Ira A. Fulton Schools of Engineering
ARIZONA STATE UNIVERSITY

SCHOOL OF COMPUTING AND AUGMENTED INTELLIGENCE
engineering.asu.edu/scai

PO Box 878809
Tempe, AZ 85287-8809
Ph: 480.965.3190
FAX: 480.965.2751

PROJECT PORTFOLIO COVER SHEET

Instructions: Students must complete all sections below. The completed form must be submitted to the Project Portfolio Canvas course with the project portfolio by the posted semester deadline.

Section I: Student Information

Last Name	First Name	ASU ID
Shah	Shachi	1217121828
Graduation Term	**All GPAs above 3.0?**	**Date submitted**
Summer 2025	☑ Yes ☐ No	06/28/2025

Section II: Project Information

PROJECT 1:

☑ Project worth at least 30% OR ☐ Project under 30% with additional work done

Course	Semester completed	Final grade in course
CSE 539: Applied Cryptography	Fall 2024	A

Instructor Name (has certified that the project is applicable for the portfolio)
Swathi Punathumkandi
☐ Verification of instructor approval is attached

☑ Instructor approval list has been sent to advising office

PROJECT 2:

☑ Project worth at least 30% OR ☐ Project under 30% with additional work done

Course	Semester completed	Final grade in course
CSE 575: Statistical Machine Learning	Spring 2025	A

Instructor Name (has certified that the project is applicable for the portfolio)
Yiran Luo
☐ Verification of instructor approval is attached

☑ Instructor approval list has been sent to advising office

Portfolio Summary

Shachi Shah
School of Computing and Augmented Intelligence
Arizona State University Online
Azusa, California, United States of America
spshah22@asu.edu

Abstract - This document contains a summary of two graduate-level courses—CSE 539 (Applied Cryptography) and CSE 575 (Statistical Machine Learning)—used for the completion of my Computer Science (MCS) portfolio. Each course included hands-on projects that emphasized algorithm design, implementation, and analysis. The summary highlights key topics, solutions, and skills acquired through real-world applications in cybersecurity and machine learning.

Index Terms - Applied Cryptography, Cryptanalysis, Steganography, RSA, Diffie-Hellman, MD5, AES, Machine Learning, Security, Statistical Analysis

I. CSE 539: Applied Cryptography

In CSE 539, I conducted an in-depth study of modern cryptographic techniques through four structured, hands-on projects that combined theoretical foundations with practical coding challenges. The course provided a progressive deep dive into the building blocks of secure communication systems, covering data hiding, encryption schemes, cryptographic attacks, and public-key infrastructure.

The first project focused on steganography, where I created a method to conceal a message within a Microsoft BMP image file. By manipulating the least significant bits of specific byte positions, I was able to embed the message without visually altering the original image. This demonstrated the subtle power of steganographic techniques and highlighted the importance of bitwise operations and file structure awareness in data concealment.

The second part of the project transitioned into cryptanalysis, where I implemented a brute-force attack to recover an encryption key generated by a time-based pseudo-random number generator (PRNG). The DES-encrypted ciphertext was generated within a known time window, which I exploited by systematically iterating through all possible time-derived seeds, converting them into keys, and testing decryption outcomes against the known plaintext. This exercise deepened my understanding of the relationship between key entropy and cryptographic strength, emphasizing the vulnerability of weak or predictable PRNG implementations.

I then investigated the security properties of hash functions by implementing a birthday attack against MD5. Using a one-byte salt and randomly generated alphanumeric strings, I searched for two inputs whose salted MD5 hashes shared identical prefixes (first five bytes). This project underscored how collision resistance is a critical requirement in cryptographic hash design, particularly in applications such as password storage and digital signatures.

The final two projects focused on public-key cryptography. In the Diffie-Hellman key exchange implementation, I simulated the secure establishment of a shared secret over an insecure channel by calculating modular exponentiations on large integers. The derived shared key was then formatted into a 256-bit key and used to perform AES encryption and decryption in CBC mode with a given IV. This project allowed me to observe how symmetric encryption can be integrated into hybrid cryptographic systems

II. CSE 575: Statistical Machine Learning

CSE 575 provided a hands-on exploration of fundamental machine learning algorithms, with a strong focus on clustering methods and deep learning through convolutional neural networks (CNNs). The projects were designed to not only teach theoretical algorithms but also demonstrate how performance can vary significantly based on parameter choices, initialization strategies, and data constraints.

The first major project was on K-Means clustering, a core unsupervised learning algorithm used to partition datasets into k distinct clusters. In Part 1, I implemented the standard K-Means algorithm using random initialization of cluster centers and examined how the clustering output and loss function (inertia) changed across values of k ranging from 2 to 10. The randomness of the initial centroid selection often led to variability in results, revealing the sensitivity of the algorithm to starting conditions.

To address this, Part 2 implemented a distance-based initialization heuristic. The first centroid was selected randomly, while each subsequent centroid was chosen to maximize its average distance from all previously selected centroids. This strategy resulted in significantly lower loss values and more stable clustering patterns across multiple runs. It illustrated how well-chosen initial conditions can mitigate the risk of converging to poor local minima and improve the interpretability of unsupervised learning outcomes.

The second half of the course focused on neural networks, specifically CNNs for image classification. In Part 1, I trained a CNN using the MNIST dataset of handwritten digits. I used the Keras library to build a baseline model and then explored the impact of modifying hyperparameters such as kernel size and the number of convolutional feature maps. By increasing kernel size from 3×3 to 5×5 and doubling feature maps, I was able to improve the model's test accuracy from 97.8% to 98.76%.

In Part 2, I trained a CNN on a reduced MNIST subset, selecting only four digit classes and using a limited number of training samples (2000) and test samples (400). Despite the smaller dataset, my model achieved 77.7% test accuracy after 10 epochs, showing how neural networks can remain robust even in constrained scenarios. I developed a custom evaluation function to compute per-epoch accuracy and loss, which enabled close monitoring of convergence trends and overfitting behavior. This part of the project reinforced the need for empirical model evaluation and effective regularization in small-data regimes

CSE 539: Applied Cryptography

Shachi Shah
School of Computing and Augmented Intelligence
Arizona State University Online
Azusa, California, United States of America
spshah22@asu.edu

III. Introduction

Throughout this course of Applied Cryptography, I have completed four programming projects individually that related to a multitude of aspects of the course that is necessary to the topic of cryptography.

A. Project 1- Steganography and Cryptanalysis

1) Part 1 - Steganography

Steganography is the practice of concealing messages and/or information within other nonsecret media; such as videos, images, files, or even messages. The first project, implementing an application of steganography, introduced a Microsoft bitmap image file (.bmp). I was able to conceal a message into that .bmp file without having to alter the image.

2) Part 2 - Cryptanalysis

Cryptanalysis involves breaking encrypted messages by identifying weaknesses in the encryption without having access to the key [4]. In the second part of this project, I was provided with a plaintext, ciphertext, date, and time corresponding to when the ciphertext was generated. The encryption function used a pseudo-random number generator to create a key and encrypt the plaintext. My task was to implement cryptanalysis to crack the encryption by exploiting the pseudo-random number generator's behavior.

B. Project 2 - Hash

Hash functions are functions where the output values are all the same number of bits in length, especially one used to encrypt or compress data or to generate incidences [2]. Passwords are usually stored as hash values, or hashes. The stored hash values are then compared against the entered value hash in order to verify and authenticate the inputted value. “Salt” values are added to make the hashes more secure as they are random bits that make up the uniqueness in the hash. Salts help in protection from enabling attackers from recomputing hashes to passwords, such as rainbow table attacks.

Within this project, I was given a one-byte salt value and had to perform a birthday attack to find two strings with the same first five bytes of the MD5 hashing algorithm.

C. Project 3 - Diffie-Hellman and Encryption

The Diffie-Hellman key exchange is a digital encryption technique that allows two parties to securely exchange cryptographic keys over a public channel without transmitting their communication across the internet [1]. This process is often illustrated using the Alice and Bob analogy, as follows:

1) Alice and Bob agree on values for g and N, where N is a prime number and g is a primitive root modulo N.
2) Alice selects a secret integer x, then gives Bob $X = g^x \bmod N$.
3) Bob selects a secret integer y, then gives Alice $Y = g^y \bmod N$.
4) Alice calculates the shared secret s using the equation $s = Y^x \bmod N$.
5) Bob calculates the shared secret s using the equation $s = X^y \bmod N$.
6) Now, Alice and Bob have the same secret, s.

I was provided with g, N, x, and $g^y \bmod N$ (base 10), along with an encrypted text and its corresponding plaintext for this project. The objective was to compute the key, decrypt the given encrypted message, and then re-encrypted the decrypted message using 256-bit AES encryption.

D. Project 4 - RSA

The Rivest-Shamir-Adleman (RSA) encryption algorithm is an asymmetric encryption algorithm that is widely used in many products and services [3]. RSA is vital in cryptography because it allows users to secure messages before they send them. For my project, I was given p and q, a ciphertext, and a plaintext to implement the RSA algorithm. The goal similarly to the prior project, I was to calculate the key, decrypt the given ciphertext. and encrypt the given plaintext.

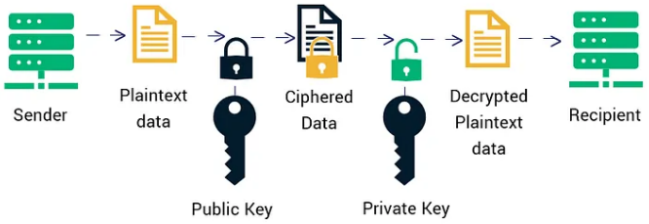

Fig. 1 - RSA Algorithm

The RSA Key generation works as follows (see Fig. 1):

1) Choose two large prime numbers: p and q.
2) Compute n by multiplying the two prime numbers to get n, where $n = p \times q$.
3) Calculate Euler’s Totient Function $\varphi(n)$, where it’s calculated as: $\varphi(n) = (p - 1) \times (q - 1)$.
4) Select an integer e where that $1 < e < 1\ \varphi(n)$ and $gcd(\varphi(n), e) = 1$.
5) Compute the decryption exponent d, using the extended euclidean algorithm such that $e \times d = 1 \bmod \varphi(n)$.
6) Now have the generated public key $\{d, n\}$ for encryption and the private key $\{n, d\}$ for decryption.

Then, in order to encrypt, you would have to represent the message as an integer such that $0 \leq m < n$ and encrypt the message with the formula $c = m^e \bmod n$. This would then transform the plaintext m into ciphertext c.

IV. Explanations of Solutions

A. Project 1

1) Part 1 - Steganography

In this solution to the steganography problem, my goal was to hide or reveal data within a bitmap image by modifying its bytes. I started by reading the hidden data provided as a command-line argument, parsing it as hexadecimal values, and storing it in a byte array. The bitmap image bytes were predefined in the program, and I set the starting index to skip over the image header, ensuring that the metadata remained unaffected.

The main logic involved processing each byte of the hidden data. I extracted 2 bits at a time from each data byte and XORed them with the corresponding 4 bytes in the bitmap. This allowed me to subtly alter the image data, embedding the hidden message without causing significant visual changes to the image.

```
for (int i = 0; i < hiddenData.Length; i++) {
        byte dataByte = hiddenData[i];
        for (int j = 0; j < 4; j++) {
           byte bits = (byte)((dataByte >> (6 - 2 * j)) & 0x03);
           bmpBytes[startIndex + i * 4 + j] ^= bits;
        }
    }
```

Finally, I printed the modified bitmap bytes as a sequence of hexadecimal values, showing how data can be embedded in an image by changing its least significant bits. This approach demonstrates the use of bitwise operations in steganography for concealing data within an image.

2) Part 2 - Cryptanalysis

In this cryptanalysis solution, my objective was to perform a brute-force attack on an encrypted message to recover the secret key used for encryption. The approach leverages a known time window during which the encryption key was generated. My task was to iterate through all possible time-based keys within this window and attempt decryption to find the correct one.

I first set up the program to accept two command-line arguments: the plaintext (expected original message) and the ciphertext (the encrypted message). I then defined the start and end times for the encryption window, calculating the total number of minutes between these two points. This gave me a clear range of possible encryption keys to test.

```
{ …
 DateTime startTime = new DateTime(2020, 7, 3, 11, 0, 0);
  DateTime endTime = new DateTime(2020, 7, 4, 11, 0, 0);
     TimeSpan timeSpan = endTime - startTime;
     int totalMinutes = (int)timeSpan.TotalMinutes;
     for (int i = 0; i <= totalMinutes; i++)
     {
       DateTime dt = startTime.AddMinutes(i);
       TimeSpan ts = dt.Subtract(new DateTime(1970, 1,
1));
        int seed = (int)ts.TotalMinutes;
        byte[] key = BitConverter.GetBytes(new
Random(seed).NextDouble());
// Attempt to decrypt the ciphertext using the generated key
if (Decrypt(ciphertext, key) == plaintext)
{
   // If decryption matches the plaintext, print the seed and
exit
    Console.WriteLine(seed);
     return;
}
```

The brute-force process involved iterating over every possible minute within the time frame. For each iteration, I calculated the total number of minutes since the Unix epoch (January 1, 1970) for the current time. I used this value as a seed to generate the encryption key, simulating how the original key might have been derived.

For each generated key, I attempted to decrypt the ciphertext. If the decryption result matched the provided plaintext, it meant I had successfully found the key, and I printed the seed used to generate it. If no matching key was found by the end of the loop, the program indicated that the key was not discovered within the provided time frame.

The decryption process itself involved converting the Base64-encoded ciphertext into bytes and using the DES (Data Encryption Standard) algorithm to attempt decryption with the generated key. I set up a memory stream and a crypto stream to process the decryption, and once complete, I compared the decrypted output to the provided plaintext to determine if the correct key was used.

B. Project 2 - Hash

In my hash project, I implemented a solution to perform a birthday attack on MD5 hashes with a specific one-byte salt. The program starts by validating that the user provides a valid one-byte salt as a command-line argument. After conversion of the salt from a hexadecimal string to a byte, the program searches for two different strings that produce the same first five bytes of their MD5 hashes with the given salt.

```
if (args.Length != 1 || args[0].Length != 2)
      {
         Console.WriteLine("Please provide a valid 1-byte
salt as a command line argument.");
         return;
      }
      byte salt = Convert.ToByte(args[0], 16);
      var collision = FindCollision(salt);
      if (collision != null)
      {

Console.WriteLine($"{collision.Item1},{collision.Item2}");
      }
      else
      {
         Console.WriteLine("Collision not found.");
      }
```

The main logic is in the FindCollision method, which generates random 8-character alphanumeric strings and computes their MD5 hashes with the salt. It then checks if the first five bytes of the hash match any previously stored hash prefixes using a dictionary. If a match is found, it returns the two strings that collided.

The ComputeMD5HashWithSalt method calculates the MD5 hash of a string combined with the salt, converting the result to a hexadecimal string. The GenerateRandomString method creates random alphanumeric strings for testing. This approach demonstrates the birthday attack by identifying collisions in hash prefixes efficiently.

```
public static string ComputeMD5HashWithSalt(string input, byte salt)
   {  using (MD5 md5 = MD5.Create())
      {
         byte[] inputBytes = Encoding.UTF8.GetBytes(input);
         byte[] saltedInputBytes = new byte[inputBytes.Length + 1];
         Buffer.BlockCopy(inputBytes, 0, saltedInputBytes, 0, inputBytes.Length);
         saltedInputBytes[saltedInputBytes.Length - 1] = salt;
         byte[] hashBytes = md5.ComputeHash(saltedInputBytes);
         StringBuilder sb = new StringBuilder();
         foreach (byte b in hashBytes)
         {
            sb.Append(b.ToString("X2"));
         }
         return sb.ToString();
      }
   }
```

C. Project 3 - Diffie-Hellman and Encryption

In my Diffie-Hellman and encryption project, I developed a solution that combines the Diffie-Hellman key exchange with AES encryption and decryption. The program first parses command-line arguments to retrieve the initialization vector (IV), parameters for the Diffie-Hellman key exchange, and the ciphertext and plaintext for encryption and decryption.

The Diffie-Hellman process begins by calculating the base g and modulus N using the provided exponents and constants. With these, I compute the shared key by raising g^y mod N to the power of x modulo N. This shared key is then used to derive a 256-bit AES key by resizing it to 32 bytes.

```
BigInteger sharedKey = BigInteger.ModPow(gyModN, x, N);
byte[] aesKey = sharedKey.ToByteArray();
Array.Resize(ref aesKey, 32);
```

The solution performs decryption of the provided ciphertext using AES-256 in CBC mode, applying the derived key and IV. It then encrypts the given plaintext with the same key and IV. The results, including the decrypted plaintext and the newly encrypted ciphertext, are output in a formatted manner.

The methods DecryptAES and EncryptAES handle the AES operations, ensuring the proper use of CBC mode and PKCS7 padding. Additionally, the ConvertHexStringToByteArray method assists in converting hexadecimal input to byte arrays for cryptographic processing. This approach integrates cryptographic techniques to securely exchange and process data.

```
static byte[] ConvertHexStringToByteArray(string hex)
{
   hex = hex.Replace(" ", ""); // Remove any spaces
   byte[] bytes = new byte[hex.Length / 2];
   for (int i = 0; i < hex.Length; i += 2)
   {
      bytes[i / 2] = Convert.ToByte(hex.Substring(i, 2), 16);
   }
   return bytes; // Return the byte array
}
```

D. Project 4 - RSA

For my RSA project, I implemented a solution to handle RSA encryption and decryption by first calculating the necessary cryptographic components based on given parameters. The program begins by parsing command-line arguments to retrieve the exponents and constants for calculating the prime numbers p and q, as well as the ciphertext and plaintext.

```
BigInteger ciphertext = BigInteger.Parse(args[4]);
BigInteger plaintext = BigInteger.Parse(args[5]);
```

I calculated the prime numbers p and q using the formula $2^e - c$ where e and c are provided as arguments. Using these primes, I computed n as the product of p and q, and $\varphi(n)$ as $(p-1)\times(q-1)(p-1)$ \times $(q-1)(p-1)\times(q-1)$. With e set to 65537, a common choice for the public exponent, I then determined the private exponent d using the modular inverse of e modulo $\varphi(n)$, which was computed through the Extended Euclidean Algorithm.

The RSA encryption formula $m^e \bmod n$ was used to encrypt the plaintext, while the decryption formula $c^d \bmod n$ was used to decrypt the ciphertext. The results are output as a comma-separated pair of the decrypted plaintext and the encrypted ciphertext.

The ModInverse method calculates the modular inverse of e using the Extended Euclidean Algorithm to ensure the proper decryption of the ciphertext. The ExtendedGCD method finds the greatest common divisor (GCD) of two numbers and the coefficients needed for the modular inverse. This approach integrates fundamental RSA principles to secure and process data efficiently.

V. Description of Results

A. Project 1 - Steganography and Cryptanalysis

1) Part 1 - Steganography

I had a bitmap made up of the bytes:

```
00 00 1A 00 00 00 0C 00 00 00 04 00 04 00 01 00 18 00 00 00 FF FF FF FF 00 00 FF FF FF FF FF FF FF 00 00 00 FF FF FF 00 00 00 FF 00 00 FF FF FF FF 00 00 FF FF FF FF FF FF 00 00 00 FF FF FF 00 00 00
```

I was able to conceal a message made up of the following bytes:

```
40 C4 D6 38 6F 52 C0 65 F9 EA 7A E5
```

Then generated an identical bitmap made up of the following bytes:

```
42 4D 4C 00 00 00 00 00 00 00 1A 00 00 00 0C 00 00 00
```

04 00 04 00 01 00 18 00 01 00 FF FF FC FF 01 00 FC FE FE FD FF FC FD 00 01 02 FC FC FE 01 00 02 FC 00 00 FF FE FD FE 01 03 FC FD FE FC FD FD 02 01 03 FD FD FC 02 01 01

This successfully contained my secret message.

2) Part 2 - Cryptanalysis

From running my program with the plaintext: “fHQ7ZEMO” and ciphertext: “VlRIuero1hGGtEgdNwvwYQ==” generated my expected seed output of “26562900”.

B. Project 2 - Hash

Running my program with the salt byte "C5" produced two strings, "KX9R3MO8" and "an9yFXRn," which resulted in identical salted hashes.

C. Project 3 - Diffie-Hellman and Encryption

For my encrypted message that was generated within the IDE, my program correctly decrypted it. Comparably, the program encrypted the given message:

d4cNqFsXRBIoU4n2bMZFfdB6rmF6rpeDUZreeeqLZg4FH

To the expected bytes:

FC FE 3F FB DE C9 9A E1 06 9C C7 23 2C AC 6F 0D BB 9D 76 D7 05 4F B5 5C C5 87 F1 54 DC 93 25 21 96 A2 A3 DB F8 46 82 4F AF E3 01 0D 48 55 AD 36

D. Project 4 - RSA

My program correctly decrypted the following value:

836791040043938201935305887215499508325700459633835106325734292585523284902502882786124466883649080318019576515661231677241610773770352816836423

5975266

To the following decrypted plaintext:

836791040043938201935305887215499508325700459633 2113912517

VI. Skills and Knowledge Acquired

Each project provided valuable insights into various aspects of cryptography. In Project 1, I gained hands-on experience with steganography, cryptanalysis, and random number generators. Project 2 focused on modern hash functions, where I learned how to detect collisions using a birthday attack. Project 3 involved using a 256-bit key for secure key exchange and encryption. Finally, Project 4 taught me how to implement the Extended Euclidean Algorithm and utilize the RSA algorithm for encryption and decryption.

References

[1] Gillis, Alexander. “Diffie-Helllman key exchange (exponential key exchange),” *TechTarget*, 2020. pp. 1-2. [Online]. Available: https://www.techtarget.com/searchsecurity/definition/Diffie-Hellman-key-exchange

[2] Grassi, Paul. “Digital Identity Guidelines,” *NIST Special Publication 800-63-3,* 2007. pp. 47. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-63-3.pdf

[3] Herring, Cardon. “Rivest, Shamir Adleman Encryption Algorithm - How RSA Encryption Works?” *Encryption Consulting LLC*, 2022. pp. 1-2. [Online]. Available: https://www.encryptionconsulting.com/education-center/what-is-rsa/

[4] Swenson, Christopher. “Modern Cryptanalysis: Techniques for Advanced Code Breaking,” *Wiley Publishing, Inc*, 2008. pp. xix-xxviii. [Online]. Available: https://books.google.com/books?hl=en&lr=&id=oLoaWgdmFJ8C&oi=fnd&pg=PP1&dq=what+is+cryptanalysis&ots=T8hGfVKggV&sig=ARbD7vQpLcbKgM6ZR_eRufhzPbk#v=onepage&q=what%20is%20cryptanalysis&f=false.

CSE 575: Statistical Machine Learning

Shachi Shah
School of Computing and Augmented Intelligence
Arizona State University Online
Azusa, California, United States of America
spshah22@asu.edu

I. Introduction

The two main projects covered in this portfolio include the K-Means-Strategy Project and Classification Using Neural Networks and Deep Learning Project. Each project consisted of two parts, each with unique objectives and implementations. This portfolio outlines the project descriptions, solution approaches, results, and skills acquired during the course of these projects. Through these projects, we explored fundamental clustering techniques, convolutional neural networks (CNNs), and optimization strategies that form the backbone of modern machine learning applications. By analyzing performance metrics and adjusting hyperparameters, we gained insights into improving model efficiency and effectiveness.

A. Project 1 - K-Means Strategy Project

1) Part 1 - Random Initialization

K-Means is a widely used clustering algorithm in the field of data mining across different disciplines in the past fifty years [Qi, 1]. The objective of this project was to implement the K-Means clustering algorithm and apply it to a dataset of 2-D points. In this part, the cluster centers were chosen randomly from the dataset, and the implementation was tested for cluster sizes ranging from 2 to 10. The centroids and the loss function values were computed for each cluster count.

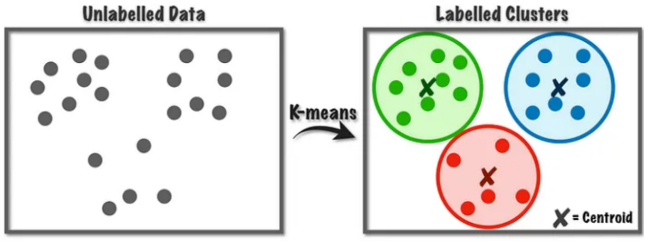

Fig. 1 - K-Means Clustering

The algorithm aimed to minimize intra-cluster variance by iteratively updating the cluster centers (see Fig. 1). Since the initial selection of cluster centers significantly affects the final result, this part explored the impact of random initialization on the clustering outcome.

The K-Means clustering algorithm works as follows from article:

1) Choose the number of clusters (K).
2) Initialize cluster centers - randomly select *K* points (centroids) from the dataset as the initial cluster centers.
3) Assign data points to the nearest cluster - each data point is assigned to the cluster whose centroid is the closest (using distance metrics like Euclidean distance).
4) Recalculate centroids - compute the new centroid (average position) for each cluster based on the points assigned to it.
5) Repeat until convergence. Steps 3 and 4 are repeated until the centroids no longer change significantly, meaning the clusters are stable.

2) Part 2 - Distance-Based Initialization

One of the most popular heuristics for solving the k-means problem is based on a simple iterative scheme for finding a locally minimal solution [Tapas, 100]. The second part improved upon the first by implementing a more informed centroid selection strategy. The first centroid was chosen randomly, while subsequent centroids were selected based on the maximum average distance from previously chosen centroids. The evaluation was done for cluster sizes ranging from 2 to 10.

B. Project 2 - Classification Using Neural Networks

1) Part 1 - Baseline CNN Model

Adaptive non-parametric neural-net classifiers work well for many real-world problems (Lippman, 47]. The goal was to analyze and modify a Convolutional Neural Network (CNN) for classifying handwritten digits from the MNIST dataset. The baseline model was tested, and experiments were conducted by varying kernel sizes and the number of feature maps in convolutional layers.

This experiment aimed to understand the impact of network architecture modifications on accuracy and loss.

2) Part 2 - Evaluation of CNN on Subset of MNIST

The MNIST dataset is well known in introducing machine learning for several reasons (i.e. to classify irregularities) [Palvanov, 128]. This part focused on training and evaluating a CNN on a subset of MNIST, selecting four random categories with 2000 training samples and 400 test samples. The goal was to optimize model performance while understanding CNN principles.
algorithm.

II. Explanations of Solutions

A. Project 1 - K-Means Strategy Project

1) Part 1 - Random Initialization

The implementation involved initializing cluster centers by randomly selecting *k* points from the dataset. The algorithm iteratively assigned points to the nearest centroid, recalculated the centroid positions, and minimized the loss function until convergence.

The following key functions were implemented:

- `initial_point_idx(id, k N)`: Generates a set of *k* random indices to select cluster centers.

- `init_point(data, idx)`: Extracts data points corresponding to the indices.
- `initial_S1(id, k)`: Uses a seeded random selection method to initialize the centroids.

```
# Compute final centroids for k in range 2
to 10
for k in range(2, 11):
    kmeans = KMeans(n_clusters=k,
init=initial_centers[k], n_init=1,
random_state=0)
    kmeans.fit(data)
    final_centroids[k] =
kmeans.cluster_center
# Compute loss (inertia) for k in range 2 to
10
for k in range(2, 11):
    kmeans = KMeans(n_clusters=k,
init=initial_centers[k], n_init=1,
random_state=0)
    kmeans.fit(data)
    loss_values[k] = kmeans.inertia_
```

This method ensures each run has a unique et reproducible initialization, which allowed me to assess how the random selection influenced the final clusters.

2) Part 2 - Distance Based Initialization

A modified initialization function was introduced to ensure better centroid separationL

- `initial_point_idx2(id, k, N)`: Selects the first centroid randomly and subsequent centroids using a distance-based heuristic.
- `initial_S2(idx, k)`: Implemented the distance-maximizing strategy to spread initial centroids more effectively.

The clustering process followed the same iterative refinement as in *Part 1*, but with an improved starting condition that helped mitigate local optima issues.

```
def initial_point_idx2(id,k, N):
    random.seed((id+k))
    return random.randint(0,N-1)

def initial_S2(idx,k):
    # print("Strategy 2: k and initial
points")
    i = int(idx)%150
    random.seed(i+800)
    init_idx2 = initial_point_idx2(i,
k,data.shape[0])
    init_s2 = data[init_idx2,:]
    return init_s2
```

B. Project 2 - Classification Using Neural Networks

1) Part 1 - Random Initialization

A CNN was implemented using Keras, consisting of convolutional layers followed by max pooling, fully connected layers, and a softmax output layer. The optimizer used was Adadelta, and the model was trained for 12 epochs.

Modifications included:

- Increasing the kernel size from 3x3 to 5x5.
- Increasing the number of feature maps in convolutional layers.

Each experiment was analyzed by tracking training loss, validation loss, and accuracy metrics.

```
Final Results Summary:
Baseline Model - Test Accuracy: 0.9782
Kernel Size 5x5 - Test Accuracy: 0.9806
Increased Feature Maps - Test Accuracy:
0.9876
```

2) Part 2 - Distance-Based Initialization

The CNN architecture was similar to Part 1, with modifications to accommodate the smaller dataset. The model was trained for 10 epochs, and accuracy and loss metrics were tracked.

The evaluation function `evaluate(net, images, labels)` was implemented to compute loss and accuracy over both training and test datasets. The results were visualized to analyze convergence trends.

```
def evaluate(net, images, labels):
    acc = 0
    loss = 0
    batch_size = 1  # Evaluating one sample
at a time
    for batch_index in range(0,
images.shape[0], batch_size):
        # Extract single image and label
        x = images[batch_index]
        y = labels[batch_index]
        # Forward pass through the network
        for layer in net.layers:
            x = layer.forward(x)
        # Compute loss using cross-entropy
        loss += cross_entropy(x, y)
        # Compute accuracy by comparing
predicted label with true label
        if np.argmax(x) == np.argmax(y):
            acc += 1
    # Normalize loss by the number of
samples
    loss /= images.shape[0]
    # Compute accuracy as a percentage
    acc /= images.shape[0]
    return acc, loss
evaluate
```

III. Description of Results

A. Project 1 - K-Means Strategy Project

1) Part 1 - Random Initialization

The results showed that as the number of clusters increased, the loss function decreased. However, the diminishing returns indicated that beyond a certain k, adding more clusters did not significantly reduce loss.

Using different random seeds resulted in slightly different clustering structures, reinforcing the sensitivity of the method to initial conditions. This provided motivation for exploring more robust initialization strategies.

2) Part 2 - Distance-Based Initialization

This method produced better clustering performance, as evidenced by lower loss values compared to Part 1. The improved initialization reduced the likelihood of poor local minima and resulted in more stable clustering patterns across different runs.

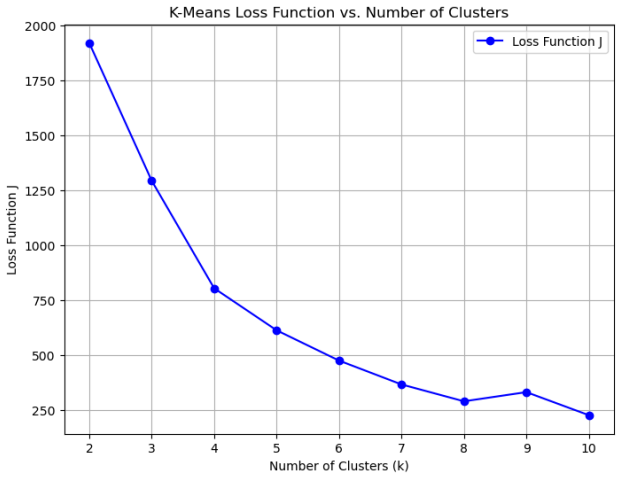

Fig. 2 - K-Means Loss Function vs. Number of Clusters

By analyzing the loss function trends (see Fig. 2), we confirmed that this strategy consistently led to better-separated clusters, highlighting the importance of intelligent initialization in K-Means clustering.

B. Project 2 - Classification Using Neural Networks

1) Part 1 - Random Initialization

The baseline model achieved an accuracy of 97.82%. Increasing the kernel size improved accuracy slightly, while increasing feature maps provided the best performance improvement, reaching 98.76% accuracy.

Graphs of accuracy and loss trends confirmed that adding feature maps led to better feature extraction and improved generalization.

2) Part 2 - Distance-Based Initialization

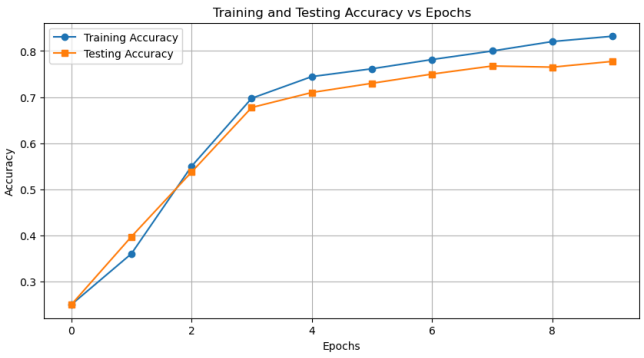

Fig. 3 - Training and Testing Accuracy vs. Epochs

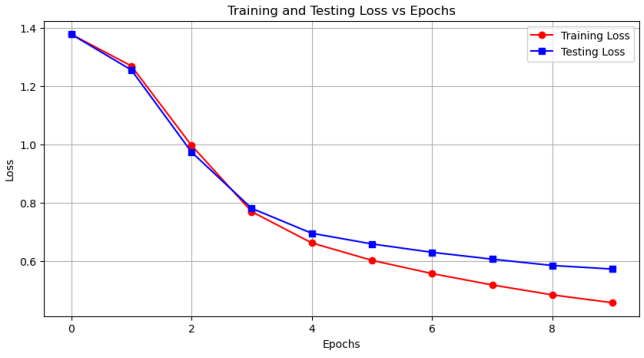

Fig. 4 - Training and Testing Loss vs. Epochs

The results were as follows:

- Initial Accuracy: 25% (random prediction level).
- Final Test Accuracy: 77.7% after 10 epochs.
- Final Training accuracy: 83.2%.
- The model exhibited stable improvement across epochs, demonstrating effective learning.

Graphs of accuracy and loss trends further illustrated the model's learning progression (see Fig. 3 and Fig. 4), showing steady optimization over time.

IV. Skills and Knowledge Acquired

Throughout this learning experience, I have gained practical knowledge and hands-on expertise in various aspects of machine learning and deep learning. One of the key areas of focus was machine learning algorithms, particularly K-Means clustering. I explored its real-world applications and different initialization strategies, improving my understanding of how data can be effectively grouped and analyzed.

In the domain of deep learning, I developed experience in designing and training convolutional neural networks (CNNs) for image classification tasks. This involved working with complex architectures, understanding feature extraction, and fine-tuning models to achieve better accuracy. The process provided deeper insight into the functioning of neural networks and their applications in computer vision.

Additionally, I explored optimization techniques, focusing on how hyperparameter tuning effects model performance. By experimenting with various parameters such as learning rates, activation functions, and batch sizes, I learned how small adjustments can significantly impact the efficiency and accuracy of a model. This knowledge is crucial for building robust machine learning models.

My data analysis skills improved as I worked with clustering results and visualized neural network training progress. I became more proficient in interpreting loss curves, accuracy trends, and cluster distributions, helping me make informed decisions about model adjustments and refinements.

Finally, I enhanced my programming skills, particularly in Python, and deepened my expertise in various packages for implementing machine learning models. Working extensively with these tools strengthened my ability to write efficient code, debug issues, and optimize machine learning workflows. These technical skills are essential for tackling real-world problems in AI and data science.

References

[1] Palvanov, Akmaljon. "Comparisons of Deep Learning Algorithms for MNIST in Real-Time Environment." *International Journal of Fuzzy Logic and Intelligent Systems,* 2018. pp.126-134. [Online]. Available: https://www.ijfis.org/journal/view.html?volume=18&number=2&spage=126.

[2] Qi, Jainpeng. "K*-Means: An Effective and Efficient k-Means Clustering Algorithm," *IEEE Xplore,* 2016. pp.xii-xiv. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7723700.

[3] R.P. Lippmann. "Pattern classification using neural networks." *IEEE Xplore*, 1989. pp. 47-50. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/41401.

[4] Tapas, Kanungo. "The analysis of a simple k-means clustering algorithm," *ACM Digital Library*, 2000. pp. 100. [Online]. Available: http://dl.acm.org/doi/abs/10.1145/336154.33618