SASTRA UNIVERSITY

ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION

(A UNIVERSITY U/S 3 OF THE UGC ACT, 1956)

ENHANCED CERVICAL CANCER CLASSIFICATION THROUGH CONVOLUTIONAL NEURAL NETWORKS AND EXTREME LEARNING MACHINES

TEAM:

DIVYA-125003073

ALMAS-125003457

GUIDE:

DR MANJULA K R

Overview of Presentation

SASTRA UNIVERSITY
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
(A UNIVERSITY U/S 3 OF THE UGC ACT, 1956)

Motivation

SASTRA UNIVERSITY
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
(A UNIVERSITY U/S 3 OF THE UGC ACT, 1956)

- Cervical cancer is the fourth most common cause of cancer death among women worldwide
- Cervical cancer is a significant health concern for women globally and ranks second after breast cancer
- If the cancer is detected in the early and precancerous stages, it is completely curable.
- The goal is to aim accurate auxiliary judgments, and improve the accuracy of diagnoses

Objective

- Increase the accuracy of detecting different stages of cervical cancer.
- Relying on strong and effective feature extraction capabilities, which can greatly improve the accuracy of object recognition
- Reduce the manual screening approach that suffers from a high false rate due to human errors and Use machine learning (ML) and DL techniques to automatically segment and categorize cervical cytology and colposcopy images

Problem Statement

SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
UNIVERSITY
(A UNIVERSITY U/S 3 OF THE UGC ACT, 1956)

- None of the existing solutions can accurately detect the early stages of cervical cancer also manual screening approach suffers from a high false rate due to human errors
- There is an intense need to improve deep learning-based digital solutions for timely and accurate cervical cancer diagnosis and detection of cervical cancer in all stages, especially in the early stages, to avoid morbidity and mortality, especially in low-income countries.

Base Paper

SASTRA UNIVERSITY
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
(A UNIVERSITY U/S 3 OF THE UGC ACT, 1956)

Cervical cancer classification using convolutional neural networks and extreme learning machines

Indexed in: ScienceDirect

Authors : Ahmed Ghoneim , Ghulam Muhammad , M. Shamim Hossain

Year : Received 28 December 2022, accepted 30 December 2022, date of publication 10 January 2023, date of current version 20 January 2023.

Literature Survey

SASTRA UNIVERSITY
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
(A UNIVERSITY U/S 3 OF THE UGC ACT, 1956)

Sl.No	Author	Findings and Techniques Used	Methodology	Gap Analysis
1	Zhang et al	SVM-based cervical cancer classification system. They used a subset of Herlev database , and the subset contained 149 cell images of which 108 were normal and 41 were cancerous	SVM based approach	The system achieved an accuracy of 98%. The subset was small, and therefore, we cannot reach to a final conclusion about the system's accuracy.
2	Wang	Proposed a segmentation and classification system for cervical cells. For the classification they used shape and texture features together with Gabor features, and the SVM.	SVM	89%(private database)
3	Harangi et al	concentrated on the segmentation of cell issues in automated Pap smear images, which is one of the prerequisites forearly diagnosis of CC detection in its primary stages.	DL-based techniques, particularly, Fully Convolutional Neural Networks (FCNNs	86.67%

Literature Survey

Sl.No	Author	Findings and Techniques Used	Methodology	Gap Analysis
4	Luo et al.	method initially uses the k-means algorithm to segregate the data into specific classes during data processing and then trained to enhance generalization ability.	Multi decision feature strategy is used along with the CNN to obtain good results.	88.9%
5	Kurnianingsih et al.	proposed a method to automatically detect the cervical cells by first dividing the cell regions using the Mask R-CNN and then classifying the cells using VGG	R-CNN	detection accuracy greater than 3%.
6	Chandran et al.	developed a model Colposcopy Ensemble Network (CYENET) to classify the images into cancerous or non-cancerous.	CYENET	classification accuracy is 19% higher than VGG19 model.

Literature Survey

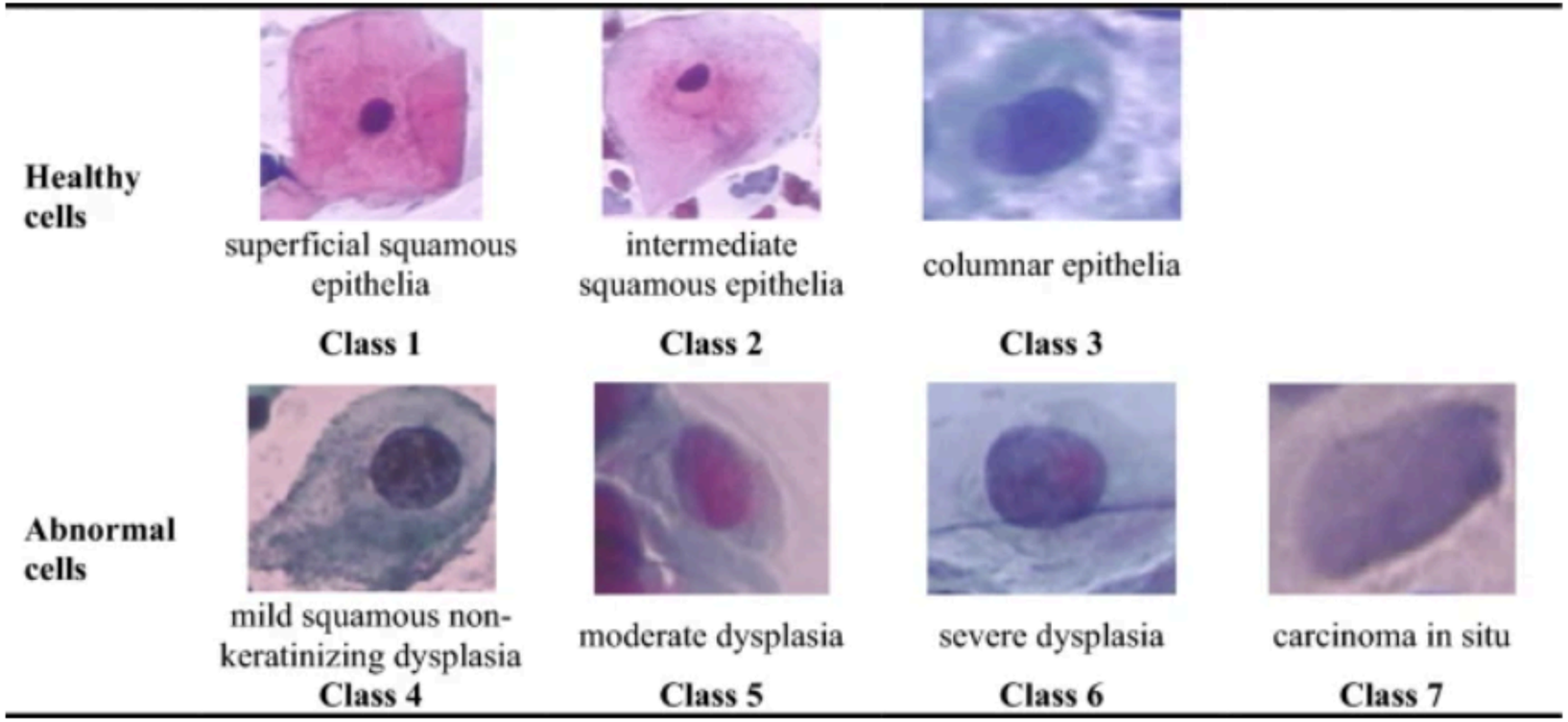
Sl.No	Author	Findings and Techniques Used	Methodology	Gap Analysis
7	Sompawong et al	used a Mask RCNN with backbone network and region proposal network to extract the features from the images to identify the Region of Interest (RoI) and thus classifying the images into normal, abnormal and background.	RCNN	The classification accuracy is high and the sensitivity is low due to lack of augmentation.
8	Xiang et al	Average precision (AP) and mean Average Precision (mAP) is used to calculate the performance of the model	R-CNN	detection accuracy greater than 3%.
9	**Ghoneim et al.**	**proposed a model that extracts the features using CNN model and these features are given to classifier based on Extreme Learning Machine (ELM) to classify the images..**	**CNN -ELM**	**99.7% accuracy is obtained in the 2-class problem using the Herlev database.97.2% accuracy is obtained in the 7-class problem.**

Data Set

- This study used the Herlev database, contains 917 pap smear images that are unevenly distributed over seven different classes of cervical cells.
- Among these seven classes, the superficial squamous epithelia, intermediate squamous epithelia, and columnar epithelia belong to normal cells, whereas the others correspond to malignant cells. The cell types are sorted from normal to abnormal cell levels, with carcinoma in situ being the highest-grade lesion in the Herlev dataset

Data Set

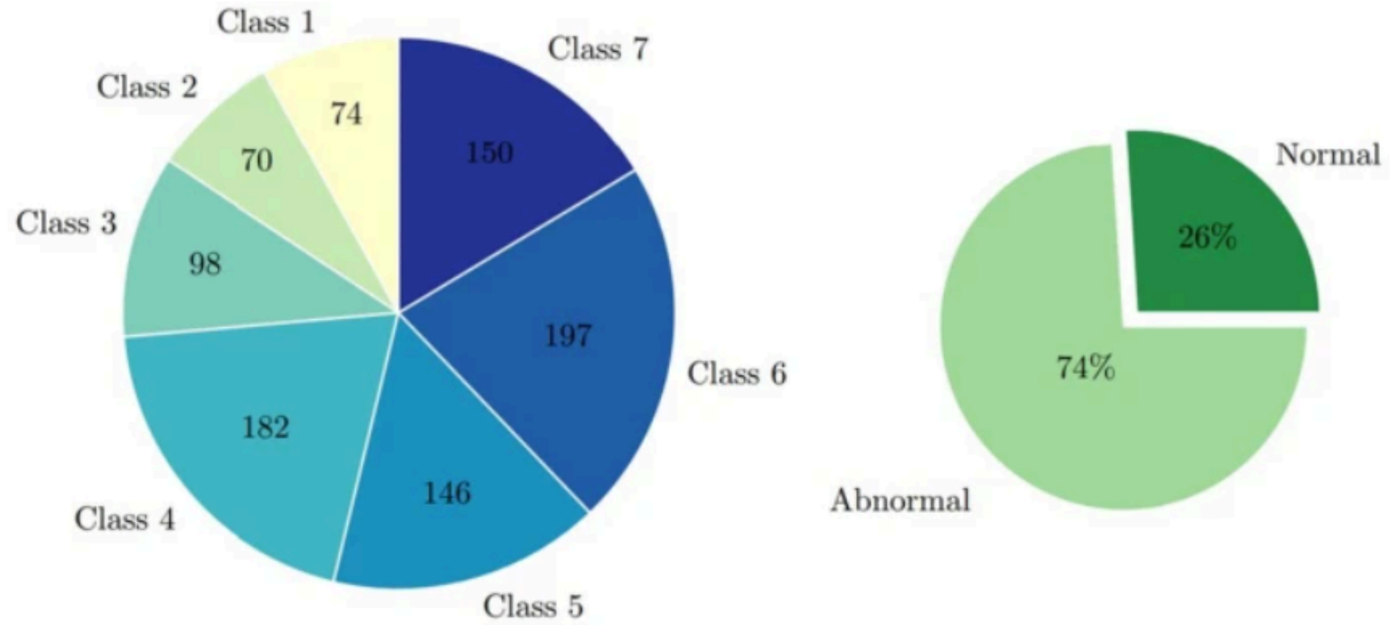
Fig. 1



Samples of Herlev dataset in seven categories

Data Set

Fig. 2



Distribution of the Herlev dataset

Methodology

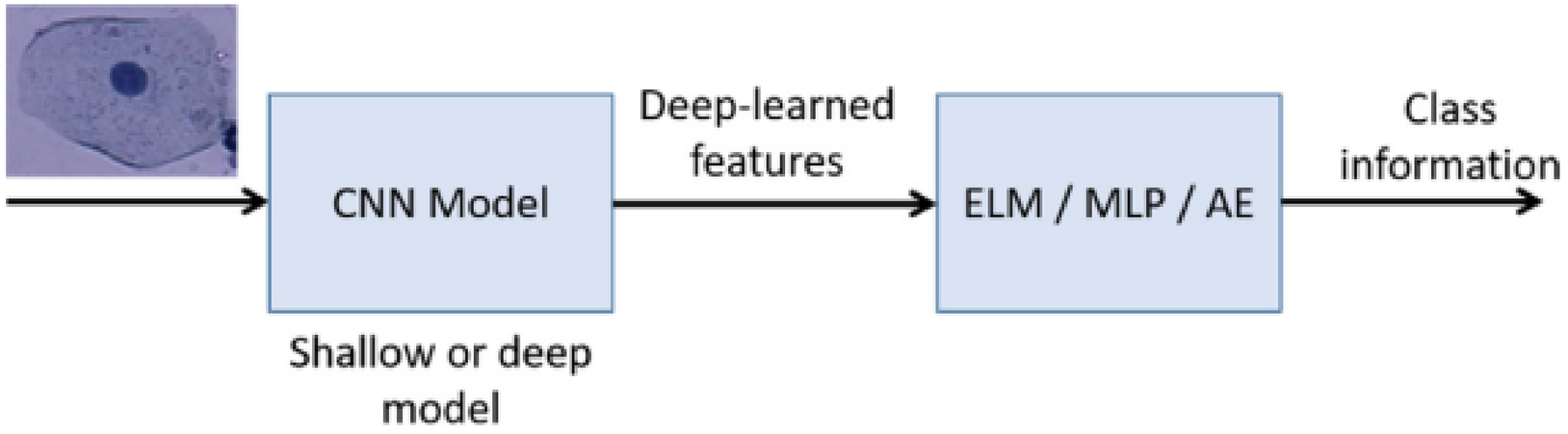
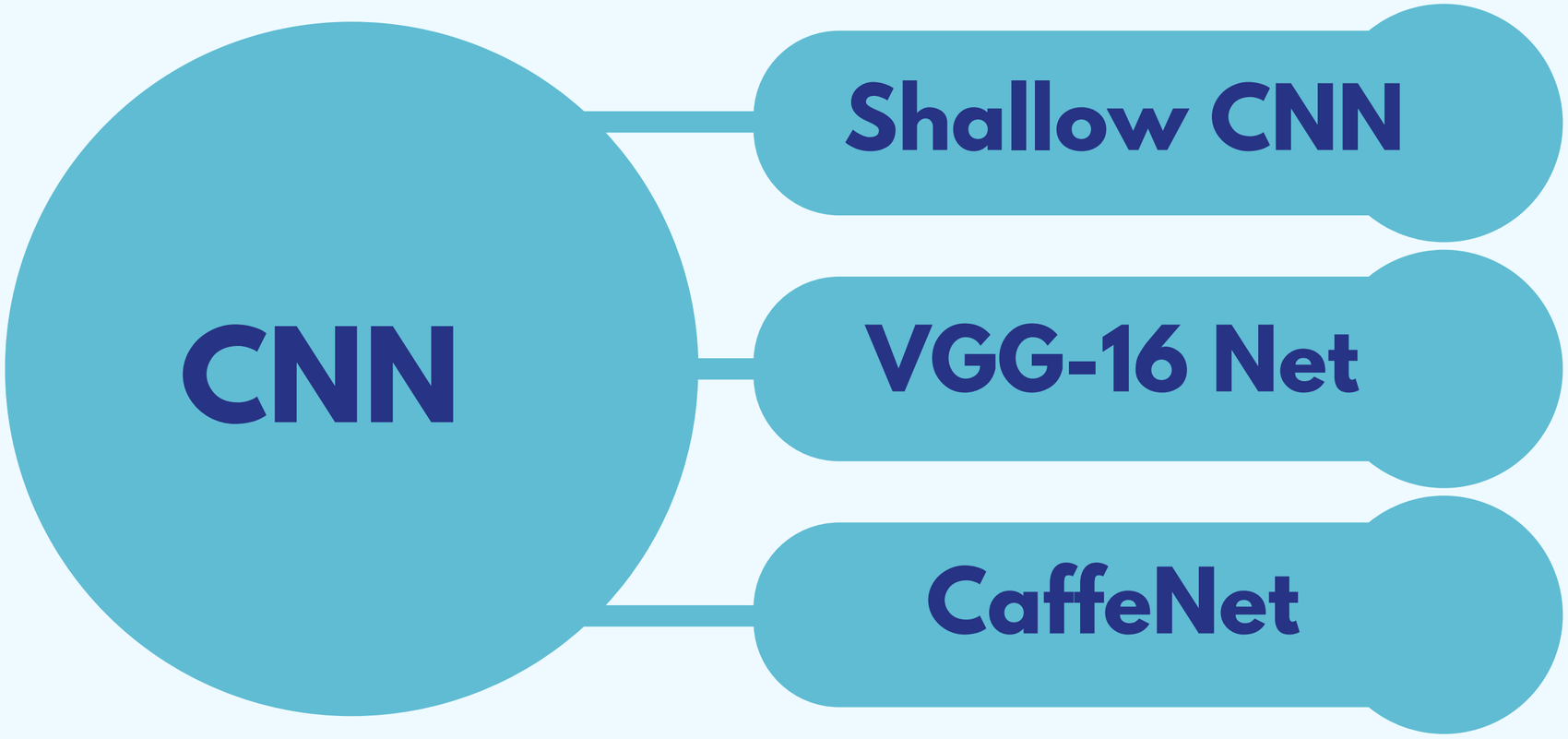


Fig. 1. Block diagram of the proposed system.

Methodology



Methodology

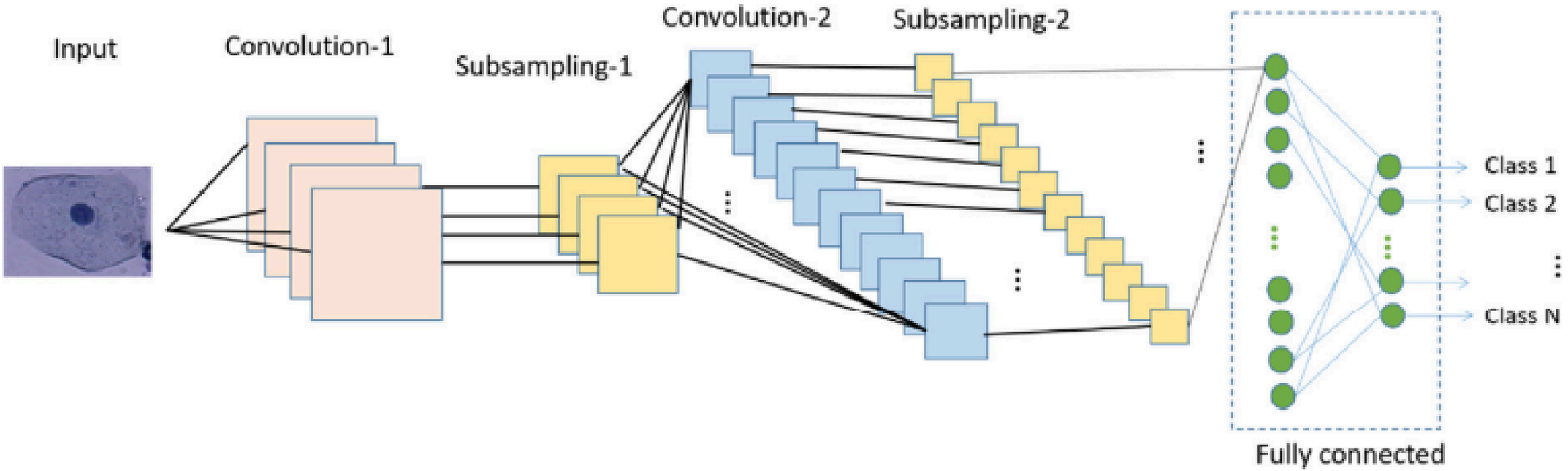


Fig. 2. Architecture of the shallow CNN model.

Methodology

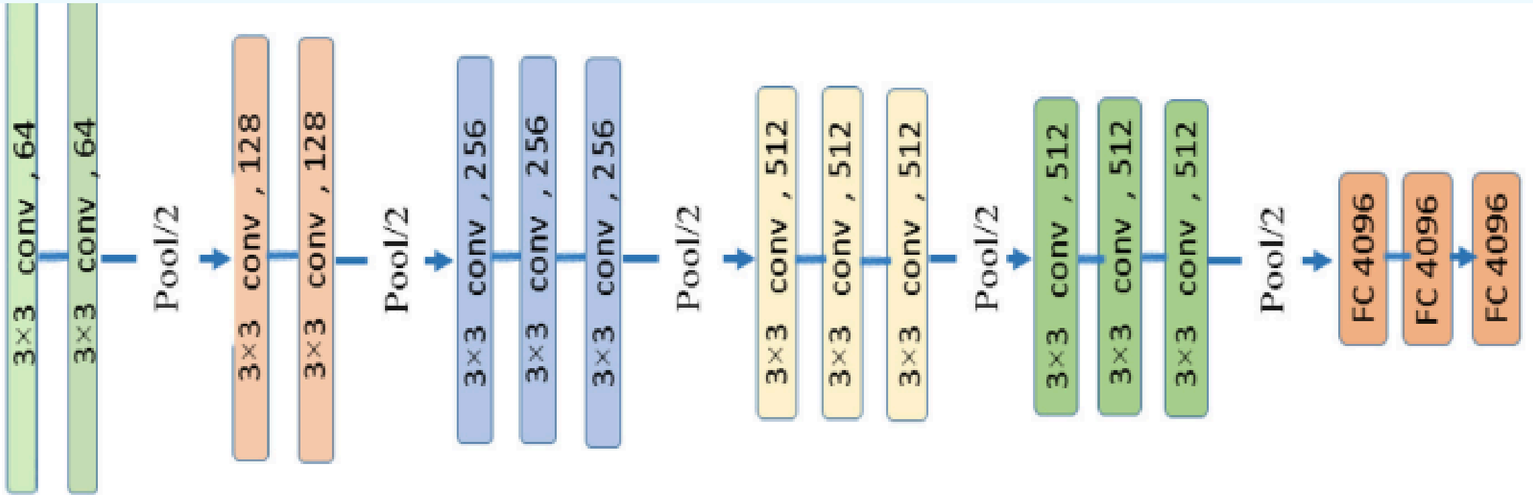


Fig. 3. Architecture of the VGG-16 Net.

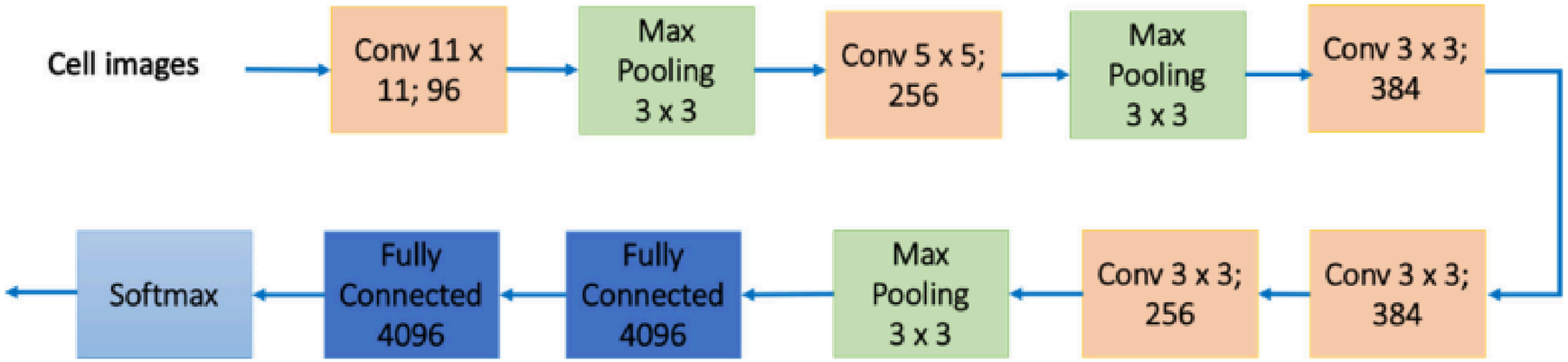


Fig. 4. Architecture of CaffeNet.

Methodology

- ELM Classifier
- AE Classifier

Work Plan

Week No.	Actions	Outcome
1-2	Paper Submission	Selecting the Journal paper and preparation of abstract for the objective of the Mini-Project
2-4	Data Collection	Collecting the Data sets of the selected algorithm
3-6	Implementing the 3 methods of CNN	Implementation of the shallow and deep architectures of CNN
6-9	Implementing the Classifiers	Implementing the two classifiers and finding out the accuracies
10-12	Evaluation	Compiling the outcomes of the data obtained and interpreting the results

Expected Outcome

SASTRA UNIVERSITY
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
(A UNIVERSITY U/S 3 OF THE UGC ACT, 1956)

We propose to implement a cervical cancer cell detection and classification system centered around convolutional neural networks (CNNs). The core workflow involves feeding cell images into a CNN model to extract deep-learned features, followed by classification using an extreme learning machine (ELM)-based classifier. Transfer learning and fine-tuning techniques are employed with the CNN model. Additionally, we explore alternatives to the ELM, autoencoder (AE)-based classifiers. Experimental validation is conducted using the Herlev database. We anticipate that our proposed CNN-ELM-based system will achieve a high accuracy rate in the detection problem (2-class)

Implementation of the Models

Shallow CNN

```
rt tensorflow as tf
tensorflow.keras import layers, models
sklearn.preprocessing import LabelEncoder
rt numpy as np
rt os

ad and preprocess data
oad_and_preprocess_data(directory, target_size=(224, 224), num_classes=2):
mage_data = []
abels = []
abel_encoder = LabelEncoder()
or label in os.listdir(directory):
    label_dir = os.path.join(directory, label)
    for filename in os.listdir(label_dir):
        image_path = os.path.join(label_dir, filename)
        image = tf.keras.preprocessing.image.load_img(image_path, target_size=target_size)
        image = tf.keras.preprocessing.image.img_to_array(image)
        image = image / 255.0
        image_data.append(image)
        labels.append(label)
mage_data = np.array(image_data)
abels = label_encoder.fit_transform(labels)

# Map labels to the nearest valid label
abels = np.clip(labels, 0, num_classes - 1)

eturn image_data, labels

ad and preprocess train and test data
_images, train_labels = load_and_preprocess_data(r"C:\Users\prasa_o5lltau\OneDrive\Desktop\miniproj\archive\Herlev Dataset\train")
```

```python
train_images, train_labels = load_and_preprocess_data(r"C:\Users\prasa_o5lltau\OneDrive\Desktop\miniproj\archive\Herlev Dataset\train")
test_images, test_labels = load_and_preprocess_data(r"C:\Users\prasa_o5lltau\OneDrive\Desktop\miniproj\archive\Herlev Dataset\test")

# Define shallow CNN model
model = models.Sequential([
    layers.Conv2D(64, (5, 5), strides=(2, 2), activation='relu', input_shape=(224, 224, 3)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(128, (5, 5), strides=(2, 2), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(64, activation='relu'),
    layers.Dense(2, activation='softmax')
])

# Compile the model
model.compile(optimizer=tf.keras.optimizers.SGD(learning_rate=0.01),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Train the model
history = model.fit(train_images, train_labels, epochs=50, batch_size=20, validation_data=(test_images, test_labels))

# Evaluate the model
test_loss, test_accuracy = model.evaluate(test_images, test_labels)

# Print test accuracy
print("Test Accuracy: {:.2f}%".format(test_accuracy * 100))
```

Output :-

```
Epoch 1/50
33/33 [==============================] - 10s 256ms/step - loss: 0.4556 - accuracy: 0.8134 - val_loss: 0.4647 - val_accuracy: 0.8358
Epoch 2/50
33/33 [==============================] - 8s 240ms/step - loss: 0.4374 - accuracy: 0.8367 - val_loss: 0.4704 - val_accuracy: 0.8358
Epoch 3/50
33/33 [==============================] - 8s 253ms/step - loss: 0.4342 - accuracy: 0.8367 - val_loss: 0.4445 - val_accuracy: 0.8358
Epoch 4/50
33/33 [==============================] - 8s 243ms/step - loss: 0.4289 - accuracy: 0.8367 - val_loss: 0.4831 - val_accuracy: 0.8358
Epoch 5/50
33/33 [==============================] - 8s 257ms/step - loss: 0.4290 - accuracy: 0.8367 - val_loss: 0.4793 - val_accuracy: 0.8358
Epoch 6/50
33/33 [==============================] - 8s 240ms/step - loss: 0.4271 - accuracy: 0.8367 - val_loss: 0.4748 - val_accuracy: 0.8358
Epoch 7/50
33/33 [==============================] - 8s 246ms/step - loss: 0.4256 - accuracy: 0.8367 - val_loss: 0.4405 - val_accuracy: 0.8358
Epoch 8/50
33/33 [==============================] - 9s 259ms/step - loss: 0.4271 - accuracy: 0.8367 - val_loss: 0.4604 - val_accuracy: 0.8358
Epoch 9/50
33/33 [==============================] - 8s 252ms/step - loss: 0.4262 - accuracy: 0.8367 - val_loss: 0.4515 - val_accuracy: 0.8358
Epoch 10/50
33/33 [==============================] - 9s 261ms/step - loss: 0.4277 - accuracy: 0.8367 - val_loss: 0.4624 - val_accuracy: 0.8358
Epoch 11/50
33/33 [==============================] - 9s 267ms/step - loss: 0.4189 - accuracy: 0.8367 - val_loss: 0.4696 - val_accuracy: 0.8358
Epoch 12/50
33/33 [==============================] - 9s 275ms/step - loss: 0.4159 - accuracy: 0.8367 - val_loss: 0.5525 - val_accuracy: 0.8358
Epoch 13/50
...
Epoch 50/50
33/33 [==============================] - 8s 240ms/step - loss: 0.3061 - accuracy: 0.8771 - val_loss: 1.2478 - val_accuracy: 0.8358
9/9 [==============================] - 1s 106ms/step - loss: 1.2478 - accuracy: 0.8358
Test Accuracy: 83.58%
```

VGG-16 CNN model

```python
import tensorflow as tf
from tensorflow.keras import layers, models
from sklearn.preprocessing import LabelEncoder
import numpy as np
import os
# Load and preprocess data
def load_and_preprocess_data(directory, target_size=(224, 224), num_classes=2):
    image_data = []
    labels = []
    label_encoder = LabelEncoder()
    for label in os.listdir(directory):
        label_dir = os.path.join(directory, label)
        for filename in os.listdir(label_dir):
            image_path = os.path.join(label_dir, filename)
            image = tf.keras.preprocessing.image.load_img(image_path, target_size=target_size)
            image = tf.keras.preprocessing.image.img_to_array(image)
            image = image / 255.0
            image_data.append(image)
            labels.append(label)
    image_data = np.array(image_data)
    labels = label_encoder.fit_transform(labels)
    # Map labels to the nearest valid label
    labels = np.clip(labels, 0, num_classes - 1)
    return image_data, labels
# Load and preprocess train and test data
train_images, train_labels = load_and_preprocess_data(r"C:\Users\prasa_o5lltau\OneDrive\Desktop\miniproj\archive\Herlev Dataset\train")
test_images, test_labels = load_and_preprocess_data(r"C:\Users\prasa_o5lltau\OneDrive\Desktop\miniproj\archive\Herlev Dataset\test")
# Define VGG-16 model
model = models.Sequential([
    layers.Conv2D(64, (3, 3), activation='relu', padding='same', input_shape=(224, 224, 3)),
```

```
    layers.MaxPooling2D((2, 2), strides=(2, 2)),
    layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
    layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D((2, 2), strides=(2, 2)),
    layers.Conv2D(256, (3, 3), activation='relu', padding='same'),
    layers.Conv2D(256, (3, 3), activation='relu', padding='same'),
    layers.Conv2D(256, (3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D((2, 2), strides=(2, 2)),
    layers.Conv2D(512, (3, 3), activation='relu', padding='same'),
    layers.Conv2D(512, (3, 3), activation='relu', padding='same'),
    layers.Conv2D(512, (3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D((2, 2), strides=(2, 2)),
    layers.Conv2D(512, (3, 3), activation='relu', padding='same'),
    layers.Conv2D(512, (3, 3), activation='relu', padding='same'),
    layers.Conv2D(512, (3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D((2, 2), strides=(2, 2)),
    layers.Flatten(),
    layers.Dense(4096, activation='relu'),
    layers.Dense(4096, activation='relu'),
    layers.Dense(4096, activation='relu'),
    layers.Dense(2, activation='softmax')
])
# Compile the model
model.compile(optimizer=tf.keras.optimizers.SGD(learning_rate=0.01),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
# Train the model
history = model.fit(train_images, train_labels, epochs=50, batch_size=20, validation_data=(test_images, test_labels))
# Evaluate the model
test_loss, test_accuracy = model.evaluate(test_images, test_labels)
# Print test accuracy
print("Test Accuracy: {:.2f}%".format(test_accuracy * 100))
```

```
Epoch 1/50
WARNING:tensorflow:From C:\Users\prasa_o5lltau\anaconda3\Lib\site-packages\keras\src\utils\tf_utils.py:492: The name tf.ragged.RaggedTensorValue is deprecated. Please use tf.c

WARNING:tensorflow:From C:\Users\prasa_o5lltau\anaconda3\Lib\site-packages\keras\src\engine\base_layer_utils.py:384: The name tf.executing_eagerly_outside_functions is depreca

33/33 [==============================] - 434s 13s/step - loss: 0.6202 - accuracy: 0.8118 - val_loss: 0.5499 - val_accuracy: 0.8358
Epoch 2/50
33/33 [==============================] - 610s 19s/step - loss: 0.5041 - accuracy: 0.8367 - val_loss: 0.4648 - val_accuracy: 0.8358
Epoch 3/50
33/33 [==============================] - 617s 19s/step - loss: 0.4491 - accuracy: 0.8367 - val_loss: 0.4454 - val_accuracy: 0.8358
Epoch 4/50
33/33 [==============================] - 586s 18s/step - loss: 0.4432 - accuracy: 0.8367 - val_loss: 0.4454 - val_accuracy: 0.8358
Epoch 5/50
33/33 [==============================] - 314s 10s/step - loss: 0.4434 - accuracy: 0.8367 - val_loss: 0.4466 - val_accuracy: 0.8358
Epoch 6/50
 5/33 [===>..........................] - ETA: 4:31 - loss: 0.4358 - accuracy: 0.8400
```

Caffenet CNN model

```python
import tensorflow as tf
from tensorflow.keras import layers, models
from sklearn.preprocessing import LabelEncoder
import numpy as np
import os
# Load and preprocess data
def load_and_preprocess_data(directory, target_size=(224, 224), num_classes=2):
    image_data = []
    labels = []
    label_encoder = LabelEncoder()
    for label in os.listdir(directory):
        label_dir = os.path.join(directory, label)
        for filename in os.listdir(label_dir):
            image_path = os.path.join(label_dir, filename)
            image = tf.keras.preprocessing.image.load_img(image_path, target_size=target_size)
            image = tf.keras.preprocessing.image.img_to_array(image)
            image = image / 255.0
            image_data.append(image)
            labels.append(label)
    image_data = np.array(image_data)
    labels = label_encoder.fit_transform(labels)
    # Map labels to the nearest valid label
    labels = np.clip(labels, 0, num_classes - 1)
    return image_data, label
# Load and preprocess train and test data
train_images, train_labels = load_and_preprocess_data(r"C:\Users\prasa_o5lltau\OneDrive\Desktop\miniproj\archive\Herlev Dataset\train")
test_images, test_labels = load_and_preprocess_data(r"C:\Users\prasa_o5lltau\OneDrive\Desktop\miniproj\archive\Herlev Dataset\test")
# Define CaffeNet-like model
model = models.Sequential([
    layers.Conv2D(96, (11, 11), strides=(4, 4), activation='relu', input_shape=(224, 224, 3)),
```

```python
model = models.Sequential([
    layers.Conv2D(96, (11, 11), strides=(4, 4), activation='relu', input_shape=(224, 224, 3)),
    layers.MaxPooling2D((3, 3), strides=(2, 2)),
    layers.Conv2D(256, (5, 5), activation='relu', padding='same'),
    layers.MaxPooling2D((3, 3), strides=(2, 2)),
    layers.Conv2D(384, (3, 3), activation='relu', padding='same'),
    layers.Conv2D(384, (3, 3), activation='relu', padding='same'),
    layers.Conv2D(256, (3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D((3, 3), strides=(2, 2)),
    layers.Flatten(),
    layers.Dense(4096, activation='relu'),
    layers.Dense(4096, activation='relu'),
    layers.Dense(2, activation='softmax')
])
# Compile the model
model.compile(optimizer=tf.keras.optimizers.SGD(learning_rate=0.01),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
# Train the model
history = model.fit(train_images, train_labels, epochs=50, batch_size=20, validation_data=(test_images, test_labels))
# Evaluate the model
test_loss, test_accuracy = model.evaluate(test_images, test_labels)
# Print test accuracy
print("Test Accuracy: {:.2f}%".format(test_accuracy * 100))
```

```
 33/33 [==============================] - 54s 1s/step - loss: 0.4846 - accuracy: 0.8212 - val_loss: 0.5020 - val_accuracy: 0.8358
 Epoch 2/50
 33/33 [==============================] - 49s 1s/step - loss: 0.4380 - accuracy: 0.8367 - val_loss: 0.4522 - val_accuracy: 0.8358
 Epoch 3/50
 33/33 [==============================] - 49s 1s/step - loss: 0.4379 - accuracy: 0.8367 - val_loss: 0.4652 - val_accuracy: 0.8358
 Epoch 4/50
 33/33 [==============================] - 49s 1s/step - loss: 0.4391 - accuracy: 0.8367 - val_loss: 0.4855 - val_accuracy: 0.8358
 Epoch 5/50
 33/33 [==============================] - 49s 2s/step - loss: 0.4316 - accuracy: 0.8367 - val_loss: 0.4525 - val_accuracy: 0.8358
 Epoch 6/50
 33/33 [==============================] - 50s 2s/step - loss: 0.4281 - accuracy: 0.8367 - val_loss: 0.4690 - val_accuracy: 0.8358
 Epoch 7/50
 33/33 [==============================] - 49s 1s/step - loss: 0.4298 - accuracy: 0.8367 - val_loss: 0.4481 - val_accuracy: 0.8358
 Epoch 8/50
 ...
 Epoch 49/50
 33/33 [==============================] - 65s 2s/step - loss: 0.3490 - accuracy: 0.8600 - val_loss: 0.6228 - val_accuracy: 0.7847
 Epoch 50/50
 33/33 [==============================] - 69s 2s/step - loss: 0.3810 - accuracy: 0.8460 - val_loss: 0.4076 - val_accuracy: 0.8248
```

VGG 16 with ELM

```python
import tensorflow as tf
from tensorflow.keras import layers, models
from sklearn.preprocessing import LabelEncoder
import numpy as np
import os
# Function to implement ELM layer
class ELM(layers.Layer):
    def __init__(self, num_hidden_units, input_shape):
        super(ELM, self).__init__()
        self.num_hidden_units = num_hidden_units
        self.random_weights = tf.Variable(tf.random.normal(shape=(input_shape, num_hidden_units)), trainable=False)
    def call(self, inputs):
        # Compute hidden layer output
        hidden_output = tf.matmul(inputs, self.random_weights)
        hidden_output = tf.nn.relu(hidden_output)
        return hidden_output
# Load and preprocess data with increased data augmentation
def load_and_preprocess_data(directory, target_size=(224, 224), num_classes=2):
    image_data = []
    labels = []
    label_encoder = LabelEncoder()
    for label in os.listdir(directory):
        label_dir = os.path.join(directory, label)
        for filename in os.listdir(label_dir):
            image_path = os.path.join(label_dir, filename)
            image = tf.keras.preprocessing.image.load_img(image_path, target_size=target_size)
            image = tf.keras.preprocessing.image.img_to_array(image)
            image = image / 255.0
            image_data.append(image)
            labels.append(label)
    image_data = np.array(image_data)
    labels = label_encoder.fit_transform(labels)
    # Map labels to the nearest valid label
    labels = np.clip(labels, 0, num_classes - 1)
    return image_data, labels
# Define dataset directory paths
train_dir = r"C:\Users\prasa_o5lltau\OneDrive\Desktop\miniproj\archive\Herlev Dataset\train"
test_dir = r"C:\Users\prasa_o5lltau\OneDrive\Desktop\miniproj\archive\Herlev Dataset\test"
# Load and preprocess train and test data
train_images, train_labels = load_and_preprocess_data(train_dir)
test_images, test_labels = load_and_preprocess_data(test_dir)
# Define VGG16-like model with ELM layers
def create_model(input_shape, num_classes):
    model = models.Sequential([
```

```python
    model = models.Sequential([
        layers.Conv2D(64, (3, 3), activation='relu', padding='same', input_shape=input_shape),
        layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
        layers.MaxPooling2D((2, 2), strides=(2, 2)),
        layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
        layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
        layers.MaxPooling2D((2, 2), strides=(2, 2)),
        layers.Conv2D(256, (3, 3), activation='relu', padding='same'),
        layers.Conv2D(256, (3, 3), activation='relu', padding='same'),
        layers.Conv2D(256, (3, 3), activation='relu', padding='same'),
        layers.MaxPooling2D((2, 2), strides=(2, 2)),
        layers.Conv2D(512, (3, 3), activation='relu', padding='same'),
        layers.Conv2D(512, (3, 3), activation='relu', padding='same'),
        layers.Conv2D(512, (3, 3), activation='relu', padding='same'),
        layers.MaxPooling2D((2, 2), strides=(2, 2)),
        layers.Conv2D(512, (3, 3), activation='relu', padding='same'),
        layers.Conv2D(512, (3, 3), activation='relu', padding='same'),
        layers.Conv2D(512, (3, 3), activation='relu', padding='same'),
        layers.MaxPooling2D((2, 2), strides=(2, 2)),
        layers.Flatten(),
        layers.Dense(4096, activation='relu'),
        layers.Dense(4096, activation='relu'),
    ])
    # Add first ELM layer
    model.add(ELM(num_hidden_units=2048, input_shape=4096))
    # Add second ELM layer
    model.add(ELM(num_hidden_units=num_classes, input_shape=2048))
    return model
# Create VGG16-like model with ELM layers
model = create_model(input_shape=train_images.shape[1:], num_classes=2)
# Adjust learning rate and optimizer
opt = tf.keras.optimizers.SGD(learning_rate=0.001, momentum=0.9)
# Compile the model with appropriate loss function
model.compile(optimizer=opt,
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
# Train the model with increased epochs
history = model.fit(train_images, train_labels, epochs=100, batch_size=20, validation_data=(test_images, test_labels))
# Evaluate the model
test_loss, test_accuracy = model.evaluate(test_images, test_labels)
# Print test accuracy
print("Test Accuracy: {:.2f}%".format(test_accuracy * 100))
```

```
Epoch 1/50
WARNING:tensorflow:From C:\Users\prasa_o5lltau\anaconda3\Lib\site-packages\keras\src\utils\tf_utils.py:492: The name tf.ragged.RaggedTensorValue is deprecated. Please use tf.

WARNING:tensorflow:From C:\Users\prasa_o5lltau\anaconda3\Lib\site-packages\keras\src\engine\base_layer_utils.py:384: The name tf.executing_eagerly_outside_functions is depre

33/33 [==============================] - 122s 4s/step - loss: 2.6547 - accuracy: 0.8072 - val_loss: 2.6237 - val_accuracy: 0.7956
Epoch 2/50
33/33 [==============================] - 126s 4s/step - loss: 2.6547 - accuracy: 0.8072 - val_loss: 2.6212 - val_accuracy: 0.7993
Epoch 3/50
33/33 [==============================] - 126s 4s/step - loss: 2.6547 - accuracy: 0.8072 - val_loss: 2.6212 - val_accuracy: 0.7993
Epoch 4/50
33/33 [==============================] - 131s 4s/step - loss: 2.6547 - accuracy: 0.8072 - val_loss: 2.6212 - val_accuracy: 0.7993
Epoch 5/50
33/33 [==============================] - 122s 4s/step - loss: 2.6547 - accuracy: 0.8072 - val_loss: 2.6212 - val_accuracy: 0.7993
Epoch 6/50
33/33 [==============================] - 124s 4s/step - loss: 2.6547 - accuracy: 0.8072 - val_loss: 2.6212 - val_accuracy: 0.7993
Epoch 7/50
33/33 [==============================] - 123s 4s/step - loss: 2.6547 - accuracy: 0.8072 - val_loss: 2.6212 - val_accuracy: 0.7993
Epoch 8/50
...
Epoch 50/50
33/33 [==============================] - 118s 4s/step - loss: 2.6547 - accuracy: 0.8072 - val_loss: 2.6212 - val_accuracy: 0.7993
9/9 [==============================] - 31s 3s/step - loss: 2.6212 - accuracy: 0.7993
Test Accuracy: 79.93%
```

Shallow with ELM

```python
import tensorflow as tf
from tensorflow.keras import layers, models
from sklearn.preprocessing import LabelEncoder
import numpy as np
import os
# Function to implement ELM layer
class ELM(layers.Layer):
    def __init__(self, num_hidden_units, input_shape):
        super(ELM, self).__init__()
        self.num_hidden_units = num_hidden_units
        self.random_weights = tf.Variable(tf.random.normal(shape=(input_shape, num_hidden_units)), trainable=False)

    def call(self, inputs):
        # Compute hidden layer output
        hidden_output = tf.matmul(inputs, self.random_weights)
        hidden_output = tf.nn.relu(hidden_output)
        return hidden_output
# Load and preprocess data with increased data augmentation
def load_and_preprocess_data(directory, target_size=(224, 224), num_classes=2):
    image_data = []
    labels = []
    label_encoder = LabelEncoder()
    for label in os.listdir(directory):
        label_dir = os.path.join(directory, label)
        for filename in os.listdir(label_dir):
            image_path = os.path.join(label_dir, filename)
            image = tf.keras.preprocessing.image.load_img(image_path, target_size=target_size)
            image = tf.keras.preprocessing.image.img_to_array(image)
            image = image / 255.0
            image_data.append(image)
            labels.append(label)
    image_data = np.array(image_data)
    labels = label_encoder.fit_transform(labels)
    # Map labels to the nearest valid label
    labels = np.clip(labels, 0, num_classes - 1)
    return image_data, labels
# Define dataset directory paths
train_dir = r"C:\Users\prasa_o5lltau\OneDrive\Desktop\miniproj\archive\Herlev Dataset\train"
test_dir = r"C:\Users\prasa_o5lltau\OneDrive\Desktop\miniproj\archive\Herlev Dataset\test"
# Load and preprocess train and test data
train_images, train_labels = load_and_preprocess_data(train_dir)
test_images, test_labels = load_and_preprocess_data(test_dir)
# Define shallow CNN model with ELM layers
def create_model(input_shape, num_classes):
    model = models.Sequential([
```

```python
# Load and preprocess train and test data
train_images, train_labels = load_and_preprocess_data(train_dir)
test_images, test_labels = load_and_preprocess_data(test_dir)
# Define shallow CNN model with ELM layers
def create_model(input_shape, num_classes):
    model = models.Sequential([
        layers.Conv2D(64, (5, 5), activation='relu', input_shape=input_shape, strides=2),
        layers.MaxPooling2D((2, 2), strides=2),
        layers.Conv2D(128, (5, 5), activation='relu', strides=2),
        layers.MaxPooling2D((2, 2), strides=2),
        layers.Flatten(),
        layers.Dense(4096, activation='relu'),
        layers.Dense(4096, activation='relu'),
        layers.Dense(num_classes, activation='softmax'),
    ])
    # Remove the softmax layer
    model.pop()
    # Add first ELM layer
    model.add(ELM(num_hidden_units=2048, input_shape=4096))
    # Add second ELM layer
    model.add(ELM(num_hidden_units=num_classes, input_shape=2048))
    return model
# Create shallow CNN model with ELM layers
model = create_model(input_shape=train_images.shape[1:], num_classes=2)
# Adjust learning rate and optimizer
opt = tf.keras.optimizers.SGD(learning_rate=0.01, momentum=0.9)
# Compile the model with appropriate loss function
model.compile(optimizer=opt,
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
# Train the model with specified number of epochs (50)
history = model.fit(train_images, train_labels, epochs=50, batch_size=20, validation_data=(test_images, test_labels))
# Evaluate the model
test_loss, test_accuracy = model.evaluate(test_images, test_labels)
# Print test accuracy
print("Test Accuracy: {:.2f}%".format(test_accuracy * 100))
```

```
Epoch 1/50
WARNING:tensorflow:From C:\Users\prasa_o5lltau\anaconda3\Lib\site-packages\keras\src\utils\tf_utils.py:492: The name tf.ragged.RaggedTensorValue is deprecated. Plea

WARNING:tensorflow:From C:\Users\prasa_o5lltau\anaconda3\Lib\site-packages\keras\src\engine\base_layer_utils.py:384: The name tf.executing_eagerly_outside_functions

33/33 [==============================] - 45s 1s/step - loss: 2.6320 - accuracy: 0.8367 - val_loss: 2.5934 - val_accuracy: 0.8358
Epoch 2/50
33/33 [==============================] - 32s 979ms/step - loss: 2.6320 - accuracy: 0.8367 - val_loss: 2.5934 - val_accuracy: 0.8358
Epoch 3/50
33/33 [==============================] - 33s 990ms/step - loss: 2.6320 - accuracy: 0.8367 - val_loss: 2.5934 - val_accuracy: 0.8358
Epoch 4/50
33/33 [==============================] - 32s 982ms/step - loss: 2.6320 - accuracy: 0.8367 - val_loss: 2.5934 - val_accuracy: 0.8358
Epoch 5/50
33/33 [==============================] - 32s 978ms/step - loss: 2.6320 - accuracy: 0.8367 - val_loss: 2.5934 - val_accuracy: 0.8358
Epoch 6/50
33/33 [==============================] - 33s 998ms/step - loss: 2.6320 - accuracy: 0.8367 - val_loss: 2.5934 - val_accuracy: 0.8358
Epoch 7/50
33/33 [==============================] - 32s 961ms/step - loss: 2.6320 - accuracy: 0.8367 - val_loss: 2.5934 - val_accuracy: 0.8358
Epoch 8/50
...
Epoch 50/50
33/33 [==============================] - 29s 866ms/step - loss: 2.6320 - accuracy: 0.8367 - val_loss: 2.5934 - val_accuracy: 0.8358
9/9 [==============================] - 2s 226ms/step - loss: 2.5934 - accuracy: 0.8358
Test Accuracy: 83.58%
```

CaffeNet with ELM

```python
import tensorflow as tf
from tensorflow.keras import layers, models
from sklearn.preprocessing import LabelEncoder
import numpy as np
import os
# Function to implement ELM layer
class ELM(layers.Layer):
    def __init__(self, num_hidden_units, input_shape):
        super(ELM, self).__init__()
        self.num_hidden_units = num_hidden_units
        self.random_weights = tf.Variable(tf.random.normal(shape=(input_shape, num_hidden_units)), trainable=False)
    def call(self, inputs):
        # Compute hidden layer output
        hidden_output = tf.matmul(inputs, self.random_weights)
        hidden_output = tf.nn.relu(hidden_output)
        return hidden_output
# Load and preprocess data with increased data augmentation
def load_and_preprocess_data(directory, target_size=(227, 227), num_classes=2):
    image_data = []
    labels = []
    label_encoder = LabelEncoder()
    for label in os.listdir(directory):
        label_dir = os.path.join(directory, label)
        for filename in os.listdir(label_dir):
            image_path = os.path.join(label_dir, filename)
            image = tf.keras.preprocessing.image.load_img(image_path, target_size=target_size)
            image = tf.keras.preprocessing.image.img_to_array(image)
            image = image / 255.0
            image_data.append(image)
            labels.append(label)
    image_data = np.array(image_data)
    labels = label_encoder.fit_transform(labels)
    # Map labels to the nearest valid label
    labels = np.clip(labels, 0, num_classes - 1)
    return image_data, labels
# Define dataset directory paths
train_dir = r"C:\Users\prasa_o5lltau\OneDrive\Desktop\miniproj\archive\Herlev Dataset\train"
test_dir = r"C:\Users\prasa_o5lltau\OneDrive\Desktop\miniproj\archive\Herlev Dataset\test"
# Load and preprocess train and test data
train_images, train_labels = load_and_preprocess_data(train_dir)
test_images, test_labels = load_and_preprocess_data(test_dir)
# Define CaffeNet-like model with ELM layers
def create_model(input_shape, num_classes):
    model = models.Sequential([
```

```python
# Define CaffeNet-like model with ELM layers
def create_model(input_shape, num_classes):
    model = models.Sequential([
        layers.Conv2D(96, (11, 11), strides=(4, 4), activation='relu', input_shape=input_shape),
        layers.MaxPooling2D((3, 3), strides=(2, 2)),
        layers.Conv2D(256, (5, 5), padding='same', activation='relu'),
        layers.MaxPooling2D((3, 3), strides=(2, 2)),
        layers.Conv2D(384, (3, 3), padding='same', activation='relu'),
        layers.Conv2D(384, (3, 3), padding='same', activation='relu'),
        layers.Conv2D(256, (3, 3), padding='same', activation='relu'),
        layers.MaxPooling2D((3, 3), strides=(2, 2)),
        layers.Flatten(),
        layers.Dense(4096, activation='relu'),
        layers.Dense(4096, activation='relu'),
    ])
    # Add first ELM layer
    model.add(ELM(num_hidden_units=2048, input_shape=4096))
    # Add second ELM layer
    model.add(ELM(num_hidden_units=num_classes, input_shape=2048))
    return model
# Create CaffeNet-like model with ELM layers
model = create_model(input_shape=train_images.shape[1:], num_classes=2)
# Adjust learning rate and optimizer
opt = tf.keras.optimizers.SGD(learning_rate=0.01, momentum=0.9)
# Compile the model with appropriate loss function
model.compile(optimizer=opt,
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
# Train the model with specified number of epochs (50)
history = model.fit(train_images, train_labels, epochs=50, batch_size=20, validation_data=(test_images, test_labels))
# Evaluate the model
test_loss, test_accuracy = model.evaluate(test_images, test_labels)
# Print test accuracy
print("Test Accuracy: {:.2f}%".format(test_accuracy * 100))
```

```
Epoch 1/50
WARNING:tensorflow:From C:\Users\prasa_o5lltau\anaconda3\Lib\site-packages\keras\src\utils\tf_utils.py:492: The name tf.ragged.RaggedTensorValue is deprecated. Plea

WARNING:tensorflow:From C:\Users\prasa_o5lltau\anaconda3\Lib\site-packages\keras\src\engine\base_layer_utils.py:384: The name tf.executing_eagerly_outside_functions

33/33 [==============================] - 45s 1s/step - loss: 2.6320 - accuracy: 0.8367 - val_loss: 2.5934 - val_accuracy: 0.8358
Epoch 2/50
33/33 [==============================] - 32s 979ms/step - loss: 2.6320 - accuracy: 0.8367 - val_loss: 2.5934 - val_accuracy: 0.8358
Epoch 3/50
33/33 [==============================] - 33s 990ms/step - loss: 2.6320 - accuracy: 0.8367 - val_loss: 2.5934 - val_accuracy: 0.8358
Epoch 4/50
33/33 [==============================] - 32s 982ms/step - loss: 2.6320 - accuracy: 0.8367 - val_loss: 2.5934 - val_accuracy: 0.8358
Epoch 5/50
33/33 [==============================] - 32s 978ms/step - loss: 2.6320 - accuracy: 0.8367 - val_loss: 2.5934 - val_accuracy: 0.8358
Epoch 6/50
33/33 [==============================] - 33s 998ms/step - loss: 2.6320 - accuracy: 0.8367 - val_loss: 2.5934 - val_accuracy: 0.8358
Epoch 7/50
33/33 [==============================] - 32s 961ms/step - loss: 2.6320 - accuracy: 0.8367 - val_loss: 2.5934 - val_accuracy: 0.8358
Epoch 8/50
...
Epoch 50/50
33/33 [==============================] - 29s 866ms/step - loss: 2.6320 - accuracy: 0.8367 - val_loss: 2.5934 - val_accuracy: 0.8358
9/9 [==============================] - 2s 226ms/step - loss: 2.5934 - accuracy: 0.8358
Test Accuracy: 83.58%
```

VGG 16 with AE

```python
import tensorflow as tf
from tensorflow.keras import layers, models
from sklearn.preprocessing import LabelEncoder
import numpy as np
import os
# Function to implement Autoencoder (AE) layer
class AE(layers.Layer):
    def __init__(self, latent_dim):
        super(AE, self).__init__()
        self.encoder = tf.keras.Sequential([
            layers.Flatten(),
            layers.Dense(latent_dim, activation='relu'),
        ])
        self.decoder = tf.keras.Sequential([
            layers.Dense(25088, activation='relu'),  # Adjust output shape to match the flattened VGG-16 output
            layers.Reshape((7, 7, 512)),  # Adjust output shape to match the output before the first FC layer in VGG-16
        ])
    def call(self, inputs):
        encoded = self.encoder(inputs)
        decoded = self.decoder(encoded)
        return decoded
# Load and preprocess data
def load_and_preprocess_data(directory, target_size=(224, 224), num_classes=2):
    image_data = []
    labels = []
    label_encoder = LabelEncoder()
    for label in os.listdir(directory):
        label_dir = os.path.join(directory, label)
        for filename in os.listdir(label_dir):
            image_path = os.path.join(label_dir, filename)
            image = tf.keras.preprocessing.image.load_img(image_path, target_size=target_size)
            image = tf.keras.preprocessing.image.img_to_array(image)
            image = image / 255.0
            image_data.append(image)
            labels.append(label)
    image_data = np.array(image_data)
    labels = label_encoder.fit_transform(labels)
    # Map labels to the nearest valid label
    labels = np.clip(labels, 0, num_classes - 1)
    return image_data, label
# Define dataset directory paths
train_dir = r"C:\Users\prasa_o5lltau\OneDrive\Desktop\miniproj\archive\Herlev Dataset\train"
test_dir = r"C:\Users\prasa_o5lltau\OneDrive\Desktop\miniproj\archive\Herlev Dataset\test"
# Load and preprocess train and test data
```

```python
    for layer in vgg_base.layers:
        layer.trainable = False
    model = models.Sequential([
        vgg_base,
        AE(latent_dim=256),  # Add AE layer
        layers.Flatten(),
        layers.Dense(4096, activation='relu'),
        layers.Dense(num_classes, activation='softmax')
    ])
    return model
# Create VGG-16 model with AE layer
model = create_vgg16_ae_model(input_shape=train_images.shape[1:], num_classes=2)
# Compile the model with SGD optimizer and appropriate loss function
model.compile(optimizer='sgd',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
# Train the model
history = model.fit(train_images, train_labels, epochs=50, batch_size=20, validation_data=(test_images, test_labels))
# Evaluate the model
test_loss, test_accuracy = model.evaluate(test_images, test_labels)
# Print test accuracy
print("Test Accuracy: {:.2f}%".format(test_accuracy * 100))
```

```
    print("Test Accuracy: {:.2f}%".format(test_accuracy * 100))
```

Python

```
WARNING:tensorflow:From C:\Users\prasa_o5lltau\anaconda3\Lib\site-packages\keras\src\losses.py:2976: The name tf.losses.sparse_softmax_cross_entropy is deprecated. Please use tf.compat.v1.losses.sparse_softmax_cross_

WARNING:tensorflow:From C:\Users\prasa_o5lltau\anaconda3\Lib\site-packages\keras\src\backend.py:1398: The name tf.executing_eagerly_outside_functions is deprecated. Please use tf.compat.v1.executing_eagerly_outside_f

WARNING:tensorflow:From C:\Users\prasa_o5lltau\anaconda3\Lib\site-packages\keras\src\layers\pooling\max_pooling2d.py:161: The name tf.nn.max_pool is deprecated. Please use tf.nn.max_pool2d instead.

WARNING:tensorflow:From C:\Users\prasa_o5lltau\anaconda3\Lib\site-packages\keras\src\optimizers\__init__.py:309: The name tf.train.Optimizer is deprecated. Please use tf.compat.v1.train.Optimizer instead.

Epoch 1/50
WARNING:tensorflow:From C:\Users\prasa_o5lltau\anaconda3\Lib\site-packages\keras\src\utils\tf_utils.py:492: The name tf.ragged.RaggedTensorValue is deprecated. Please use tf.compat.v1.ragged.RaggedTensorValue instead

WARNING:tensorflow:From C:\Users\prasa_o5lltau\anaconda3\Lib\site-packages\keras\src\engine\base_layer_utils.py:384: The name tf.executing_eagerly_outside_functions is deprecated. Please use tf.compat.v1.executing_ea

33/33 [==============================] - 180s 5s/step - loss: 1.5323 - accuracy: 0.7854 - val_loss: 0.4025 - val_accuracy: 0.8358
Epoch 2/50
26/33 [======================>.......] - ETA: 28s - loss: 0.3385 - accuracy: 0.8442
```

Shallow with AE

```python
import tensorflow as tf
from tensorflow.keras import layers, models
from sklearn.preprocessing import LabelEncoder
import numpy as np
import os
# Function to implement Autoencoder (AE) layer
class AE(layers.Layer):
    def __init__(self, latent_dim):
        super(AE, self).__init__()
        self.encoder = tf.keras.Sequential([
            layers.Flatten(),
            layers.Dense(latent_dim, activation='relu'),
        ])
        self.decoder = tf.keras.Sequential([
            layers.Dense(4096, activation='sigmoid'),  # Adjust output shape to match the flattened CNN output
            layers.Reshape((64, 64, 1)),  # Adjust output shape to match the input shape before the first ELM layer
        ])

    def call(self, inputs):
        encoded = self.encoder(inputs)
        decoded = self.decoder(encoded)
        return decoded
# Load and preprocess data with increased data augmentation
def load_and_preprocess_data(directory, target_size=(224, 224), num_classes=2):
    image_data = []
    labels = []
    label_encoder = LabelEncoder()
    for label in os.listdir(directory):
        label_dir = os.path.join(directory, label)
        for filename in os.listdir(label_dir):
            image_path = os.path.join(label_dir, filename)
            image = tf.keras.preprocessing.image.load_img(image_path, target_size=target_size)
            image = tf.keras.preprocessing.image.img_to_array(image)
            image = image / 255.0
            image_data.append(image)
            labels.append(label)
    image_data = np.array(image_data)
    labels = label_encoder.fit_transform(labels)
    # Map labels to the nearest valid label
    labels = np.clip(labels, 0, num_classes - 1)
    return image_data, labels
# Define dataset directory paths
train_dir = r"C:\Users\prasa_o5lltau\OneDrive\Desktop\miniproj\archive\Herlev Dataset\train"
test_dir = r"C:\Users\prasa_o5lltau\OneDrive\Desktop\miniproj\archive\Herlev Dataset\test"
```

```python
test_dir = r"C:\Users\prasa_o5lltau\OneDrive\Desktop\miniproj\archive\Herlev Dataset\test"
# Load and preprocess train and test data
train_images, train_labels = load_and_preprocess_data(train_dir)
test_images, test_labels = load_and_preprocess_data(test_dir)
# Define shallow CNN model with AE layers
def create_model(input_shape, num_classes):
    model = models.Sequential([
        # Convolutional layer 1
        layers.Conv2D(128, (5, 5), activation='relu', strides=2),
        layers.MaxPooling2D((2, 2), strides=2),
        layers.Flatten(),
        layers.Dense(4096, activation='relu'),
    ])
    # Add AE layer
    model.add(AE(latent_dim=128))
    # Flatten the output before feeding to dense layers
    model.add(layers.Flatten())
    # Add fully connected layers
    model.add(layers.Dense(4096, activation='relu'))
    model.add(layers.Dense(num_classes, activation='softmax'))
    return model
# Create shallow CNN model with AE layers
model = create_model(input_shape=train_images.shape[1:], num_classes=2)
# Adjust learning rate and optimizer
opt = tf.keras.optimizers.SGD(learning_rate=0.01, momentum=0.9)
# Compile the model with appropriate loss function
model.compile(optimizer=opt,
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
# Train the model with specified number of epochs (50)
history = model.fit(train_images, train_labels, epochs=50, batch_size=20, validation_data=(test_images, test_labels))
# Evaluate the model
test_loss, test_accuracy = model.evaluate(test_images, test_labels)
# Print test accuracy
print("Test Accuracy: {:.2f}%".format(test_accuracy * 100))
```

```
Epoch 1/50
WARNING:tensorflow:From C:\Users\prasa_o5lltau\anaconda3\Lib\site-packages\keras\src\utils\tf_utils.py:492: The name tf.ragged.RaggedTensorValue is deprecated. Please use tf.compat.v

WARNING:tensorflow:From C:\Users\prasa_o5lltau\anaconda3\Lib\site-packages\keras\src\engine\base_layer_utils.py:384: The name tf.executing_eagerly_outside_functions is deprecated. Pl

33/33 [==============================] - 35s 927ms/step - loss: 0.7595 - accuracy: 0.7869 - val_loss: 0.4571 - val_accuracy: 0.8358
Epoch 2/50
33/33 [==============================] - 28s 854ms/step - loss: 0.4877 - accuracy: 0.8367 - val_loss: 0.4759 - val_accuracy: 0.8358
Epoch 3/50
33/33 [==============================] - 29s 879ms/step - loss: 0.5814 - accuracy: 0.8367 - val_loss: 0.4866 - val_accuracy: 0.8358
Epoch 4/50
33/33 [==============================] - 28s 849ms/step - loss: 0.4603 - accuracy: 0.8367 - val_loss: 0.4491 - val_accuracy: 0.8358
Epoch 5/50
33/33 [==============================] - 28s 856ms/step - loss: 0.4454 - accuracy: 0.8367 - val_loss: 0.4466 - val_accuracy: 0.8358
Epoch 6/50
33/33 [==============================] - 29s 876ms/step - loss: 0.4453 - accuracy: 0.8367 - val_loss: 0.4466 - val_accuracy: 0.8358
Epoch 7/50
33/33 [==============================] - 28s 840ms/step - loss: 0.4464 - accuracy: 0.8367 - val_loss: 0.4470 - val_accuracy: 0.8358
Epoch 8/50
...
33/33 [==============================] - 44s 1s/step - loss: 0.4456 - accuracy: 0.8367 - val_loss: 0.4467 - val_accuracy: 0.8358
Epoch 48/50
33/33 [==============================] - 42s 1s/step - loss: 0.4454 - accuracy: 0.8367 - val_loss: 0.4466 - val_accuracy: 0.8358
Epoch 49/50
 9/33 [=======>......................] - ETA: 28s - loss: 0.4050 - accuracy: 0.8611
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

CaffeNet with AE

```python
import tensorflow as tf
from tensorflow.keras import layers, models
from sklearn.preprocessing import LabelEncoder
import numpy as np
import os
# Function to implement Autoencoder (AE) layer
class AE(layers.Layer):
    def __init__(self, latent_dim):
        super(AE, self).__init__()
        self.encoder = tf.keras.Sequential([
            layers.Flatten(),
            layers.Dense(latent_dim, activation='relu'),
        ])
        self.decoder = tf.keras.Sequential([
            layers.Dense(4096, activation='sigmoid'),  # Adjust output shape to match the flattened CNN output
            layers.Reshape((32, 32, 4)),  # Adjust output shape to match the input shape before the first ELM layer
        ])
    def call(self, inputs):
        encoded = self.encoder(inputs)
        decoded = self.decoder(encoded)
        return decoded
# Load and preprocess data with increased data augmentation
def load_and_preprocess_data(directory, target_size=(227, 227), num_classes=2):
    image_data = []
    labels = []
    label_encoder = LabelEncoder()
    for label in os.listdir(directory):
        label_dir = os.path.join(directory, label)
        for filename in os.listdir(label_dir):
            image_path = os.path.join(label_dir, filename)
            image = tf.keras.preprocessing.image.load_img(image_path, target_size=target_size)
            image = tf.keras.preprocessing.image.img_to_array(image)
            image = image / 255.0
            image_data.append(image)
            labels.append(label)
    image_data = np.array(image_data)
    labels = label_encoder.fit_transform(labels)
    # Map labels to the nearest valid label
    labels = np.clip(labels, 0, num_classes - 1)
    return image_data, labels
# Define dataset directory paths
train_dir = r"C:\Users\prasa_o5lltau\OneDrive\Desktop\miniproj\archive\Herlev Dataset\train"
test_dir = r"C:\Users\prasa_o5lltau\OneDrive\Desktop\miniproj\archive\Herlev Dataset\test"
```

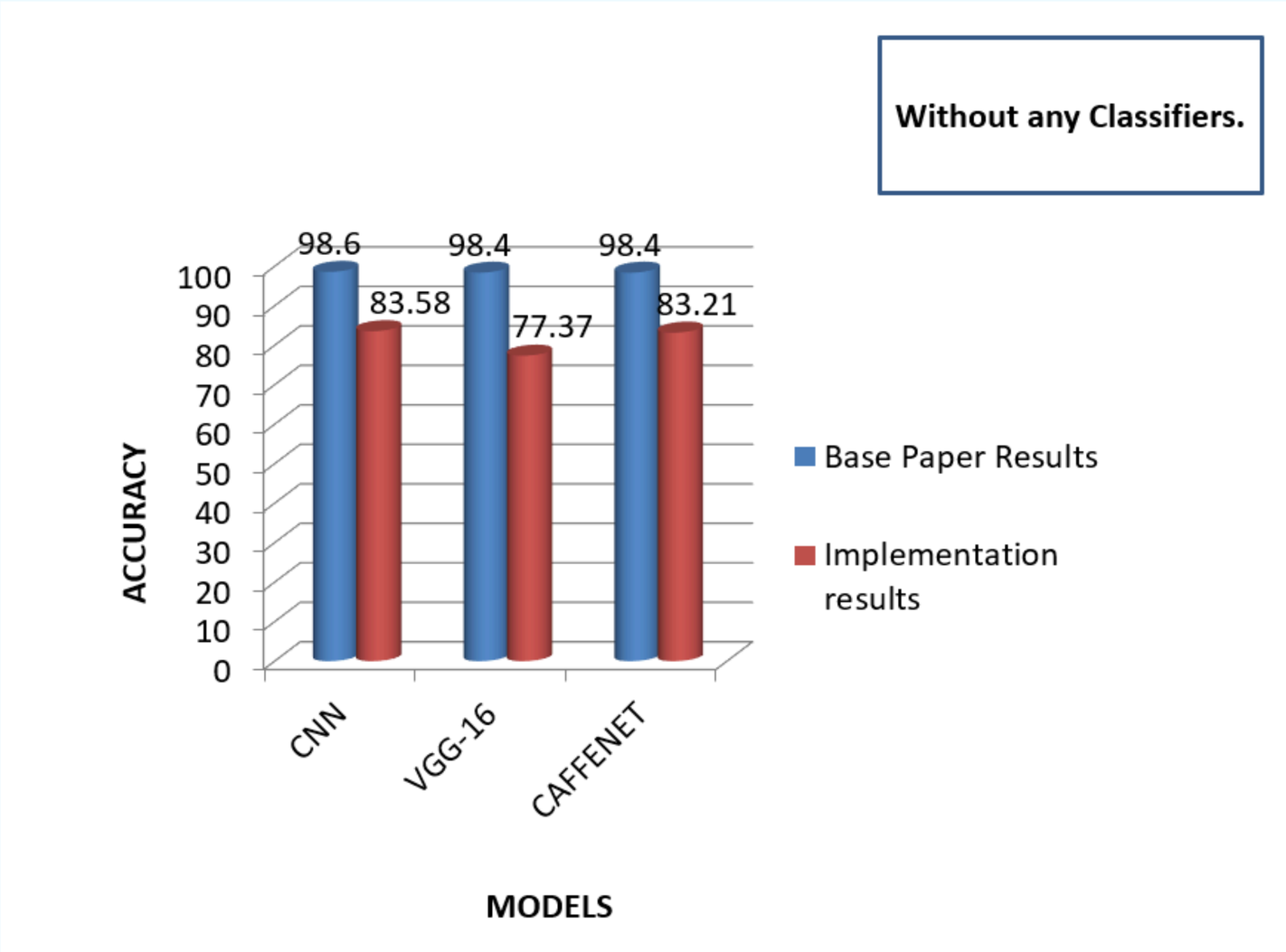
```python
train_images, train_labels = load_and_preprocess_data(train_dir)
test_images, test_labels = load_and_preprocess_data(test_dir)
# Define CaffeNet-like model with AE layers
def create_model(input_shape, num_classes):
    model = models.Sequential([
        layers.Conv2D(96, (11, 11), strides=(4, 4), activation='relu', input_shape=input_shape),
        layers.MaxPooling2D((3, 3), strides=(2, 2)),
        layers.Conv2D(256, (5, 5), padding='same', activation='relu'),
        layers.MaxPooling2D((3, 3), strides=(2, 2)),
        layers.Conv2D(384, (3, 3), padding='same', activation='relu'),
        layers.Conv2D(384, (3, 3), padding='same', activation='relu'),
        layers.Conv2D(256, (3, 3), padding='same', activation='relu'),
        layers.MaxPooling2D((3, 3), strides=(2, 2)),
        layers.Flatten(),
        layers.Dense(4096, activation='relu'),
    ])
    # Add AE layer
    model.add(AE(latent_dim=512))  # Adjust latent dimension according to your requirement
    # Flatten the output before feeding to dense layers
    model.add(layers.Flatten())
    # Add fully connected layers
    model.add(layers.Dense(4096, activation='relu'))
    model.add(layers.Dense(num_classes, activation='softmax'))
    return model
# Create CaffeNet-like model with AE layers
model = create_model(input_shape=train_images.shape[1:], num_classes=2)
# Adjust learning rate and optimizer
opt = tf.keras.optimizers.SGD(learning_rate=0.01, momentum=0.9)
# Compile the model with appropriate loss function
model.compile(optimizer=opt,
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
# Train the model with specified number of epochs (50)
history = model.fit(train_images, train_labels, epochs=50, batch_size=20, validation_data=(test_images, test_labels))
# Evaluate the model
test_loss, test_accuracy = model.evaluate(test_images, test_labels)
```

```
WARNING:tensorflow:From C:\Users\prasa_o5lltau\anaconda3\Lib\site-packages\keras\src\losses.py:2976: The name tf.losses.sparse_softmax_cross_entropy is deprecated. Please use t

WARNING:tensorflow:From C:\Users\prasa_o5lltau\anaconda3\Lib\site-packages\keras\src\backend.py:873: The name tf.get_default_graph is deprecated. Please use tf.compat.v1.get_de

WARNING:tensorflow:From C:\Users\prasa_o5lltau\anaconda3\Lib\site-packages\keras\src\layers\pooling\max_pooling2d.py:161: The name tf.nn.max_pool is deprecated. Please use tf.n

Epoch 1/50
WARNING:tensorflow:From C:\Users\prasa_o5lltau\anaconda3\Lib\site-packages\keras\src\utils\tf_utils.py:492: The name tf.ragged.RaggedTensorValue is deprecated. Please use tf.co

WARNING:tensorflow:From C:\Users\prasa_o5lltau\anaconda3\Lib\site-packages\keras\src\engine\base_layer_utils.py:384: The name tf.executing_eagerly_outside_functions is deprecate

33/33 [==============================] - 156s 4s/step - loss: 0.9095 - accuracy: 0.7123 - val_loss: 0.4579 - val_accuracy: 0.8358
Epoch 2/50
33/33 [==============================] - 84s 3s/step - loss: 0.4470 - accuracy: 0.8367 - val_loss: 0.4631 - val_accuracy: 0.8358
Epoch 3/50
33/33 [==============================] - 84s 3s/step - loss: 0.4565 - accuracy: 0.8367 - val_loss: 0.5410 - val_accuracy: 0.8358
Epoch 4/50
33/33 [==============================] - 85s 3s/step - loss: 0.4735 - accuracy: 0.8367 - val_loss: 0.4507 - val_accuracy: 0.8358
Epoch 5/50
33/33 [==============================] - 85s 3s/step - loss: 0.4445 - accuracy: 0.8367 - val_loss: 0.4625 - val_accuracy: 0.8358
Epoch 6/50
33/33 [==============================] - 84s 3s/step - loss: 0.4872 - accuracy: 0.8367 - val_loss: 0.5008 - val_accuracy: 0.8358
Epoch 7/50
33/33 [==============================] - 84s 3s/step - loss: 0.4678 - accuracy: 0.8367 - val_loss: 0.4487 - val_accuracy: 0.8358
Epoch 8/50
...
33/33 [==============================] - 85s 3s/step - loss: 0.4455 - accuracy: 0.8367 - val_loss: 0.4469 - val_accuracy: 0.8358
Epoch 14/50
33/33 [==============================] - 85s 3s/step - loss: 0.4455 - accuracy: 0.8367 - val_loss: 0.4467 - val_accuracy: 0.8358
Epoch 15/50
22/33 [===================>..........] - ETA: 25s - loss: 0.4644 - accuracy: 0.8250
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

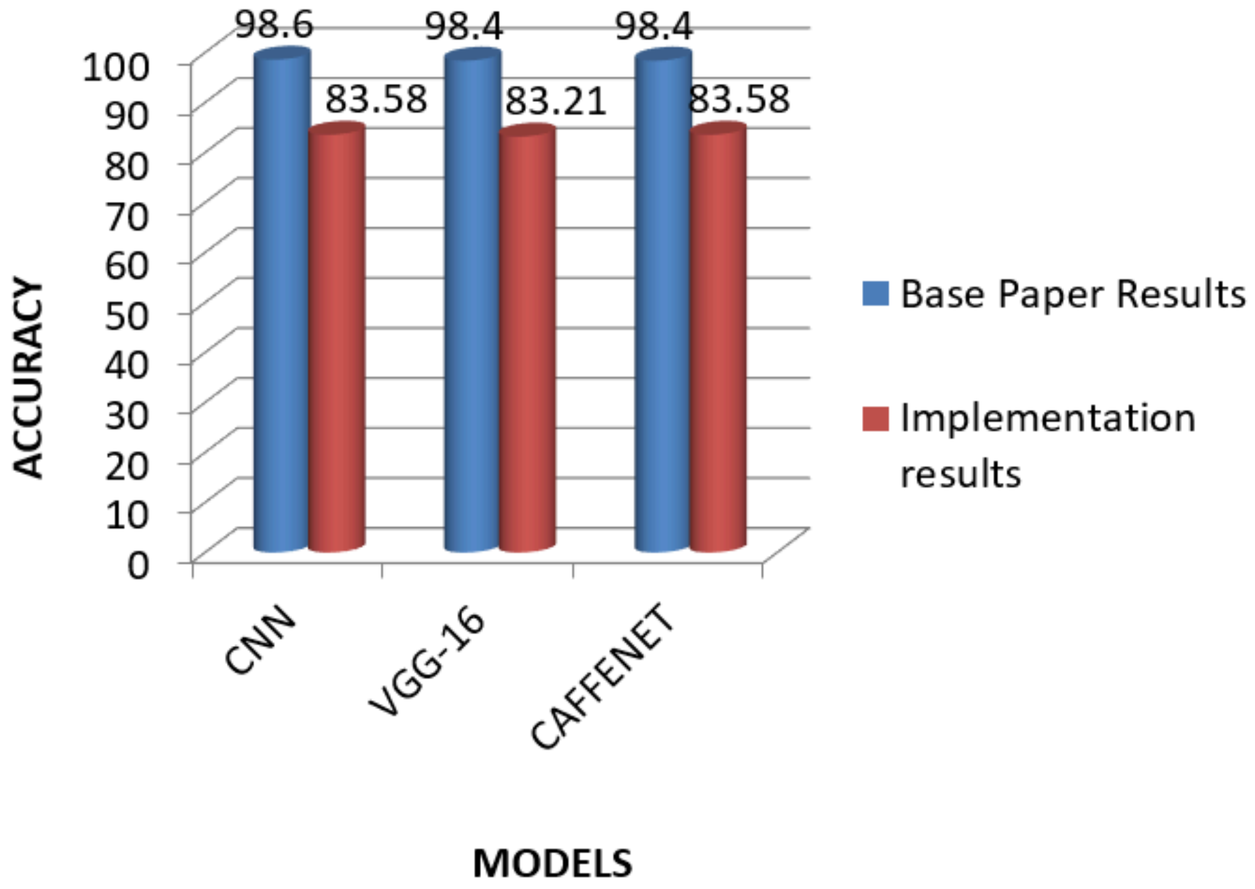
Final Results

Proposed system for 2-class	Shallow CNN	VGG-16	Caffenet
Without Classifier	83.58	77.37	83.21
With ELM	83.58	83.21	83.58
With AE	83.58	83.71	83.58

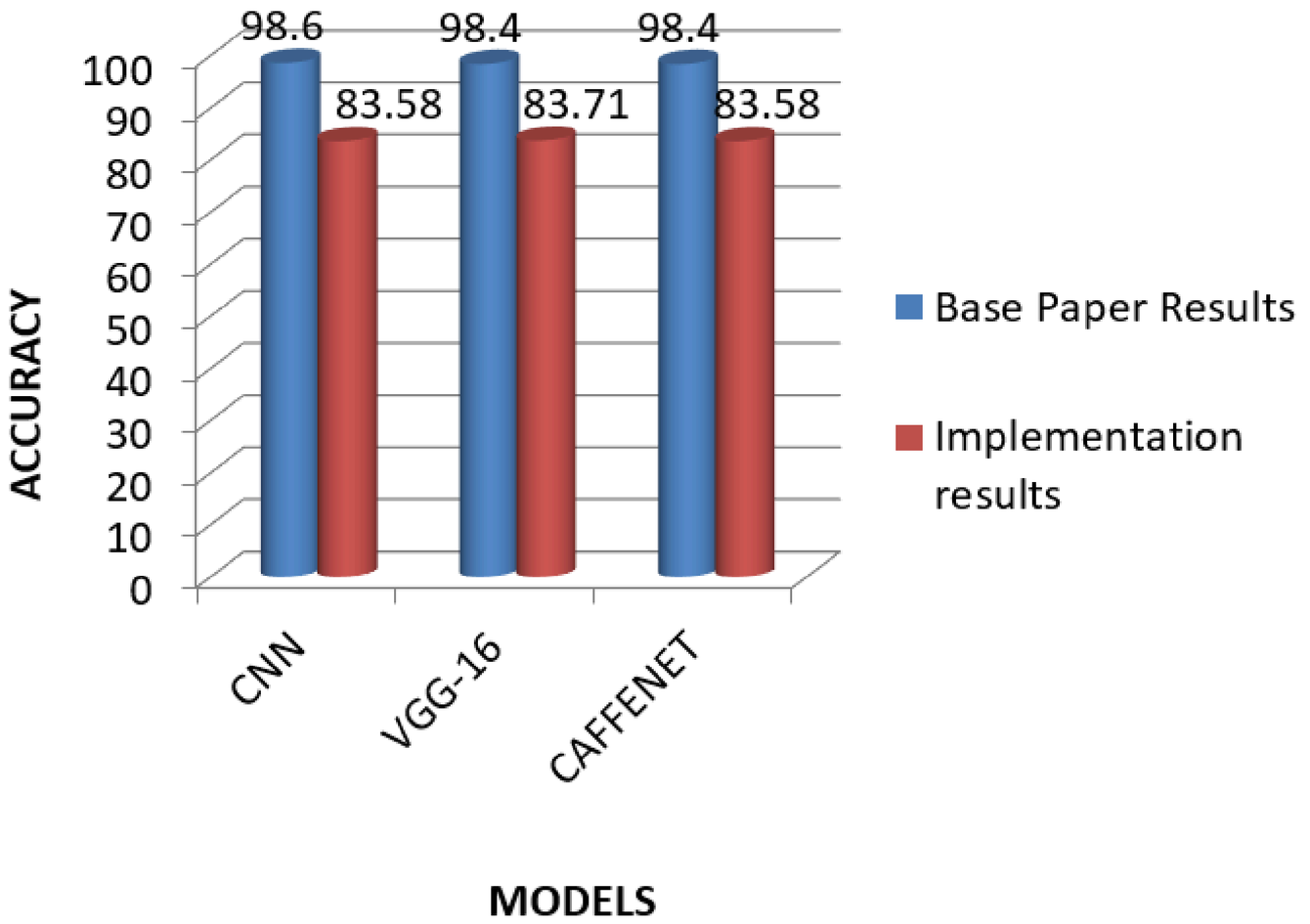
Comparision Graphs



ELMs IN 2-CLASS PROBLEM



AE IN 2-CLASS PROBLEM



Conclusion

In conclusion, the proposed cervical cancer detection and classification system, integrating with the classifiers like ELM and AE classifier after the CNN model, has yielded remarkable results. Through rigorous investigation involving both shallow and deep CNN models VGG-16 and Caffenet. Our implemented system achieved unprecedented accuracies on the Herlev database. Specifically, it attained very good accuracy in the 2-class problem surpassing all previously reported accuracies. These findings underscore the efficiency and potential of our approach in enhancing cervical cancer detection and classification accuracy.

References

1) Z. Alyafeai and L. Ghouti, “A fully-automated deep learning pipeline for cervical cancer classification,” Exp. Syst. Appl., vol. 141, Mar. 2020, Art. no. 112951, doi: 10.1016/j.eswa.2019.112951.

2) W. Yang, X. Gou, T. Xu, X. Yi, and M. Jiang, “Cervical cancer risk prediction model and analysis of risk factors based on machine learning,” in Proc. 11th Int. Conf. Bioinf. Biomed. Technol., May 2019, pp. 50–54, doi: 10.1145/3340074.3340078.

3) H. A. Almubarak, R. J. Stanley, R. Long, S. Antani, G. Thoma, R. Zuna, and S. R. Frazier, “Convolutional neural network based localized classifi□cation of uterine cervical cancer digital histology images,” Proc. Comput. Sci., vol. 114, pp. 281–287, 2017, doi: 10.1016/j.procs.2017.09.044.

4) A. Ghoneim, G. Muhammad, and M. S. Hossain, “Cervical cancer classification using convolutional neural networks and extreme learning machines,” Future Gener. Comput. Syst., vol. 102, pp. 643–649, Jan. 2020, doi: 10.1016/j.future.2019.09.015.

Thank You