Handwriting Recognition: Architectural Impact Analysis

Optimizing CNN-RNN Models with CTC Loss

https://github.com/canonee/BIAI-proj

Authors:

Kamil Sobociński

Jakub Boduch

Introduction

Handwritten text recognition represents one of the most challenging problems in computer vision and natural language processing. The complexity arises from the inherent variability in human handwriting styles, character formations, spacing, and writing angles. Traditional optical character recognition (OCR) systems, while effective for printed text, struggle with the nuanced variations present in handwritten documents.

- Objective: Develop an optimal CNN-RNN architecture for handwritten text recognition using CTC (Connectionist Temporal Classification) loss function.

- Key Focus: Understanding how specific architectural modifications impact model performance and accuracy through systematic experimentation and analysis.

Task Analysis

- Pure CNN Approaches

    - Pros: Excellent feature extraction capabilities, parallel processing efficiency
    - Cons: Limited ability to handle sequential dependencies, struggles with variable-length outputs

- Traditional RNN/LSTM Methods

    - Pros: Strong sequential modeling, handles variable-length sequences naturally
    - Cons: Requires pre-extracted features, limited spatial understanding

- CNN-RNN Hybrid Architecture (Selected)

    - Pros: Combines spatial feature extraction with temporal sequence modeling, handles both character recognition and sequence alignment
    - Cons: Increased computational complexity, more parameters to optimize
    - Why Selected: Optimal balance between spatial feature understanding and sequential processing capabilities

Dataset Description

- Source: Kaggle - iam_handwriting_word_database

- Dataset Structure:

    - Total samples: 38,305 test samples (based on evaluation data)
    - Training/Validation/Test Split: 27,579 / 3,065 / 7,661

Methodology:

The experimental approach follows a systematic model tuning strategy:

- Baseline Establishment: Implementation of a fundamental CNN-RNN architecture to serve as performance benchmark
- Individual Component Analysis: Systematic modification of single parameters while keeping others constant to isolate the impact of each architectural choice
- Strategic Combination: Integration of best-performing individual modifications to achieve optimal architecture
- Performance Evaluation: Comprehensive analysis of accuracy improvements

This methodology ensures that each architectural decision is data-driven and that the final model represents the synergistic combination of proven improvements rather than arbitrary parameter selection.

Used Libraries

- OpenCV - Image preprocessing
- NumPy - Numerical computations and array operati
- Tensorflow -> Keras, Layers - CNN, RNN, Dense layers and CTC loss implementation
- Scikit-learn -> train_test_split - Data splitting

Base Architecture Overview

Layer (type)	Output Shape	Param #	Connected to
image (InputLayer)	(None, 32, 128, 1)	0	-
conv2d (Conv2D)	(None, 32, 128, 32)	320	image[0][0]
max_pooling2d (MaxPooling2D)	(None, 16, 64, 32)	0	conv2d[0][0]
conv2d_1 (Conv2D)	(None, 16, 64, 64)	18,496	max_pooling2d[0][0]
max_pooling2d_1 (MaxPooling2D)	(None, 8, 32, 64)	0	conv2d_1[0][0]
conv2d_2 (Conv2D)	(None, 8, 32, 128)	73,856	max_pooling2d_1[0][0]
batch_normalization (BatchNormalization)	(None, 8, 32, 128)	512	conv2d_2[0][0]
max_pooling2d_2 (MaxPooling2D)	(None, 4, 32, 128)	0	batch_normalization[0][0]
conv2d_3 (Conv2D)	(None, 4, 32, 128)	147,584	max_pooling2d_2[0][0]
batch_normalization_1 (BatchNormalization)	(None, 4, 32, 128)	512	conv2d_3[0][0]
max_pooling2d_3 (MaxPooling2D)	(None, 2, 32, 128)	0	batch_normalization_1[0][…
reshape (Reshape)	(None, 32, 256)	0	max_pooling2d_3[0][0]
dense (Dense)	(None, 32, 128)	32,896	reshape[0][0]
bidirectional (Bidirectional)	(None, 32, 256)	263,168	dense[0][0]
bidirectional_1 (Bidirectional)	(None, 32, 128)	164,352	bidirectional[0][0]
label (InputLayer)	(None, None)	0	-
dense2 (Dense)	(None, 32, 79)	10,191	bidirectional_1[0][0]
ctc_loss (CTCLayer)	(None, 32, 79)	0	label[0][0], dense2[0][0]

Image 1 – Baseline Architecture

Key Components (Image 1):

- CNN Feature Extraction: Hierarchical feature learning from raw pixels
- RNN Sequence Modeling: Bidirectional LSTM for temporal dependencies
- CTC Loss: Handles variable-length sequences without explicit alignment

Charts

for Individual Component Analysis

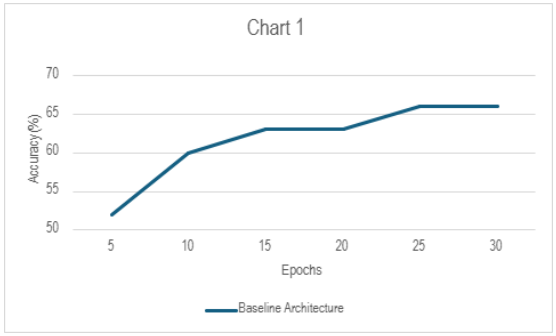

Chart 1 – 1. Baseline Architecture

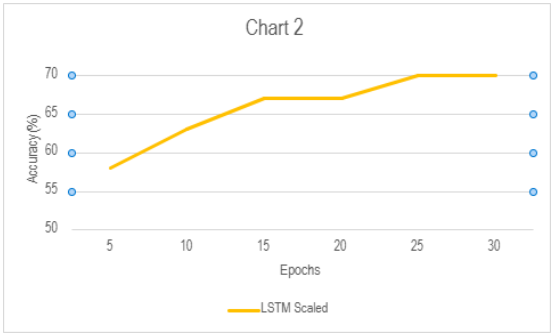

Chart 2 - LSTM Scaling Impact

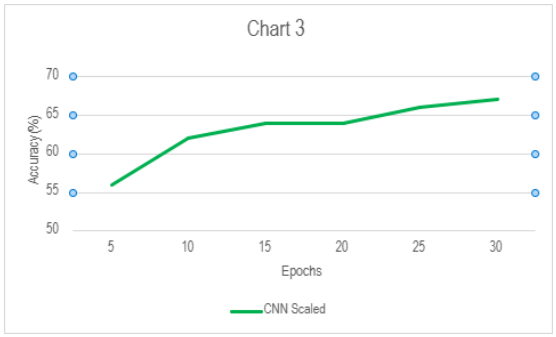

Chart 3 – CNN Filter Scaling

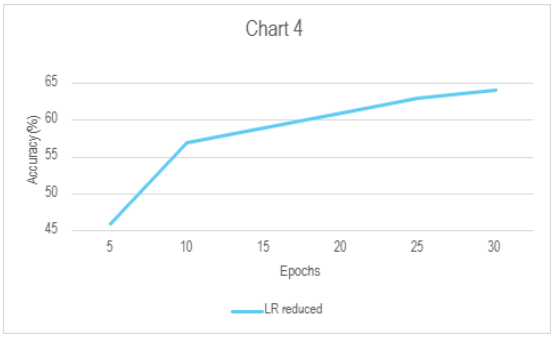

Chart 4 – Learning rate Reduced

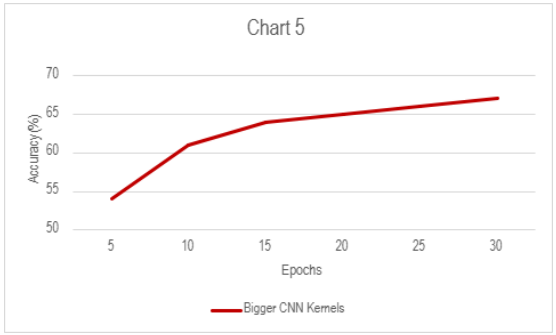

Chart 5 – Bigger Kernel sizes Impact

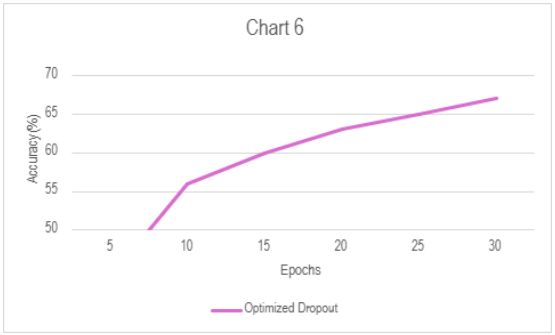

Chart c – Dropout Optimization

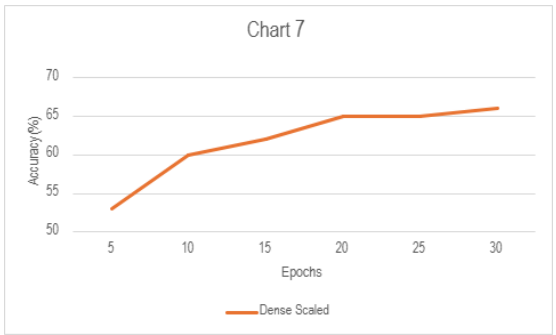

Chart 7 – Dense Layer Scaling

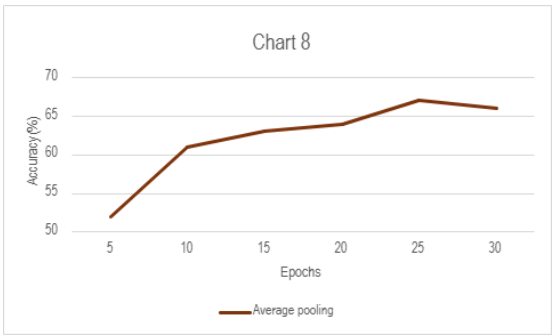

Chart 8 – Average vs Max pooling

Individual Component Analysis

1. Baseline Architecture (Test 1, Chart 1)

Configuration: CNN(32→64→128→128) filters, LSTM(128,64), Dense(128)

Results: 52.47% → 65.94% accuracy (epochs 5→30)

Analysis: Solid foundation but limited by smaller network capacity

2. LSTM Scaling Impact (Test 2, Chart 2)

Modification: LSTM layers: 128,64 → 256,128

Results: 57.66% → 71.38% accuracy (significant +5.44% improvement)

Why it worked:

- Increased memory capacity for longer sequences
- Better gradient flow in bidirectional processing
- Enhanced feature representation in temporal domain

3. CNN Filter Scaling (Test 3, Chart 3)

Modification: Filters: 32→64→128→128 to 64→128→256→256

Results: 56.16% → 65.91% accuracy (minimal improvement)

Analysis: Diminishing returns - computational cost increase without proportional accuracy gain suggests bottleneck was in sequence modeling, not feature extraction

4. Learning Rate Reduction (Test 4, Chart 4)

Modification: LR: 0.001 → 0.0005

Results: Slower initial learning, final accuracy: 63.05%

Critical Insight: Lower learning rate improved stability but reduced convergence speed, indicating the original rate was near-optimal for this architecture

5. Kernel Size Impact (Test 5, Chart 5)

Modification: 3×3 → 5×5 kernels in first two layers

Results: 54.25% → 67.32% accuracy

Why larger kernels helped:

- Increased receptive field captures more context per character
- Better handling of character variations and spacing
- Reduced sensitivity to small positional shifts

6. Dropout Optimization (Test 6, Chart 6)

Modifications:

CNN dropout: 0.2, 0.3 in 2 last layers

LSTM dropout: 0.25 → 0.4

Results: 43.65% → 67.08% accuracy + reduced val_loss (~90 → ~80)

Key Finding: Aggressive regularization reduced overfitting significantly but hurt initial learning capacity

7. Dense Layer Scaling (Test 7, Chart 7)

Modification: Dense: 128 → 256 neurons

Results: Marginal improvement (65.79% final)

Interpretation: Dense layer wasn't the limiting factor; confirms RNN capacity was the primary bottleneck

#Pooling Strategy Impact

8. Average vs Max Pooling (Test 8, Chart 8)

Modification: MaxPool → AvgPool in first two layers

Results: 52.28% → 67.02% accuracy

Mechanism: Average pooling preserves more spatial information, beneficial for fine-grained character features that max pooling might discard

Charts

for Stategic Combinations

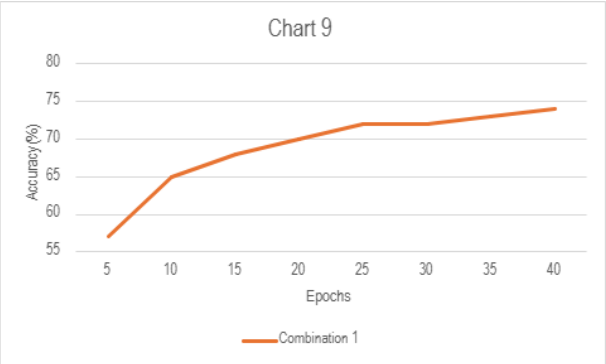

Chart S – Balanced Enhancement

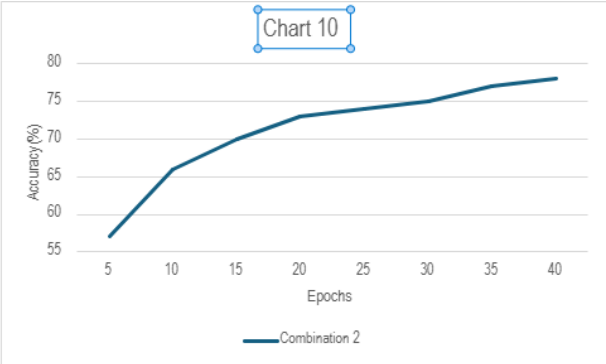

Chart 10 – Maximum Capacity

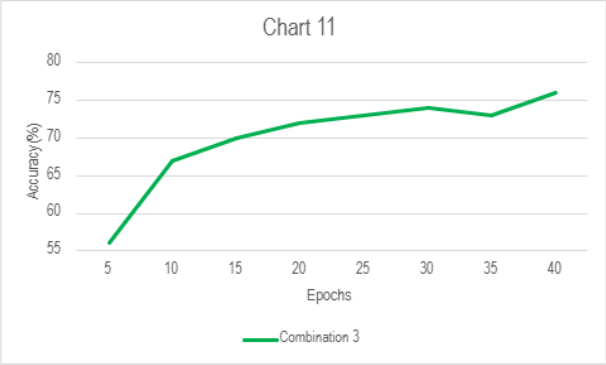

Chart 11 – Experimental Approach

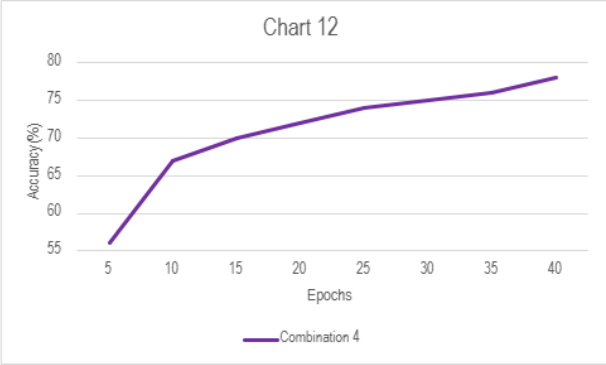

Chart 12 – Deep Architecture

Strategic Combinations

Combination 1: Balanced Enhancement (Chart 9)

Strategy: LSTM scaling + larger kernels + controlled dropout + dense scaling – Image 2

Layer (type)	Output Shape	Param #	Connected to
image (InputLayer)	(None, 32, 128, 1)	0	-
conv2d (Conv2D)	(None, 32, 128, 32)	832	image[0][0]
max_pooling2d (MaxPooling2D)	(None, 16, 64, 32)	0	conv2d[0][0]
conv2d_1 (Conv2D)	(None, 16, 64, 64)	51,264	max_pooling2d[0][0]
max_pooling2d_1 (MaxPooling2D)	(None, 8, 32, 64)	0	conv2d_1[0][0]
conv2d_2 (Conv2D)	(None, 8, 32, 128)	73,856	max_pooling2d_1[0][0]
batch_normalization (BatchNormalization)	(None, 8, 32, 128)	512	conv2d_2[0][0]
dropout (Dropout)	(None, 8, 32, 128)	0	batch_normalization[0][0]
max_pooling2d_2 (MaxPooling2D)	(None, 4, 32, 128)	0	dropout[0][0]
conv2d_3 (Conv2D)	(None, 4, 32, 128)	147,584	max_pooling2d_2[0][0]
batch_normalization_1 (BatchNormalization)	(None, 4, 32, 128)	512	conv2d_3[0][0]
dropout_1 (Dropout)	(None, 4, 32, 128)	0	batch_normalization_1[0][…
max_pooling2d_3 (MaxPooling2D)	(None, 2, 32, 128)	0	dropout_1[0][0]
reshape (Reshape)	(None, 32, 256)	0	max_pooling2d_3[0][0]
dense (Dense)	(None, 32, 256)	65,792	reshape[0][0]
dropout_2 (Dropout)	(None, 32, 256)	0	dense[0][0]
bidirectional (Bidirectional)	(None, 32, 512)	1,050,624	dropout_2[0][0]
bidirectional_1 (Bidirectional)	(None, 32, 256)	656,384	bidirectional[0][0]
label (InputLayer)	(None, None)	0	-
dense2 (Dense)	(None, 32, 79)	20,303	bidirectional_1[0][0]
ctc_loss (CTCLayer)	(None, 32, 79)	0	label[0][0], dense2[0][0]

Image 2 -Balanced Enhancement Architecture

Result: 74.24% accuracy

Design Logic:

- Address sequence modeling bottleneck (larger LSTM)
- Improve feature quality (5×5 kernels)
- Prevent overfitting without hampering learning
- Comprehensive capacity increase

Combination 2: Maximum Capacity - Best combination (Chart 10)

Strategy: CNN scaling + LSTM enhancement + optimized regularization – Image 3

Layer (type)	Output Shape	Param #	Connected to
image (InputLayer)	(None, 32, 128, 1)	0	-
conv2d (Conv2D)	(None, 32, 128, 64)	640	image[0][0]
max_pooling2d (MaxPooling2D)	(None, 16, 64, 64)	0	conv2d[0][0]
conv2d_1 (Conv2D)	(None, 16, 64, 128)	73,856	max_pooling2d[0][0]
max_pooling2d_1 (MaxPooling2D)	(None, 8, 32, 128)	0	conv2d_1[0][0]
conv2d_2 (Conv2D)	(None, 8, 32, 256)	295,168	max_pooling2d_1[0][0]
batch_normalization (BatchNormalization)	(None, 8, 32, 256)	1,024	conv2d_2[0][0]
dropout (Dropout)	(None, 8, 32, 256)	0	batch_normalization[0][0]
max_pooling2d_2 (MaxPooling2D)	(None, 4, 32, 256)	0	dropout[0][0]
conv2d_3 (Conv2D)	(None, 4, 32, 256)	590,080	max_pooling2d_2[0][0]
batch_normalization_1 (BatchNormalization)	(None, 4, 32, 256)	1,024	conv2d_3[0][0]
dropout_1 (Dropout)	(None, 4, 32, 256)	0	batch_normalization_1[0][…
max_pooling2d_3 (MaxPooling2D)	(None, 2, 32, 256)	0	dropout_1[0][0]
reshape (Reshape)	(None, 32, 512)	0	max_pooling2d_3[0][0]
dense (Dense)	(None, 32, 256)	131,328	reshape[0][0]
dropout_2 (Dropout)	(None, 32, 256)	0	dense[0][0]
bidirectional (Bidirectional)	(None, 32, 512)	1,050,624	dropout_2[0][0]
bidirectional_1 (Bidirectional)	(None, 32, 256)	656,384	bidirectional[0][0]
label (InputLayer)	(None, None)	0	-
dense2 (Dense)	(None, 32, 79)	20,303	bidirectional_1[0][0]
ctc_loss (CTCLayer)	(None, 32, 79)	0	label[0][0], dense2[0][0]

Image 3 – Maximum Capacity Architecture

Result: 78.27% accuracy (best performance)

Why this worked best:

- Feature Extraction: 64→128→256→256 filters capture complex character variations
- Sequence Processing: Enhanced LSTM capacity handles longer text sequences
- Balanced Regularization: Prevents overfitting while maintaining learning capacity
- Synergistic Effect: Each component addresses different bottlenecks

Combination 3: Experimental Approach (Chart 11)

Strategy: Mixed pooling + non-standard filter sizes (48,96,192) – Image 4

Layer (type)	Output Shape	Param #	Connected to
image (InputLayer)	(None, 32, 128, 1)	0	-
conv2d (Conv2D)	(None, 32, 128, 48)	1,248	image[0][0]
average_pooling2d (AveragePooling2D)	(None, 16, 64, 48)	0	conv2d[0][0]
conv2d_1 (Conv2D)	(None, 16, 64, 96)	115,296	average_pooling2d[0][0]
average_pooling2d_1 (AveragePooling2D)	(None, 8, 32, 96)	0	conv2d_1[0][0]
conv2d_2 (Conv2D)	(None, 8, 32, 192)	166,080	average_pooling2d_1[0][0]
batch_normalization (BatchNormalization)	(None, 8, 32, 192)	768	conv2d_2[0][0]
dropout (Dropout)	(None, 8, 32, 192)	0	batch_normalization[0][0]
max_pooling2d (MaxPooling2D)	(None, 4, 32, 192)	0	dropout[0][0]
conv2d_3 (Conv2D)	(None, 4, 32, 192)	331,968	max_pooling2d[0][0]
batch_normalization_1 (BatchNormalization)	(None, 4, 32, 192)	768	conv2d_3[0][0]
dropout_1 (Dropout)	(None, 4, 32, 192)	0	batch_normalization_1[0][…
max_pooling2d_1 (MaxPooling2D)	(None, 2, 32, 192)	0	dropout_1[0][0]
reshape (Reshape)	(None, 32, 384)	0	max_pooling2d_1[0][0]
dense (Dense)	(None, 32, 256)	98,560	reshape[0][0]
dropout_2 (Dropout)	(None, 32, 256)	0	dense[0][0]
bidirectional (Bidirectional)	(None, 32, 512)	1,050,624	dropout_2[0][0]
bidirectional_1 (Bidirectional)	(None, 32, 256)	656,384	bidirectional[0][0]
label (InputLayer)	(None, None)	0	-
dense2 (Dense)	(None, 32, 79)	20,303	bidirectional_1[0][0]
ctc_loss (CTCLayer)	(None, 32, 79)	0	label[0][0], dense2[0][0]

Image 4 – Experimental Approach Architecture

Result: 76.33% accuracy

Analysis: An approach attempting to optimize computational requirements

with good results but less systematic than Combination 2

Combination 4: Deep Architecture (Chart 12)

Strategy: Progressive scaling + additional dense layers + recurrent dropout - Image 5 and Image 6

Layer (type)	Output Shape	Param #	Connected to
image (InputLayer)	(None, 32, 128, 1)	0	-
conv2d (Conv2D)	(None, 32, 128, 32)	320	image[0][0]
conv2d_1 (Conv2D)	(None, 32, 128, 32)	9,248	conv2d[0][0]
max_pooling2d (MaxPooling2D)	(None, 16, 64, 32)	0	conv2d_1[0][0]
dropout (Dropout)	(None, 16, 64, 32)	0	max_pooling2d[0][0]
conv2d_2 (Conv2D)	(None, 16, 64, 64)	18,496	dropout[0][0]
conv2d_3 (Conv2D)	(None, 16, 64, 64)	36,928	conv2d_2[0][0]
max_pooling2d_1 (MaxPooling2D)	(None, 8, 32, 64)	0	conv2d_3[0][0]
dropout_1 (Dropout)	(None, 8, 32, 64)	0	max_pooling2d_1[0][0]
conv2d_4 (Conv2D)	(None, 8, 32, 128)	73,856	dropout_1[0][0]
conv2d_5 (Conv2D)	(None, 8, 32, 128)	147,584	conv2d_4[0][0]
batch_normalization (BatchNormalization)	(None, 8, 32, 128)	512	conv2d_5[0][0]
max_pooling2d_2 (MaxPooling2D)	(None, 4, 32, 128)	0	batch_normalization[0][0]
dropout_2 (Dropout)	(None, 4, 32, 128)	0	max_pooling2d_2[0][0]
conv2d_6 (Conv2D)	(None, 4, 32, 256)	295,168	dropout_2[0][0]
batch_normalization_1 (BatchNormalization)	(None, 4, 32, 256)	1,024	conv2d_6[0][0]
max_pooling2d_3 (MaxPooling2D)	(None, 2, 32, 256)	0	batch_normalization_1[0][…
dropout_3 (Dropout)	(None, 2, 32, 256)	0	max_pooling2d_3[0][0]
reshape (Reshape)	(None, 32, 512)	0	dropout_3[0][0]
dense (Dense)	(None, 32, 512)	262,656	reshape[0][0]
dropout_4 (Dropout)	(None, 32, 512)	0	dense[0][0]
dense_1 (Dense)	(None, 32, 256)	131,328	dropout_4[0][0]
dropout_5 (Dropout)	(None, 32, 256)	0	dense_1[0][0]

Image 5 – Deep Architecture part 1

bidirectional (Bidirectional)	(None, 32, 512)	1,050,624	dropout_5[0][0]
bidirectional_1 (Bidirectional)	(None, 32, 256)	656,384	bidirectional[0][0]
label (InputLayer)	(None, None)	0	-
dense2 (Dense)	(None, 32, 79)	20,303	bidirectional_1[0][0]
ctc_loss (CTCLayer)	(None, 32, 79)	0	label[0][0], dense2[0][0]

Image c – Deep Architecture part 2

Result: 77.76% accuracy

Additional complexity helped but didn't surpass the more focused Combination 2

Prediction examples

In Image 10 we can see the results of prediction generated by model from Combination 2. There are listed sequentially paths to the images (Image 7 – 9) and followed by the Tensorflow library logs the predictions of model are presented in Table 1.

Image	Text	Predicted Text
Image 7	„Lucky”	„Lucky”
Image 8	„Federal”	„Fedesal”
Image 9	„amount”	„amount”

Table 1 – Predicted vs true text comparison

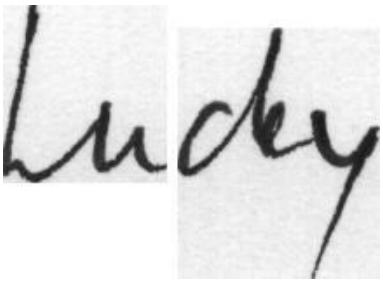

Image 7 – Handwritten word „Lucky”

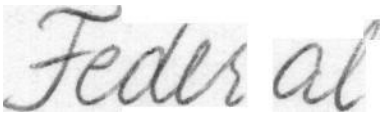

Image 8 – Handwritten word „Federal”

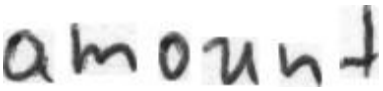

Image S - Handwritten word „amount”

```
(.venv) PS C:\Users\kamso\HandWrittingRecognition> py testModel.py C:\Users\kamso\HandWrittingRecognition\iam_words\words\e01\e01-022\e01-022-00-00.png
WARNING:tensorflow:From C:\Users\kamso\HandWrittingRecognition\.venv\Lib\site-packages\keras\src\backend\tensorflow\core.py:232: The name tf.placeholder i
 tf.compat.v1.placeholder instead.

WARNING:tensorflow:From C:\Users\kamso\HandWrittingRecognition\.venv\Lib\site-packages\keras\src\legacy\backend.py:666: The name tf.nn.ctc_loss is deprecat
t.v1.nn.ctc_loss instead.

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 542ms/step
Predykcja dla 'C:\Users\kamso\HandWrittingRecognition\iam_words\words\e01\e01-022\e01-022-00-00.png': Lucky
(.venv) PS C:\Users\kamso\HandWrittingRecognition> py testModel.py C:\Users\kamso\HandWrittingRecognition\iam_words\words\a01\a01-014x\a01-014x-00-03.png
WARNING:tensorflow:From C:\Users\kamso\HandWrittingRecognition\.venv\Lib\site-packages\keras\src\backend\tensorflow\core.py:232: The name tf.placeholder i
 tf.compat.v1.placeholder instead.

WARNING:tensorflow:From C:\Users\kamso\HandWrittingRecognition\.venv\Lib\site-packages\keras\src\legacy\backend.py:666: The name tf.nn.ctc_loss is deprecat
t.v1.nn.ctc_loss instead.

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 630ms/step
Predykcja dla 'C:\Users\kamso\HandWrittingRecognition\iam_words\words\a01\a01-014x\a01-014x-00-03.png': Fedesal
(.venv) PS C:\Users\kamso\HandWrittingRecognition> py testModel.py C:\Users\kamso\HandWrittingRecognition\iam_words\words\l04\l04-113\l04-113-04-09.png
WARNING:tensorflow:From C:\Users\kamso\HandWrittingRecognition\.venv\Lib\site-packages\keras\src\backend\tensorflow\core.py:232: The name tf.placeholder i
 tf.compat.v1.placeholder instead.

WARNING:tensorflow:From C:\Users\kamso\HandWrittingRecognition\.venv\Lib\site-packages\keras\src\legacy\backend.py:666: The name tf.nn.ctc_loss is deprecat
t.v1.nn.ctc_loss instead.

1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 628ms/step
Predykcja dla 'C:\Users\kamso\HandWrittingRecognition\iam_words\words\l04\l04-113\l04-113-04-09.png': amount
(.venv) PS C:\Users\kamso\HandWrittingRecognition>
```

Image 10 – Prediction results

Conclusions

- The primary bottleneck was sequence modeling—expanding LSTM layers yielded the largest accuracy gains.

- Larger convolutional kernels (5×5) provided more spatial context than simply increasing filter counts.

- Moderate, well-placed dropout effectively reduced overfitting; excessive regularization hindered learning.

- The best performance (78.27 %) came from synergistically combining CNN scaling, LSTM enhancement, and balanced regularization.

- Integrated architectural changes produced a greater overall improvement than individual modifications alone.

Sources

- Python Tensorflow/Keras Documentation: https://www.tensorflow.org/api_docs/python/tf/keras/layers

- Tutorial – Machine Learning Mastery: https://machinelearningmastery.com/handwritten-digit-recognition-using-convolutional-neural-networks-python-keras/

- Youtube playlist - Python Lessons –> Machine Learning Training Utilities https://www.youtube.com/playlist?list=PLbMO9c_jUD46i756J63JJ1_XahCU9Mdaj