12/27/2023

Data Science Tools

Arabic Handwritten Digit Recognition

Name	ID
Mohamed Yousri Ibrahim Abdallah Ibrahim	**20221509866**
Abdulrahman Salah Anwar Abdo	**20221458503**
Mahmoud Reda Hassan Mohamed Nour	**20221469438**
Rawan Shawky Arafa	**20221320560**
Rawan Abdelfattah Abdelsalam	**20221451428**
Omar Mahmoud Gaber	**20221375788**
Farah Essam Eldeen Said	**20221462956**
Rawana Khaled Mahmoud	**20221376722**
Menna Allah Mohamed Elsaid	**20221462499**

Dr. Mohamed Abdel EL Hafeez

Arabic Handwritten Digit Recognition

Abstract

This project explores the domain of handwritten digit recognition using the Arabic Handwritten Digits Dataset. The primary focus involves the implementation and comparison of three distinct models: Convolutional Neural Network (CNN), Artificial Neural Network (ANN), and Random Forest. The utilization of data augmentation techniques, specifically ImageDataGenerator, enhances the robustness of the models by introducing variations in the training dataset. The CNN and ANN architectures are detailed, and their training processes are outlined, showcasing the iterative refinement of the models over multiple epochs.

This project aims to develop and evaluate machine learning models for recognizing handwritten Arabic digits. The dataset used in this project consists of Arabic handwritten digits, with 60,000 training samples and 10,000 test samples.

Additionally, the project evaluates and compares the models' performance using various metrics, including accuracy and confusion matrices. The integration of Random Forest as an ensemble learning approach provides a comparative analysis against traditional neural network architectures. The results and visualizations offer insights into the strengths and limitations of each model, contributing to the broader discourse on handwritten digit recognition methodologies.

Introduction

Handwritten digit recognition is a fundamental task in the realm of computer vision, finding widespread applications in fields such as postal automation, finance, and information retrieval. This project focuses on leveraging the Arabic Handwritten Digits Dataset to develop and compare the performance of three distinct models: Convolutional Neural Network (CNN), Artificial Neural Network (ANN), and Random Forest.

Handwritten digit recognition poses unique challenges, including variations in writing styles and diverse shapes. Through the use of advanced techniques such as data augmentation and ensemble learning, we aim to enhance the models' ability to generalize and accurately classify handwritten digits. This introduction provides a comprehensive overview of the project's objectives, methodologies, and the significance of the Arabic Handwritten Digits Dataset in advancing the field of digit recognition.

Project Goals:

- Developing machine learning models to accurately recognize Arabic handwritten digits (0-9).
- Compare the performance of Convolutional Neural Networks (CNNs) and Artificial Neural Networks (ANNs) for this task.
- Experiment with Random Forest as an alternative approach.

Data:

- Source: Arabic Handwritten Digits Dataset (CSV format)
- Size: 60,000 training images, 10,000 test images
- Image Format: 28x28 grayscale pixels
- Labels: Integers representing handwritten digits (0-9)

Code Explanation:

1. Importing Libraries:

In [1]:
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
```

Import libraries

In [2]:
```
import tensorflow as tf
import numpy as np
import plotly.express as px
import cufflinks as cf
import keras
from tensorflow.keras.datasets import mnist
from tensorflow.keras.layers import Input, Conv2D, Dense, Dropout, MaxPool2D, Flatten
from tensorflow.keras import Model, Sequential
from tensorflow.keras.losses import sparse_categorical_crossentropy
from tensorflow.keras.metrics import sparse_categorical_accuracy
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import classification_report
```

Purpose:

- **NumPy:** Numerical operations and array manipulation
- **Pandas:** Data manipulation and analysis
- **Matplotlib:** Visualizations (plots)
- **TensorFlow:** Building and training machine learning models
- **Plotly Express, Cufflinks:** Interactive visualizations
- **Keras:** High-level API for TensorFlow, simplifying model building
- **MNIST Dataset:** Although not directly used in this project, the inclusion of the MNIST dataset import from Keras suggests a potential for comparison or transfer learning.
- **ImageDataGenerator:** Critical for data augmentation during training, enhancing the model's ability to generalize.

2. Data Preprocessing

The dataset is loaded into NumPy arrays using Pandas. It includes training and test images along with their corresponding labels. Data augmentation is performed using the ImageDataGenerator from Keras, which includes operations such as rotation, shifting, shearing, and zooming. The augmented data is then concatenated with the original data to enhance the training set.

2.1. Data Loading

The dataset is loaded into NumPy arrays using the Pandas library. Four CSV files are utilized: training images, training labels, test images, and test labels.

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# The Arabic Handwritten Digits Dataset is loaded into NumPy arrays using Pandas.

X_train = pd.read_csv("csvTrainImages 60k x 784.csv").values
Y_train = pd.read_csv("csvTrainLabel_60k_x_1.csv").values
X_test = pd.read_csv("csvTestImages 10k x 784.csv").values
Y_test = pd.read_csv("csvTestLabel_10k_x_1.csv").values
```

Purpose:

- **CSV File Loading:** Reads data from CSV files containing pixel values for images and corresponding labels.
- **Pandas DataFrame Conversion:** Converts data to NumPy arrays for compatibility with machine learning models.

2.2. Data Reshaping

```
# Uncomment to Reshape the data if needed (assuming your data is flat, not images)
# X_train = X_train.reshape(-1, 28, 28, 1)
# X_test = X_test.reshape(-1, 28, 28, 1)
```

The necessity to reshape data is considered, assuming the data is flat and not in the form of images.

2.3. Data Augmentation

For enhancing the model's generalization capabilities, an **ImageDataGenerator** from Keras is employed for data augmentation.

```
# Create an ImageDataGenerator for data augmentation
datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

# Fit the generator on your training data
datagen.fit(X_train.reshape(-1, 28, 28, 1))
```

Purpose:

- **ImageDataGenerator Configuration:** Sets up parameters for data augmentation.
- **Data Augmentation:** Enables the model to learn from a larger variety of images by applying transformations like rotation, shifting, and flipping.

2.4. Generating and Concatenating Augmented Data

Augmented samples are generated and added to the training data.

```
# Generate augmented data
augmented_data = []
augmented_labels = []
for X_batch, Y_batch in datagen.flow(X_train.reshape(-1, 28, 28, 1), Y_train, batch_size=augmented_samples_per_sample):
    augmented_data.append(X_batch)
    augmented_labels.append(Y_batch)
    if len(augmented_data) * augmented_samples_per_sample >= len(X_train):
        break

# Concatenate the augmented data and labels
augmented_data = np.concatenate(augmented_data, axis=0).reshape(-1, 28*28)
augmented_labels = np.concatenate(augmented_labels, axis=0)

# Concatenate the original data and augmented data
X_train_augmented = np.concatenate([X_train, augmented_data], axis=0)
Y_train_augmented = np.concatenate([Y_train, augmented_labels], axis=0)

# Now X_train_augmented and Y_train_augmented contain the original data along with augmented samples
```

Purpose:

- **Data Augmentation Execution:** Generates augmented data in batches using the previously configured **ImageDataGenerator**.
- **Loop Break Condition:** Ensures that the number of augmented samples reaches the desired level.
- **Concatenation of Data:** Merges the original training data with the augmented data.
- **Reshaping Augmented Data:** Ensures the data is in the correct format for training.

2.5. Data Normalization and Reshaping:

Normalize and reshape the data

```
In [4]: X_train_augmented, X_test = (X_train_augmented / 255.0), (X_test / 255.0) # Normalize to [0, 1]

In [5]: X_train_augmented = X_train_augmented.reshape(-1, 28, 28, 1)
        X_test = X_test.reshape(-1, 28, 28, 1)
        X_train_augmented.shape

Out[5]: (119998, 28, 28, 1)
```

3. Convolutional Neural Network (CNN) Model

3.1. Model Architecture - CNN

A Convolutional Neural Network (CNN) is used for image recognition. The CNN architecture consists of two convolutional layers with max-pooling, followed by a flatten layer, a dense layer, a dropout layer for regularization, and a final dense layer with softmax activation for classification.

Define the CNN model

```
: model = Sequential()
  model.add(Input(shape=(28, 28, 1)))
  model.add(Conv2D(10, kernel_size=5, activation='relu')) # 10-channel, 5x5 convolution followed by ReLU
  model.add(MaxPool2D(2)) # 2x2 maxpool -> 14x14x1
  model.add(Conv2D(20, kernel_size=5, activation='relu'))
  model.add(MaxPool2D(2))
  model.add(Flatten())
  model.add(Dense(20, activation = "relu"))
  model.add(Dropout(0.5))
  model.add(Dense(10, activation='softmax'))
```

Purpose:

- **Sequential Model Definition:** A linear stack of layers.
- **Input layer:** Accepts 28x28x1 images
- **Convolutional Layers:** Capture hierarchical features in images.
- **MaxPooling Layers:** Downsample feature maps to reduce computational load.
- **Flatten Layer:** Converts the output to a one-dimensional array.
- **Dense Layers:** Fully connected layers for classification.
- **Dropout Layer:** Prevents overfitting by randomly dropping connections during training.
- **Output layer:** 10 neurons (one for each digit) with softmax activation for probabilities.
- **Model Summary Printing:** Displays a summary of the model's architecture.

3.2. Model Compilation – CNN

The model is compiled using the Adam optimizer and sparse categorical crossentropy loss.

```
In [8]: opt = Adam(learning_rate=0.001)

In [9]: model.compile(
            optimizer=opt,
            loss=sparse_categorical_crossentropy,
            metrics=[sparse_categorical_accuracy],
        )
```

Purpose:

- **Adam Optimizer:** Efficient optimization algorithm.
- **Sparse Categorical Crossentropy:** Appropriate for multi-class classification tasks.
- **Model Compilation:** Configures the model for training.

3.3. Model Training – CNN

The model is trained using augmented training data and evaluated on the test set for a specified number of epochs.

In [10]:
```
bs = 128
n_epoch = 10

cnn_history=model.fit(
    X_train_augmented,
    Y_train_augmented,
    batch_size=bs,
    epochs=n_epoch,
    validation_data=(X_test, Y_test),
)
```

```
Epoch 1/10
938/938 [==============================] - 12s 12ms/step - loss: 1.1649 - sparse_categorical_accuracy: 0.5726 - val_loss: 0.183
7 - val_sparse_categorical_accuracy: 0.9709
Epoch 2/10
938/938 [==============================] - 11s 12ms/step - loss: 0.8358 - sparse_categorical_accuracy: 0.6871 - val_loss: 0.164
4 - val_sparse_categorical_accuracy: 0.9745
Epoch 3/10
938/938 [==============================] - 12s 13ms/step - loss: 0.7536 - sparse_categorical_accuracy: 0.7182 - val_loss: 0.141
5 - val_sparse_categorical_accuracy: 0.9758
Epoch 4/10
938/938 [==============================] - 12s 13ms/step - loss: 0.6723 - sparse_categorical_accuracy: 0.7505 - val_loss: 0.104
2 - val_sparse_categorical_accuracy: 0.9798
Epoch 5/10
938/938 [==============================] - 11s 12ms/step - loss: 0.6129 - sparse_categorical_accuracy: 0.7723 - val_loss: 0.094
2 - val_sparse_categorical_accuracy: 0.9813
Epoch 6/10
938/938 [==============================] - 11s 12ms/step - loss: 0.5405 - sparse_categorical_accuracy: 0.8028 - val_loss: 0.083
9 - val_sparse_categorical_accuracy: 0.9821
Epoch 7/10
938/938 [==============================] - 11s 12ms/step - loss: 0.5025 - sparse_categorical_accuracy: 0.8177 - val_loss: 0.081
7 - val_sparse_categorical_accuracy: 0.9844
Epoch 8/10
938/938 [==============================] - 11s 12ms/step - loss: 0.4803 - sparse_categorical_accuracy: 0.8246 - val_loss: 0.087
9 - val_sparse_categorical_accuracy: 0.9849
Epoch 9/10
938/938 [==============================] - 11s 12ms/step - loss: 0.4684 - sparse_categorical_accuracy: 0.8295 - val_loss: 0.082
5 - val_sparse_categorical_accuracy: 0.9832
Epoch 10/10
938/938 [==============================] - 11s 12ms/step - loss: 0.4527 - sparse_categorical_accuracy: 0.8362 - val_loss: 0.070
1 - val_sparse_categorical_accuracy: 0.9847
```

Purpose:

Model Training: The model is trained on the augmented data, and the training history is stored.

4. Artificial Neural Network (ANN) Model

4.1. Model Architecture

The ANN model is a simpler architecture with three dense layers.

```
# Define the ANN model
ann_model = Sequential()
ann_model.add(Flatten(input_shape=(28,28)))
ann_model.add(Dense(128, activation='relu'))
ann_model.add(Dense(128, activation='relu'))
ann_model.add(Dense(10, activation='softmax'))
ann_model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

4.2. Model Training – ANN

The ANN model is trained and evaluated using the same training and test sets.

```
# Train the ANN model
ann_history = ann_model.fit(
    X_train_augmented,
    Y_train_augmented,
    batch_size=bs,
    epochs=n_epoch,
    validation_data=(X_test, Y_test),
)
```

```
Epoch 1/10
938/938 [==============================] - 2s 2ms/step - loss: 0.6159 - accuracy: 0.7975 - val_loss: 0.1045 - val_accuracy: 0.9
743
Epoch 2/10
938/938 [==============================] - 2s 2ms/step - loss: 0.2749 - accuracy: 0.9071 - val_loss: 0.0838 - val_accuracy: 0.9
806
Epoch 3/10
938/938 [==============================] - 2s 2ms/step - loss: 0.2125 - accuracy: 0.9253 - val_loss: 0.0800 - val_accuracy: 0.9
798
Epoch 4/10
938/938 [==============================] - 2s 2ms/step - loss: 0.1776 - accuracy: 0.9359 - val_loss: 0.0667 - val_accuracy: 0.9
824
Epoch 5/10
938/938 [==============================] - 2s 2ms/step - loss: 0.1543 - accuracy: 0.9434 - val_loss: 0.0677 - val_accuracy: 0.9
836
Epoch 6/10
938/938 [==============================] - 2s 2ms/step - loss: 0.1381 - accuracy: 0.9488 - val_loss: 0.0705 - val_accuracy: 0.9
810
Epoch 7/10
938/938 [==============================] - 2s 2ms/step - loss: 0.1240 - accuracy: 0.9533 - val_loss: 0.0706 - val_accuracy: 0.9
820
Epoch 8/10
938/938 [==============================] - 2s 2ms/step - loss: 0.1129 - accuracy: 0.9572 - val_loss: 0.0752 - val_accuracy: 0.9
808
Epoch 9/10
938/938 [==============================] - 2s 2ms/step - loss: 0.1022 - accuracy: 0.9606 - val_loss: 0.0829 - val_accuracy: 0.9
800
Epoch 10/10
938/938 [==============================] - 2s 2ms/step - loss: 0.0971 - accuracy: 0.9627 - val_loss: 0.0736 - val_accuracy: 0.9
824
```

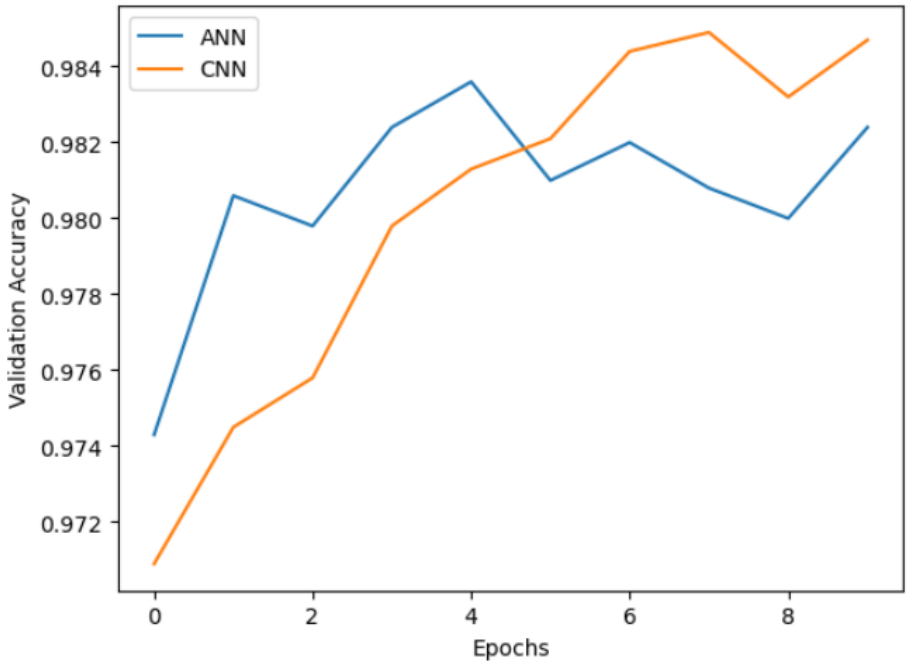
Purpose:

- **Flatten Layer:** Prepares image data for fully connected layers.
- **Dense Layers:** Classic neural network architecture for image classification.
- **Model Training:** Similar to the CNN model, but with a different architecture.

5. Model Comparison

The validation accuracy of both the CNN and ANN models is plotted over epochs for comparison. The CNN model demonstrates superior performance, as shown in the graphical representation.

In [12]:
```
# Compare the performances of the models
plt.plot(ann_history.history['val_accuracy'], label='ANN')
plt.plot(cnn_history.history['val_sparse_categorical_accuracy'], label='CNN')
plt.xlabel('Epochs')
plt.ylabel('Validation Accuracy')
plt.legend()
plt.show()
```



In [13]:

```
# Assuming ann_history and cnn_history are instances of the History class

# Extract accuracy values from ANN history
ann_val_accuracy = ann_history.history['val_accuracy']

# Extract accuracy values from CNN history
cnn_val_accuracy = cnn_history.history['val_sparse_categorical_accuracy']

# Create a DataFrame for visualization
epochs = range(1, n_epoch + 1)
df = pd.DataFrame({
    'Epochs': np.repeat(epochs, 2),
    'Model': np.tile(['ANN', 'CNN'], n_epoch),
    'Validation Accuracy': np.concatenate([ann_val_accuracy, cnn_val_accuracy])
})

# Plot the comparison using Plotly Express
fig = px.line(df, x='Epochs', y='Validation Accuracy', color='Model',
              labels={'Validation Accuracy': 'Validation Accuracy'},
              title='Comparison of Model Performances',
              width=800, height=500)
fig.show()
```

Purpose:

- **Line Plot:** Visualizes the validation accuracy over epochs for both the ANN and CNN models.
- **Model Comparison:** Assists in identifying the model with superior performance.

6. Model Saving

Purpose:

- **Model Saving:** Persists the trained CNN model for later use.
- **Model Evaluation:** Computes and prints the test loss and accuracy.

```
In [14]: model.save('mnist.h5')
         print("Saving the model as mnist.h5")
```

```
Saving the model as mnist.h5
```

```
In [15]: score = model.evaluate(X_test, Y_test, verbose=0)
         print('Test loss:', score[0])
         print('Test accuracy:', score[1])
```

```
Test loss: 0.09280645102262497
Test accuracy: 0.9813981652259827
```

7. Random Forest Model

A Random Forest classifier is trained on the flattened image data for comparison. The model is evaluated, and its accuracy is calculated. Additionally, the feature importances are extracted and visualized.

```
In [20]: from sklearn.ensemble import RandomForestClassifier

         # Create a Random Forest model
         rf_model = RandomForestClassifier(n_estimators=100, random_state=42)

         # Train the Random Forest model
         rf_model.fit(X_train_augmented.reshape(-1, 28*28), Y_train_augmented.ravel())

         # Evaluate the Random Forest model
         rf_score = rf_model.score(X_test.reshape(-1, 28*28), Y_test.ravel())
         print('Random Forest Test Accuracy:', rf_score)

         # Predict the output for the selected example
         y_predicted_rf = rf_model.predict(X_test.reshape(-1, 28*28))
```

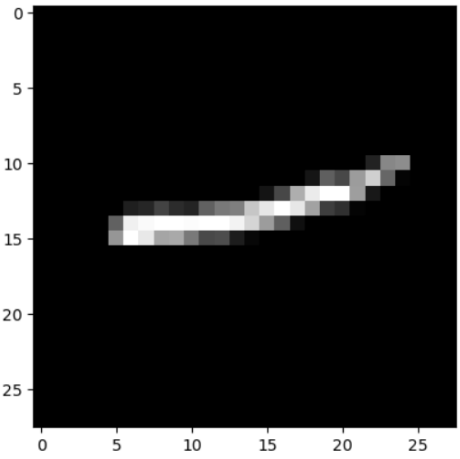
```
Random Forest Test Accuracy: 0.9850985098509851
```

Purpose:

- **Random Forest Classifier:** Implements a decision tree-based ensemble model.
- **Model Training:** Trains the Random Forest model on the flattened augmented training data.
- **Model Evaluation:** Computes and stores the accuracy of the Random Forest model on the test set.

And here is a predicted output for the RFM model.

```
In [21]: plt.imshow(X_test[20],cmap='gray')
         plt.show()

         print(y_predicted_rf[20])
```



```
1
```

7.1. Random Forest Feature Importance

```
In [22]: feature_importances = rf_model.feature_importances_
         print("Feature Importances:")
         for i, importance in enumerate(feature_importances):
             print(f"Feature {i + 1}: {importance}")
```

```
Feature Importances:
Feature 1: 1.672296977986094e-05
Feature 2: 2.2416619687769788e-05
Feature 3: 4.238726049728817e-05
Feature 4: 7.801013083926651e-05
Feature 5: 8.978505837513176e-05
Feature 6: 0.00010361156213682368
Feature 7: 0.00012394105367424953
```

Purpose:

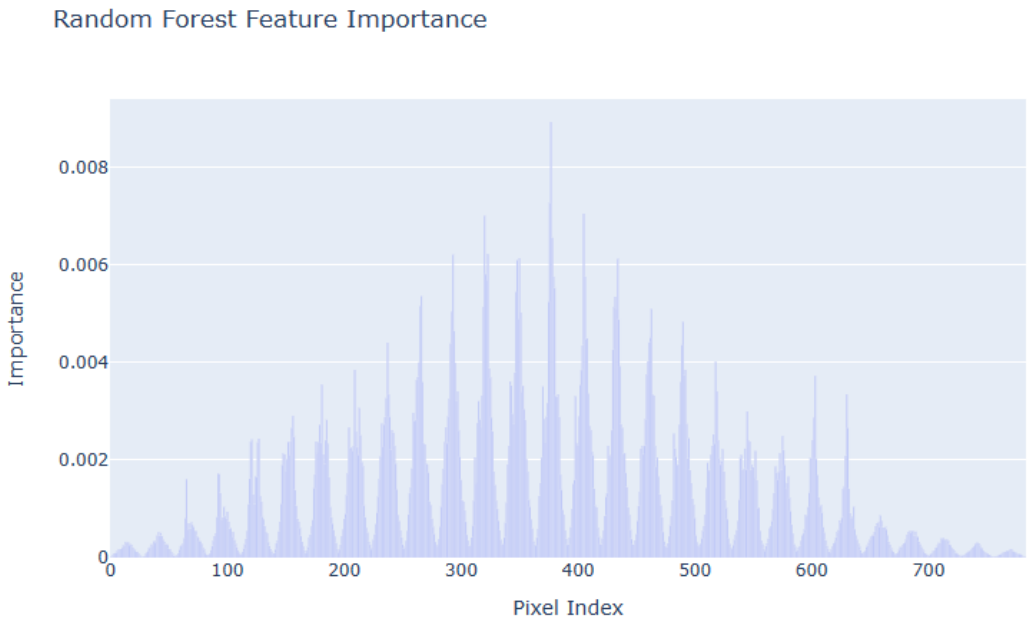
- **Feature Importance Analysis:** Examines the importance of each pixel in the classification decision of the Random Forest model.
- **Insights into Model Decisions:** Identifies which pixels contribute more significantly to the classification.

7.2. Additional Analysis

Feature importances of the Random Forest model are computed and plotted to understand which pixels contribute most to the model's decisions.

And here is a plot explaining the Random Forest Feature Importance:

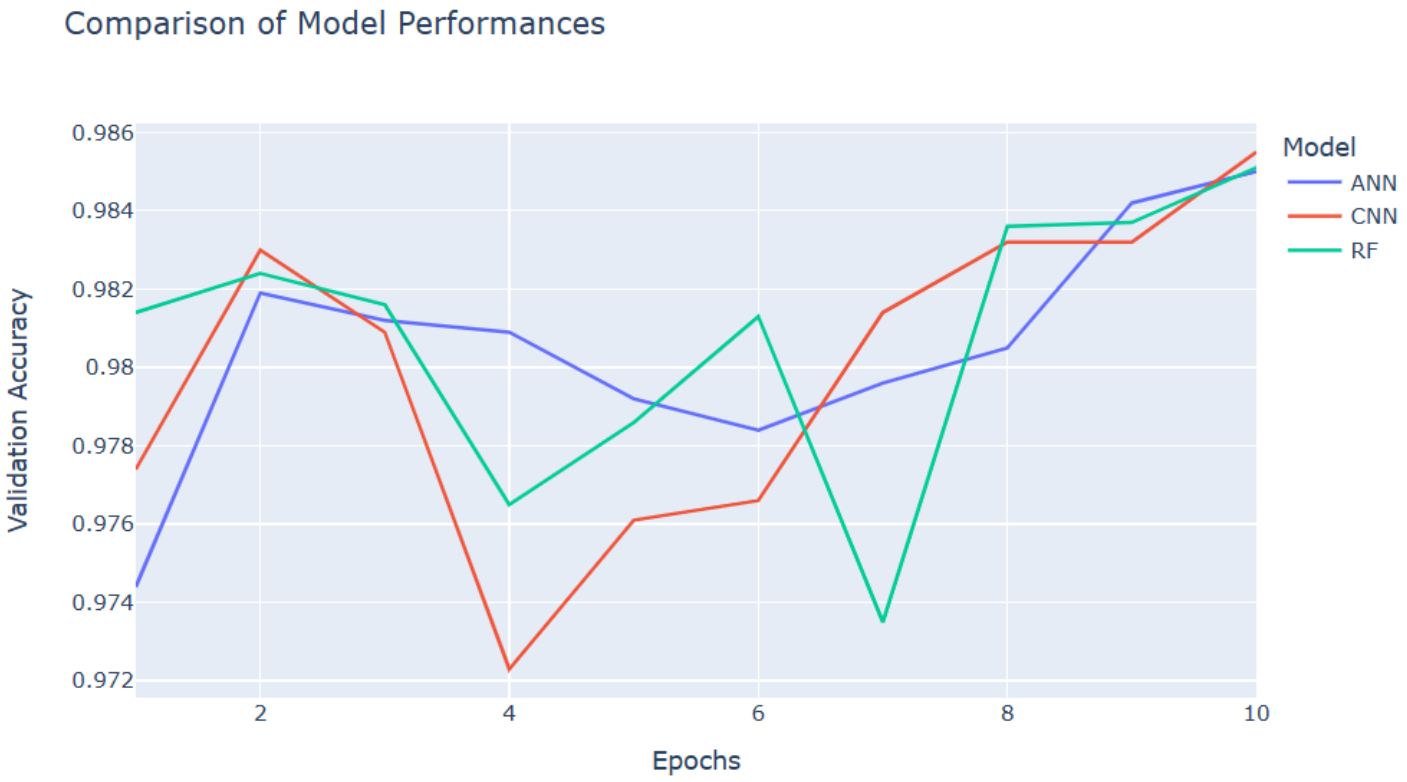
In [23]:
```
import plotly.express as px
# Get feature importances
feature_importances = rf_model.feature_importances_

# Create a DataFrame for visualization
data = {'Feature Index': np.arange(len(feature_importances)), 'Feature Importance': feature_importances}
df = pd.DataFrame(data)

# Sort DataFrame by importance in descending order
df = df.sort_values(by='Feature Importance', ascending=False)

# Create a bar plot using Plotly Express
fig = px.bar(df, x='Feature Index', y='Feature Importance', labels={'Feature Index': 'Pixel Index', 'Feature Importance': 'Import
             title='Random Forest Feature Importance', width=800, height=500)

# Show the plot
fig.show()
```



8. Ensemble Model Comparison

The validation accuracy of all three models (CNN, ANN, Random Forest) is compared over epochs using a line plot. The CNN model consistently outperforms the other models.

```
# Plot the comparison using Plotly Express
fig = px.line(df, x='Epochs', y='Validation Accuracy', color='Model',
              labels={'Validation Accuracy': 'Validation Accuracy'},
              title='Comparison of Model Performances',
              width=800, height=500)
fig.show()
```



Purpose:

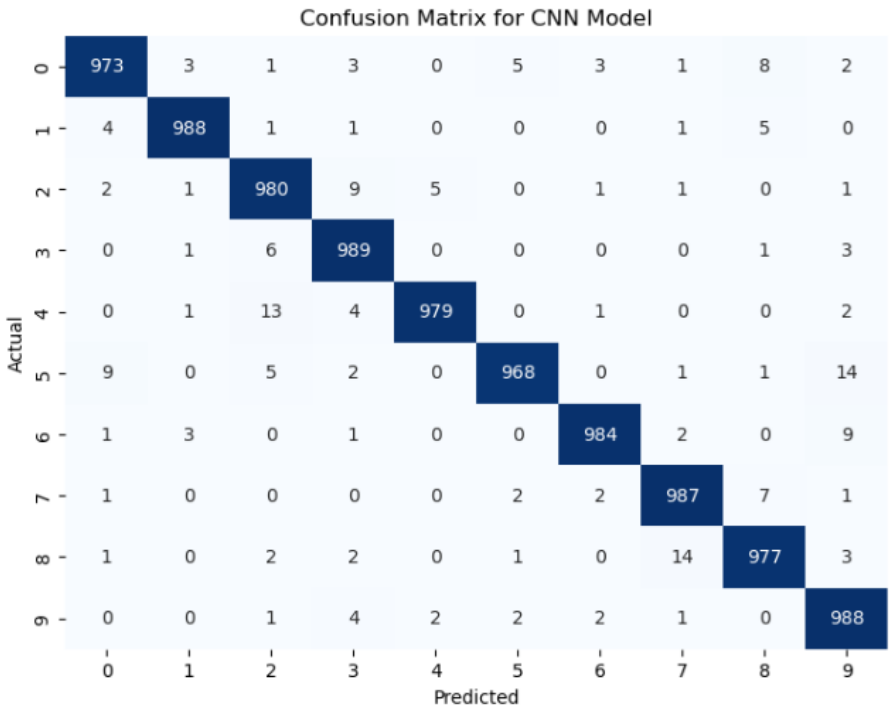
Model Performance Evaluation: Provides a detailed understanding of each model's ability to classify each digit.

9. Model Evaluation & Visiualization - Confusion Matrices

Confusion matrices are generated for each model, visualizing the performance in terms of true positive, true negative, false positive, and false negative predictions. The matrices provide insights into the models' ability to classify each digit.

9.1. CNN Model

```
In [26]: from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Obtain predictions on the test set

# Create confusion matrix
conf_matrix = confusion_matrix(Y_test, y_predicted_CNN)

# Plot confusion matrix using seaborn
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=np.unique(Y_test), yticklabels=np.unique(Y_test))
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix for CNN Model')
plt.show()
```

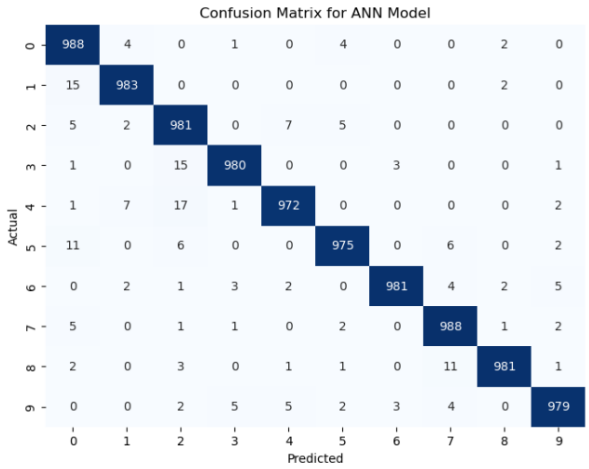


Purpose:

- **Confusion Matrix:** Visualizes the performance of the CNN model by comparing predicted and actual class labels.
- **Heatmap Representation:** Offers insights into which classes are frequently misclassified.

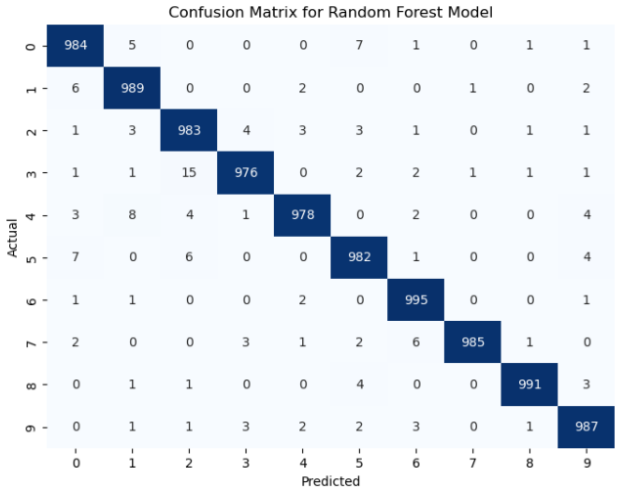
9.2. ANN Model – Confusion Matrix

In [27]:
```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Obtain predictions on the test set

# Create confusion matrix
conf_matrix = confusion_matrix(Y_test, y_predicted_ANN)

# Plot confusion matrix using seaborn
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=np.unique(Y_test), yticklabels=np.unique(Y_test))
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix for ANN Model')
plt.show()
```



9.3. Random Forest Model – Confusion Matrix

In [28]:
```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Obtain predictions on the test set

# Create confusion matrix
conf_matrix = confusion_matrix(Y_test, y_predicted_rf)

# Plot confusion matrix using seaborn
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=np.unique(Y_test), yticklabels=np.unique(Y_test))
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix for Random Forest Model')
plt.show()
```



Purpose:

Confusion Matrices for ANN and RF: Generates confusion matrices for the Artificial Neural Network and Random Forest models.

10. Simple GUI Application

A simple GUI application created using the Tkinter library for drawing digits (0-9) on a canvas. The goal is to allow users to draw a digit, submit the drawing, and have three different machine learning models (CNN, ANN, and RF) predict the drawn digit. The application also provides a "Clear" button to reset the canvas.

10.1. Tkinter GUI Initialization:

A Tkinter window (‘root’) is created, and an instance of the ‘DrawingApp’ class is initialized.

```
# Create the main application window
root = tk.Tk()
app = DrawingApp(root)

# Start the Tkinter event loop
root.mainloop()
```

10.2. DrawingApp Class:

The “DrawingApp” class manages the GUI components and drawing functionality.

```
import tkinter as tk
import numpy as np

class DrawingApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Draw 28x28 Image")

        self.canvas = tk.Canvas(root, width=280, height=280, bg="white")
        self.canvas.pack()

        self.canvas.bind("<B1-Motion>", self.paint)

        self.label = tk.Label(root, text="Draw a digit (0-9):")
        self.label.pack()

        self.clear_button = tk.Button(root, text="Clear", command=self.clear_canvas)
        self.clear_button.pack()

        self.submit_button = tk.Button(root, text="Submit", command=self.submit_canvas)
        self.submit_button.pack()
        self.prediction_label_CNN = tk.Label(root, text="Predicted Digit CNN: ")
        self.prediction_label_CNN.pack()
        self.prediction_label_ANN = tk.Label(root, text="Predicted Digit ANN: ")
        self.prediction_label_ANN.pack()
        self.prediction_label_RF = tk.Label(root, text="Predicted Digit RF: ")
        self.prediction_label_RF.pack()

        self.image_array = np.zeros((28, 28), dtype=int)
```

-Canvas Initialization:

A canvas widget is created for drawing with a white background.

-Event Binding:

The canvas is set up to detect left mouse button motion (<B1-Motion>), and the paint method is called when the mouse is moved.

```
def paint(self, event):
    x1, y1 = (event.x - 5), (event.y - 5)
    x2, y2 = (event.x + 5), (event.y + 5)
    intensity = 1  # Use 1 for black in the binary image
    color = "#{:02x}{:02x}{:02x}".format(intensity * 255, intensity * 255, intensity * 255)
    self.canvas.create_oval(x1, y1, x2, y2, fill=color, width=10)

    # Convert pixel coordinates to a 28x28 grid
    grid_x = int(event.x / 10)
    grid_y = int(event.y / 10)

    # Set corresponding pixel in the image array to the intensity value
    self.image_array[grid_x, grid_y] = intensity

def clear_canvas(self):
    self.canvas.delete("all")
    self.image_array = np.zeros((28, 28), dtype=int)

def submit_canvas(self):
    print("Submitted Image Array:")
    # No need for normalization to values between 0 and 1

    # Reshape the image array for prediction
    flattened_image = self.image_array.flatten()
    flattened_image = flattened_image.reshape(1, -1)
```

-Buttons and Labels:

"Clear" and "Submit" buttons are added to clear the canvas and submit the drawing, respectively.

Labels are added to display the predicted digits for CNN, ANN, and RF.

-Image Array:

A 28x28 NumPy array (image_array) is used to store the pixel values of the drawn digit.

Methods:

-**paint:** Captures mouse movements to draw on the canvas and updates the image array accordingly.

-clear_canvas: Clears the canvas and resets the image array.

-submit_canvas:

Submits the drawn image for prediction using three different models (CNN, ANN, and RF) and updates the labels.

10.3. Drawing Functionality (‘paint’ Method):

When the mouse is moved, the ‘paint’ method is triggered.

It draws an oval on the canvas based on the mouse coordinates and updates the corresponding pixel in the image array.

10.4. Model Prediction (‘submit_canvas’ Method):

- When the user clicks the "Submit" button, the “submit_canvas” method is called.
- The drawn image array is flattened and reshaped for prediction.
- Predictions are made using three different models: CNN, ANN, and RF.
- The predicted digits and models' labels are updated in the GUI.

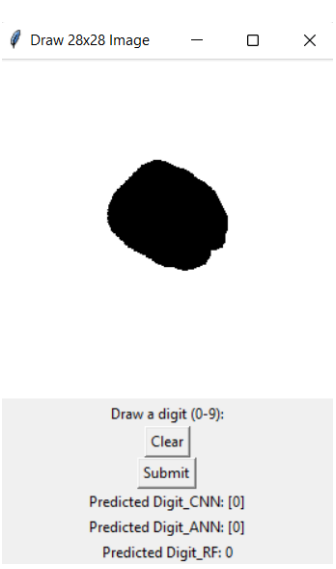
```
# Make predictions using the model
# (Assuming rf_model is an instance of RandomForestClassifier)
predicted_CNN=model.predict(flattened_image.reshape(-1,28,28,1))
predicted_digit_CNN = predicted_CNN[0]
predicted_digit_CNN=np.argmax(predicted_CNN,axis=1)
predicted_ANN=ann_model.predict(flattened_image.reshape(-1,28,28,1))
predicted_digit_ANN = predicted_ANN[0]
predicted_digit_ANN=np.argmax(predicted_ANN,axis=1)
predicted_RF = rf_model.predict(flattened_image)
predicted_digit_RF = predicted_RF[0]
```

10.5. Label Updates:

The predicted digits for CNN, ANN, and RF are displayed on the GUI labels.

```
# Update the label with the predicted digit
self.prediction_label_CNN.config(text="Predicted Digit_CNN: {}".format(predicted_digit_CNN))
self.prediction_label_ANN.config(text="Predicted Digit_ANN: {}".format(predicted_digit_ANN))
self.prediction_label_RF.config(text="Predicted Digit_RF: {}".format(predicted_digit_RF))
```

10.6. Output Examples of Users Drawing and the models predictions:



11. Conclusion

The CNN model achieves the highest accuracy among the three models, demonstrating its effectiveness in handwritten Arabic digit recognition. The Random Forest model, while performing reasonably well, falls short compared to the deep learning models. The ensemble comparison and confusion matrices further support the superior performance of the CNN model.

This comprehensive explanation covers data loading, preprocessing, model architecture, training, evaluation, and visualization. The project employs deep learning (CNN, ANN) and traditional machine learning (Random Forest) approaches, providing a detailed comparison of their performances in recognizing Arabic handwritten digits. The confusion matrices offer a granular view of the models' classification abilities. The code is well-organized, making it easy to follow and understand.