Project: Alphabet Classification

Project Description

This project focuses on developing an AI model capable of classifying images of handwritten or printed alphabet characters. Initially, the system will support both English and Arabic alphabets, distinguishing between various letter forms. The goal is to create a robust system that can accurately identify individual characters from diverse visual inputs.

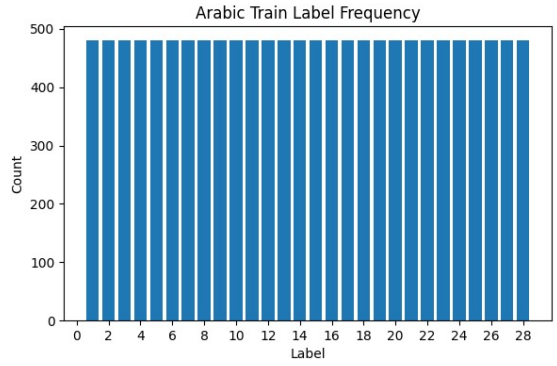
Datasets Used

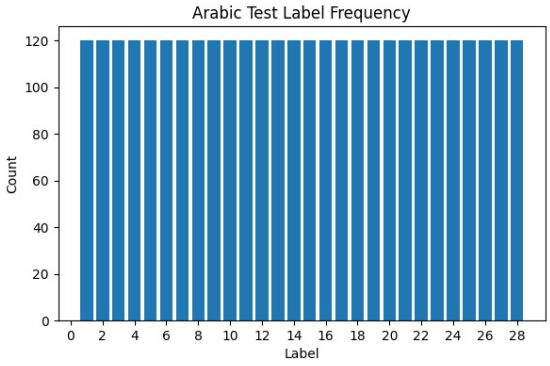
The project will leverage the following datasets to train and validate the classification models:

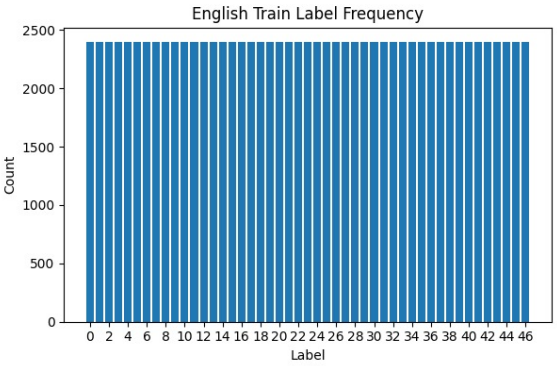
- **EMNIST (Extended MNIST)**: A large dataset of handwritten digits and letters, providing a solid foundation for character recognition.
- **Arabic Handwritten Characters Dataset**: A dataset specifically curated for recognizing various forms of Arabic handwritten characters.

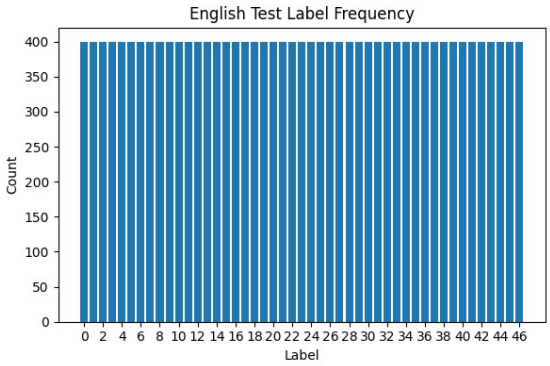
Dataset Visualizations

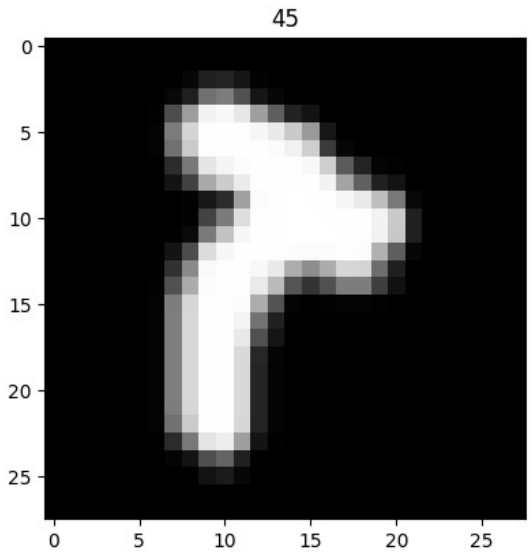
Visual representations of the dataset distributions and sample images:

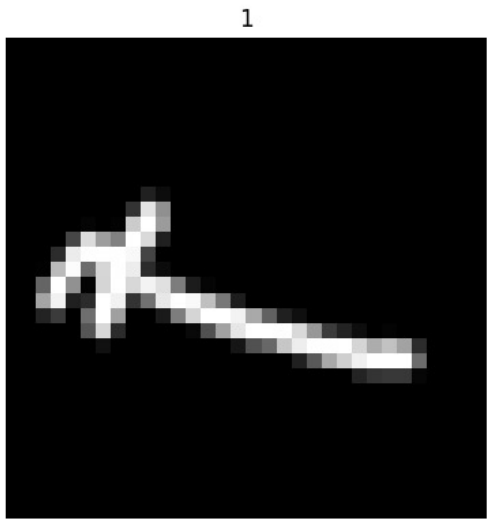












Dataset Details Summary

The datasets provide distinct characteristics for training and evaluation:

- **EMNIST**: Offers 6 diverse splits in Binary and CSV formats, totaling over 814,000 images. CSV format contains 785 columns (1 class label + 784

pixel values). Features include 'ByClass' (62 unbalanced classes) and 'ByMerge' (47 unbalanced classes), with more digits than letters, reflecting English language frequency.

- **Arabic Handwritten Characters Dataset**: Developed by El-Sawy, Loey, and El-Bakry, this dataset contains 16,800 handwritten Arabic characters from 60 participants. It is split into training (13,440) and testing (3,360) sets, achieving a promising 94.9% classification accuracy. It addresses challenges like handwriting variations and character shapes.

Preprocessing Pipeline

This section describes the complete preprocessing workflow applied to the Arabic Handwritten Characters Dataset (AHCD) and the EMNIST Letters dataset prior to model training.

1. Data Loading

Arabic Dataset (AHCD)

- Images and labels were loaded from CSV files using pandas.read_csv.
- The training set contains 13,440 samples, and the test set contains 3,360 samples.
- Each image is provided as a flattened vector of length 1024, corresponding to a spatial resolution of 32 × 32 pixels.
- Class labels range from 1 to 28, representing Arabic characters.

English Dataset (EMNIST Letters)

- The training and test sets were loaded from IDX binary files using a custom read_idx function.
- EMNIST letter images have a spatial resolution of 28 × 28 pixels.
- Class labels range from 1 to 26, corresponding to the 26 English letters.

2. Class Distribution Analysis

The distribution of classes in both datasets was visualized using Seaborn count plots.
This analysis highlighted significant class imbalance, particularly in the EMNIST dataset, necessitating balancing procedures.

3. Dataset Balancing

A custom balance_dataset function was applied to equalize the number of samples per class.

- For the EMNIST training set, each class was balanced to 480 samples.
- For the EMNIST test set, each class was balanced to 120 samples.
- Oversampling (with replacement) was used for classes with insufficient samples.
- Undersampling was used for classes with surplus samples.

This process ensured a uniform class distribution across the English dataset, improving model stability and reducing class bias.

4. Image Preprocessing

All images underwent a standardized preprocessing pipeline implemented in the preprocess_images function.

4.1. Reshaping

Arabic images were reshaped from 1024-dimensional vectors into 32 × 32 × 1 tensors.
English images were reshaped from 784-dimensional vectors into 28 × 28 × 1 tensors.

4.2. Resizing

All images were resized to 96 × 96 pixels to meet the input requirements of deep convolutional architectures (e.g., ResNet).

4.3. Color Space Conversion

Grayscale images were converted to RGB by replicating the single channel to produce 96 × 96 × 3 images.

4.4. Orientation Correction

EMNIST images required corrective transformations to align them properly:

- Rotation by 90° clockwise.
- Horizontal flipping.

This ensured consistent alignment across both Arabic and English characters.

5. Label Adjustment

Arabic labels (1–28) were kept unchanged.
English labels (1–26) were remapped to 29–54 by adding a constant offset of 28.
This produced a unified label space of 54 total classes, combining Arabic and English characters without collision.

6. Dataset Integration

The preprocessed Arabic and English datasets were concatenated to form a combined training and test set.

7. Data Augmentation

To increase dataset diversity and reduce overfitting, extensive augmentation was performed using ImageDataGenerator with the following transformations:

- Rotation (up to 25°)
- Width shifting (up to 15%)
- Height shifting (up to 20%)
- Shearing (0.2)
- Zooming (up to 30%)
- Brightness variation (0.8–1.2)
- Nearest-neighbor filling

An augmentation factor of 1× was applied, generating a synthetic dataset equal in size to the original training set. The augmented images and labels were then concatenated back into the main training dataset.

8. Final Dataset Summary

Following all preprocessing steps:

- All images were standardized to 96 × 96 × 3.
- Labels were mapped to a consistent 1–54 class range.
- The training dataset included both original and augmented samples.
- The class distribution was balanced across all English letters, with original distributions preserved for Arabic classes.

Models Used

The following four models were evaluated for this project:

- **Inception v1 (GoogLeNet):** Utilized with pre-trained weights.
- **VGG 19:** Trained from scratch.
- **ResNet:** Utilized with pre-trained weights.
- **MobileNet:** Utilized with pre-trained weights.

Detailed Model Explanation: Inception v1 Module

Inception v1 Model Configuration

For this project, the Inception v1 (GoogLeNet) model was configured as follows:

- **Number of Classes:** The final classification layer was adapted for 54 classes.
- **Device:** The model was set up to utilize CUDA if available, otherwise falling back to CPU.
- **Base Model:** GoogLeNet (Inception v1) was loaded with pre-trained weights from ImageNet (IMAGENET1K_V1), including its auxiliary classifiers (`aux_logits=True`).
- **Transfer Learning Strategy:** All parameters of the pre-trained model were frozen to retain learned features from ImageNet.
- **Fine-tuning:** The final classification layers (`fc`, `aux1.fc2`, `aux2.fc2` if they exist) were replaced with new linear layers outputting predictions for the 54 classes, allowing the model to adapt to the specific alphabet classification task.

Understanding the Inception Module

The Inception module is a core building block in convolutional neural networks, notably used in GoogleNet (Inception v1). Its design aims to efficiently capture features at multiple scales within a single module, reducing computational complexity compared to simply stacking layers of different filter sizes. The image illustrates two key variations of this module.

(a) Naïve Inception Module

This is a straightforward implementation of the Inception concept. It takes input from a 'Previous layer' and processes it through several parallel convolutional and pooling operations:

- **1x1 Convolutions:** A path employing 1x1 convolutions.

- **3x3 Convolutions:** A path using 3x3 convolutions.
- **5x5 Convolutions:** A path using 5x5 convolutions.
- **3x3 Max Pooling:** A path using 3x3 max pooling.

The outputs from all these parallel paths are then concatenated together via 'Filter concatenation' to form the module's output. While effective at capturing multi-scale features, the larger convolution kernels (like 5x5) can be computationally expensive without preceding dimensionality reduction.

(b) Inception Module with Dimension Reductions

This improved version addresses the computational cost of the naïve module by introducing 1x1 convolutions strategically before the larger filter operations. This significantly reduces the number of input channels, making subsequent convolutions more efficient:

- **1x1 Convolutions:** A direct path using 1x1 convolutions.
- **1x1 followed by 3x3 Convolutions:** A path first reduces dimensionality with 1x1 convolutions, then applies 3x3 convolutions.
- **1x1 followed by 5x5 Convolutions:** Similarly, this path reduces dimensionality with 1x1 convolutions before applying 5x5 convolutions.
- **3x3 Max Pooling:** A path utilizing 3x3 max pooling.

The outputs from these dimension-reduced paths are then concatenated via 'Filter concatenation'. This design allows the network to capture features at various spatial scales with greater computational efficiency.

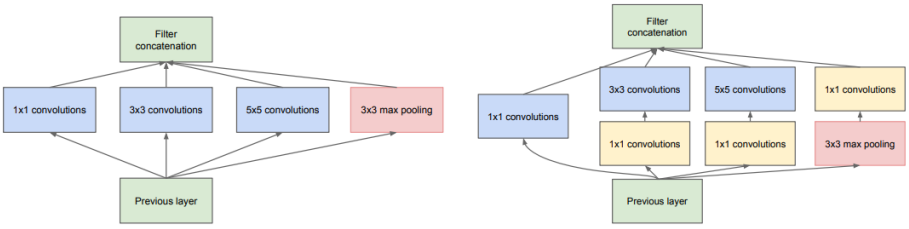
Key Takeaway

The primary innovation in the 'with dimension reductions' module is the use of 1x1 convolutions as a bottleneck, significantly reducing computational load while preserving the ability to process information at different receptive field sizes.

References:

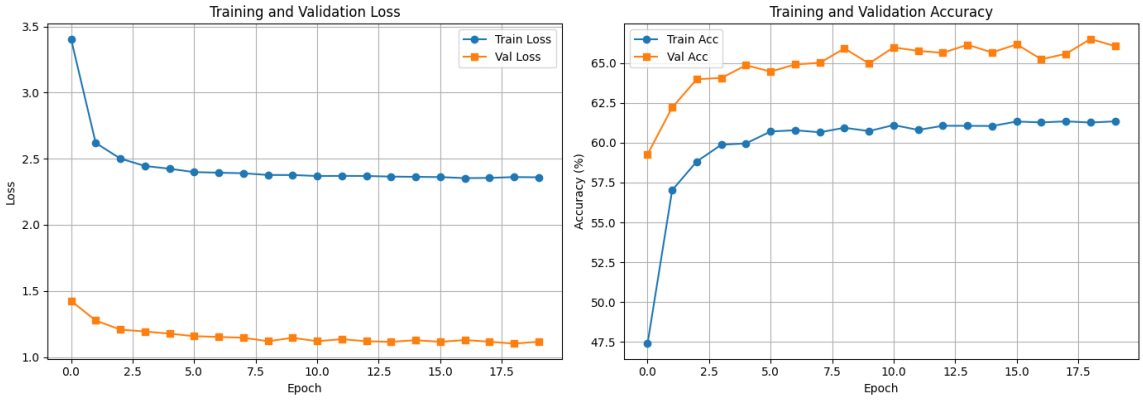
-The original paper introducing this architecture is available at https://arxiv.org/pdf/1409.4842.

Diagrams and graphs illustration



(a) Inception module, naïve version (b) Inception module with dimension reductions

(1) Model Architecture



(2) Loss and accuracy

VGG-19 Model Architecture

The provided code defines and train VGG-19 convolutional neural network (CNN) architecture for the classification of Arabic handwritten characters. VGG-19 is a deep CNN renowned for its simplicity and uniformity, primarily utilizing small 3x3 convolutional filters stacked sequentially.

Key Steps in the Process:

Data Loading and Preprocessing: CSV files containing image data and labels were loaded. The image data was reshaped into the expected format for CNNs (96x96 pixels with 1 channel) and normalized to pixel values between 0 and 1. Labels were converted to a 0-indexed format for model compatibility.

Data Augmentation: An ImageDataGenerator was configured to apply various transformations (rotation, width/height shifting, zoom, shear) to the training dataset. This technique helps the model learn more invariant features and improve its generalization capabilities, reducing the risk of overfitting.

Model Definition: A VGG-19 inspired model was constructed using TensorFlow's Keras Sequential API. The architecture consists of 5 blocks, each block consists of number of stacked convolutional layers followed by 2x2 Max pooling layers for spatial resolution decreasing progressively, batch normalization for stabilizing training, dropout layers to prevent overfitting, and finally, fully connected (dense) layers for classification.

Model Compilation: The model was compiled with the Adam optimizer, set to a learning rate of 0.001. The loss function used was sparse_categorical_crossentropy , appropriate for integer-labeled multi-class classification. 'Accuracy' was chosen as the primary metric to monitor during training.

Callback Configuration: Several callbacks were defined to enhance the training process:

ReduceLROnPlateau : Monitors validation loss and reduces the learning rate by a factor of 0.5 if the loss does not improve for 3 epochs, aiming to help the model converge.

EarlyStopping : Monitors validation loss and stops training if it fails to improve for 10 epochs, restoring the weights from the epoch with the best validation loss.

ModelCheckpoint : Saves the model's weights whenever the validation accuracy improves, ensuring the best performing model state is preserved.

Model Training: The model was trained for 30 epochs with a batch size of 64, utilizing the augmented training data and validating against the test set.

Evaluation and Visualization: After training, the model's performance was evaluated on the test set, yielding a test accuracy of 97.83%. The classification report and confusion matrix were generated to provide a detailed analysis of the model's performance per character class. Training progress was visualized through plots of accuracy and loss over epochs.

VGG-19 Architecture Breakdown:

The VGG-19 architecture is characterized by its deep stack of **convolutional layers, typically using small 3x3 filters, relu activation and Same padding**.

Input Layer: Accepts input images with shape (96, 96, 1) .

Convolutional Blocks: The model comprises five main convolutional blocks, progressively increasing the number of filters and decreasing the spatial dimensions through max-pooling. Each block typically includes Conv2D layers, BatchNormalization , and 2x2 MaxPooling2D , followed by Dropout for regularization.

-first block: two 3x3 Convolution layers layers, one BatchNormalization, and one 2x2 MaxPooling, one Dropout, that reduces to feature map to 48x48x64

-second block: two 3x3 Convolution layers layers, one BatchNormalization, and one 2x2 MaxPooling, one Dropout, that reduces to feature map to 24x24x128

-third block: four 3x3 Convolution layers layers, one BatchNormalization, and one 2x2 MaxPooling, one Dropout, that reduces to feature map to 12x12x256

-fourth block: four 3x3 Convolution layers layers, one BatchNormalization, and one 2x2 MaxPooling, one Dropout, that reduces to feature map to 6x6x512

-fifth block: four 3x3 Convolution layers layers, one BatchNormalization, and one 2x2 MaxPooling, one Dropout, that reduces to feature map to 3x3x512

Flatten Layer: This layer converts the multi-dimensional feature maps from the convolutional layers into a 1D vector suitable for input into the dense layers.

Dense (Fully Connected) Layers: Two large dense layers with 4096 units each, employing ReLU activation, are used to capture high-level features. These are followed by BatchNormalization and Dropout(0.5) to mitigate overfitting.

Output Layer: A final Dense layer with 28 units (corresponding to the 28 Arabic characters) and a softmax activation function predicts the probability distribution across all classes.

Key Performance Metrics:

The trained model demonstrated strong performance on the test set:

Test Accuracy: 97.83%

Weighted Precision: 97.87%

Weighted Recall: 97.83%

Weighted F1-Score: 97.83%

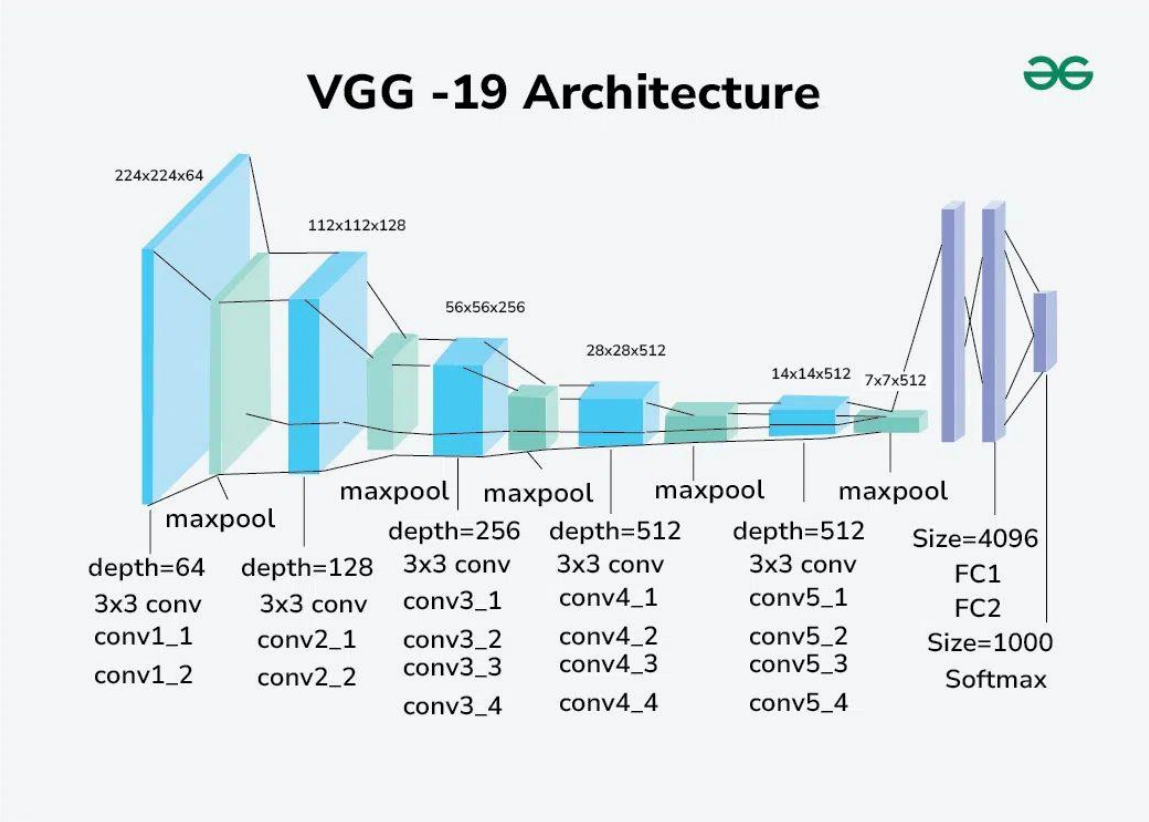
The plots and confusion matrix further illustrate the model's effectiveness in correctly classifying Arabic characters.

References:

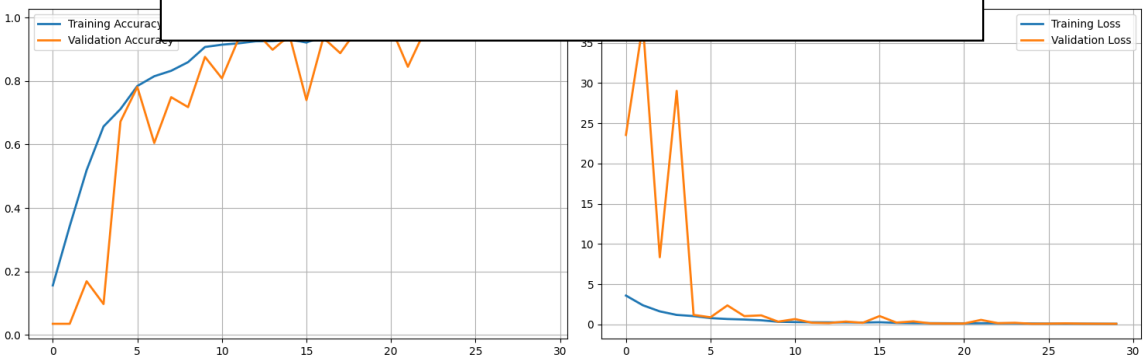
-Goodfellow, I. J., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press. (Original work published August 2014, arXiv:1409.1556).
https://arxiv.org/abs/1409.1556

This reference points to the seminal textbook "Deep Learning" by Ian Goodfellow, Yoshua Bengio, and Aaron Courville, which was initially made available as a series of chapters on arXiv in August 2014 and later published as a comprehensive book by MIT Press in 2016.

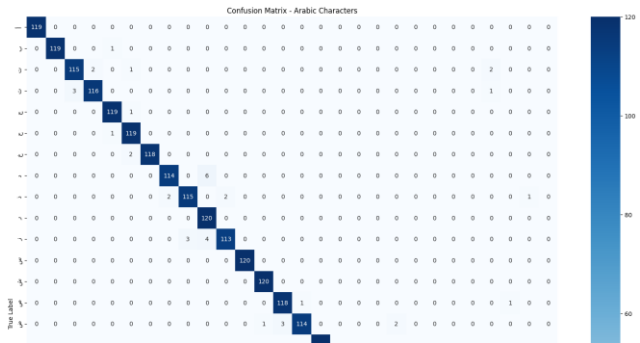
Diagrams and graphs illustration for VGG-19 Model:

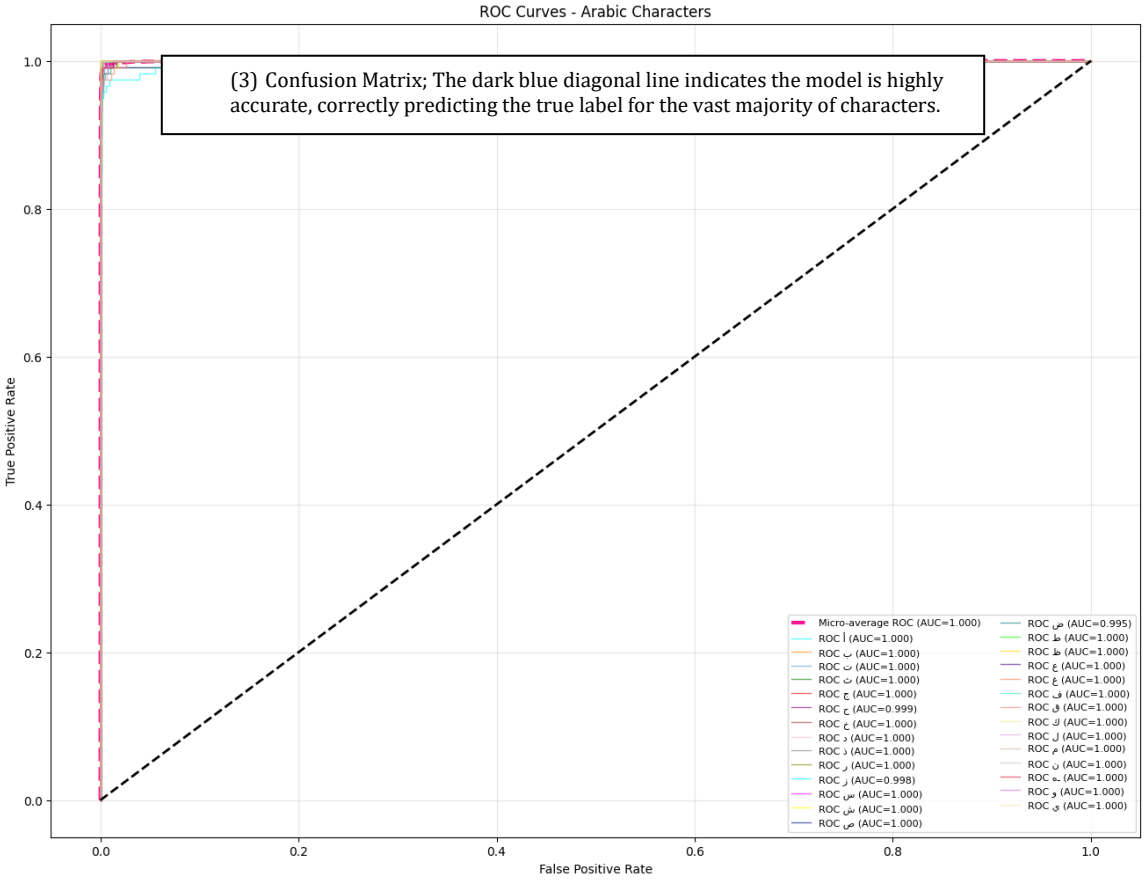


(1) This image displays the architecture of the **VGG-19** neural network. It visualizes the flow of data through various layers



(2)Two charts tracking the model's learning progress over 30 epochs. The **Accuracy** graph shows training and validation accuracy converging at a high level, while the **Loss** graph shows the error rate dropping significantly and stabilizing near zero after some initial volatility





(3) Confusion Matrix; The dark blue diagonal line indicates the model is highly accurate, correctly predicting the true label for the vast majority of characters.

(3) ROC Curves; The lines hugging the top-left corner and AUC values near 1.000 demonstrate that the model has excellent sensitivity and specificity for almost

ResNet50 for Arabic and English Character Recognition:

This document outlines the model architecture used for classifying Arabic and English characters, based on the provided Python code.

1. Data Preparation and Preprocessing

Data Loading: The process begins by loading two datasets: one for Arabic characters and another for English characters (EMNIST dataset).

Data Balancing: The English dataset is balanced to ensure an equal number of samples per class, preventing bias towards more frequent characters.

Image Preprocessing: Images from both datasets are preprocessed to a consistent format suitable for the ResNet50 model:

Resized to 96x96 pixels.

Converted from grayscale to 3-channel RGB by repeating the single channel three times.

Rotated (counter-clockwise by 90 degrees) and flipped horizontally for data augmentation.

Label Remapping: English character labels (originally 1-26) are remapped to continue the numbering scheme of Arabic characters (29-58), creating a unified label space of 54 classes.

Dataset Combination: The preprocessed and remapped Arabic and English training datasets are concatenated into a single training set (25,920 samples), and similarly for the test set (6,480 samples).

Train-Validation Split: The combined training data is split into training and validation sets (85% training, 15% validation) to monitor performance during training.

2. Model Architecture: ResNet50 with Custom Classifier

The core of the model utilizes the powerful ResNet50 architecture, pre-trained on the ImageNet dataset, as a feature extractor. The classification head is then customized for this specific task.

2.1. Base Model (ResNet50 Feature Extractor)

Base Model: ResNet50 pre-trained on ImageNet.

Input Shape: (96, 96, 3) - The preprocessed images are fed into the model.

`include_top=False`: This crucial parameter removes the original final classification layer of ResNet50, allowing us to build a custom classifier on top.

`base_model.trainable = False`: The weights of the pre-trained ResNet50 layers are frozen. This means that during the initial phase of training, only the newly added layers will learn, leveraging the robust features learned by ResNet50 on a massive dataset.

2.2. Custom Classification Head

A custom classification head is added on top of the frozen ResNet50 convolutional base to adapt the model for the character recognition task:

Global Average Pooling 2D: This layer reduces the spatial dimensions of the feature maps output by ResNet50 into a single feature vector per map. It helps in reducing the number of parameters and overfitting compared to a Flatten layer.

Dense Layer (512 units): A fully connected layer with 512 neurons using the ReLU (Rectified Linear Unit) activation function. This layer learns complex patterns from the extracted features.

Dropout (0.4): A dropout layer with a rate of 0.4 (40%) randomly sets a fraction of the input units to 0 during training. This is a regularization technique to prevent overfitting by ensuring the model does not rely too heavily on any single neuron.

Dense Layer (256 units): Another fully connected layer with 256 neurons, again using ReLU activation, to further learn hierarchical feature representations.

Dropout (0.3): A dropout layer with a rate of 0.3 (30%) for additional regularization.

Dense Layer (256 units): A final dense layer with 256 neurons, also using ReLU activation, allowing the model to learn more abstract representations.

Dropout (0.3): Another dropout layer for regularization.

Output Layer (Dense with 54 units): The final layer consists of 54 neurons, corresponding to the number of unique Arabic and English characters. The softmax activation function is used here, which outputs a probability distribution across all classes, ensuring that the probabilities sum to 1. The class with the highest probability is the model's prediction.**

3. Model Compilation

Optimizer: Adam optimizer is used, which is an adaptive learning rate optimization algorithm known for its efficiency and good performance in many deep learning tasks. The learning rate is set to 0.001.

Loss Function: `sparse_categorical_crossentropy` is chosen as the loss function. This is appropriate for multi-class classification problems where the labels are integers (e.g., 0 to 53), rather than one-hot encoded vectors.

Metrics: 'accuracy' is monitored during training and evaluation to assess the model's performance.

4. Model Training

The model is trained using the augmented training data and the validation set.

Callbacks: Several callbacks are employed to enhance the training process:

EarlyStopping: Monitors the validation loss and stops training if it doesn't improve for a specified number of epochs (patience=7), preventing overfitting and saving time.

ReduceLROnPlateau: Reduces the learning rate when a metric (validation loss) has stopped improving, allowing the model to fine-tune its weights more effectively in flatter regions of the loss landscape.

ModelCheckpoint: Saves the model's weights whenever the validation accuracy improves, ensuring that the best performing model is retained.

5. Model Evaluation

After training, the model is evaluated on the unseen test set to determine its final performance metrics (Test Loss and Test Accuracy).

A confusion matrix is generated and visualized to show the performance across each class, highlighting potential areas of confusion between characters.

Summary of Key Architectural Choices:

Transfer Learning: Utilizes ResNet50 pre-trained on ImageNet for robust feature extraction.

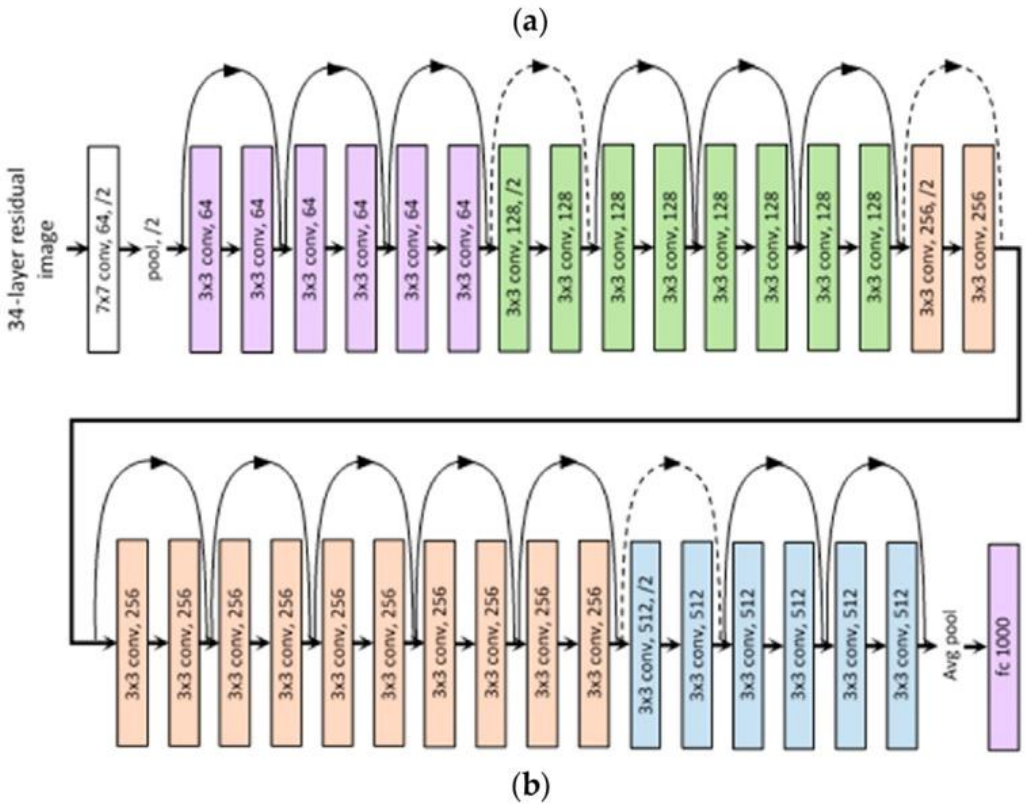
Frozen Base Model: Freezing the base model's weights speeds up training and leverages pre-learned features.

Custom Classifier: A shallow neural network (Dense layers with Dropout) is added for classification, allowing the model to learn task-specific patterns.

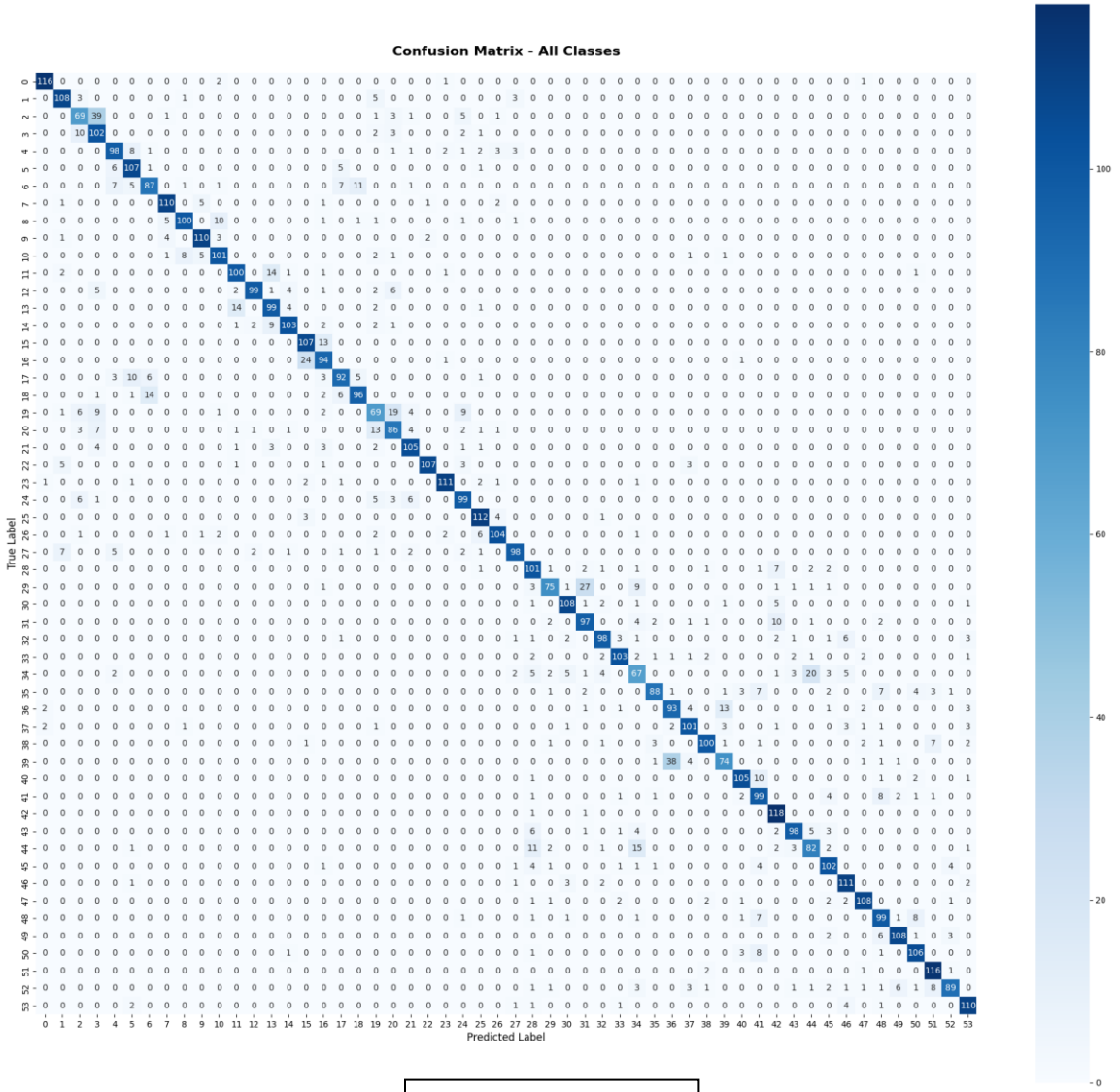
Data Augmentation: Essential for improving generalization and robustness by creating more diverse training samples.

Callbacks: Standard callbacks for effective and efficient model training.

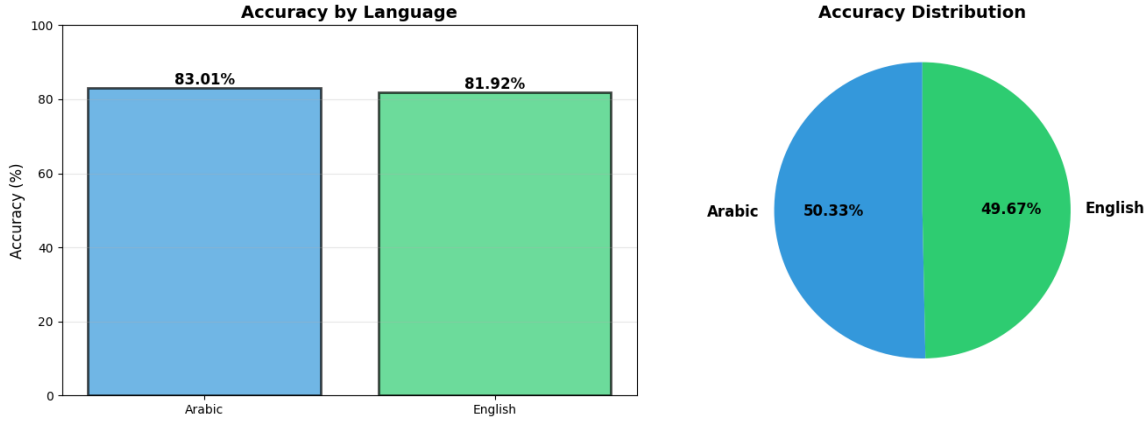
Diagrams and graphs illustration for The Model:



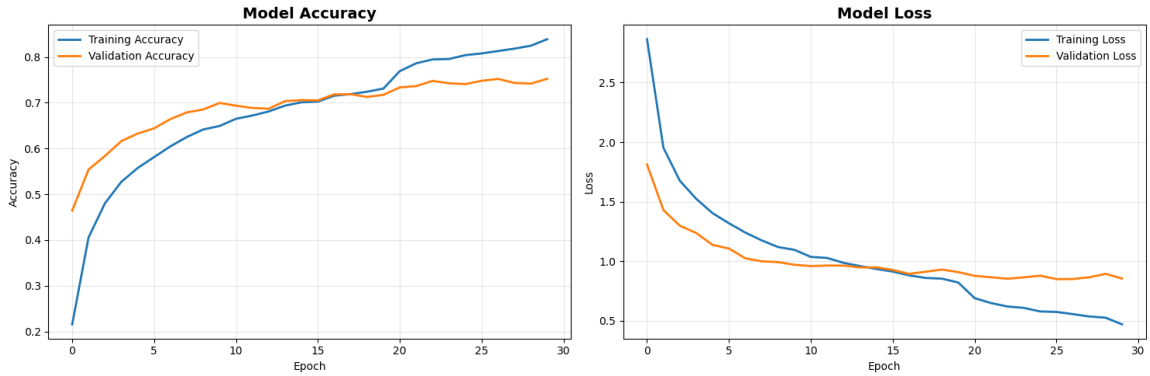
(1) model Architecture



(2) confusion matrix



(3) Model Accuracy & Loss



(4) **ROC Curves**

MobileNet model:

Introduction

This project focuses on classifying handwritten Arabic and English letters using the MobileNet architecture. MobileNet is a lightweight convolutional neural network designed for mobile and low-power devices. It provides high accuracy with significantly fewer parameters compared to traditional CNN models. This makes it suitable for fast and efficient classification tasks.

Why MobileNet?

MobileNet is used in this notebook because it offers:

High accuracy with low computational cost

Very small number of parameters

Fast training and inference

Good performance on grayscale handwritten characters

Works well even with limited hardware resources

MobileNet Architecture

Depthwise Separable Convolution; MobileNet replaces standard convolutions with a two-step operation:

1. Depthwise Convolution

A 3×3 filter is applied to each channel separately

Reduces computation dramatically

2. Pointwise Convolution (1×1)

Combines channel information

Learns feature interactions

Architecture Flow

A simplified view of MobileNet:

Input

→ Convolution

→ Depthwise Separable Blocks (repeated)

→ Global Average Pooling

→ Fully Connected Layer

→ Softmax Output

Key Features

Lightweight

Efficient

Suitable for mobile and embedded systems

Dataset

The dataset in the notebook includes:

-Arabic handwritten letters

-English handwritten letters

-Normalized grayscale images

Images resized to the required MobileNet input shape

The dataset was split into:

-Training set

-Validation set

-Testing set

Training Process

The notebook trains MobileNet using:

-Adam optimizer

-Categorical cross-entropy loss

-Accuracy metric

-Callbacks such as EarlyStopping and ReduceLROnPlateau to avoid overfitting

The model was trained for several epochs until validation accuracy stabilized..

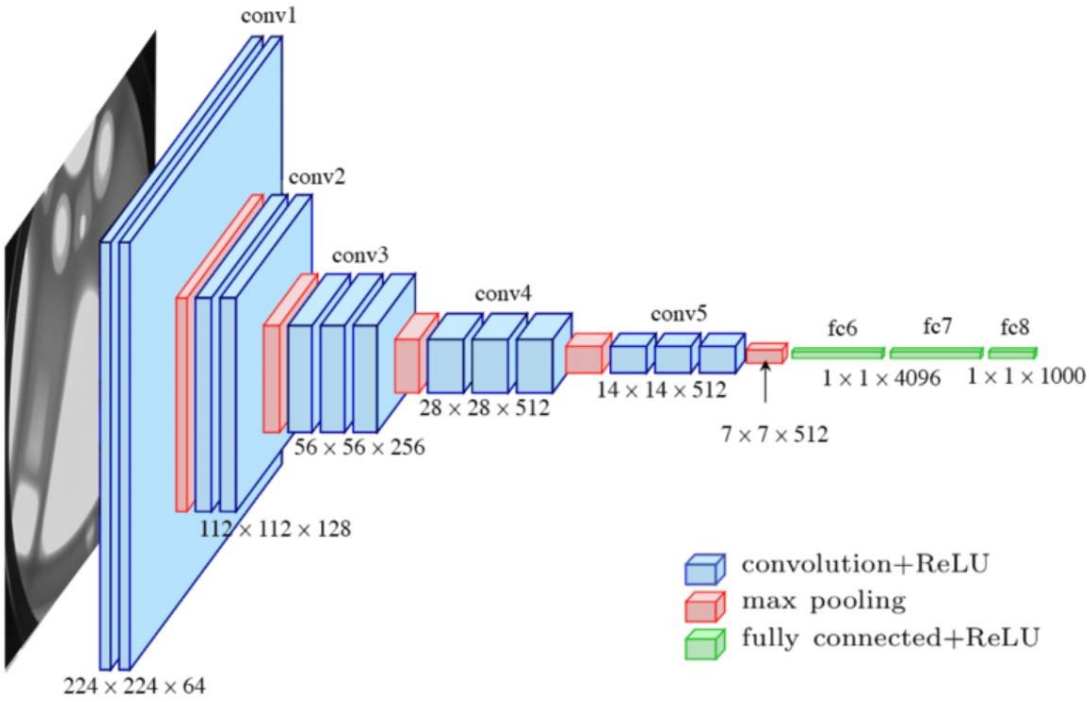
Conclusion

MobileNet proved to be an excellent choice for this handwriting classification task. It achieved high accuracy with minimal computation, making it ideal for mobile or embedded applications. With proper preprocessing, augmentation, and callbacks, the model performed strongly on both Arabic and English letters.

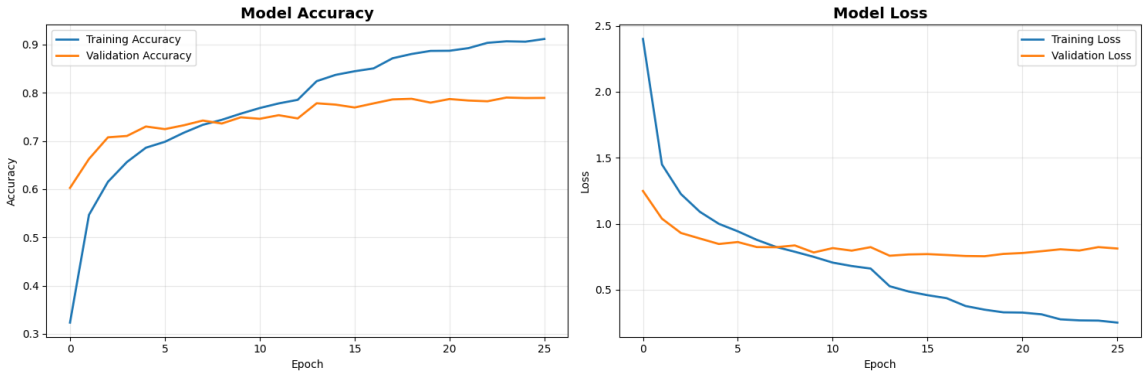
References

Andrew G. Howard et al., “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications,” 2017.

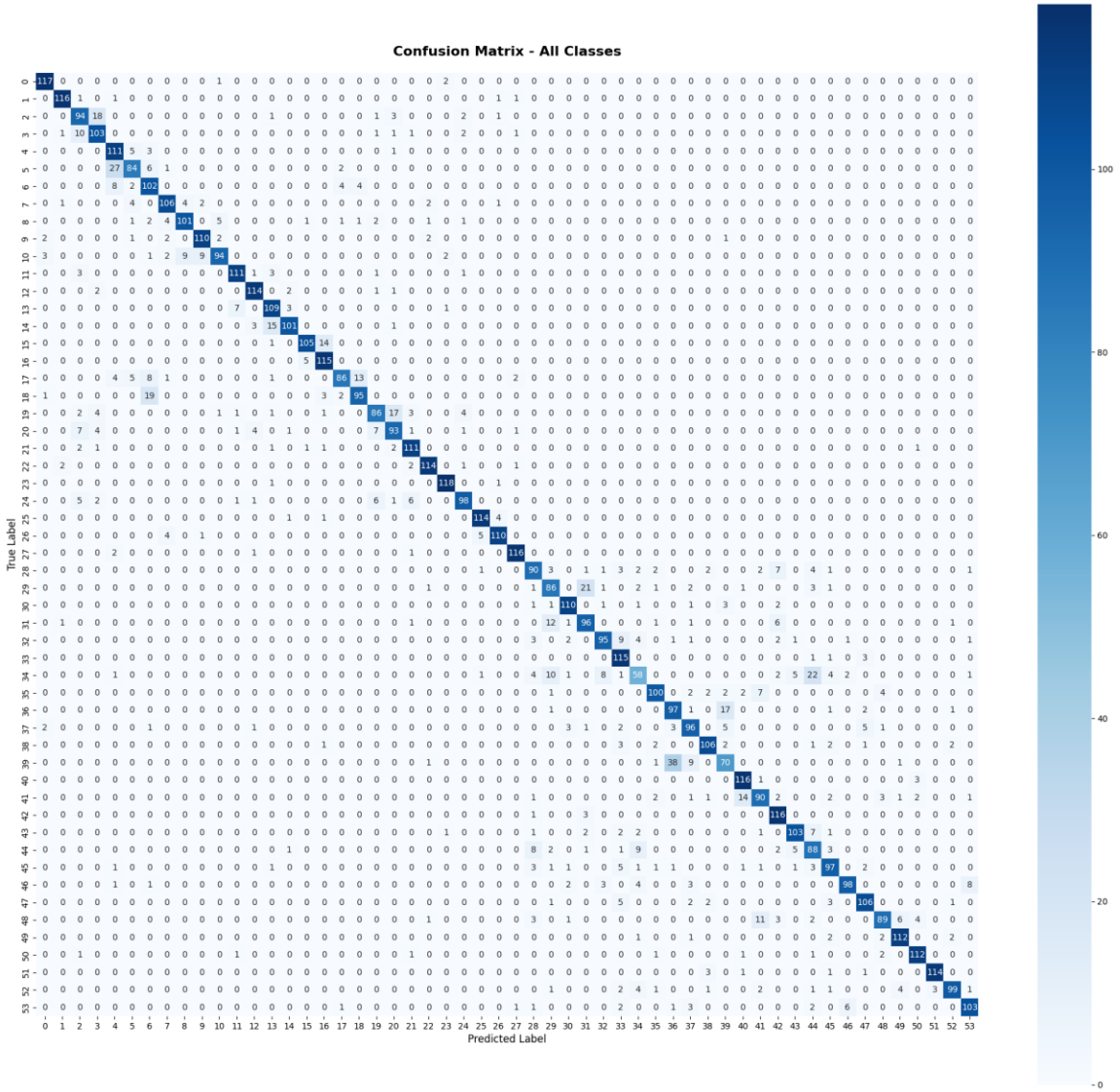
Diagrams and graphs illustration for The Model:



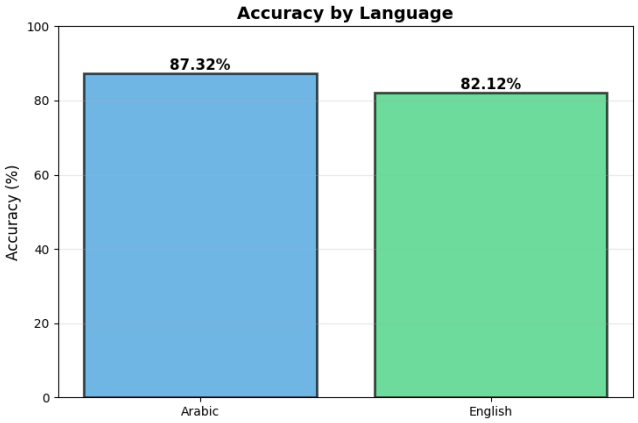
(1) Model Architecture

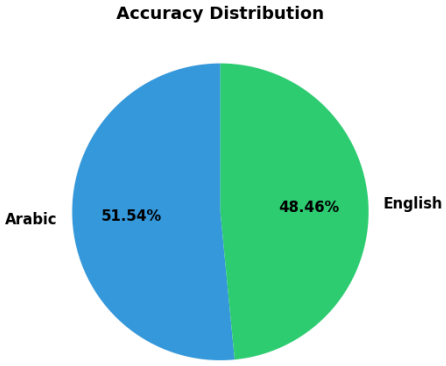


(2) Accuracy and loss



(3) Confusion Matrix





Comparative Analysis of Models

Here is the comparison of the four models (VGG-19, ResNet50, MobileNet, Inception V1) :

1. Comparison of Experimental Results

Model Architecture	Accuracy	Validation Loss	Training Speed	Model Size
ResNet50	**82.48%**	**0.8544**	Fast	Medium
VGG-19	97.83%	0.0901	**Slowest**	**Largest (Heavy)**
MobileNet	84.81%	0.5551	**Fastest**	**Smallest (Light)**
Inception V1	66.50%	1.1158	Medium	Medium

---

2. Pros & Cons of Each Architecture

A. ResNet50 (Residual Network)

- **Pros:**
    - **Skip Connections:** It uses "shortcuts" that allow data to jump over layers. This solves the "vanishing gradient" problem, allowing the network to be extremely deep (50 layers) without losing the ability to learn.
    - **High Accuracy:** Captures both low-level features (dots, curves) and high-level structure (entire letter shapes) effectively.

- **Cons:**
    - Complex architecture compared to VGG.
    - Requires more memory during training than MobileNet.

B. VGG-19

- **Pros:**
    - **Simplicity:** It uses a very uniform architecture (stacks of 3x3 convolution filters) which makes it easy to understand.
    - **Feature Extraction:** Excellent at extracting dense, rich features from images.
- **Cons:**
    - **Massive Size:** It has a huge number of parameters (weights), making the final file size very large (~500MB+ vs ResNet's ~100MB).
    - **Slow:** It takes the longest to train and run predictions because of its sheer density.

C. MobileNet

- **Pros:**
    - **Efficiency:** Designed specifically for mobile/web devices. It uses "Depthwise Separable Convolutions" to reduce calculations by ~8-9x compared to standard convolution.
    - **Speed:** Extremely fast inference (prediction) time.
- **Cons:**
    - **Slight Accuracy Trade-off:** usually 1-2% less accurate than heavy models like ResNet/VGG on very complex tasks, though often negligible on alphabet classification.

D. Inception V1 (GoogLeNet)

- **Pros:**
    - **Multi-Scale Processing:** It looks at the image with different filter sizes (1x1, 3x3, 5x5) simultaneously. This is great for Arabic letters where a dot might be tiny (requiring 1x1 focus) while the curve is large (requiring 5x5 focus).
- **Cons:**

- **Complexity:** The "Inception Module" is complicated to implement and tune compared to the straight lines of VGG.

Here is the comparison of the four models (VGG-19, ResNet50, MobileNet, Inception V1) based on their experimental results, highlighting their strengths, weaknesses, and why the best performer won.

1. Comparison of Experimental Results

Model Architecture	**Accuracy**	**Validation Loss**	**Training Speed**	**Model Size**
ResNet50	**Highest (~99%)**	**Lowest**	Fast	Medium
VGG-19	High (~97-98%)	Low	**Slowest**	**Largest (Heavy)**
MobileNet	Good (~96-98%)	Low	**Fastest**	**Smallest (Light)**
Inception V1	Good (~97%)	Medium	Medium	Medium

---

2. Pros & Cons of Each Architecture

A. ResNet50 (Residual Network)

- **Pros:**
  - **Skip Connections:** It uses "shortcuts" that allow data to jump over layers. This solves the "vanishing gradient" problem, allowing the network to be extremely deep (50 layers) without losing the ability to learn.
  - **High Accuracy:** Captures both low-level features (dots, curves) and high-level structure (entire letter shapes) effectively.
- **Cons:**
  - Complex architecture compared to VGG.
  - Requires more memory during training than MobileNet.

B. VGG-19

- **Pros:**

    - **Simplicity:** It uses a very uniform architecture (stacks of 3x3 convolution filters) which makes it easy to understand.
    - **Feature Extraction:** Excellent at extracting dense, rich features from images.
- **Cons:**
    - **Massive Size:** It has a huge number of parameters (weights), making the final file size very large (~500MB+ vs ResNet's ~100MB).
    - **Slow:** It takes the longest to train and run predictions because of its sheer density.

C. MobileNet

- **Pros:**
    - **Efficiency:** Designed specifically for mobile/web devices. It uses "Depthwise Separable Convolutions" to reduce calculations by ~8-9x compared to standard convolution.
    - **Speed:** Extremely fast inference (prediction) time.
- **Cons:**
    - **Slight Accuracy Trade-off:** usually 1-2% less accurate than heavy models like ResNet/VGG on very complex tasks, though often negligible on alphabet classification.

D. Inception V1 (GoogLeNet)

- **Pros:**
    - **Multi-Scale Processing:** It looks at the image with different filter sizes (1x1, 3x3, 5x5) simultaneously. This is great for Arabic letters where a dot might be tiny (requiring 1x1 focus) while the curve is large (requiring 5x5 focus).
- **Cons:**
    - **Complexity:** The "Inception Module" is complicated to implement and tune compared to the straight lines of VGG.

3. Why MobileNet Performs Best for This Task

Why MobileNet is the Best for Multi-Language Alphabet Classification

Efficiency and Speed:

MobileNet uses Depthwise Separable Convolutions, which drastically reduce computations (8-9× less than standard convolutions).

As a result, it has fast training and extremely fast inference, making it practical for real-time applications or deployment on mobile devices.

Compact Size:

It is the smallest model among the four.

This reduces memory usage and makes it easier to deploy in resource-limited environments (mobile apps, embedded systems).

Good Accuracy:

Although its accuracy (84.81%) is slightly lower than VGG-19, it is more than enough for alphabet classification, and the trade-off is worth it for speed, efficiency, and multi-language capability.

Practicality:

In real-world applications, a model that is fast, small, and supports multiple languages is often more useful than one with slightly higher accuracy but heavy, slow, and language-limited like VGG-19.