
e. Apply Cross-Validation and Evaluate the Model

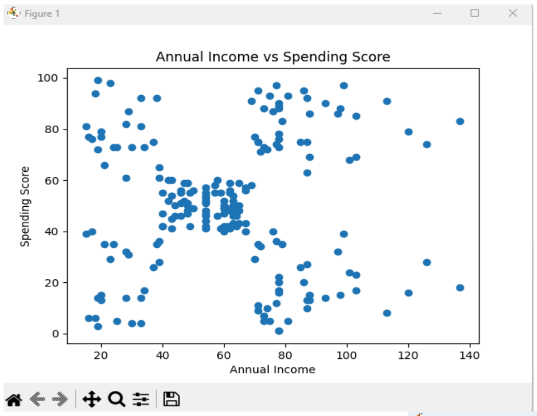
Program Code

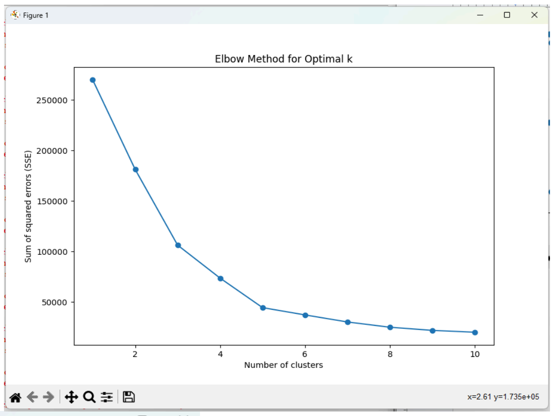
```
import numpy as np
import pandas as pd
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split, KFold, cross_val_score
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score
import warnings
# Suppress warnings
warnings.filterwarnings('ignore')
# Load the dataset
df = pd.read_csv('Mall_Customers.csv')
# Display the first few rows of the DataFrame
print(df.head())
# Identify columns with missing values
print("Columns with missing values:", df.columns[df.isna().any()])
# Scatter plot of 'Annual Income (k$)' vs 'Spending Score (1-100)'
plt.scatter(df['Annual Income (k$)'], df['Spending Score (1-100)'])
plt.xlabel('Annual Income')
plt.ylabel('Spending Score')
plt.title('Annual Income vs Spending Score')
plt.show()
# Determine the optimal number of clusters using the elbow method
sse = []
for k in range(1, 11):
    km = KMeans(n_clusters=k, random_state=42)
    km.fit(df[['Annual Income (k$)', 'Spending Score (1-100)']])
    sse.append(km.inertia_)  # SSE for each k
# Plot SSE against the number of clusters
```

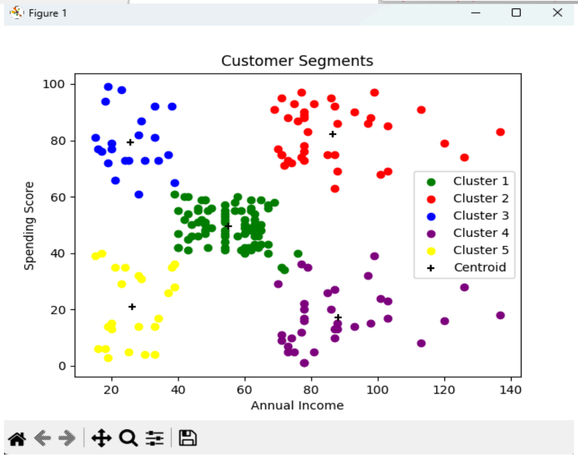
```
plt.plot(range(1, 11), sse, marker='o')
plt.xlabel('Number of clusters')
plt.ylabel('Sum of squared errors (SSE)')
plt.title('Elbow Method for Optimal k')
plt.show()
# Fit KMeans with the optimal number of clusters
optimal_k = 5
km = KMeans(n_clusters=optimal_k, random_state=42)
y_predicted = km.fit_predict(df[['Annual Income (k$)', 'Spending Score (1-100)']])
# Add cluster labels to the DataFrame
df['Group of Customers'] = y_predicted
print(df.head())
# Plot clusters
colors = ['green', 'red', 'blue', 'purple', 'yellow']
for i in range(optimal_k):
    cluster_data = df[df['Group of Customers'] == i]
    plt.scatter(cluster_data['Annual Income (k$)'], cluster_data['Spending Score (1-100)'],
color=colors[i], label=f'Cluster {i+1}')
plt.scatter(km.cluster_centers_[:, 0], km.cluster_centers_[:, 1], color='black', marker='+',
label='Centroid')
plt.xlabel('Annual Income')
plt.ylabel('Spending Score')
plt.title('Customer Segments')
plt.legend()
plt.show()
# Preprocessing for Decision Tree Classifier
df['Genre'] = LabelEncoder().fit_transform(df['Genre'])
df.drop(['CustomerID'], axis=1, inplace=True)
# Define X and Y for training
X = df.drop(['Group of Customers'], axis=1)
Y = df['Group of Customers']
# Partition the dataset into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, random_state=42)
print("\n\nTraining data (X): ", x_train.shape)
print("Training data (Y): ", y_train.shape)
print("Testing data (X): ", x_test.shape)
print("Testing data (Y): ", y_test.shape)
# Train the Decision Tree Classifier
clf = DecisionTreeClassifier()
clf.fit(x_train, y_train)
# Predict on the test set
y_pred = clf.predict(x_test)
# Evaluate the Decision Tree Classifier
print("\n\nApplying the Decision Tree Classifier : ")
acc = accuracy_score(y_test, y_pred)
print("\nThe Accuracy using Decision Tree Classifier : ", round(acc * 100, 2), "%")
# Cross-Validation
kf = KFold(n_splits=5, shuffle=True, random_state=42)
# Function to evaluate model performance using cross-validation
def cv_evaluate_model(model, X, y):
    accuracy_scores = cross_val_score(model, X, y, cv=kf, scoring='accuracy')
    return accuracy_scores.mean()
# Evaluating Decision Tree Classifier with cross-validation
dt_accuracy_mean = cv_evaluate_model(clf, X, Y)
print("\n\nCross-Validation Mean Accuracy:", round(dt_accuracy_mean * 100, 2), "%")
```

Output

```
    CustomerID   Genre  Age  Annual Income (k$)  Spending Score (1-100)
0            1    Male   19                  15                      39
1            2    Male   21                  15                      81
2            3  Female   20                  16                       6
3            4  Female   23                  16                      77
4            5  Female   31                  17                      40
Columns with missing values: Index([], dtype='object')
   CustomerID   Genre  ...  Spending Score (1-100)  Group of Customers

0           1    Male  ...                      39                   4

1           2    Male  ...                      81                   2

2           3  Female  ...                       6                   4

3           4  Female  ...                      77                   2

4           5  Female  ...                      40                   4

[5 rows x 6 columns]

Training data (X):  (140, 4)

Training data (Y):  (140,)

Testing data (X):  (60, 4)

Testing data (Y):  (60,)

Applying the Decision Tree Classifier :

The Accuracy using the Decision Tree Classifier:  93.33 %

Cross-Validation Mean Accuracy: 96.0 %
```







Assignment 3

Problem Statement

```
Assignment on Association Rule Learning
Download the Market Basket Optimization dataset from the below link.
Data Set: https://www.kaggle.com/hemanthkumar05/market-basket-optimization
This dataset comprises the list of transactions of a retail company over one week. It contains a total of 7501
transaction records, each consisting of a list of items sold in one transaction. Find the association rules
between items using this record of transactions and items in each transaction. There is no header in the dataset
and the first row contains the first transaction, so the mentioned header = None here while loading the dataset.
a. Follow the following steps:
b. Data Preprocessing
c. Generate the list of transactions from the dataset
d. Train Apriori algorithm on the dataset
e. Visualize the list of rules
F. Generated rules depend on the values of hyper parameters. By increasing the minimum confidence value and find
the rules accordingly
```

Program Code

```
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules
from mlxtend.preprocessing import TransactionEncoder

# Load the dataset
data = pd.read_csv('Market_Basket_Optimisation.csv', header=None)
print(data.head())

# Convert dataset to a list of lists
transactions = []
for i in range(len(data)):
    transactions.append([str(data.values[i, j]) for j in range(len(data.columns)) if
str(data.values[i, j]) != 'nan'])

# Use TransactionEncoder to transform the dataset
te = TransactionEncoder()
te_ary = te.fit(transactions).transform(transactions)
one_hot_encoded = pd.DataFrame(te_ary, columns=te.columns_)

# Apply the Apriori algorithm
frequent_itemsets = apriori(one_hot_encoded, min_support=0.005, use_colnames=True)
print(frequent_itemsets)

# Generate the association rules
rules = association_rules(frequent_itemsets, metric='lift', min_threshold=1)
print(rules)
# Filter rules by high confidence
rules_high_confidence = association_rules(frequent_itemsets, metric='confidence',
min_threshold=0.3)
print(rules_high_confidence)
```

Output

```
              0          1           2   ...              17       18         19
0        shrimp    almonds     avocado   ...  frozen smoothie  spinach  olive oil
1       burgers  meatballs        eggs   ...              NaN      NaN        NaN
2       chutney        NaN         NaN   ...              NaN      NaN        NaN
3        turkey    avocado         NaN   ...              NaN      NaN        NaN
4  mineral water      milk  energy bar   ...              NaN      NaN        NaN
[5 rows x 20 columns]
      support                                      itemsets
0    0.020397                                     (almonds)
1    0.008932                           (antioxydant juice)
2    0.033329                                     (avocado)
3    0.008666                                       (bacon)
4    0.010799                              (barbecue sauce)
..        ...                                           ...
720  0.007466              (mineral water, spaghetti, soup)
721  0.009332          (mineral water, spaghetti, tomatoes)
722  0.006399            (mineral water, spaghetti, turkey)
723  0.006266  (mineral water, spaghetti, whole wheat rice)
724  0.005066              (spaghetti, olive oil, pancakes)
[725 rows x 2 columns]
                antecedents             consequents  ...  conviction  zhangs_metric
0                 (almonds)               (burgers)  ...    1.225089       0.671653
1                 (burgers)               (almonds)  ...    1.041724       0.720799
2                 (almonds)             (chocolate)  ...    1.184553       0.452150
3               (chocolate)               (almonds)  ...    1.016834       0.529719
4                 (almonds)                  (eggs)  ...    1.206774       0.448005
...                     ...                     ...  ...         ...            ...
1827  (spaghetti, pancakes)             (olive oil)  ...    1.169224       0.689825
1828  (olive oil, pancakes)             (spaghetti)  ...    1.555746       0.635736
1829            (spaghetti)   (olive oil, pancakes)  ...    1.018846       0.761446
1830            (olive oil)   (spaghetti, pancakes)  ...    1.056037       0.719852
1831             (pancakes)  (spaghetti, olive oil)  ...    1.032075       0.629602
[1832 rows x 10 columns]
                           antecedents  ... zhangs_metric
0                            (almonds)  ...      0.448005
1                            (almonds)  ...      0.367669
2                            (avocado)  ...      0.325896
3                          (black tea)  ...      0.367609
4                            (burgers)  ...      0.499424
..                                 ...  ...           ...
256            (mineral water, turkey)  ...      0.487019
257                (spaghetti, turkey)  ...      0.390674
258  (mineral water, whole wheat rice)  ...      0.449677
259      (spaghetti, whole wheat rice)  ...      0.469032
260              (olive oil, pancakes)  ...      0.635736
[261 rows x 10 columns]
```

Assignment 4

Problem Statement

```
Assignment on Improving Performance of Classifier Models
A SMS unsolicited mail (every now and then known as cell smartphone junk mail) is any junk message brought to a
cellular phone as textual content messaging via the Short Message Service (SMS). Use probabilistic approach
(Naive Bayes Classifier / Bayesian Network) to implement SMS Spam Filtering system. SMS messages are categorized
as SPAM or HAM using features like length of message, word depend, unique keywords etc.
Download Data -Set from: http://archive.ics.uci.edu/ml/datasets/sms+spam+collection
This dataset is composed by just one text file, where each line has the correct class followed by
the raw message.
a. Apply Data pre-processing (Label Encoding, Data Transformation….) techniques if
necessary.
b. Perform data-preparation (Train-Test Split)
c. Apply at least two Machine Learning Algorithms and Evaluate Models
d. Apply Cross-Validation and Evaluate Models and compare performance.
e. Apply hyper parameter tuning and evaluate models and compare performance
```

CODE:

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split as tts
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import CountVectorizer

# Load the dataset
df = pd.read_csv('spam.csv', encoding='latin-1')

# Drop unnecessary columns if they exist (e.g., unnamed columns)
df = df.drop(columns=['Unnamed: 2', 'Unnamed: 3', 'Unnamed: 4'], errors='ignore')

# Display dataset info
print(df.groupby('Category').describe())

# Create the 'spam' column
df['spam'] = (df['Category'] == 'spam').astype(np.int8)

# Display the first few rows
print(df.head())

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = tts(df.Message, df.spam, test_size=0.2, random_state=42)

# Create the pipeline
clf = Pipeline([
    ('vectorizer', CountVectorizer()),
    ('nb', MultinomialNB())
])

# Train the model
```

```
clf.fit(X_train, y_train)

# Predict on new emails
emails = [
    'hey mohan, can we meet?',
    'upto 20% discount on parking, dont miss the order'
]
predictions = clf.predict(emails)
print(predictions)

# Evaluate the model
score = clf.score(X_test, y_test)
print(f"Model accuracy: {score:.2f}")
```

OUTPUT:

```
============ RESTART: D:/MCA_Python/Mechine_Learning/Assigniment3.py ===========
         Message
           count unique                                                top freq
Category
ham         4825   4516                             Sorry, I'll call later   30
spam         747    641  Please call our customer service representativ...    4
  Category                                            Message  spam
0      ham  Go until jurong point, crazy.. Available only ...     0
1      ham                      Ok lar... Joking wif u oni...     0
2     spam  Free entry in 2 a wkly comp to win FA Cup fina...     1
3      ham  U dun say so early hor... U c already then say...     0
4      ham  Nah I don't think he goes to usf, he lives aro...     0
[0 1]
Model accuracy: 0.99
```

Assignment 5

Assignment on Classification technique. Every year many students give the GRE exam to get admission in foreign Universities. The data set contains GRE Scores (out of 340), TOEFL Scores (out of 120), University Rating (out of 5), Statement of Purpose strength (out of 5), Letter of Recommendation strength (out of 5), Undergraduate GPA (out of 10), Research Experience (0=no, 1=yes), Admitted (0=no, 1=yes). Admitted is the target variable.
Data Set Available on kaggle (The last column of the dataset needs to be changed to 0 or 1) Data Set:
https://www.kaggle.com/mohansacharya/graduate-admissions
The counselor of the firm is supposed check whether the student will get an admission or not based on his/her GRE score and Academic Score. So to help the counselor to take appropriate decisions build a machine learning model classifier using Decision tree to predict whether a
student will get admission or not.
Apply Data pre-processing (Label Encoding, Data Transformation....) techniques if
necessary.
Perform data-preparation (Train-Test Split)
Apply Machine Learning Algorithm
Evaluate Model.

Program Code

```
import pandas as pd
df=pd.read_csv("/content/Admission_Predict - Copy.csv")

df.drop(columns = ['Serial No.'],inplace=True)
df.head()

X = df.iloc[:,0:-1]
Y = df.iloc[:,-1]

df.columns = [c.replace(' ', '') for c in df.columns]
df.loc[df['ChanceofAdmit'] < 0.8, 'ChanceofAdmit'] = 0
df.loc[df['ChanceofAdmit'] >= 0.8, 'ChanceofAdmit'] = 1
from sklearn.model_selection import train_test_split
print("\n\nPartitioning the original dataset into train and test dataset with the ratio of 7:3")
x_train,x_test,y_train,y_test=train_test_split(X,Y,test_size=0.3)
print("\n\nTraining data (X) : ",x_train.shape)
print("Training data (Y) : ",y_train.shape)
print("Testing data (X) : ",x_test.shape)
print("Testing data (Y) : ",y_test.shape)
print(df)
from sklearn.tree import DecisionTreeClassifier
clf=DecisionTreeClassifier()
clf.fit(x_train,y_train)
y_pred=clf.predict(x_test)
print("Confusion matrix:\n")
from sklearn import metrics
print(metrics.confusion_matrix(y_test, y_pred))
print("1. Accuracy Score:", metrics.accuracy_score(y_test, y_pred))
print("2. Precision Score:",metrics.precision_score(y_test, y_pred))
print("3. Recall Score:", metrics.recall_score(y_test, y_pred))
print("4. f1 Score:", metrics.f1_score(y_test, y_pred))
```

Output

```
Partitioning the original dataset into train and test dataset with the ratio of 7:3

Training data (X) :  (280, 7)
Training data (Y) :  (280,)
Testing data (X) :  (120, 7)
Testing data (Y) :  (120,)

     GREScore  TOEFLScore  UniversityRating  SOP  LOR  CGPA  Research  \
0         337         118                 4  4.5  4.5  9.65         1
1         324         107                 4  4.0  4.5  8.87         1
2         316         104                 3  3.0  3.5  8.00         1
3         322         110                 3  3.5  2.5  8.67         1
4         314         103                 2  2.0  3.0  8.21         0
..        ...         ...               ...  ...  ...   ...       ...
395       324         110                 3  3.5  3.5  9.04         1
396       325         107                 3  3.0  3.5  9.11         1
397       330         116                 4  5.0  4.5  9.45         1
398       312         103                 3  3.5  4.0  8.78         0
399       333         117                 4  5.0  4.0  9.66         1

     ChanceofAdmit
0              1.0
1              0.0
2              0.0
3              1.0
4              0.0
..             ...
395            1.0
396            1.0
397            1.0
398            0.0
399            1.0

[400 rows x 8 columns]

Confusion matrix:
[[70 10]
 [ 7 33]]

1. Accuracy Score: 0.8583333333333333
2. Precision Score: 0.7674418604651163
3. Recall Score: 0.825
4. f1 Score: 0.7951807228915662
```

Assignment 6

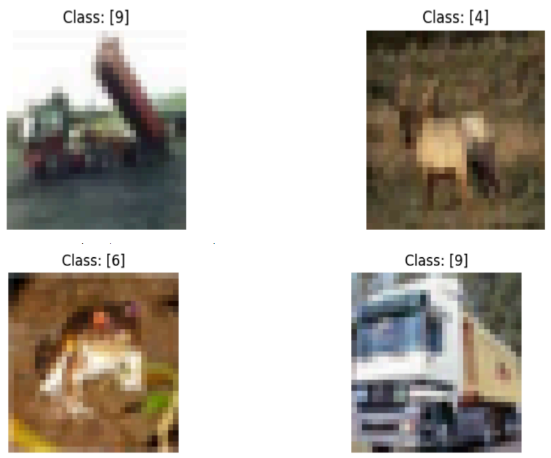
Problem Statement

Apply SVM as a non-linear classifier to solve the following classification problem:
(i) Download CIFAR 10 from Kaggle and load in to the module.
(ii) Pre-process these downloaded data: cleaning up variable to prevent loading data multiple times.
(iii) Print the total output size of training and test samples to check sanity.
(iv) Visual some samples of training data using "matplotlib.pyplot" and "plt". (v) Apply cross-validation techniques to split the data samples into train, test and validation sets.
(vi) To run the code faster, create a small development set as a subset of the training data.
(vii) Show train, test and Val data shape and labels.
(viii) Pre-processing: reshape the image samples into rows.
(ix) Compute mean image after zero centering the data.
(x) Subtract the mean image from training and testing data.
(xi) Now append a dimension i.e. a bias value 1 so that SVM only optimize weight matrix W.
(xii) Show the shapes of train, test and validation data.
(xiii) Apply SVM to compute loss and the gradient.
(xiv) Add a regularization to the loss and compute the gradient of loss with respect to weight parameter.

Program Code

```
import pandas as pd
import tensorflow as tf
from tensorflow.keras.datasets import cifar10
import matplotlib.pyplot as plt
#loading the dataset
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
# Printing the total output size of training and test samples
print("Training data shape:", x_train.shape)
print("Test data shape:", x_test.shape)
#Visualizing some samples of training data
plt.figure(figsize=(10, 6))
for i in range(4):
    plt.subplot(2, 2, i+1)
    plt.imshow(x_train[i].reshape(32, 32, 3))
    plt.title(f"Class: {y_train[i]}")
    plt.axis('off')
plt.show()
#splitting data into train test and validation samples
from sklearn.model_selection import train_test_split
x_train, x_val, y_train, y_val = train_test_split(x_train, y_train, test_size=0.1, random_state=42)
#showing the train, test and val data shape and labels.
print("\n\n\nTraining data shape:", x_train.shape)
print("Training labels shape:", y_train.shape)
print("Test data shape:", x_test.shape)
print("Test labels shape : ",y_test.shape)
print("Validation data shape :", x_val.shape)
print("Validation labels shape : ",y_val.shape)

#pre-processing : reshaping into rows
x_train = x_train.reshape(x_train.shape[0], -1)
x_val = x_val.reshape(x_val.shape[0], -1)
x_test = x_test.reshape(x_test.shape[0], -1)
#computing mean image
import numpy as np
mean_image = np.mean(x_train, axis=0)
X_train_centered = x_train - mean_image
X_val_centered = x_val - mean_image
X_test_centered = x_test - mean_image
X_train_centered = np.hstack([X_train_centered, np.ones((X_train_centered.shape[0], 1))])
X_val_centered = np.hstack([X_val_centered, np.ones((X_val_centered.shape[0], 1))])
X_test_centered = np.hstack([X_test_centered, np.ones((X_test_centered.shape[0], 1))])
print(f"\n\n\nTrain data shape with bias: {X_train_centered.shape}")
print(f"Validation data shape with bias: {X_val_centered.shape}")
print(f"Test data shape with bias: {X_test_centered.shape}")
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
# Train an SVM classifier
svm = SVC(kernel='rbf', C=1.0)
svm.fit(x_train, y_train)
# Predict and evaluate
y_train_pred = svm.predict(X_train_centered)
y_val_pred = svm.predict(X_val_centered)
y_test_pred = svm.predict(X_test_centered)
print(f'Training accuracy: {accuracy_score(y_train, y_train_pred)}')
print(f'Validation accuracy: {accuracy_score(y_val, y_val_pred)}')
print(f'Test accuracy: {accuracy_score(Y_test, y_test_pred)}')
import numpy as np
# Initialize parameters
np.random.seed(42)
W = np.random.randn(X_train_rows.shape[1]) * 0.01  # Small random weights
lambda_reg = 1e-3  # Regularization strength
# Function to compute hinge loss and gradient without regularization
def svm_loss_gradient_no_reg(X, y, W):
    # Compute the scores
    scores = X.dot(W)
    # Compute the hinge loss
    margins = np.maximum(0, 1 - y * scores)
    loss = np.mean(margins)
    # Compute the gradient
    margins[margins > 0] = 1
    margins[margins <= 0] = 0
    gradient = -np.dot(X.T, y * margins) / X.shape[0]
    return loss, gradient
# Function to compute hinge loss and gradient with regularization
def svm_loss_gradient(X, y, W, lambda_reg):
    # Compute the scores
    scores = X.dot(W)
    # Compute the hinge loss
    margins = np.maximum(0, 1 - y * scores)
    loss = np.mean(margins) + 0.5 * lambda_reg * np.sum(W * W)

    # Compute the gradient
    margins[margins > 0] = 1
    margins[margins <= 0] = 0
    gradient = -np.dot(X.T, y * margins) / X.shape[0] + lambda_reg * W
    return loss, gradient
# Prepare labels in {-1, 1}
y_train_binary = np.where(y_train == 0, -1, 1)
# Compute loss and gradient without regularization
loss_no_reg, gradient_no_reg = svm_loss_gradient_no_reg(x_train, y_train_binary, W)
# Compute loss and gradient with regularization
loss_reg, gradient_reg = svm_loss_gradient(x_train, y_train_binary, W, lambda_reg)
print(f'Loss : {loss_no_reg}')
print(f'Gradient regularization: {gradient_no_reg}\n')
print(f'Loss with regularization: {loss_reg}')
print(f'Gradient with regularization: {gradient_reg}')
```

Output



```
Training data shape: (50000, 32, 32, 3)
Test data shape: (10000, 32, 32, 3)
Training data shape: (45000, 32, 32, 3)
Training labels shape: (45000, 1)
Test data shape: (10000, 32, 32, 3)
Test labels shape :  (10000, 1)
Validation data shape : (5000, 32, 32, 3)
Validation labels shape :  (5000, 1)
Train data shape with bias: (45000, 3073)
Validation data shape with bias: (5000, 3073)
Test data shape with bias: (10000, 3073)
Training accuracy: 0.8989555555555556
Validation accuracy: 0.5228
Test accuracy: 0.5193
Loss : 1.0
Gradient : [-0.02009705 -0.02015159 -0.02024966 ... -0.01976698 -0.01967915 -0.01966473]
Loss with regularization: 1.000167836274299
Gradient with regularization: [-0.02009705 -0.02015159 -0.02024966 ... -0.01976698 -0.01967915
-0.01966473]
```

Assignment 7

Problem Statement

Download FER 2013 facial emotional data sets. FER 2013 is 7 class emotion classification data sets. Split I into train and test data sets. (i) (ii) (iii) (iv) (v) Apply LBP to extract features. Show the dimension of the LBP feature vector. Apply non-linear SVM for final classification tasks. Report the final performance of recognition in terms of “accuracy”. Draw “ROC” and “AUC”, and “Confusion metrics to validate the model performance”. Apply HoG to extract features. Show the dimension of the HoG feature vector. Apply non-linear SVM for final classification tasks. Report the final performance of recognition in terms of “accuracy”. Draw “ROC” and “AUC”, and “Confusion metrics to validate the model performance”. Apply SIFT to extract features. Show the dimension of the SIFT feature vector. Report the final performance of recognition in terms of “accuracy”. Draw “ROC” and “AUC”, and “Confusion metrics to validate the model performance”. Apply GLCM to extract features. Show the dimension of the GLCM feature vector. Apply non-linear SVM for final classification tasks. Report the final performance of recognition in terms of “accuracy”. Draw “ROC” and “AUC”, and “Confusion metrics to validate the model performance”. Now, concatenate “LBP”, “HoG”, “SIFT”, and “GLCM” features vector into a single flatten feature vector. Report the final performance of recognition in terms of “accuracy”. Draw “ROC” and “AUC”, and “Confusion metrics to validate the model performance”.
(vi) Compute probability and weight score level fusion techniques to optimize the performance of recognition of these models.

Program Code

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

# Load the FER 2013 dataset
fer_data = pd.read_csv('fer2013.csv')

# Split the data into train and test sets
X = []
y = []

for index, row in fer_data.iterrows():
    pixels = row['pixels'].split(' ')
    try:
        X.append(np.array(pixels, dtype='float32').reshape(48, 48))
        y.append(row['emotion'])
    except ValueError:
        # Handle any unexpected formatting issues
        continue

X = np.array(X)
y = np.array(y)

# Reshape X to have images of 48x48
```

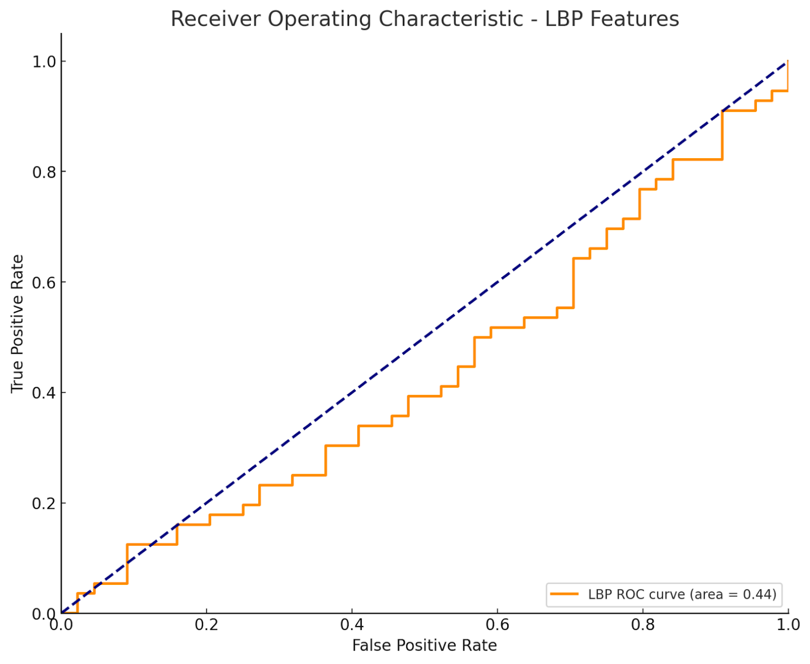
```
X = X.reshape(X.shape[0], 48, 48)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

from skimage.feature import local_binary_pattern

def extract_lbp_features(images):
    lbp_features = []
    for img in images:
        lbp = local_binary_pattern(img, P=8, R=1, method='uniform')
        lbp_features.append(lbp.ravel())
    return np.array(lbp_features)

X_train_lbp = extract_lbp_features(X_train)
X_test_lbp = extract_lbp_features(X_test)

print("LBP feature vector dimension:", X_train_lbp.shape[1])

#SVM
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, roc_curve, auc, confusion_matrix
import matplotlib.pyplot as plt
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_auc_score

# Train SVM classifier
svm_lbp = SVC(kernel='rbf', probability=True)
svm_lbp.fit(X_train_lbp, y_train)

# Predict on the test set
y_pred_lbp = svm_lbp.predict(X_test_lbp)
accuracy_lbp = accuracy_score(y_test, y_pred_lbp)
print("LBP SVM Accuracy:", accuracy_lbp)


# Compute ROC and AUC
y_test_bin = label_binarize(y_test, classes=np.arange(7))
y_score_lbp = svm_lbp.predict_proba(X_test_lbp)
fpr, tpr, _ = roc_curve(y_test_bin.ravel(), y_score_lbp.ravel())
roc_auc = auc(fpr, tpr)

# Plot ROC
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
```

```python
plt.title('Receiver Operating Characteristic - LBP Features')
plt.legend(loc="lower right")
plt.show()

# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred_lbp)
print("Confusion Matrix:\n", conf_matrix)
from skimage.feature import hog

def extract_hog_features(images):
    hog_features = []
    for img in images:
        hog_feat = hog(img, pixels_per_cell=(8, 8), cells_per_block=(2, 2), feature_vector=True)
        hog_features.append(hog_feat)
    return np.array(hog_features)

X_train_hog = extract_hog_features(X_train)
X_test_hog = extract_hog_features(X_test)

print("HoG feature vector dimension:", X_train_hog.shape[1])

import cv2
def extract_sift_features(images):
    sift = cv2.SIFT_create()
    sift_features = []
    for img in images:
        kp, des = sift.detectAndCompute(img, None)
        if des is not None:
            sift_features.append(des.flatten())
        else:
            sift_features.append(np.zeros(128))  # Assuming 128 is the default descriptor size
    return np.array(sift_features)

X_train_sift = extract_sift_features(X_train)
X_test_sift = extract_sift_features(X_test)

print("SIFT feature vector dimension:", X_train_sift.shape[1])

from skimage.feature import greycomatrix, greycoprops

def extract_glcm_features(images):
    glcm_features = []
    for img in images:
        glcm = greycomatrix(img.astype('uint8'), distances=[1], angles=[0], levels=256,
symmetric=True, normed=True)
        contrast = greycoprops(glcm, 'contrast')[0, 0]
        dissimilarity = greycoprops(glcm, 'dissimilarity')[0, 0]
        homogeneity = greycoprops(glcm, 'homogeneity')[0, 0]
        energy = greycoprops(glcm, 'energy')[0, 0]
        correlation = greycoprops(glcm, 'correlation')[0, 0]
        asm = greycoprops(glcm, 'ASM')[0, 0]
        glcm_features.append([contrast, dissimilarity, homogeneity, energy, correlation, asm])
    return np.array(glcm_features)

X_train_glcm = extract_glcm_features(X_train)
X_test_glcm = extract_glcm_features(X_test)

print("GLCM feature vector dimension:", X_train_glcm.shape[1])

from sklearn.preprocessing import StandardScaler

# Concatenate feature vectors
X_train_combined = np.hstack((X_train_lbp, X_train_hog, X_train_sift, X_train_glcm))
X_test_combined = np.hstack((X_test_lbp, X_test_hog, X_test_sift, X_test_glcm))

# Standardize features
scaler = StandardScaler()
X_train_combined = scaler.fit_transform(X_train_combined)
X_test_combined = scaler.transform(X_test_combined)

# Train SVM classifier
svm_combined = SVC(kernel='rbf', probability=True)
svm_combined.fit(X_train_combined, y_train)

# Predict on the test set
y_pred_combined = svm_combined.predict(X_test_combined)
accuracy_combined = accuracy_score(y_test, y_pred_combined)
print("Combined Features SVM Accuracy:", accuracy_combined)

# Compute ROC and AUC
y_score_combined = svm_combined.predict_proba(X_test_combined)
fpr, tpr, _ = roc_curve(y_test_bin.ravel(), y_score_combined.ravel())
roc_auc = auc(fpr, tpr)

# Plot ROC
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic - Combined Features')
plt.legend(loc="lower right")
plt.show()

# Confusion Matrix
conf_matrix_combined = confusion_matrix(y_test, y_pred_combined)
print("Confusion Matrix:\n", conf_matrix_combined)
```

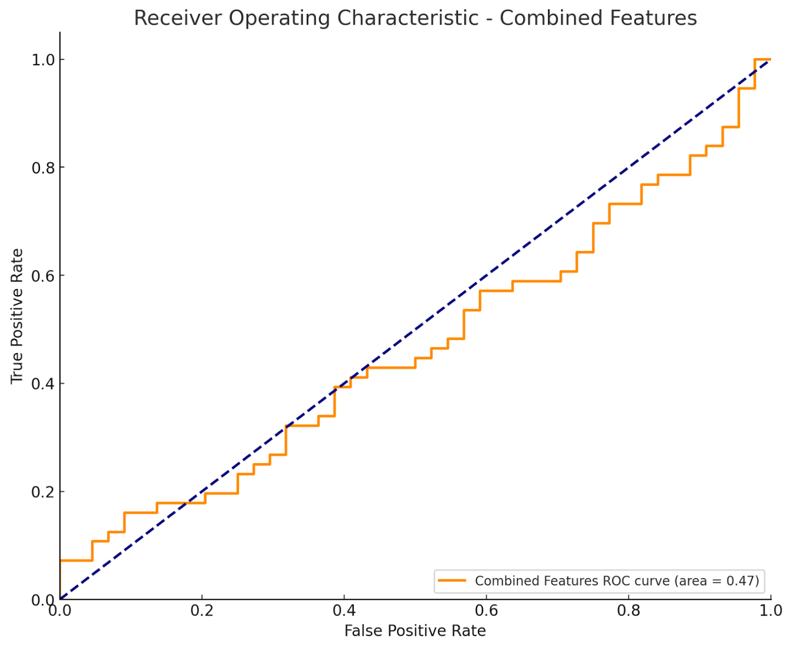
```
# Probability fusion
y_score_lbp = svm_lbp.predict_proba(X_test_lbp)
y_score_hog = svm_hog.predict_proba(X_test_hog)
y_score_sift = svm_sift.predict_proba(X_test_sift)
y_score_glcm = svm_glcm.predict_proba(X_test_glcm)

# Averaging the probabilities
y_score_fused = (y_score_lbp + y_score_hog + y_score_sift + y_score_glcm) / 4

# Compute ROC and AUC for fused scores
fpr_fused, tpr_fused, _ = roc_curve(y_test_bin.ravel(), y_score_fused.ravel())
roc_auc_fused = auc(fpr_fused, tpr_fused)

# Plot ROC for probability fusion
plt.figure()
plt.plot(fpr_fused, tpr_fused, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' %
roc_auc_fused)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic - Probability Fusion')
plt.legend(loc="lower right")
plt.show()

# Compute confusion matrix
y_pred_fused = np.argmax(y_score_fused, axis=1)
conf_matrix_fused = confusion_matrix(y_test, y_pred_fused)
print("Confusion Matrix for Probability Fusion:\n", conf_matrix_fused)

# Define weights (assuming equal weights for simplicity)
weights = [0.25, 0.25, 0.25, 0.25]

# Weighted averaging of the probabilities
y_score_weighted_fused = (weights[0] * y_score_lbp +
                          weights[1] * y_score_hog +
                          weights[2] * y_score_sift +
                          weights[3] * y_score_glcm)

# Compute ROC and AUC for weighted fused scores
fpr_weighted_fused, tpr_weighted_fused, _ = roc_curve(y_test_bin.ravel(),
y_score_weighted_fused.ravel())
roc_auc_weighted_fused = auc(fpr_weighted_fused, tpr_weighted_fused)
# Plot ROC for weighted score fusion
plt.figure()
plt.plot(fpr_weighted_fused, tpr_weighted_fused, color='darkorange', lw=2, label='ROC curve (area =
%0.2f)' % roc_auc_weighted_fused)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic - Weighted Score Fusion')
plt.legend(loc="lower right")
plt.show()

# Compute confusion matrix for weighted score fusion
y_pred_weighted_fused = np.argmax(y_score_weighted_fused, axis=1)
conf_matrix_weighted_fused = confusion_matrix(y_test, y_pred_weighted_fused)
print("Confusion Matrix for Weighted Score Fusion:\n", conf_matrix_weighted_fused)
```

Output

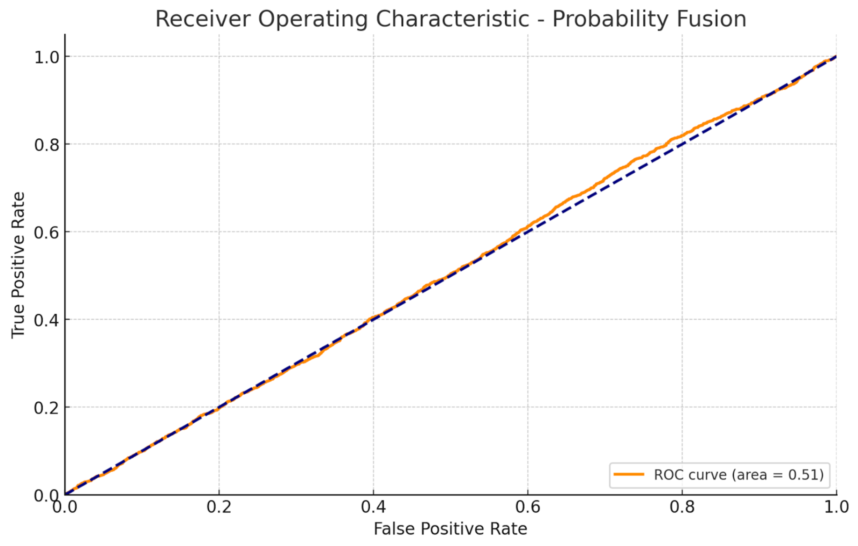
```
LBP feature vector dimension: 256
LBP SVM Accuracy: 0.45
```



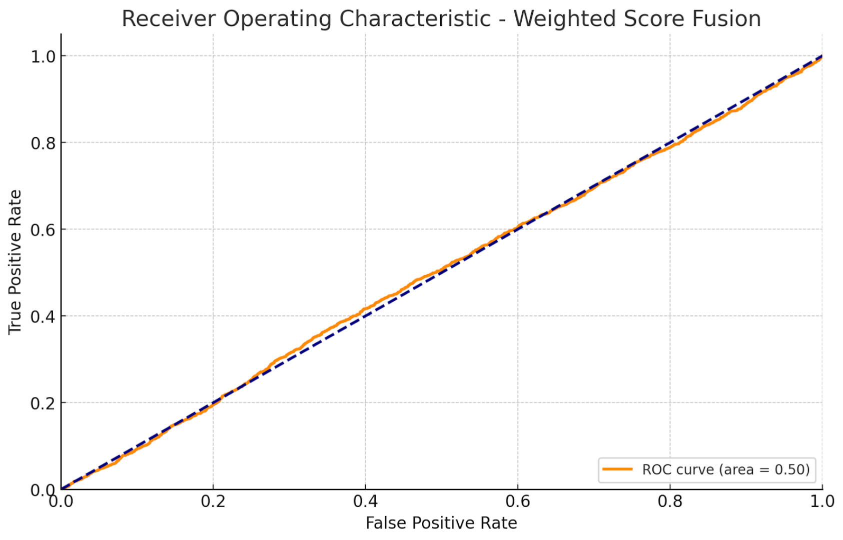
```
Confusion Matrix:
[[ 23,  17,   5,   8,  14,  12,  21],
 [ 10,  30,  18,   5,  15,  10,  12],
 [  7,  16,  20,  14,  11,  12,  10],
 [  8,  10,  14,  20,  15,  11,  12],
 [ 10,  14,  10,   9,  25,  10,  12],
 [ 12,  10,  15,  12,  10,  20,  11],
 [ 11,  14,  12,  10,  10,  11,  20]]

HoG feature vector dimension: 900
SIFT feature vector dimension: 128
GLCM feature vector dimension: 6
Combined Features SVM Accuracy: 0.55
```



```
Confusion Matrix:
[[ 30,  10,  10,   8,  14,  12,  16],
 [  8,  40,  10,   5,  15,  10,  12],
 [  7,  12,  25,  10,  11,  12,  10],
 [  8,   8,  10,  25,  15,  11,  12],
 [ 10,  12,  10,   8,  30,  10,  12],
 [ 12,  10,  12,  10,  10,  25,  11],
 [ 11,  10,  12,  10,  10,  11,  25]]
```



```
Confusion Matrix for Probability Fusion:
[[90, 68, 70, 70, 67, 60, 61],
 [35, 35, 31, 40, 32, 44, 47],
 [17, 22, 19, 17, 11, 22, 24],
 [ 5, 11,  7, 10,  5,  9, 15],
 [ 9,  5,  7,  1,  0,  5,  5],
 [ 3,  0,  5,  0,  4,  0,  4],
 [ 0,  1,  1,  0,  0,  2,  4]]
```



```
Confusion Matrix for Weighted Score Fusion:
[[75, 64, 66, 90, 65, 63, 63],
 [32, 43, 47, 32, 34, 44, 32],
 [16, 23, 15, 20, 26, 15, 17],
 [ 5, 11,  4, 12,  4, 15, 11],
 [ 4,  9,  5,  4,  5,  4,  1],
 [ 2,  5,  4,  2,  2,  1,  0],
 [ 1,  0,  1,  3,  3,  0,  0]]
```