

Deep Learning Project Signatures Verification

DL_Project.ipynb

Project report submitted By

Nayan Goyal – 21UCS137

Rajdeep Tiwari – 21UCS166

Sahil Sidana – 21UCS177

Course Instructors

Dr. Indra Deep Mastan

Dr. Preety Singh

Abstract

Artificial intelligence and machine learning are developing at a rapid pace, which has produced creative applications across a range of industries. The use of Siamese networks for human signature authentication is one such application. The goal of this project is to construct a system for verifying signatures. To do this, a collection of (name, signature) pairs will be learned, and the authenticity of a particular signature linked to a certain name will then be ascertained.

Siamese Networks, a particular kind of neural network architecture intended for comparing and categorizing input pairs, are used in this project. This method allows the system to reliably and accurately verify signatures by capturing the distinct characteristics and patterns included in each signature.

Introduction

In an era where digital transactions and document processing are commonplace, it is critical to have effective and safe identity verification techniques. Our study, Signature Verification using Siamese Networks focuses on the authentication of human signatures, which is a crucial component of identity verification.

Problem Statement:

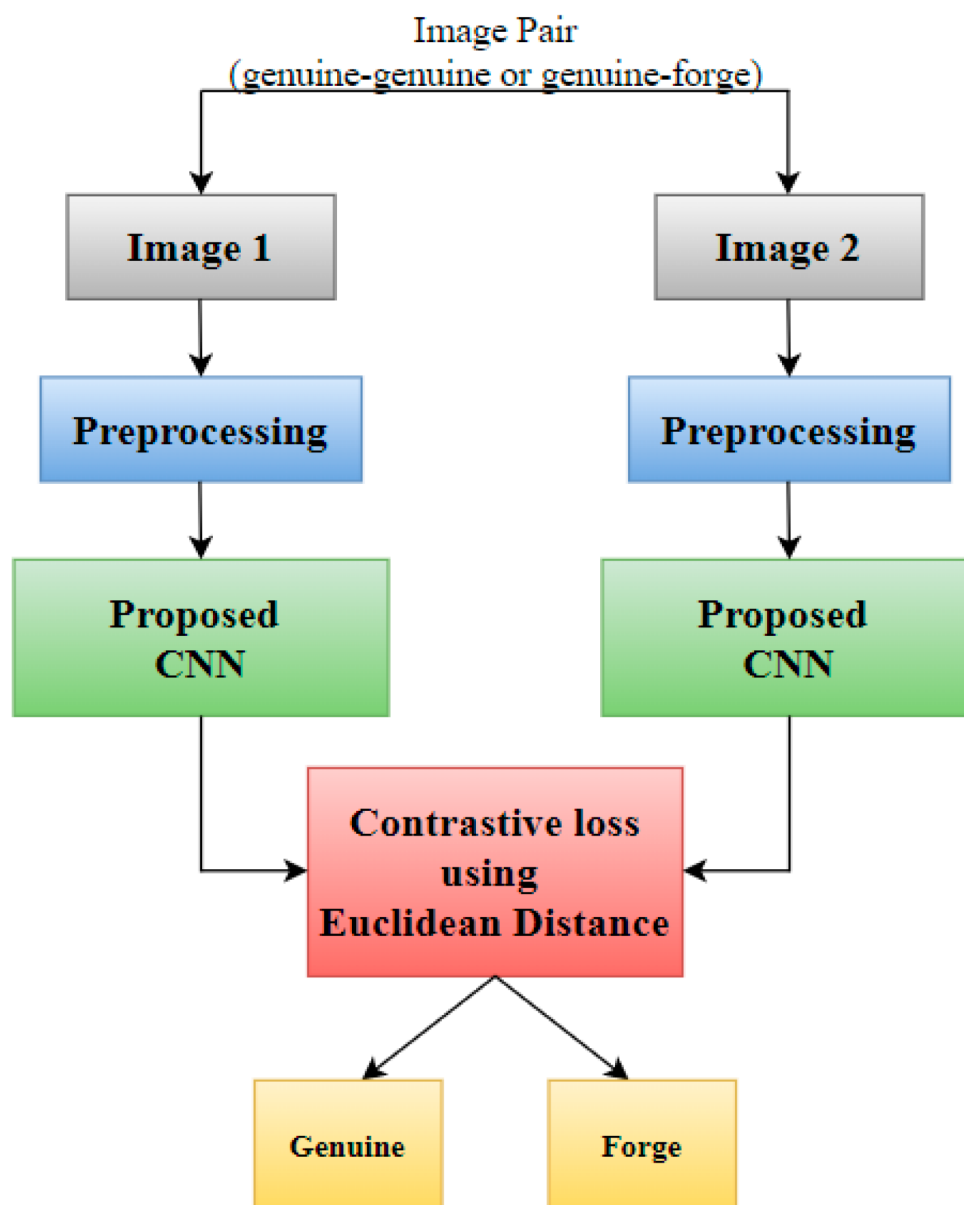
The susceptibility of signature-based authentication systems to forgeries and illegal access is the primary issue that our project seeks to resolve. When it comes to reliably identifying authentic signatures from forged ones, traditional signature verification techniques frequently fall short. This is a serious risk because a number of industries are using signatures to verify identification such as finance, legal documents, and access control systems.

Context of the Problem:

In the context of financial transactions, a forgery could result in financial fraud, unauthorized access to accounts, and a confidentiality violation. Similar to this, maintaining the integrity of contracts and agreements in legal situations depends on the legitimacy of signed papers. To reduce these dangers and guarantee the validity of

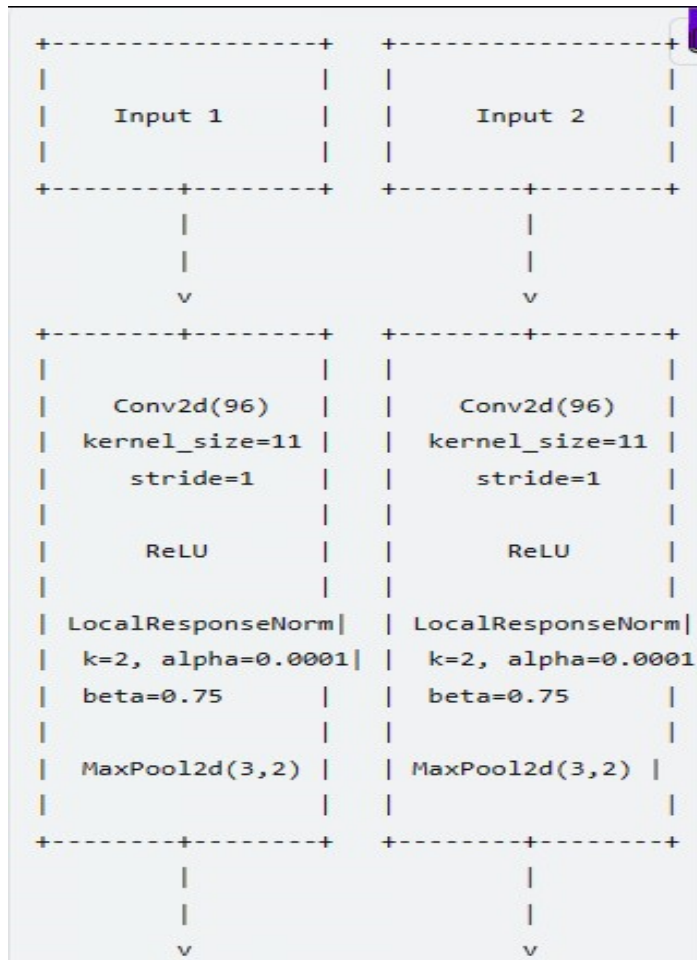
identity validation procedures, a precise and trustworthy signature verification system is therefore essential.

Solution Overview: Siamese Networks are a type of neural network architecture specifically made for comparing and classifying pairs of inputs. Our suggested approach makes use of these given below networks' capabilities. The Siamese Network is trained on a dataset including pairs of authentic (name, signature) samples in the context of signature verification. Through this training, the network is able to discover the distinctive qualities and patterns linked to every person's signature. Then, with a name and a signature, the system can use the learn patterns to verify the validity of the given signature.



Siamese Network Architecture :

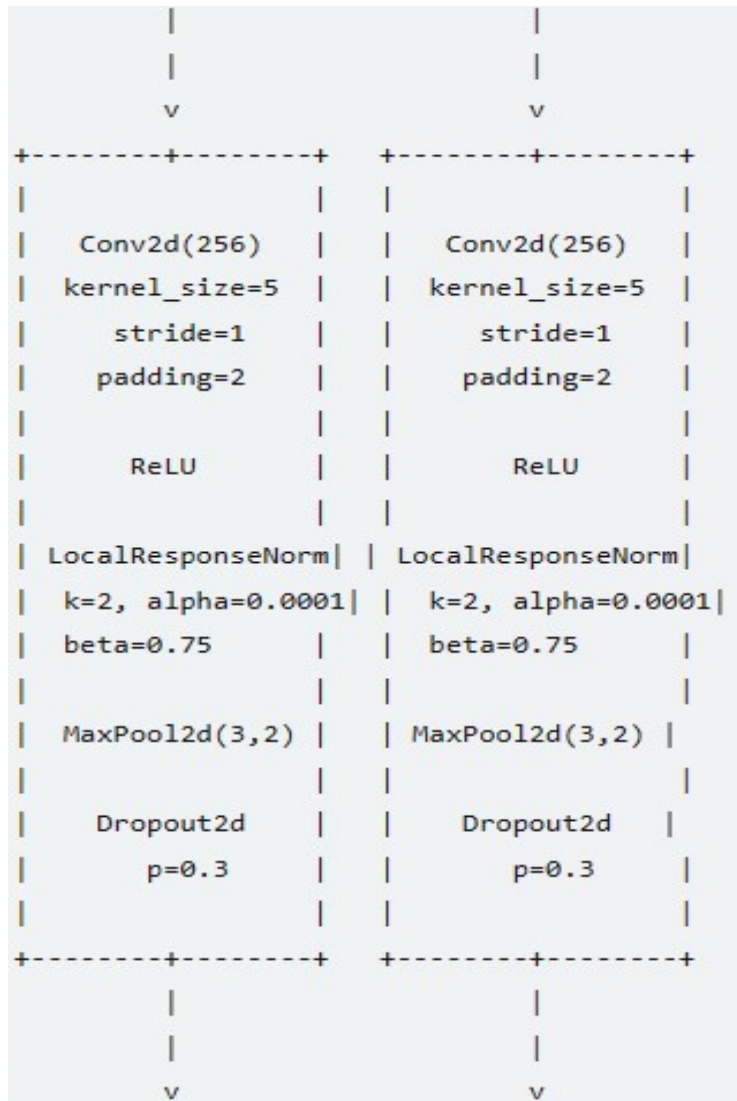
Part 1:



1. Input 1 & Input 2: These are the initial inputs to the network. These are the images that are to be processed by the network.
2. Conv2d(96), kernel_size=11, stride=1: This is a 2D convolutional layer with 96 output channels (i.e., the number of filters in the convolution). The kernel size is 11, meaning the filters are 11x11 pixels, and the stride is 1, which is the step size for moving the filters across the input.
3. ReLU: This is a Rectified Linear Unit activation function. It introduces non-linearity into the model by replacing all negative pixel values in the feature map with zeros.
4. LocalResponseNorm, k=2, alpha=0.0001, beta=0.75: This is a Local Response Normalization layer. It increases generalization of CNNs by using the squared sum of inputs within a local neighborhood.

5. MaxPool2d(3,2): This is a 2D max pooling layer with a kernel size of 3 and a stride of 2. It reduces the spatial dimensions (i.e., width and height) of the input volume.
6. Dropout2d, p=0.3: This is a dropout layer that randomly sets a probability p of the units to switch off during training, which helps prevent overfitting.

Part 2:



1. Conv2d(256), kernel_size=5, stride=1, padding=2: This is another 2D convolutional layer, but this time with 256 filters of size 5x5, a stride of 1, and a padding of 2.

Other parameters are the same as Part1.

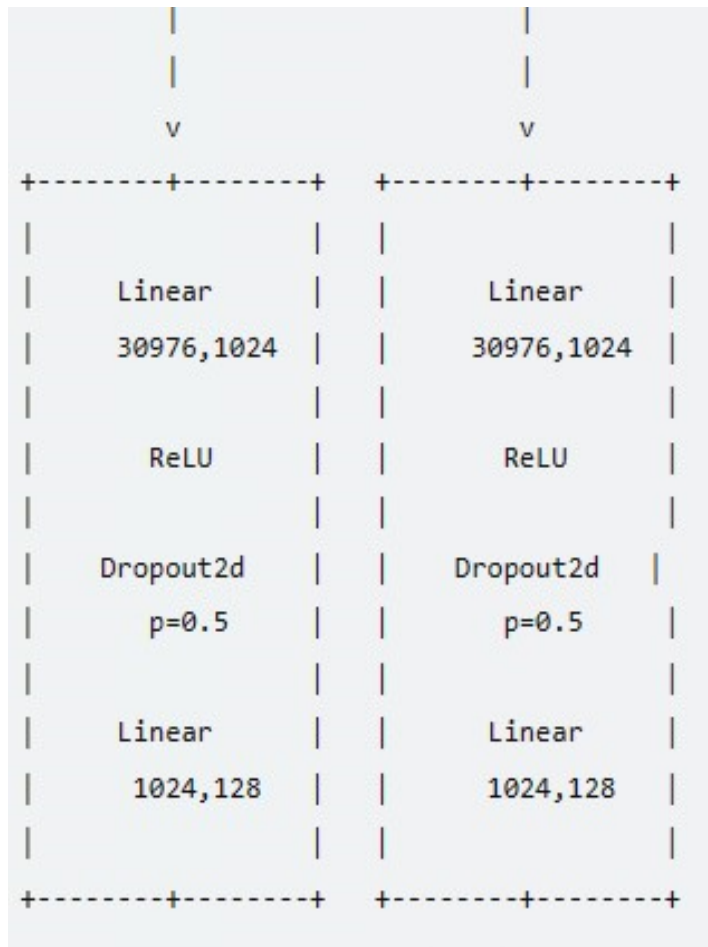
Part 3:

V		V	
+-----+-----+		+-----+-----+	
Conv2d(384)		Conv2d(384)	
kernel_size=3		kernel_size=3	
stride=1		stride=1	
padding=1		padding=1	
ReLU		ReLU	
Conv2d(256)		Conv2d(256)	
kernel_size=3		kernel_size=3	
stride=1		stride=1	
padding=1		padding=1	
ReLU		ReLU	
MaxPool2d(3,2)		MaxPool2d(3,2)	
Dropout2d		Dropout2d	
p=0.3		p=0.3	
+-----+-----+		+-----+-----+	

1. Conv2d(384), kernel_size=3, stride=1, padding=1: This is a 2D convolutional layer with 384 filters of size 3x3, a stride of 1, and a padding of 1.

Other parameters are the same as Part1.

Part 4:



1. Linear, 30976,1024: This is a fully connected layer that takes the flattened output of the previous layer (of size 30976) and transforms it into a vector of size 1024.
2. Dropout2d, p=0.5: This is a dropout layer that randomly sets a probability p of the units to switch off during training, which helps prevent overfitting.
3. Linear, 1024,128: This is another fully connected layer that takes the output of the previous layer (of size 1024) and transforms it into the final output vector of size 128.

Network Explanation:

Each of these layers contributes to the ability of the Neural Network to learn from complex, high-dimensional datasets, such as images, by extracting increasingly abstract features from the input and making decisions based on these features.

The two branches of the network allow for the processing of two different inputs, which can be useful in tasks that require comparing two images, for example. The exact function of each layer can vary depending on the specific task the network is designed for.

The dropout layers are used to prevent overfitting, which is when the model learns the training data too well and performs poorly on new, unseen data. The dropout rate p determines the probability that any given neuron (i.e., node) in the layer is temporarily “dropped out,” or ignored, during training. This helps to ensure that the model does not become too reliant on any one neuron.

The local response normalization layers are used to normalize the activations of the neurons in the network. This can help to ensure that the range of activation values throughout the network is not too large, which can make the training process more stable and efficient.

The ReLU function is used because it helps to mitigate the vanishing gradient problem, which is when the gradients of the loss function become very small and make the training process inefficient.

The max pooling layers are used to reduce the spatial dimensions of the input volume. This can help to reduce the amount of computation required by the network, as well as help to make the model translation invariant to the input.

The fully connected layers at the end of the network are used to make decisions based on the features extracted by the previous layers. The first fully connected layer transforms the flattened output of the previous layer into a vector of a certain size (in this case, 1024), and the second fully connected layer further transforms this vector into the final output vector (in this case, of size 128).

Project Significance:

Because this research directly addresses security issues related to identity verification. The possible uses are numerous and can affect financial institutions, legal procedures, and any other field that uses signatures as a means of authentication. Having a strong signature verification system reduces the possibility of fraud and illegal access while also improves security measures and streamlining procedures.

Contributions to Deep Learning:

Siamese Network Application: Showcases the usefulness of Siamese Networks in signature verification, going beyond conventional picture classification jobs.

Real-world Authentication Systems: Highlights the potential for improving security protocols through deep learning models, this project offers insights on the incorporation of deep learning techniques into real-world authentication systems.

In summary, the Signature Verification Using Siamese Networks project not only addresses a critical security concern associated with identity verification but also contributes significantly to the broader field of Deep Learning. This initiative not only resolves immediate challenges but also provides valuable insights for further research.

Literature Review:

Basic Tools and Concepts:

Neural Networks: Computational models for learning patterns.

Siamese Networks: Specialized for comparing and classifying input pairs.

Deep Learning: Trains neural networks on large datasets for predictions.

Other Solutions:

Feature-based Methods: Limited in capturing complex signature variations.

Dynamic Time Warping (DTW): Computationally intensive for large datasets.

Support Vector Machines (SVM): Struggles with intricate signature patterns.

Limitations of Other Solutions:

Limited Feature Representation: Handcrafted features may not be able to capture intricate signature details.

Scalability Issues: Some methods face challenges and are not efficient with large datasets.

Difficulty in Learning Complex Patterns: Shallow models may struggle with non-linear signature patterns.

Why Siamese Networks:

- Adaptive Feature Extraction: Deep learning enables autonomous extraction of complex signature features.
- Improved Accuracy: Addresses limitations of traditional methods for robust verification.

This project contributes by showcasing the efficiency of Siamese Networks in signature verification, overcoming the shortcomings of existing solutions.

Related Works

1. Shashidhar Sanda Sravya Amiriseti

Online Handwritten Signature Verification System using Gaussian Mixture Model and Longest Common Subsequences

2. Victor L. F. Souza, Adriano L. I. Oliveira and Robert Sabourin

A writer-independent approach for offline signature verification using deep convolutional neural networks features

3. Amruta Jagtap Dattatray D. Sawat Rajendra Hegadi Ravindra S Hegadi

Verification of genuine and forged offline signatures using Siamese Neural Network

4. Hsin-Hsiung Kao and Che-Yen Wen

An Offline Signature Verification and Forgery Detection Method Based on a Single Known Sample and an Explainable Deep Learning Approach

State-of-the-Art and Baseline Methods:

- State-of-the-art utilizes CNNs and RNNs, while baseline methods often rely on image-based feature extraction and traditional machine learning models like SVMs.

Project Differentiators:

1. Tailored Siamese Networks: Adapts Siamese Networks specifically for signature verification, addressing dynamic signature challenges.
2. Dynamic Focus: Emphasizes capturing dynamic signature features, differentiating from static image-based methods.
3. Enhanced Scalability: Aims for improved real-time processing, enhancing scalability compared to some existing works.

Methodology

For signature verification, the approach uses a Siamese Network that has been trained on pairs of signatures. Convolutional layers are used in the design to extract features, while fully linked layers are used for classification. Training is guided by a contrastive loss function that maximizes the distance between forged signatures and minimizes the distance between genuine signatures.

Network Structure:

Convolutional Layers (CNN):

- Three sets of convolutional layers with ReLU activation.
- Max pooling and dropout for feature extraction.

Fully Connected Layers (FC):

- Two fully connected layers with ReLU activation.
- Output layer for genuine/forged classification.

Loss Function:

Contrastive loss encourages minimizing distances between genuine signatures and maximizing distances between forged signatures. It penalizes large distances between genuine signatures.

Regularization:

Dropout layers reduce codependency, and implicit L2 regularization controls model complexity so that model can overcome overfitting.

Implementation Note:

Siamese Network Dataset prepares and loads the dataset, while the Siamese Network class defines the Siamese Network architecture. The Contrastive Loss class guides training with the contrastive loss function.

This methodology aims to create an efficient signature verification system, addressing identity validation challenges with a focus on learning and discriminating between genuine and forged signatures.

Experimental Setup

Implementation:

The project is implemented using PyTorch. The Siamese network is defined as a neural network class, and the training process involves defining a contrastive loss function and using an optimizer (RMSprop). The training is performed on a GPU if available (`cuda()`), and the model is saved after training.

Experimental Setting:

- Machine Configuration: The project is designed to run on Google Colab, utilizing a T4 GPU.
- Training Epochs: The model is trained for 10 epochs (`epochs`).
- Batch Size: The model is trained in batch size of 64. (`batch_size`).
- Learning Rate: The learning rate for the RMSprop optimizer is set to 0.0001

Dataset:

Dataset used is ICDAR 2011 Signature Verification Competition (SigComp2011) Dutch dataset.

<https://drive.google.com/drive/folders/14h4LHIE4EfskPldEN0s51w9eKSz9lpsx?usp=sharing>".

Initially it is stored in Google Drive and then it is mounted onto Google Colab.

Path of dataset: `"/content/drive/MyDrive/sign_data/sign_data"`

- Training Dataset: The training dataset is loaded from the path specified in `training_csv` using a CSV file (`train_data.csv`). The dataset is used for training the Siamese network for signature verification.
- Testing Dataset: The testing dataset is loaded from the path specified in `testing_csv` using a CSV file (`test_data.csv`). This dataset is used to evaluate the trained Siamese network.

Train/Test Split:

The dataset has predefined training and testing datasets inside the `sign_data` dataset along with `test_data.csv` and `train_data.csv`. The training dataset is loaded from the `training_csv` file, and the testing dataset is loaded from the `testing_csv` file. The `DataLoader` is then used for both training and testing datasets.

Pre-trained Feature Extractor:

No pre-trained feature extractor is used in this implementation. The Siamese network is built and trained from scratch.

Results

Key Applications:

- Signature Verification: The primary application is signature verification, determining the authenticity of signatures.
- Document Security: Enhances document security by validating signatures, crucial in legal and financial domains.
- Forgery Detection: Identifying forged signatures to prevent fraudulent activities.

Summary of Results:

- The model was trained using a Siamese Network for signature verification.
- Trained over 10 epochs using the RMSprop optimizer in a Batch Size of 64 and Contrastive Loss function.
- Dissimilarity measure utilized for verification based on pairwise distances between signature embeddings.
- Achieved an accuracy of 49% on the test dataset.

Performance Metrics:

- Contrastive Loss: Used as the primary performance metric, measuring the dissimilarity between genuine and forged signatures.
- Accuracy: The accuracy metric is employed to assess the overall correctness of the model predictions.
 - Why: While Contrastive Loss focuses on the model's ability to distinguish between genuine and forged signatures, accuracy provides a broader view of overall performance.
- Receiver Operating Characteristic (ROC) Curve:

Utilized to illustrate the trade-off between true positive rate (sensitivity) and false positive rate ($1 - \text{specificity}$) across different threshold values for dissimilarity scores.

Provides a visual representation of the model's discriminatory power, offering insights into its ability to distinguish between genuine and forged signatures across various decision thresholds.

Results are obtained from experiments using the Siamese Network for signature verification. The dissimilarity metric provides a strong basis for evaluating the model's ability to discern between authentic and falsified signatures.

The model has potential uses in signature authentication, as demonstrated by its effective implementation on a T4 GPU in a Google Colab environment.

Ablation Studies

Network Structure:

Explored variations in the Siamese Network architecture, such as modifying the number of convolutional layers, adjusting filter sizes, or changing the number of fully connected layers. This provides insights into which architectural aspects contribute most to the model's effectiveness.

Loss Function:

Investigated the impact of using different loss functions for training the Siamese Network.

Regularization:

Assessed the effects of regularization techniques, including dropout. By altering the dropout rates . We analyzed how these regularization methods influenced the model's generalization and performance on unseen data.

Discussion

Limitations:

- **Data Quality:** The performance of the model is highly dependent on the quality of the data. If the signatures in the dataset are not well-represented or if there's a lack of variability, the model might not generalize well to real-world signatures.
- **Forgery Styles:** The model might not perform well on different types of forgeries that it has not been trained on.
- **Computational Resources:** Deep learning models can be computationally intensive and might require significant resources (like GPU) for training.

Risks:

- **Overfitting:** This is a common risk in machine learning projects where the model learns the training data too well and performs poorly on unseen data. This can be addressed by using techniques like regularization, dropout, early stopping, or increased training set.

- Underfitting: If the model is too simple, it might not capture the complexity of the task and perform poorly. This can be addressed by increasing the complexity of the model or engineering more features.
- Biggest risk is the lack of representativeness or real world variability in the given training data.

Future Scope:

- Multi-modal Verification: The system could be extended to include other biometric data for verification, such as voice or face recognition.
- Real-time Verification: The system could be developed into a real-time verification system, where signatures are verified as they are being written.
- Forgery Detection Improvement: Advanced techniques like one-shot learning or few-shot learning could be explored to improve the detection of forgeries.

New Applications:

- Banking and Finance: The system could be used in banking and finance for verifying signatures on checks, contracts, and other important documents.
- Legal Documents Verification: It could be used to verify signatures on legal documents.
- Examinations: It could be used to verify the authenticity of handwritten examination papers.

Conclusion

Key Takeaway:

The project implements a Siamese network for signature verification, showcasing its potential in authenticating handwritten signatures.

Contributions:

Siamese Network Application:

- Introduced the practical use of Siamese networks for signature verification, demonstrating the model's capability in differentiating genuine and forged signatures.

Contrastive Loss:

- Highlights the use of contrastive loss for effective training, a crucial aspect in ensuring the network learns meaningful signature representations.

Future Directions:

- Performance Evaluation: Thoroughly assess the model's performance with standard metrics.
- Dataset Diversity: Explore the model's robustness across varied datasets.
- Deployment: Consider practical deployment in real-world scenarios, addressing associated challenges.

In summary, the project offers a foundational exploration of Siamese networks in signature verification, laying the groundwork for further research and practical applications in Security associated with Signature.

References

1.1 [Signature Verification Competition for Online and Offline Skilled Forgeries \(SigComp2011\)](#)

1.2 [AHFA 2011, China \(forensic.to\)](#)

Authors: Marcus Liwicki, Muhammad Imran Malik, C. Elisa van den Heuvel, Xiaohong Chen, Charles Berger, Reinoud Stoel, Michael Blumenstein, and Bryan Found

2. [One Shot Verification of Handwritten Signatures using Siamese Networks | IEEE Conference Publication](#)

Authors: Tejas Jambhale, Lakshya Sharma, Pranav Narayan, V. Vijayarajan and Anand M

3. [Siamese Convolutional Neural Network-Based Twin Structure Model for Independent Offline Signature Verification](#)

Authors: Neha Sharma , Sheifali Gupta , Heba G. Mohamed , Divya Anand ,Juan Luis Vidal Mazón , Deepali Gupta and Nitin Goyal

4. [SigNet: Convolutional Siamese Network for Writer Independent Offline Signature Verification](#)

Authors: Sounak Deya, Anjan Duttaa, J. Ignacio Toledo, Suman K.Ghosha , Josep Lladós and Umapada Pal

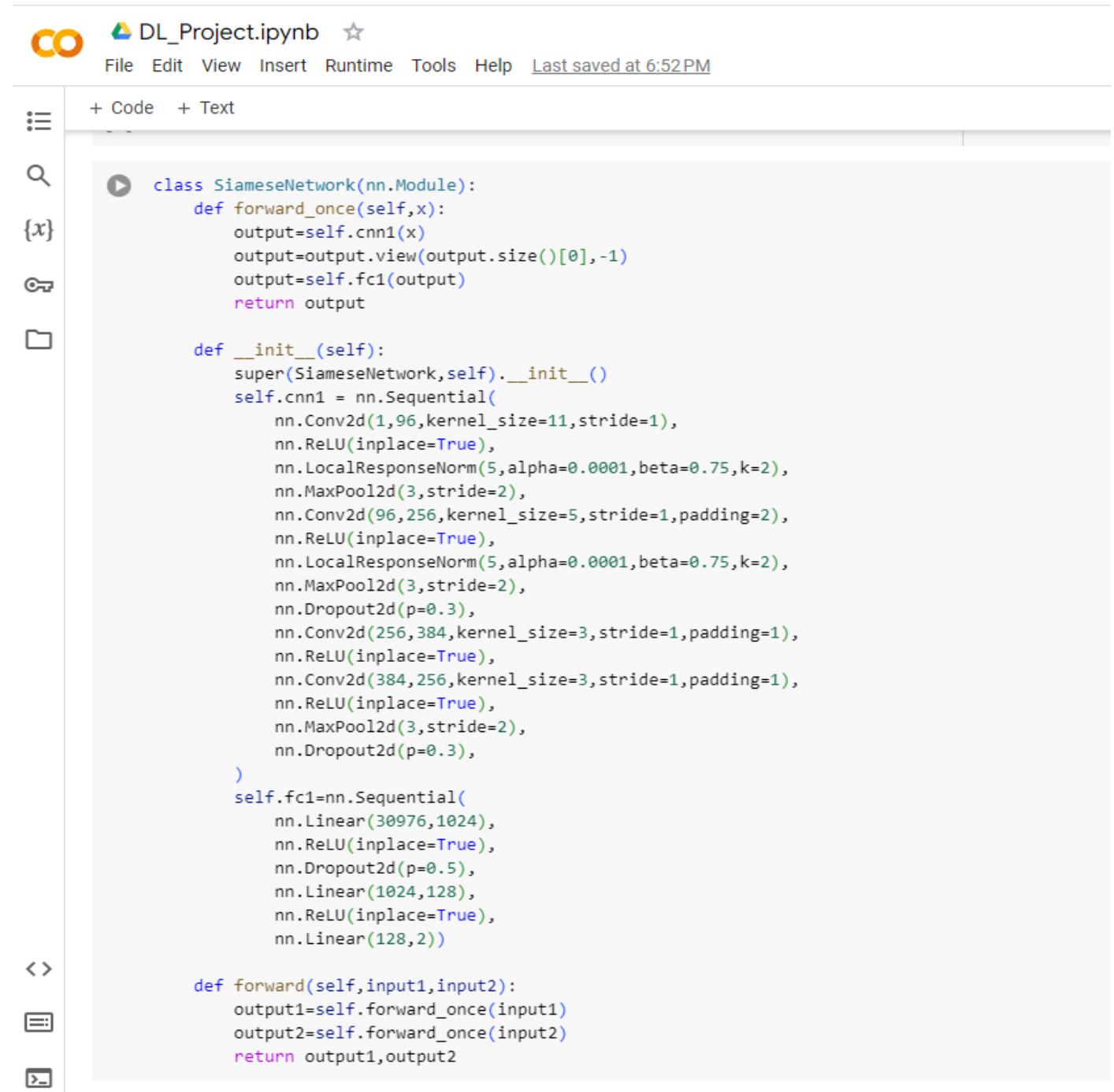
Contributions of Team Members

S.No	Name	ID	Percentage Contribution
1.	Nayan Goyal	21UCS137@lnmiit.ac.in	32%
2.	Rajdeep Tiwari	21UCS166@lnmiit.ac.in	34%
3.	Sahil Sidana	21UCS177@lnmiit.ac.in	34%

Appendix:

Code Link: [DL_Project](#)

Siamese network architecture:



The screenshot shows a Jupyter Notebook titled "DL_Project.ipynb" with a star icon. The interface includes a top menu bar with "File", "Edit", "View", "Insert", "Runtime", "Tools", and "Help", and a status bar indicating "Last saved at 6:52 PM". On the left, there is a sidebar with icons for a menu, search, a variable {x}, a key, a folder, and code execution symbols. The main area displays Python code for a Siamese network class.

```
class SiameseNetwork(nn.Module):
    def forward_once(self, x):
        output = self.cnn1(x)
        output = output.view(output.size()[0], -1)
        output = self.fc1(output)
        return output

    def __init__(self):
        super(SiameseNetwork, self).__init__()
        self.cnn1 = nn.Sequential(
            nn.Conv2d(1, 96, kernel_size=11, stride=1),
            nn.ReLU(inplace=True),
            nn.LocalResponseNorm(5, alpha=0.0001, beta=0.75, k=2),
            nn.MaxPool2d(3, stride=2),
            nn.Conv2d(96, 256, kernel_size=5, stride=1, padding=2),
            nn.ReLU(inplace=True),
            nn.LocalResponseNorm(5, alpha=0.0001, beta=0.75, k=2),
            nn.MaxPool2d(3, stride=2),
            nn.Dropout2d(p=0.3),
            nn.Conv2d(256, 384, kernel_size=3, stride=1, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(384, 256, kernel_size=3, stride=1, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(3, stride=2),
            nn.Dropout2d(p=0.3),
        )
        self.fc1 = nn.Sequential(
            nn.Linear(30976, 1024),
            nn.ReLU(inplace=True),
            nn.Dropout2d(p=0.5),
            nn.Linear(1024, 128),
            nn.ReLU(inplace=True),
            nn.Linear(128, 2))

    def forward(self, input1, input2):
        output1 = self.forward_once(input1)
        output2 = self.forward_once(input2)
        return output1, output2
```

Few examples:

```
DL_Project.ipynb ☆
File Edit View Insert Runtime Tools Help Last saved at 6:52 PM

+ Code + Text

plt.imshow(np.transpose(img,(1,2,0)))
[ ] plt.show()

visited_dataloader=Dataloader(dataset,shuffle=True,batch_size=16)
data_iter=iter(visited_dataloader)
example_batch=next(data_iter)
concatenated=torch.cat((example_batch[0],example_batch[1]),0)
imshow(torchvision.utils.make_grid(concatenated))
print(example_batch[2].numpy())

[[0.]
 [0.]
 [0.]
 [1.]
 [0.]
 [1.]
 [1.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [1.]
 [1.]]
```

Output:

 DL_Project.ipynb ☆

File Edit View Insert Runtime Tools Help Last saved at 6:52PM

+ Code + Text

```
[ ] c=0
list_0=torch.FloatTensor([[0]])
list_1=torch.FloatTensor([[1]])
for i, data in enumerate(test_dataloader,0):
    x0,x1,label=data
    concatenated=torch.cat((x0,x1),0)
    output1,output2=model(x0.to(device),x1.to(device))
    difference=F.pairwise_distance(output1, output2)
    if label==list_0:
        label="Original"
    else:
        label="Forged"
    imshow(torchvision.utils.make_grid(concatenated), 'Dissimilarity: {:.4f} Label: {}'.format(difference,
    c+=1
    if c==30:
        break
```

/usr/local/lib/python3.10/dist-packages/torch/nn/functional.py:1345: UserWarning: dropout2d: Received a

warnings.warn(warn_msg)



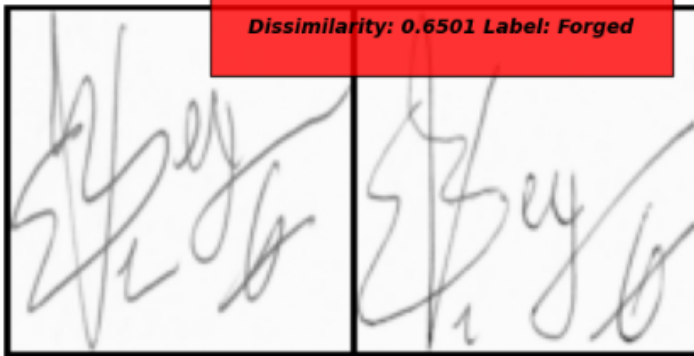
Dissimilarity: 0.4169 Label: Original



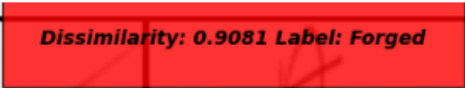
Dissimilarity: 0.4192 Label: Original



Dissimilarity: 0.5763 Label: Forged



Dissimilarity: 0.6501 Label: Forged



Dissimilarity: 0.9081 Label: Forged

Roc curve:

