



RGB Arabic Alphabets Sign Language

MEM 524– Midterm

Advanced Artificial Intelligence: Generative AI

Supervised by:

Dr. Areej Suleiman Al-Wabil

Prepared by:

Sarah Mohammed Bin Haqqan ID# G236093



I.	Introduction:	3
II.	Project goals:.....	3
III.	Dataset Description:.....	3
IV.	Data Preprocessing	4
V.	Model Selection, Description of Platform, Accuracy, Algorithm, and Evaluation	4
VI.	Results obtained from the evaluation.....	13
VII.	Discussion	15
VIII.	Conclusion	16
IX.	Run it on The Google Colab	17
X.	Set of experiments	21
XI.	link for Testing the Model	23
XII.	References	23



I. INTRODUCTION:

Sign language is considered one of the most important means of communication for people with hearing disabilities, as it represents an effective means of expression that enables them to communicate and interact with the world around them. Through this Mid-Term project, I seek to develop a system capable of converting ten sign language letters from the letter alif to the letter r into written texts, making it easier for others to understand and interact with it.

II. PROJECT GOALS:

Identifying and classifying sign language: Designing a model capable of identifying and classifying Arabic sign language for ten specific letters from the Alif to the letter Ra.

Evaluating the accuracy of the system: testing and evaluating the accuracy of the system and its ability to recognize sign language with high accuracy, by comparing the results calculated by it with the correct results.

Improving performance and accuracy: Exploring and applying techniques to improve performance and increase classification accuracy and signal transformation.

Providing social communication: enabling users to communicate better socially by improving the current means of communication.

III. DATASET DESCRIPTION:

The “RGB Arabic Alphabets Sign Language” dataset contains images representing the letters of the Arabic alphabet in sign language. These images are shot in RGB to provide a comprehensive range of colors. The collection contains many 31 categories, each category represents a letter from the Arabic alphabet, and each category contains at least 250 number of samples. This collection aims to provide rich and diverse data to train machine learning models to classify letters of the Arabic alphabet in sign language.

The labeling of the dataset is based on the Arabic alphabet. Each image is associated with a specific label indicating the corresponding letter in the Arabic alphabet. This labeling scheme is crucial for supervised learning, where the model learns to associate visual features of the sign language gestures with their respective alphabet letters during the training process.

The output of the model is a categorical variable representing the predicted Arabic letter.

<https://www.kaggle.com/datasets/muhammalbrham/rgb-arabic-alphabets-sign-language-dataset?resource=download>



IV. DATA PREPROCESSING

- Ten letters from the letter Alif to the letter Ra were chosen to be the subject of study in this Mid-Term project. This data set was uploaded due to its large size, not all characters were used in the training model. Instead, I selected the first ten letters, created folders with the names of these letters, and added 41 samples of these letters to its folder.
- So, Data Preparation Approach I used:

Data Acquisition: I downloaded the dataset from Kaggle.

Data Cleansing: I checked for and removed corrupted images or images with incorrect labels.

Data Wrangling: I selected a subset of the data (first 10 letters).

I have resized the images to a uniform size for consistency as Teachable Machine requires it.

I have visualized sample images to understand the variations in hand postures, backgrounds, and lighting conditions.

V. MODEL SELECTION, DESCRIPTION OF PLATFORM, ACCURACY, ALGORITHM, AND EVALUATION

- Google Teachable Machine is a web-based tool that allows users to easily train machine learning models without the need for programming knowledge. It uses a technique called transfer learning, where a pre-trained model is adapted to a new task with a relatively small amount of new data.
- Teachable Machine's core functionality for training machine learning models relies on cloud-based resources. When uploading data and train a model, the computations happen on Teachable Machine's servers, not directly on the device's local hardware.
- Teachable Machine typically uses a pre-defined neural network architecture, such as a Convolutional Neural Network (CNN) “which was used here”, to train the model. The exact architecture and number of layers can vary depending on the specific implementation and version of Teachable Machine being used. However, CNNs commonly used in Teachable Machine for image classification tasks may include several convolutional layers followed by pooling layers, and then one or more fully connected layers at the end for classification.
- Convolutional Neural Networks (CNNs) themselves are typically used in supervised learning. This means they require a labeled dataset for training.
- In addition, here are the reason why CNNs are suited for supervised learning:
Labeled Data: Supervised learning algorithms need data that is already classified into categories. In the case of CNNs for image recognition, this means having images with corresponding labels indicating what the image contains (e.g., "cat," "dog," "car").



Learning Features: During training, the CNN's convolutional layers learn features from the labeled images. These features are like building blocks, allowing the network to recognize patterns and identify objects in new images.

Classification: With the learned features, the fully connected layers at the end of the CNN take over. They use the extracted features to classify unseen images into predefined categories based on the training data.

While CNNs are primarily used in supervised learning, there are some emerging applications in the realm of unsupervised learning:

Dimensionality Reduction: CNNs can be used as a pre-processing step for unsupervised tasks. By extracting features from unlabeled images, they can reduce the data's dimensionality, making it easier for unsupervised algorithms to handle complex image data.

Anomaly Detection: Unsupervised CNNs can be trained to identify patterns that deviate from the norm. For example, they could be used to detect unusual objects in security footage.

However, these unsupervised applications are less common than supervised learning tasks for CNNs. Teachable Machine itself focuses on supervised image classification where providing labeled training data.

- While Teachable Machine uses a pre-built CNN architecture, understanding the underlying algorithms can be helpful. Here's a breakdown of the algorithms involved in Convolutional Neural Networks (CNNs):

1. Convolution:

This is the core operation of a CNN. It involves applying a small filter (also called a kernel) that slides across the input image, multiplying element-wise with a specific region of the image.

The purpose is to detect specific features in the image, like edges, lines, or shapes. By moving the filter across the entire image, the CNN learns how these features appear in different locations.

Mathematically, it's a dot product operation between the filter and the image data.

2. Pooling:

This step downsamples the output of the convolution layer, reducing the image size and computational complexity.

Common pooling methods include average pooling and max pooling.

Average pooling takes the average value within the filter's receptive field, summarizing the presence of a feature.

Max pooling picks the maximum value, emphasizing the strongest activation for that feature.

3. Activation Functions:

These functions introduce non-linearity into the network, allowing it to learn complex patterns.

A popular activation function used in CNNs is ReLU (Rectified Linear Unit), which sets negative values to zero and keeps positive values unchanged. This helps the network learn which features are important and adds decision-making capabilities.

4. Backpropagation:

This algorithm is used to train the CNN. It compares the model's predictions with the actual labels and calculates the error.

The error is then propagated backward through the network, adjusting the weights and biases of the filters in each layer.

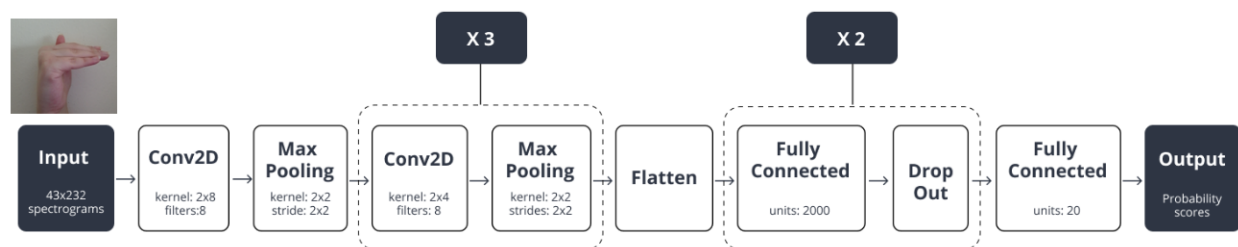
Over multiple training iterations, the CNN iteratively refines its filters to better recognize the features that differentiate the image classes.

5. Fully Connected Layers:

In the later stages of a CNN, there are typically fully connected layers similar to regular neural networks.

These layers take the high-level features extracted by the convolutional layers and combine them to make the final prediction about the image class.

Overall, these algorithms work together in a CNN to achieve impressive image recognition capabilities. By learning features through convolutions, reducing complexity through pooling, introducing non-linearity with activation functions, and fine-tuning through backpropagation, CNNs can become very effective at identifying objects and patterns in images.



- Reflect on the AI-enabled system

In refining the AI-enabled system for Arabic sign language classification, several alternative approaches could be explored for optimization. For example, within the limitations of Google Teachable Machine, it's worthwhile to investigate fine-tuning hyperparameters. If the platform allows any control over hyperparameters, experimenting with different values, such as learning rates and batch sizes, could be instrumental in optimizing the overall performance of the model.



- Reflect on the algorithm , “A reflection on Convolutional Neural Networks (CNNs) alongside other approaches for image recognition” :

Strengths of CNNs:

Feature Learning: A major advantage of CNNs is their ability to automatically learn features from the data. This eliminates the need for manual feature engineering, a time-consuming and expert-driven process in traditional computer vision approaches.

High Accuracy: CNNs have achieved state-of-the-art performance in various image recognition tasks, like object detection and classification. They excel at capturing spatial relationships between pixels in images, crucial for accurate recognition.

Scalability: CNNs can be effectively scaled to handle large and complex datasets. By adding more layers or increasing the number of filters, they can learn increasingly intricate features, improving recognition capabilities.

Other Approaches to Consider:

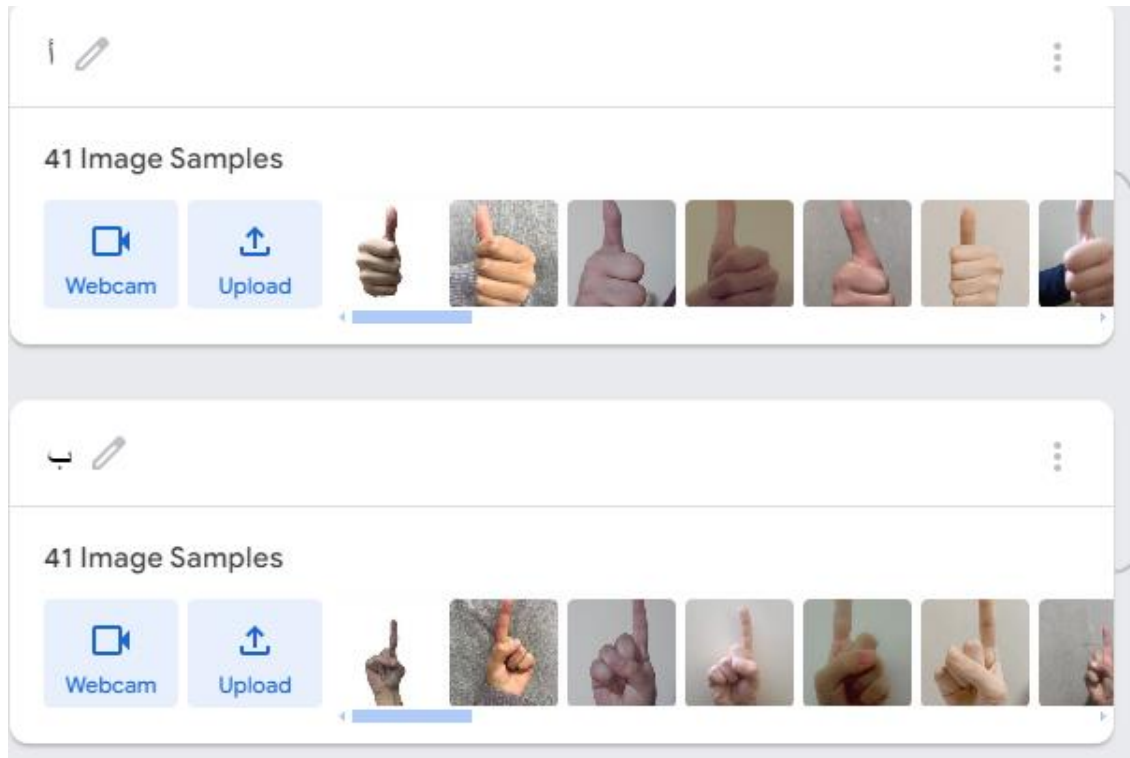
Support Vector Machines (SVMs): SVMs are powerful supervised learning algorithms that can be used for image classification. They work by finding a hyperplane that best separates different classes in the data. While effective, SVMs often require manual feature engineering and might not scale as well as CNNs for highly complex datasets.

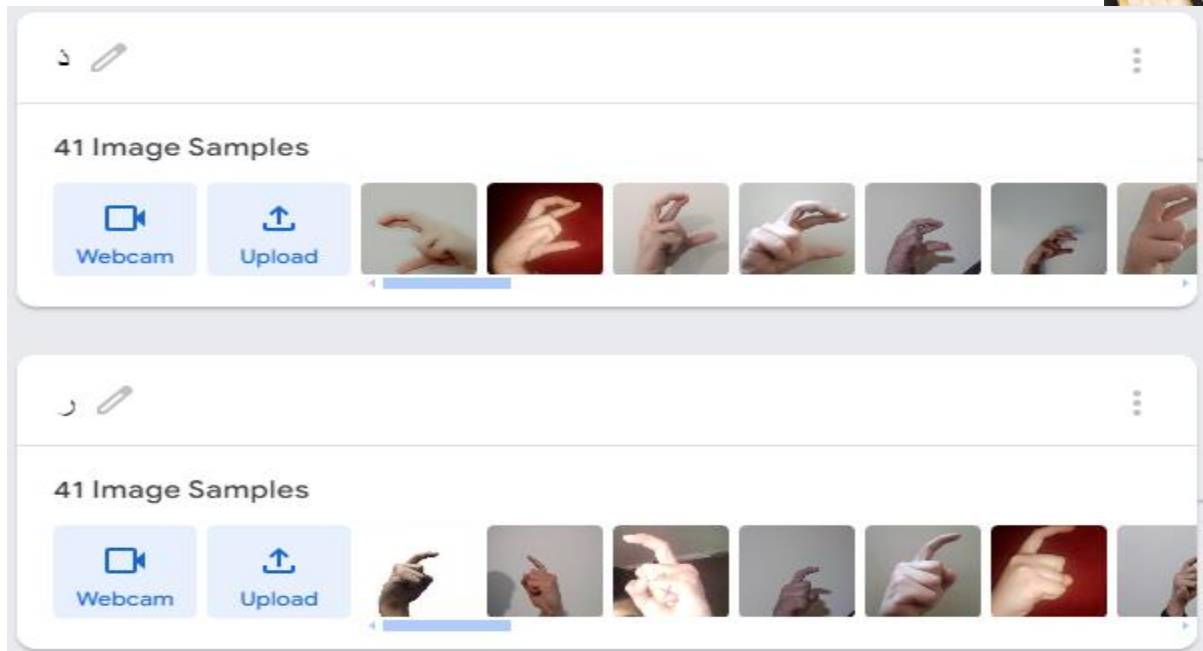
K-Nearest Neighbors (KNN): This is a simple non-parametric algorithm that classifies data points based on their similarity to labeled data points in the training set. KNN can be effective for small datasets but can become computationally expensive with large image datasets.

Decision Trees: These tree-like structures can be used for image classification by learning a series of rules based on image features. However, decision trees might struggle to capture the complex relationships between pixels in images, potentially limiting their accuracy compared to CNNs.

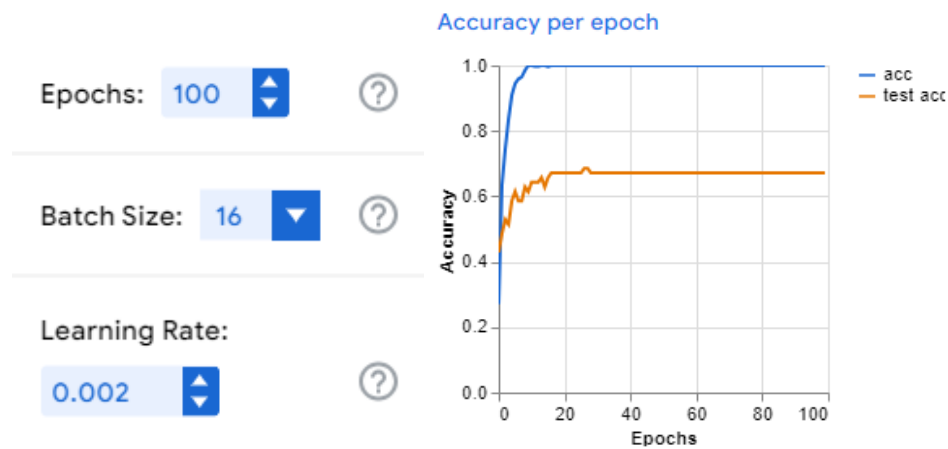


- After accessing the teachable machine model, I created 10 classes to represent the first ten letters of the Arabic alphabet, from Alif (ا) to Ra (ر). Each class corresponds to one letter. For each class, I uploaded 41 images that contain that letter in different contexts or styles. This dataset will be used to train the model to recognize and classify these letters in sign language.





- As I mentioned, Teachable Machine provides a simple interface for training the model. I adjusted the training factors, such as the number of epochs (refers to one complete pass through the entire training dataset during the training phase of a model) to be 100 and batch size (which refers to the number of training examples utilized in one iteration) to be 16 and learning rate (controls how much we are adjusting the weights of our network with respect to the loss gradient) to be 0.002, the model was able to classify alphabets with an accuracy rate for each category as shown below, with an overall accuracy of 67%.



Training Parameters

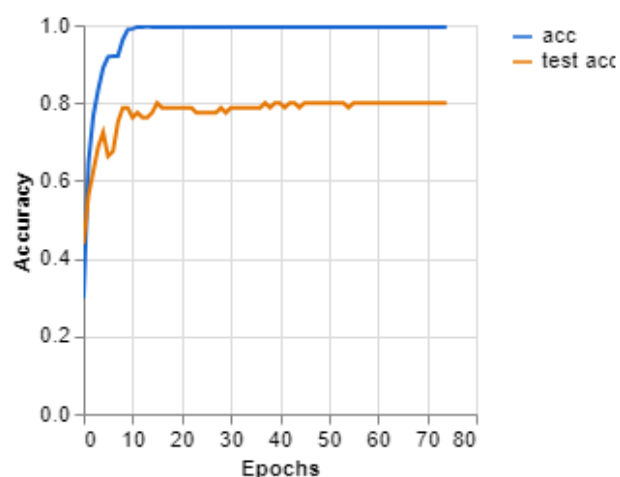
over all accuracy

Accuracy per class

CLASS	ACCURACY	# SAMPLES
ا	0.57	7
ب	0.71	7
ت	0.71	7
ث	1.00	7
ج	0.57	7
ح	1.00	7
خ	0.86	7
د	0.57	7
ذ	0.43	7
ر	0.29	7

- To improve the system's performance. After multi times training, I found that if I add more sample to my data set (change it from 42 to 51) and then change the epochs to 75 ,batch size to be 32 and keep the learning rate 0.002 the model was able to classify RGB Arabic Alphabets with an accuracy rate of 75% to 100% for each category as shown below , with an overall accuracy of 83.3%.

Accuracy per epoch





Training

Model Trained

Advanced



Epochs: 75



Batch Size: 32



Learning Rate:

0.002

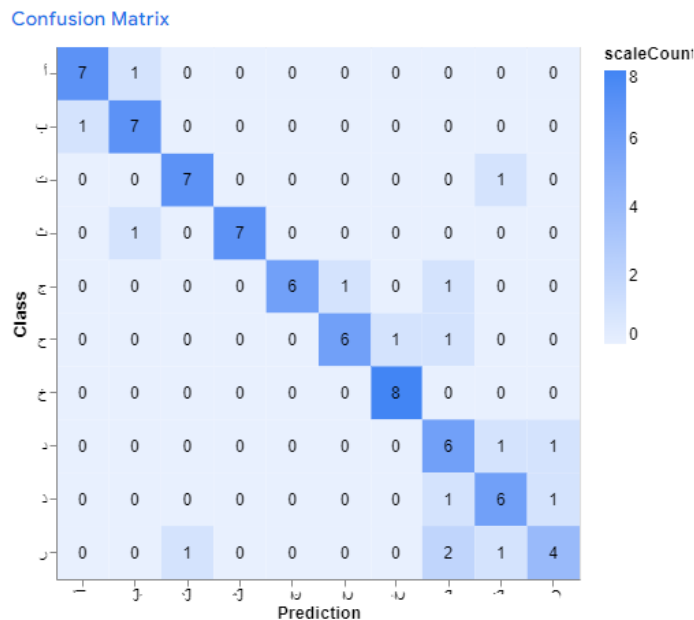


Accuracy per class

CLASS	ACCURACY	# SAMPLES
أ	0.88	8
ب	0.88	8
ك	0.88	8
ت	0.88	8
ج	0.75	8
ح	0.75	8
خ	1.00	8
د	0.75	8
ذ	0.75	8

- Evaluation

The confusion matrix provided is related to a classification task letters from Alf to Ra. The matrix represents the classification results, where each row corresponds to a true class and each column corresponds to a predicted class.



Rows: represent the actual classes (true labels) of the samples.

Columns: represent the predicted classes (predicted labels) by the model for the samples.

Diagonal Values (True Positives): The diagonal values represent the number of samples that were correctly classified for each class. For example, class Alf has 7 correct predictions, class Ba has 7 correct predictions, and so on.

Off-Diagonal Values (False Positives and False Negatives): The off-diagonal values represent the number of misclassifications. For example, the (Alf, Ba) cell has a value of 1, indicating that there was 1 sample that belonged to class Alf but was classified as class Ba.

Overall Performance:

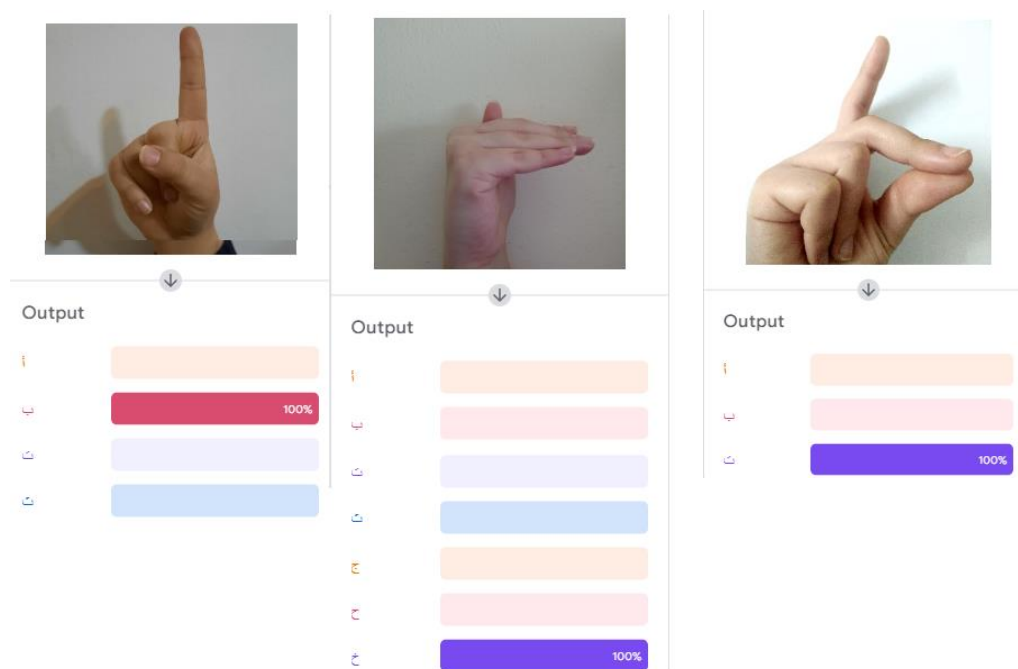
- The model performs well for most classes, with high numbers of true positives and low numbers of false positives and false negatives.
- Classes Kha (8 correct predictions), Tah (7 correct predictions), and Tha (7 correct predictions) are classified with high accuracy, as indicated by the high diagonal values.
- Classes Jeem, Hea, Dal, Thal, and Ra have a few misclassifications, but overall, the model's performance is still quite good.

Areas of Improvement:

- Classes Jeem and Hae have some confusion with each other, indicating that the model may struggle to distinguish between these two classes. Additional features or fine-tuning of the model may help improve this.
- Class Ra has a relatively higher number of false negatives compared to other classes, indicating that the model may need more training data or a different approach to better distinguish this class.

VI. RESULTS OBTAINED FROM THE EVALUATION.

Below are screenshots for the results obtained from the evaluation:





Output



Output

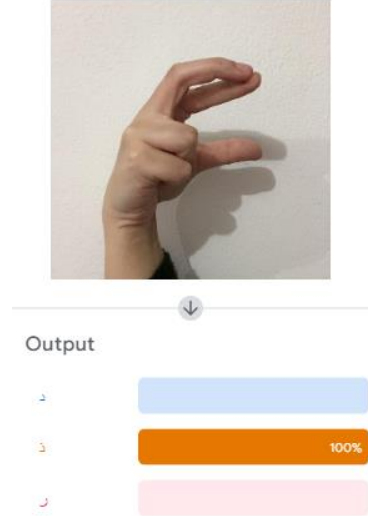
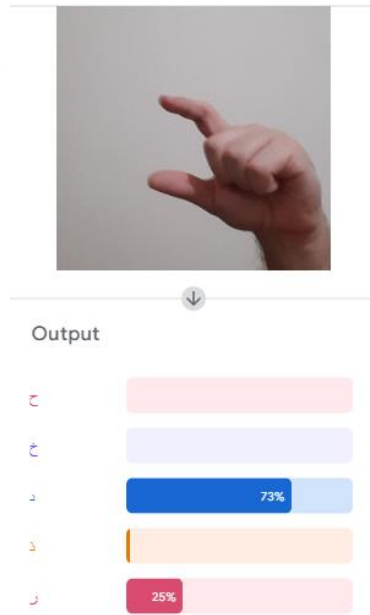


Output



Output





VII. DISCUSSION

Based on the results obtained, I was able to improve the performance of my model by increasing the dataset size and adjusting the training parameters and some observations.

Increased Dataset Size: Increasing the dataset size from 42 to 51 samples likely helped improve the model's performance by providing more diverse examples for each class. This can help the model generalize better to unseen data.

1. Adjusted Training Parameters:

- **Epochs:** Increasing the number of epochs to 75 allowed the model to train for longer, potentially capturing more patterns in the data. However, it's essential to monitor for overfitting when increasing the number of epochs.
- **Batch Size:** A batch size of 32 can help the model converge faster during training, especially with a larger dataset. It can also help improve the model's generalization ability.
- **Learning Rate:** Keeping the learning rate at 0.002 is a good practice, as it ensures a moderate rate of learning without causing the model to diverge or get stuck in local minima.



2. **Improved Accuracy:** With the adjustments made, the model was able to achieve an overall accuracy of 83.3%, with individual class accuracies ranging from 75% to 100%. This indicates that the model is performing well, with high accuracy for most classes.
3. **Room for Further Improvement:** While the model's performance has improved, there may still be room for further optimization. Experimenting with different architectures, data augmentation techniques, or hyperparameter tuning could potentially lead to even higher accuracy.

VIII. CONCLUSION

In conclusion, the improvements made to the model, including increasing the dataset size, adjusting the training parameters, and maintaining a moderate learning rate, have led to a substantial enhancement in performance. The model now achieves an overall accuracy of 83.3% for classifying Arabic alphabets from 'Alif' to 'Ra', with individual class accuracies ranging from 75% to 100%.

These results indicate that the model is effective in distinguishing between the different Arabic characters, with high accuracy for most classes. However, there is still room for further improvement, and it may be beneficial to continue refining the model through additional experimentation and optimization.

Overall, the success of these improvements highlights the importance of dataset size and training parameters in the performance of a machine learning model. By carefully tuning these aspects, it is possible to achieve significant enhancements in model accuracy and performance.

IX. RUN IT ON THE GOOGLE COLAB

Export the Model

I choose TensorFlow as framework and ensure that Keras is selected. Then, clicked the download button to retrieve my model, which will be saved as Keras.h5 along with labels.txt.



Export your model to use it in projects. X

Tensorflow.js ⓘ **Tensorflow** ⓘ Tensorflow Lite ⓘ

Model conversion type:

☒ Keras ☐ Savedmodel [Download my model](#)

Converts your model to a keras .h5 model. Note the conversion happens in the cloud, but your training data is not being uploaded, only your trained model.

Code snippets to use your model:

Keras OpenCV Keras [Contribute on Github](#)

```
from keras.models import load_model # TensorFlow is required for Keras to work
from PIL import Image, ImageOps # Install pillow instead of PIL
import numpy as np

# Disable scientific notation for clarity
np.set_printoptions(suppress=True)

# Load the model
model = load_model("keras_Model.h5", compile=False)
```

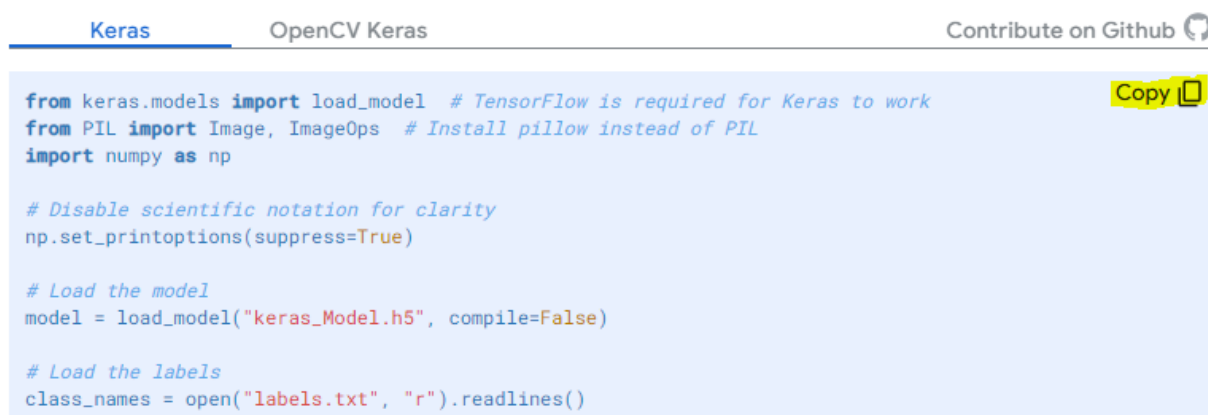
[Copy](#)

[Export Model](#)

tain a model on the left can preview it here.

Clicking 'Copy' to copy the Python code for the model

Code snippets to use your model:



Keras OpenCV Keras [Contribute on Github](#)

```
from keras.models import load_model # TensorFlow is required for Keras to work
from PIL import Image, ImageOps # Install pillow instead of PIL
import numpy as np

# Disable scientific notation for clarity
np.set_printoptions(suppress=True)

# Load the model
model = load_model("keras_Model.h5", compile=False)

# Load the labels
class_names = open("labels.txt", "r").readlines()
```

[Copy](#)



Next, opening Google Colab and create a new notebook. Pasting the copied code into the notebook and follow the instructions highlighted in green, including uploading the test image, labels.txt, and keras_moodle.h5 to Colab.

```
from keras.models import load_model # TensorFlow is required for Keras to work
from PIL import Image, ImageOps # Install pillow instead of PIL
import numpy as np

# Disable scientific notation for clarity
np.set_printoptions(suppress=True)

# Load the model
model = load_model("/content/keras_model.h5", compile=False)

# Load the labels
class_names = open("/content/labels.txt", "r").readlines()

# Create the array of the right shape to feed into the keras model
# The 'length' or number of images you can put into the array is
# determined by the first position in the shape tuple, in this case 1
data = np.ndarray(shape=(1, 224, 224, 3), dtype=np.float32)

# Replace this with the path to your image
image = Image.open("/content/Jeem_34.jpg").convert("RGB")

# Replace this with the path to your image
image = Image.open("/content/Jeem_34.jpg").convert("RGB")

# resizing the image to be at least 224x224 and then cropping from the center
size = (224, 224)
image = ImageOps.fit(image, size, Image.Resampling.LANCZOS)

# turn the image into a numpy array
image_array = np.asarray(image)

# Normalize the image
normalized_image_array = (image_array.astype(np.float32) / 127.5) - 1

# Load the image into the array
data[0] = normalized_image_array

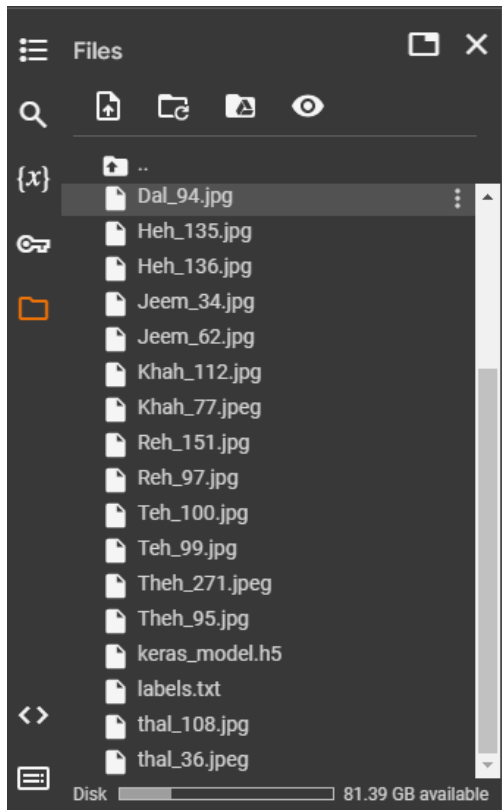
# Predicts the model
prediction = model.predict(data)
index = np.argmax(prediction)
class_name = class_names[index]
confidence_score = prediction[0][index]

# Print prediction and confidence score
print("Class:", class_name[2:], end="")
print("Confidence Score:", confidence_score)

1/1 [=====] - 1s 564ms/step
Class: ج
Confidence Score: 0.8899728
```



Uploading the Test image, the label.txt and keras_moodle.h5 to colab



```
from keras.models import load_model # TensorFlow is required for Keras to work
```

```
from PIL import Image, ImageOps # Install pillow instead of PIL
```

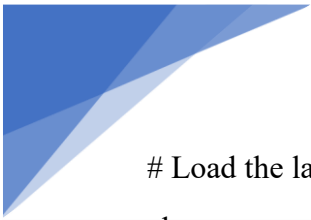
```
import numpy as np
```

```
# Disable scientific notation for clarity
```

```
np.set_printoptions(suppress=True)
```

```
# Load the model
```

```
model = load_model("/content/keras_model.h5", compile=False)
```



```
# Load the labels
class_names = open("/content/labels.txt", "r").readlines()

# Create the array of the right shape to feed into the keras model
# The 'length' or number of images you can put into the array is
# determined by the first position in the shape tuple, in this case 1
data = np.ndarray(shape=(1, 224, 224, 3), dtype=np.float32)

# Replace this with the path to your image
image = Image.open("/content/Jeem_34.jpg").convert("RGB")

# resizing the image to be at least 224x224 and then cropping from the center
size = (224, 224)
image = ImageOps.fit(image, size, Image.Resampling.LANCZOS)

# turn the image into a numpy array
image_array = np.asarray(image)

# Normalize the image
normalized_image_array = (image_array.astype(np.float32) / 127.5) - 1

# Load the image into the array
data[0] = normalized_image_array

# Predicts the model
prediction = model.predict(data)
index = np.argmax(prediction)
class_name = class_names[index]
```



```
confidence_score = prediction[0][index]
```

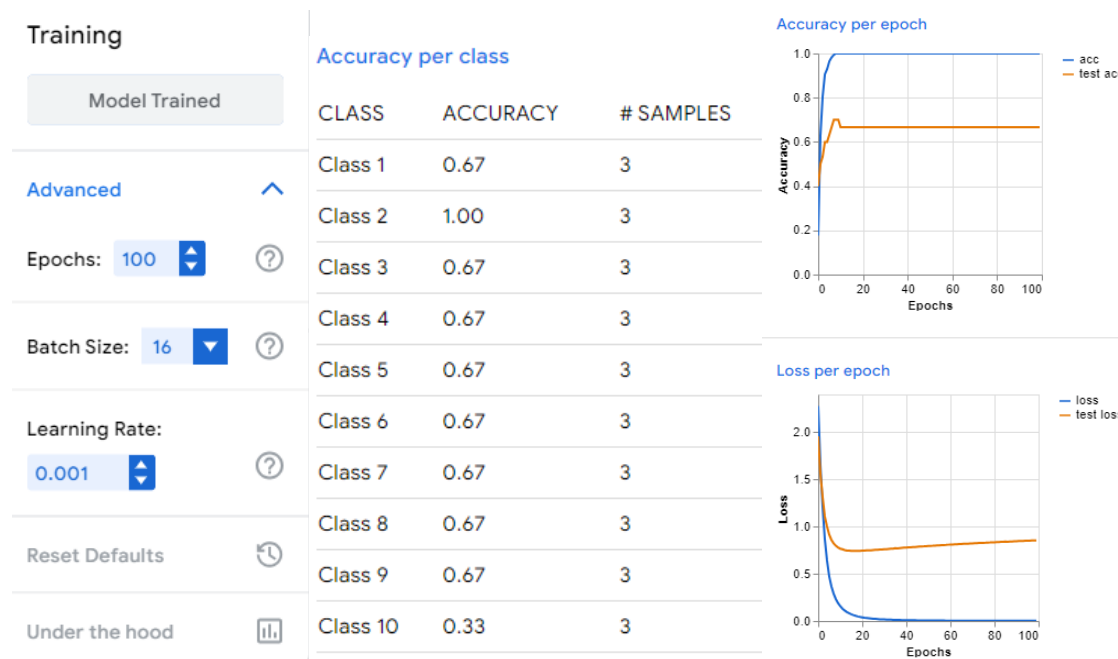
```
# Print prediction and confidence score
```

```
print("Class:", class_name[2:], end="")
```

```
print("Confidence Score:", confidence_score)
```

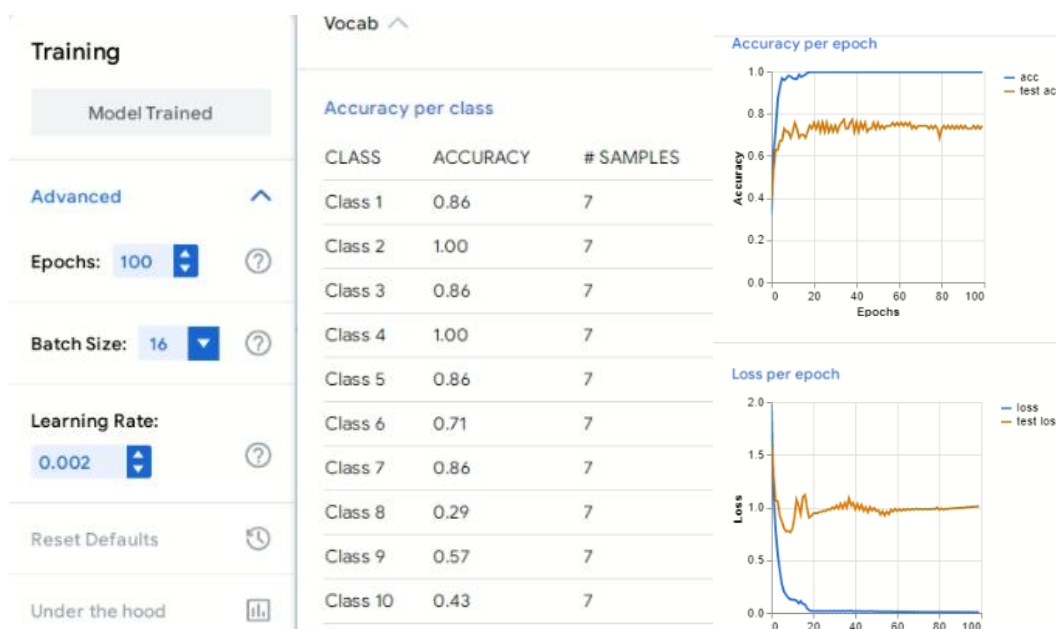
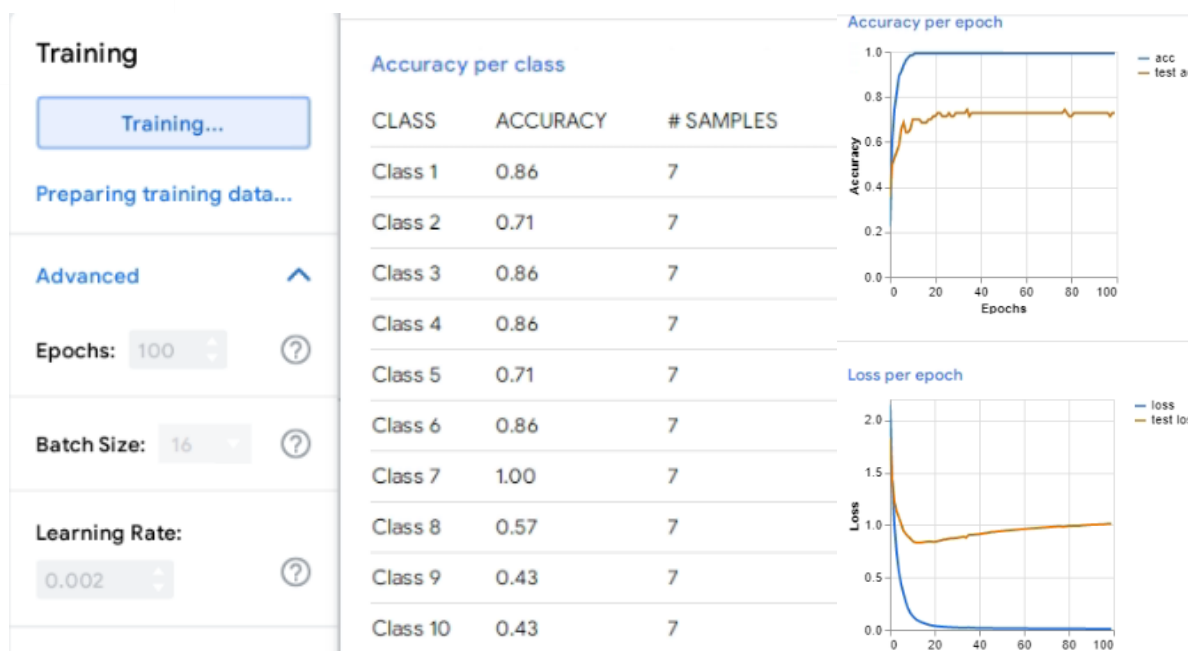
X. SET OF EXPERIMENTS

This is a set of experiments I worked on to improve the model's performance and achieve the best possible results. I changed many settings, including the number of epochs, batch size, and learning rate.





جامعة الفيصل
Alfaisal University





XI. LINK FOR TESTING THE MODEL

The model and associated resources were uploaded to a public repository on GitHub [<https://github.com/SarahAlhaqan1/-MEM-524-Advanced-Artificial-Intelligence-Generative-AI.git>]. This platform allows for efficient testing and version control, enabling the tracking of changes made to the model and facilitating collaboration with other researchers who may wish to contribute or build upon the work.

XII. REFERENCES

<https://teachablemachine.withgoogle.com/>

<https://www.kaggle.com/datasets/muhammadalbrham/rgb-arabic-alphabets-sign-language-dataset?resource=download>

<https://www.sciencedirect.com/topics/engineering/confusion-matrix#:~:text=A%20confusion%20matrix%20is%20a,performance%20of%20a%20classification%20algorithm.>

<https://www.geeksforgeeks.org/how-does-epoch-affect-accuracy-in-deep-learning-model/>

<https://towardsdatascience.com/parameters-and-hyperparameters-aa609601a9ac>

<https://github.com/>