Handwriting to Digital Text Conversion (H2DTC)

JV Aditya (12140840),C Nikhil(12140530), B Sri Bhargav Ram (12140460)

REPORT

For the latest version of our project, please visit our GitHub repository at:
`https://github.com/sribhargav1345/ML-Project`

Our project successfully identified handwriting in images, achieving a commendable accuracy of approximately 93%. We conducted extensive experiments, exploring various models such as Convolutional Neural Networks (CNN), Support Vector Machines (SVM), K-Means clustering, and Hidden Markov Models (HMMs) for character recognition. Initial focus was on SVM, and we employed feature extraction techniques alongside pre-processing tasks like contrast enhancement and image normalization to enhance input data quality. Our ensemble strategy combined predictions from diverse models, including Sequential Neural Networks, ResNet variations, and a Convolutional Recurrent Neural Network (C-RNN), optimizing accuracy and adaptability across diverse input data

Data Pre-processing :

We have utilized the MNIST and EMNIST datasets, which together comprise over four hundred thousand handwritten words
Character Recognition utilizes image processing technologies to convert characters on scanned documents into digital forms. It typically performs well in machine-printed fonts.
There are 206,799 words in total. The data was divided into a training set (331,059), testing set (41,382), and validation set (41,382) respectively.

```
# Extract features from images
n_samples = len(train_X)
n_features = img_size * img_size
train_X = np.array(train_X).reshape(n_samples, -1)

n_samples_val = len(val_X)
val_X = np.array(val_X).reshape(n_samples_val, -1)

# Scale the data
n_components = 64  # Adjust the number of components as needed
pca = PCA(n_components=n_components, svd_solver='randomized', whiten=True).fit(train_X)

train_X = pca.transform(train_X)
val_X = pca.transform(val_X)
```

The provided code segment prepares image to flaten into 1D array.Subsequently, the Principal Component Analysis (PCA) for dimensionality reduction. It aims to reduce the data's complexity by setting `n_components` to 64, meaning it wants to represent the data using only 64 principal components.This dimensionality reduction aids in reducing computational

complexity while preserving essential information, making the data more suitable for machine learning tasks involving high-dimensional image data.
We trained our models on dataset containing characters, and then we have segmented the

Figure 1: Character Dataset

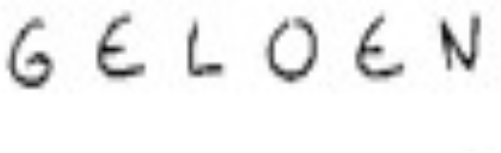

Figure 2: Word Dataset

Model's Performance Summary

Model	Validation Accuracy (%)
SVM	79.76%
K-Means + NN	63.30%
Seq1	92.77%
Seq2	90.26%
Seq3	92.14%
Seq4	40.83%
Seq5	90.65%
Seq6	90.39%
Seq7	65.61%
ResNet1	91.87%
ResNet2	91.89%
ResNet3	92.33%
CRNN	91.54%

Table 1: Validation Accuracy of Different Models

Model Descriptions

Sequential Neural Network

1. **mod1:**
    - ReLU activation for Conv2D layers.
    - LeakyReLU activation after one Conv2D layer.
    - MaxPooling2D for down-sampling.
    - Dropout for regularization.
    - Softmax activation in the final layer for multiclass classification.
2. **mod12:**

- ReLU activation for Conv2D layers.
- GlobalAveragePooling2D for spatial downsampling.
- Dropout for regularization.
- Softmax activation in the final layer for multiclass classification.
- Categorical crossentropy loss and Adam optimizer during training.

3. **mod13:**
   - ReLU activation for Conv2D layers.
   - SpatialDropout2D after one Conv2D layer.
   - GlobalAveragePooling2D for spatial downsampling.
   - Dropout for regularization.
   - Softmax activation in the final layer for multiclass classification.
   - Categorical crossentropy loss and Adam optimizer during training.
4. **mod14:**
   - Various activation functions (ReLU, Sigmoid, Tanh) for different Conv2D layers.
   - GlobalAveragePooling2D for spatial downsampling.
   - Dropout for regularization.
   - Softmax activation in the final layer for multiclass classification.
   - Categorical crossentropy loss and Adam optimizer during training.
5. **mod15:**
   - Different activation functions (ReLU, SGD) for Conv2D and Dense layers.
   - GlobalAveragePooling2D for spatial downsampling.
   - Dropout for regularization.
   - Softmax activation in the final layer for multiclass classification.
   - Categorical crossentropy loss and SGD optimizer during training.
6. **mod16:**
   - Different activation functions (ReLU, Leaky ReLU) for Conv2D layers.
   - GlobalAveragePooling2D for spatial downsampling.
   - Dropout for regularization.
   - Softmax activation in the final layer for multiclass classification.
   - Categorical crossentropy loss and SGD optimizer during training.
7. **mod17:**
   - Different activation functions (ReLU, ELU, Tanh) for Conv2D layers.
   - GlobalAveragePooling2D for spatial downsampling.
   - Dropout for regularization.
   - Softmax activation in the final layer for multiclass classification.
   - Categorical crossentropy loss and Adadelta optimizer during training.
   - Conv2D layer with padding="same" for zero-padding.

ResNet

1. **ResNetOCR (mod2):**
   - Basic ResNet block with two convolutional layers in each block.
   - Global average pooling for spatial downsampling.
   - Batch normalization after each convolutional layer.
   - Softmax activation in the final layer for multiclass classification.
   - Learning rate: 0.001 for Adam optimizer.
   - Batch size during training: 64.
2. **ResNetV1 (mod22):**
   - Similar to ResNetOCR with a different learning rate (0.0005) for the Adam optimizer.
   - Shares the same basic architecture and configuration as ResNetOCR.
3. **ResNetV2 (mod3):**
   - Differs from ResNetV1 by increasing the number of residual blocks in each layer (3, 4, 6).
   - Higher learning rate (0.002) for the Adam optimizer compared to ResNetV1.
   - Overall structure remains consistent with ResNet, featuring convolutional layers, batch normalization, and global average pooling.

HMM

- **Initial and Transition Probabilities:** The `init_trans` function reads the training text file to calculate initial state probabilities and transition probabilities. The initial state probabilities represent the likelihood of each character being the first alphabet of a word, while the transition probabilities model the likelihood of transitioning from one state (character) to another.
- **Emission Probabilities:** The `calc_emission_prob` function calculates emission probabilities based on the observed and reference pixels of characters. It considers noise in the observed pixels and computes the probability of observing a certain set of pixels given a reference character.
- **Simplified Model:** The `simplified` function uses a simplified model to make predictions. It calculates the emission probabilities for each observation and selects the character with the maximum probability for each observation.
- **Variable Elimination (VE):** The `hmm_ve` function implements the variable elimination algorithm to perform MAP inference. It calculates probabilities for each state at each observation and selects the state with the maximum probability.
- **Viterbi Decoding:** The `viterbi` function implements the Viterbi decoding algorithm, which calculates the most likely path of hidden states given observed data. It uses dynamic programming to efficiently find the optimal sequence of hidden states.

- **Recognition and Output:** The code prints the results of three recognition methods: simplified model, HMM with variable elimination (VE), and HMM with MAP inference using Viterbi decoding. The recognized characters are printed for each method.

C-RNN

- The CRNN class includes two Conv2D layers with 64 and 128 filters, respectively, both using a 3x3 kernel and ReLU activation. MaxPooling2D layers follow each convolutional layer for spatial downsampling.
- An LSTM layer with 128 units and return_sequences=True is employed to capture temporal dependencies in the data. This layer operates on the reshaped output of the convolutional layers.
- The output from the LSTM layer is reshaped and flattened.Two Dense layers follow, with 256 neurons and ReLU activation in the first layer and num_classes neurons with softmax activation in the final layer.
- The model is compiled with the Adam optimizer and categorical crossentropy loss, suitable for multi-class classification.
- Finally, we achieved an accuaracy of 91.54% in this model.

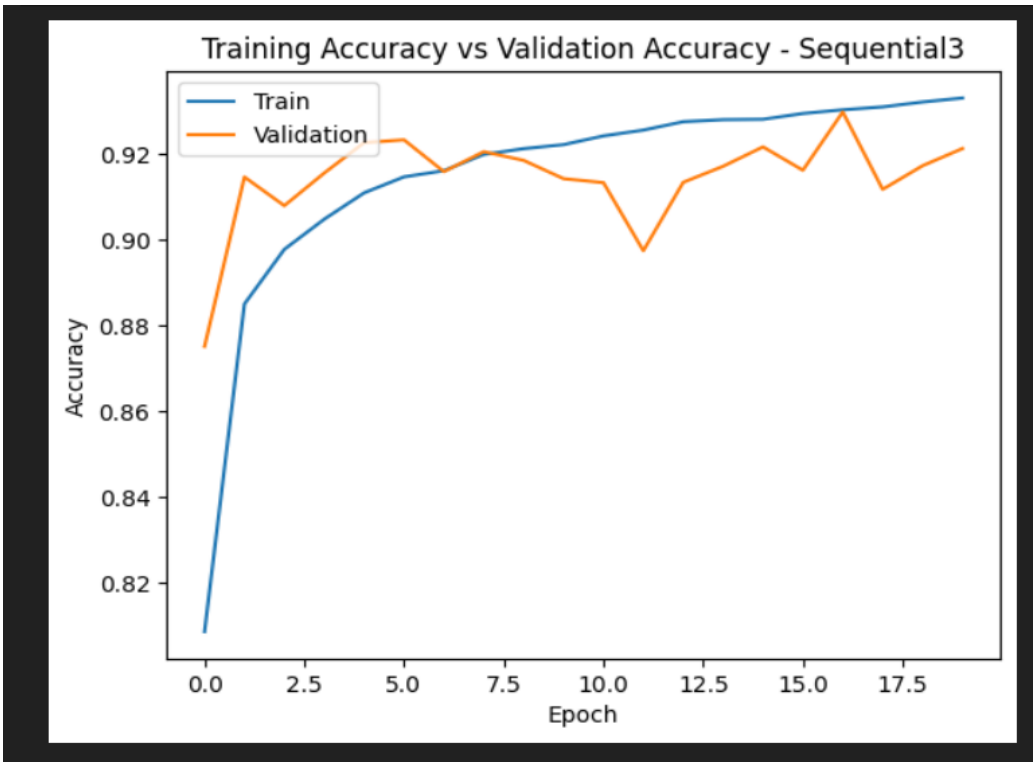

Figure 3: Training Accuracy vs Validation Accuracy

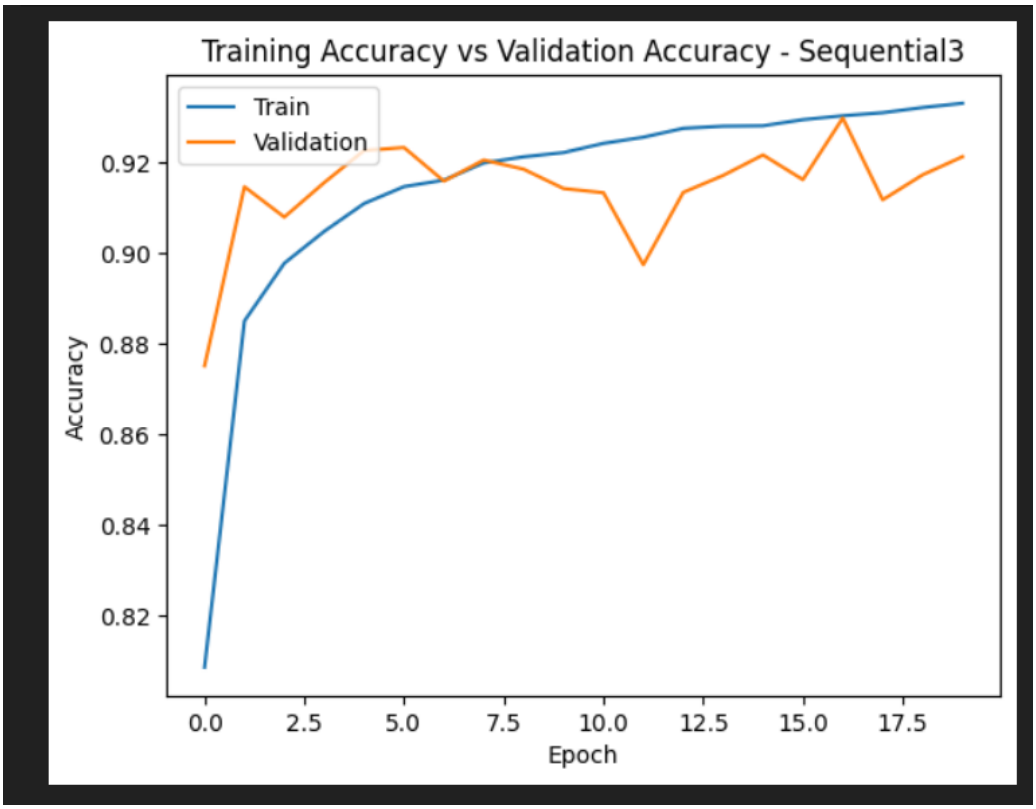

Figure 4: Training accuracy Vs Validation Accuracy.

Ensemble

The ensemble model comprises three neural network architectures: Sequential Model 1, a conventional sequential network incorporating convolutional and dense layers with ReLU activations and dropout for spatial feature extraction and regularization; a Recurrent Convolutional Neural Network (RCCN) designed to capture both spatial and temporal dependencies; and a Residual Neural Network (ResNet) with residual blocks and skip connections to address vanishing gradients in deep networks, specifically effective for image classification. Employing a soft voting strategy, the ensemble combines predictions from these diverse models, harnessing their unique strengths to enhance overall performance and robustness. Sequential Model 1 focuses on spatial features, RCCN incorporates temporal dependencies, and ResNet excels in capturing hierarchical features, collectively providing a comprehensive approach for complex tasks. The ensemble strategy maximizes the collective knowledge of the models, contributing to improved accuracy and adaptability in handling diverse input data.

Conclusion

In conclusion, our project aimed at transforming handwritten text into a digital format has shown promising results, achieving high accuracy in character recognition through the integration of advanced machine learning models like CNNs, ResNets, and an ensemble strategy. Looking ahead, our focus extends beyond character recognition to the more complex task of deciphering entire sentences. We envision addressing challenges related to spatial relationships between words and cursive writing styles. Key areas for improvement include segmentation individual character, exploring additional models, and expanding the dataset.

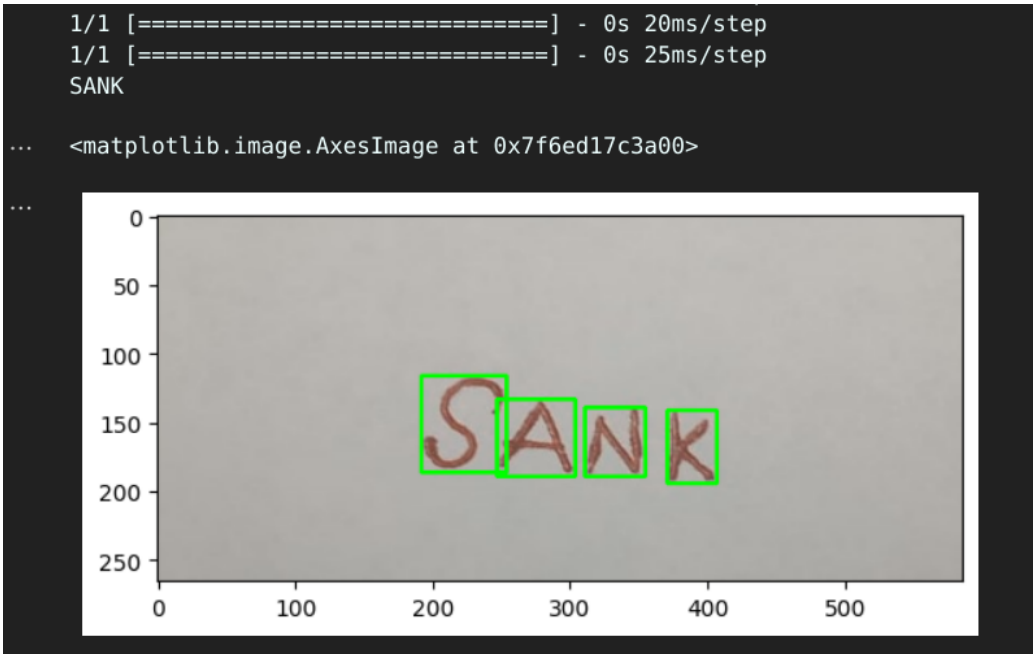

Figure 5: Result of the ensemble on handwritten name

Individual Contributions:

Josyula Venkata Aditya

1. **Pre-processing and Image Enhancement:** Took charge of the initial pre-processing of the handwritten images. This involved tasks such as contrast enhancement, and image normalization. By ensuring that the input data was of the highest quality, we enhanced the model's ability to extract meaningful features.

2. **SVM Model:** Explored the effectiveness of various SVM kernels. My goal was to identify the kernel that best suited the characteristics of our dataset.

3. **Grapheme Segmentation:** Grapheme segmentation was a crucial step in breaking down handwritten text into its constituent characters or graphemes. Worked on developing and implementing a grapheme segmentation method to accurately separate individual characters within the text, which was essential for character-level recognition.

4. **CNN model:** Implemented a model of CNN by experimenting with one specific learning rate and added some new layers, which makes different from other CNN models.

5. **Ensembling:** In the concluding phase, I conducted soft-voting ensembling by combining predictions from the best-performing models. This process aimed to leverage the strengths of individual models, fostering enhanced accuracy and robustness in character recognition.

Chiruvolu Nikhil

1. **Feature Extraction:** The process of feature extraction was pivotal in character recognition. I implemented techniques to extract relevant features from the pre-processed images. These features included structural characteristics, statistical properties, and other discriminative patterns that were essential for accurate recognition. Used PCA for dimensional reduction, generally for regularization.

2. **CNN models:** Implemented 5 different models of CNNs by experimenting with different learning rates, varying the number of layers, and exploring various compilation configurations to construct the best-performing model.

3. **Hidden Markov Models :** using Hidden Markov Models (HMMs) involves breaking the handwriting into discrete states, typically characters or sub-symbols. HMMs model the temporal dependencies between these states as the handwriting evolves. Training the HMM involves estimating transition and emission probabilities from labeled data.

Bollapragada Sri Bhargav Ram

1. **ResNet Models:** Implemented several variations of ResNet models by experimenting with different learning rates, varying the number of layers, and exploring various compilation configurations to construct the best-performing model.

2. **CNN model:** Implemented a model of CNN by experimenting with one specific learning rate and added some new layers, which makes different from other CNN models.

3. **C-RNN Networks:** Additionally, I implemented a C-RNN (Convolutional Recurrent Neural Network) model. I gathered the achieved accuracies from individual models. I got an accuracy of 92% here.

4. **K-Means Clustering:** The dataset was divided into clusters using K-Means clustering, and prototypes were created for each cluster. Characters were recognized by matching new samples to the nearest cluster prototypes. But here, I got a very less accuarcy, which i ignored it later.

In the concluding phase, we conducted ensembling by combining predictions from the best-performing models. This process aimed to leverage the strengths of individual models, fostering enhanced accuracy and robustness in character recognition.