

“An AI-based Arabic handwritten character recognition module as a foundational component for automated exam grading systems”

Domain:

Artificial Intelligence / Machine Learning – Computer Vision
(Handwritten Text Recognition)

Problem Statement:

Manual grading of Arabic handwritten exam papers is time-consuming, error-prone, and inconsistent due to handwriting variations. Automated grading systems require accurate recognition of handwritten Arabic text as a prerequisite step. However, Arabic handwriting recognition is challenging because of complex character shapes and visual similarity between letters. This project addresses this challenge by developing a Convolutional Neural Network (CNN) model capable of recognizing individual Arabic handwritten letters, which serves as a critical building block toward an automated Arabic exam grading system.

Project Statement

This project implements a CNN-based system to classify Arabic handwritten letters from scanned images. The trained model accurately recognizes 28 Arabic characters from grayscale images and is deployed using a Streamlit interface for real-time prediction. The system demonstrates the feasibility of Arabic handwritten character recognition as a first step toward automatic grading of handwritten exams.

Data collection and Processing using Python

1-anaconda installed, I used Anaconda prompt

2-Jupyter Notebook installed

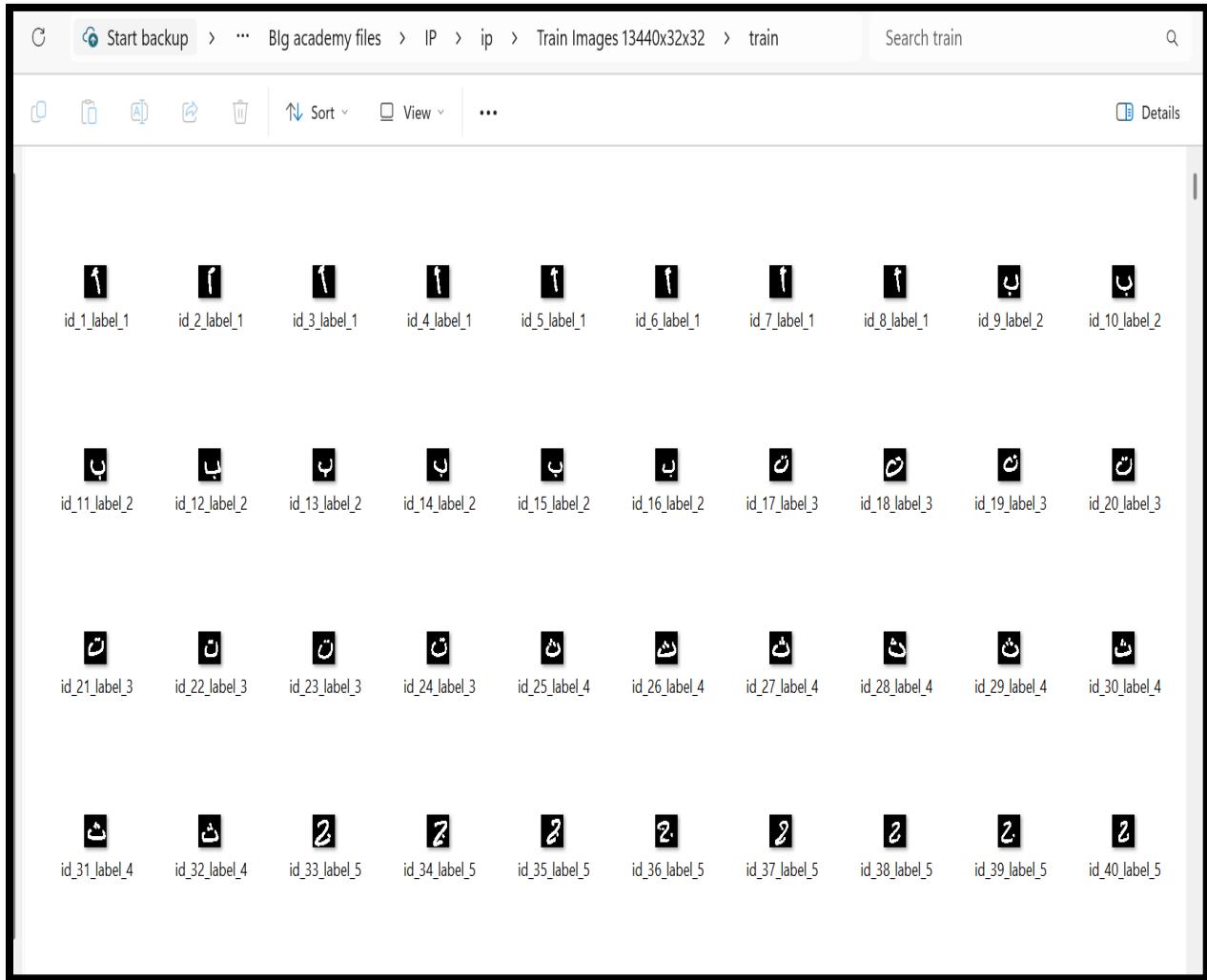
3- Python libraries installed

Data source Link :

The dataset used for training and testing was sourced from a publicly available GitHub repository consisting of Arabic handwritten letter images labeled into 28 classes.

[Arabic-Handwritten-Characters-Dataset/Train Images 13440x32x32.zip at master · mloey/Arabic-Handwritten-Characters-Dataset](https://github.com/mloey/Arabic-Handwritten-Characters-Dataset)

Example screenshot of the images dataset



Data Preprocessing

Data preprocessing was adapted to the image-based nature of the dataset. Unnecessary information such as color channels and file metadata was removed by converting all images to **grayscale**. Images without valid labels were excluded from the dataset, and corrupted or unreadable images were ignored during data loading. All images were resized to 32×32 and **normalized** to ensure consistent input for machine learning models

Coding part of CNN :

Link to my files: train.py and app.py

<https://github.com/UniverseTalker/AR-HW-CNN.git>

- Input: 32x32 grayscale images
- Classes: 28 Arabic letters
- Model: **Convolutional Neural Network (CNN)**
- Framework: **TensorFlow / Keras**
- Deployment: Streamlit

In this picture part of the code related to the CNN Model training

```
56 # Dataset pipeline
57 train_ds = tf.data.Dataset.from_tensor_slices((X_train, y_train_i)).shuffle(5000).batch(BATCH_SIZE).prefetch(tf.data.AUTOTUNE)
58 test_ds = tf.data.Dataset.from_tensor_slices((X_test, y_test_i)).batch(BATCH_SIZE).prefetch(tf.data.AUTOTUNE)
59
60 # CNN model (32 / 32x32)
61 model = tf.keras.Sequential([
62     tf.keras.layers.Input(shape=(IMG_H, IMG_W, CHANNELS)),
63
64     tf.keras.layers.Conv2D(32, 3, padding="same", activation="relu"),
65     tf.keras.layers.BatchNormalization(),
66     tf.keras.layers.MaxPooling2D(),
67
68     tf.keras.layers.Conv2D(64, 3, padding="same", activation="relu"),
69     tf.keras.layers.BatchNormalization(),
70     tf.keras.layers.MaxPooling2D(),
71
72     tf.keras.layers.Conv2D(128, 3, padding="same", activation="relu"),
73     tf.keras.layers.BatchNormalization(),
74     tf.keras.layers.MaxPooling2D(),
75
76     tf.keras.layers.Flatten(),
77     tf.keras.layers.Dense(256, activation="relu"),
78     tf.keras.layers.Dropout(0.3),
79     tf.keras.layers.Dense(num_classes, activation="softmax"),
80 ])
81
82 model.compile(
83     optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3),
84     loss="sparse_categorical_crossentropy",
85     metrics=["accuracy"]
86 )
87
```

How to run the application:

1. Activate environment (mine is: ip-project1)
2. Run training: python train.py (then model file and json file created)
3. Run streamlit app : python -m streamlit run app.py

Anaconda Prompt X + v - X

```
(base) C:\Users\Lenovo>conda activate ip-project1

(ip-project1) C:\Users\Lenovo>cd C:\Users\Lenovo\Desktop\Big academy files\IP\ip

(ip-project1) C:\Users\Lenovo\Desktop\Big academy files\IP\ip>
```

localhost:8889/lab

File Edit View Run Kernel Tabs Settings Help

Untitled.ipynb Notebook Python (ip-project1)

Name Modified

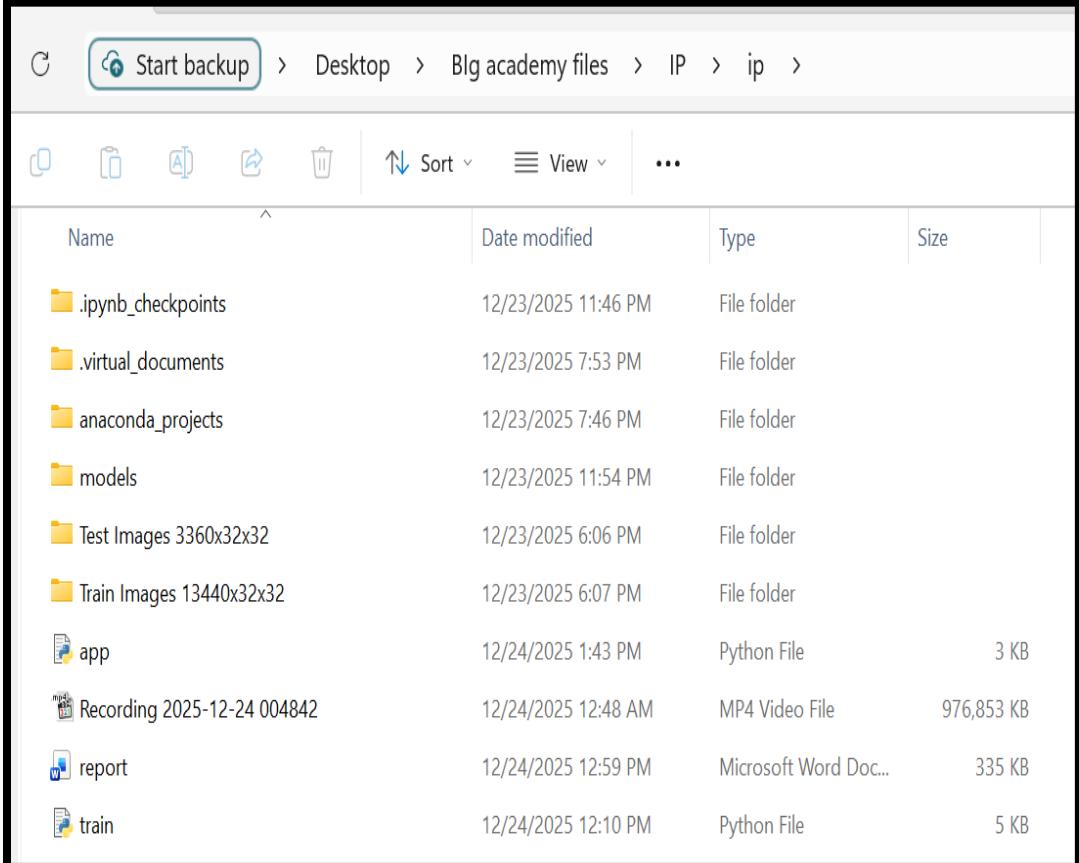
anaconda_proje...	12h ago
anaconda3	12h ago
Contacts	last yr.
Desktop	yesterday
Documents	4h ago
Downloads	5h ago
Favorites	last yr.
Folder	last yr.

```
[1]: import sys
print(sys.executable)

C:\Users\Lenovo\anaconda3\envs\ip-project1\python.exe

[*]: pip install tensorflow streamlit pillow numpy scikit-learn matplotlib
```

My project folder structure:



The screenshot shows a Windows File Explorer window with a black border. The path bar at the top shows: C: > Start backup > Desktop > Blg academy files > IP > ip >. Below the path bar is a toolbar with icons for copy, move, search, and delete, followed by 'Sort' and 'View' dropdowns and an ellipsis button. The main area is a table listing files and folders:

Name	Date modified	Type	Size
.ipynb_checkpoints	12/23/2025 11:46 PM	File folder	
.virtual_documents	12/23/2025 7:53 PM	File folder	
anaconda_projects	12/23/2025 7:46 PM	File folder	
models	12/23/2025 11:54 PM	File folder	
Test Images 3360x32x32	12/23/2025 6:06 PM	File folder	
Train Images 13440x32x32	12/23/2025 6:07 PM	File folder	
app	12/24/2025 1:43 PM	Python File	3 KB
Recording 2025-12-24 004842	12/24/2025 12:48 AM	MP4 Video File	976,853 KB
report	12/24/2025 12:59 PM	Microsoft Word Doc...	335 KB
train	12/24/2025 12:10 PM	Python File	5 KB

Examples of Prediction :

Arabic Letter Recognizer (CNN)

Upload a letter image (32x32 recommended). The model will predict the letter.

Upload an image

Drag and drop file here
Limit 200MB per file • PNG, JPG, JPEG

Browse files

 id_7_label_4.png 171.08

X



Uploaded image

Predicted letter



Label: 4

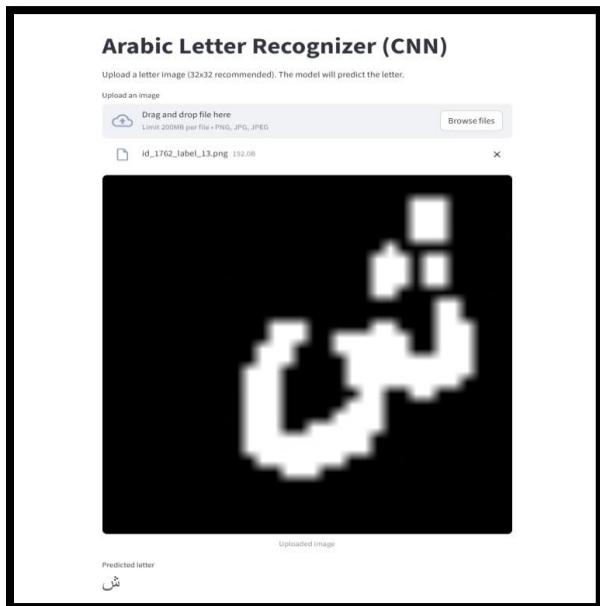
Prediction

Confidence

99.12%

Top 3 predictions

1. ت (label 4) - 99.12%
2. ج (label 13) - 0.66%
3. ع (label 25) - 0.11%



Model Evaluation : Accuracy, Confusion Matrix, and Precision/Recall:

Model accuracy : 0.954

In the code we can see it inside train.py file :

```
acc = accuracy_score(y_test_i, y_pred)
print("\nTest Accuracy:", acc)
```

Results while running file :train.py

```
Test Accuracy: 0.9541666666666667

Classification report:
      precision    recall  f1-score   support

          0       1.00     1.00     1.00      120
          1       0.99     0.98     0.99      120
          2       0.92     0.95     0.93      120
          3       0.93     0.93     0.93      120
          4       0.99     0.97     0.98      120
          5       0.93     0.98     0.96      120
          6       0.97     0.97     0.97      120
          7       0.94     0.95     0.95      120
          8       0.90     0.91     0.90      120
          9       0.91     0.97     0.94      120
```

```

accuracy                           0.95      3360
macro avg                           0.95      3360
weighted avg                          0.95      3360

Confusion matrix:
[[120   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [ 0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [ 0   118   0   0   0   0   0   0   0   0   0   0   0   0   0   1   0   0   0   0]
 [ 0   0   0   0   0   0   0   0   0   0   0   0   0   0   1]
 [ 0   0   114   5   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [ 0   0   1   0   0   0   0   0   0   0   0   0   0   0   0]
 [ 0   0   5   112   0   0   0   0   0   0   0   0   0   0   0   2   0   0   0   0   0]
 [ 0   0   0   1   0   0   0   0   0   0   0   0   0   0]
 [ 0   0   0   0   117   2   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [ 0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [ 0   0   0   0   1   118   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1]
 [ 0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [ 0   0   0   0   0   3   116   0   0   0   0   0   0   0   0   0   0   0   0   0   1]
 [ 0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [ 0   1   0   0   0   0   1   0   114   2   1   0   0   0   0   0   0   0   0   0]
 [ 0   0   0   0   0   0   0   0   0   0   1   0]
 [ 0   0   0   0   0   0   0   0   4   109   0   2   0   0   0   1   1   0   0   0]
 [ 0   0   1   0   1   0   0   0   0   1   0]
 [ 0   0   0   0   0   0   0   0   1   0   116   3   0   0   0   0   0   0   0   0]
 [ 0   0   0   0   0   0   0   0   0   0   0   0   0]
 [ 0   0   0   0   0   0   0   1   7   9   101   0   0   0]
 [ 0   0   0   0   1   0   1   0   0   0   0   0   0]

```

Jupyter Notebook

ML Models work (Logistic Regression & SVM)

Original Data are images but to deal with ML Models

images converted into 32X32 =1024 features vector to store as a CSV so ML models can work on .

Note: I chose LR and LSVM because both are good with large number of features 1024 and both are known base ML for comparison with CNN.

Screenshots for ML Work :

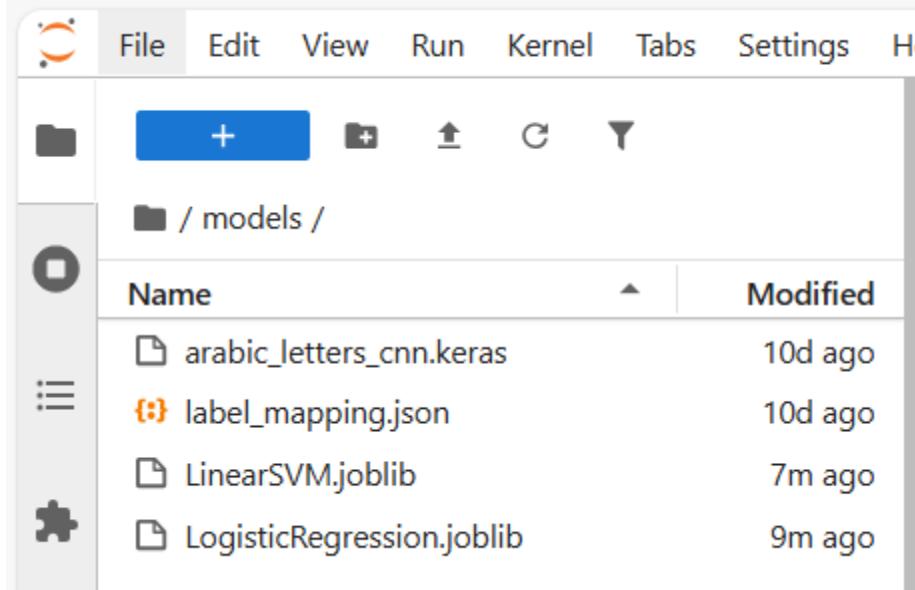
The screenshot shows a Jupyter Notebook interface with two tabs: "train_ml_models.py" and "Terminal 1".

The "train_ml_models.py" tab displays the following output:

```
Training LogisticRegression ...
===
Accuracy: 0.33690476190476193
Precision (macro): 0.33703266895613637
Recall (macro): 0.33690476190476193
F1 (macro): 0.33652415096423843

Classification report:
precision    recall   f1-score   support
          1      0.70     0.71     0.71      120
          2      0.46     0.47     0.47      120
          3      0.24     0.24     0.24      120
          4      0.32     0.33     0.33      120
          5      0.23     0.23     0.23      120
          6      0.26     0.27     0.26      120
          7      0.22     0.24     0.23      120
          8      0.45     0.48     0.47      120
          9      0.34     0.28     0.31      120
         10      0.52     0.57     0.54      120
         11      0.38     0.39     0.39      120
         12      0.36     0.33     0.35      120
         13      0.26     0.27     0.26      120
         14      0.23     0.25     0.24      120
         15      0.23     0.24     0.24      120
         16      0.30     0.34     0.32      120
         17      0.36     0.36     0.36      120
         18      0.32     0.31     0.31      120
         19      0.35     0.34     0.35      120
         20      0.26     0.24     0.25      120
         21      0.27     0.26     0.26      120
         22      0.33     0.35     0.34      120
         23      0.40     0.38     0.39      120
         24      0.39     0.35     0.37      120
         25      0.29     0.26     0.27      120
         26      0.31     0.31     0.31      120
         27      0.30     0.30     0.30      120
         28      0.36     0.33     0.34      120

accuracy           0.34      0.34      0.34      3360
macro avg           0.34      0.34      0.34      3360
```



Model Comparison and Analysis:

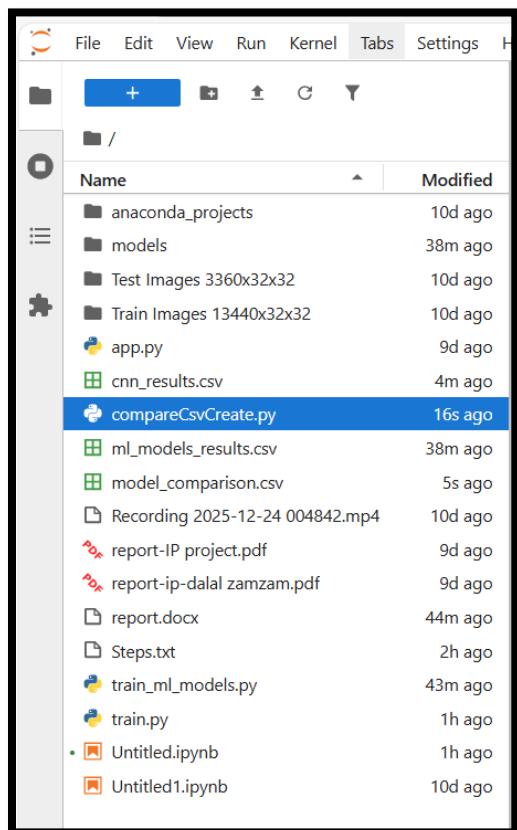
-Screenshot of Csv file for comparison

The screenshot shows a code editor with two tabs open. The left tab contains the code for `train_ml_models.py`, which reads two CSV files ('ml_models_results.csv' and 'cnn_results.csv') into pandas DataFrames, concatenates them, and saves the result to 'model_comparison.csv'. The right tab contains the code for `compareCsvCreate.py`. A callout box provides details about this script: Name: compareCsvCreate.py, Path: compareCsvCreate.py, Last Saved: 1/3/26, 9:21 AM, Last Checkpoint: 1/3/26, 9:21 AM.

```
1 import pandas as pd
2
3 df_ml = pd.read_csv("ml_models_results.csv")
4 df_cnn = pd.read_csv("cnn_results.csv")
5
6 df_all = pd.concat([df_ml, df_cnn], ignore_index=True)
7 df_all.to_csv("model_comparison.csv", index=False)
8 df_all
```

Name: compareCsvCreate.py
Path: compareCsvCreate.py
Last Saved: 1/3/26, 9:21 AM
Last Checkpoint: 1/3/26, 9:21 AM

-Screenshot of My Work Folder content:



CSV results Analysis

	Model	Accuracy	Precision_macro	Recall_macro	F1_macro	
1	LogisticRegression	0.3369047619047619	0.3370326689561363	0.3369047619047619	0.3365241509642384	
2	LinearSVM	0.3113095238095238	0.3044829585854569	0.3113095238095238	0.3056717788009361	
3	CNN	0.9508928571428572	0.9513500378436686	0.9508928571428572	0.950807053444805	

Based on the evaluation results shown in *model_comparison.csv*, a clear performance difference can be observed between traditional machine learning models and the CNN model.

Logistic Regression

- Accuracy: ~33.7%
- Precision (macro): ~33.7%
- Recall (macro): ~33.7%
- F1-score (macro): ~33.7%

Linear SVM

- Accuracy: ~31.1%
- Precision (macro): ~30.4%
- Recall (macro): ~31.1%
- F1-score (macro): ~30.6%

Convolutional Neural Network (CNN)

- Accuracy: ~95.1%
- Precision (macro): ~95.1%

- Recall (macro): ~95.1%
- F1-score (macro): ~95.1%

The CNN significantly outperformed both traditional machine learning models across all evaluation metrics. This is because CNNs are specifically designed for image-based tasks and are able to learn spatial and structural features such as edges, strokes, and shapes that are critical for recognizing Arabic handwritten characters.

Overall Comparison

The results clearly demonstrate that while traditional machine learning models can be used as baseline classifiers, they are not well suited for image-based handwritten character recognition. The CNN model achieves substantially higher accuracy and robustness by preserving spatial information and automatically learning hierarchical visual features.

Key Conclusion:

The comparison confirms that convolutional neural networks are significantly more effective than traditional machine learning models for Arabic handwritten letter recognition, making CNNs the most suitable choice as a core component in automated handwritten exam grading systems.

Dalal Adnan zamzam