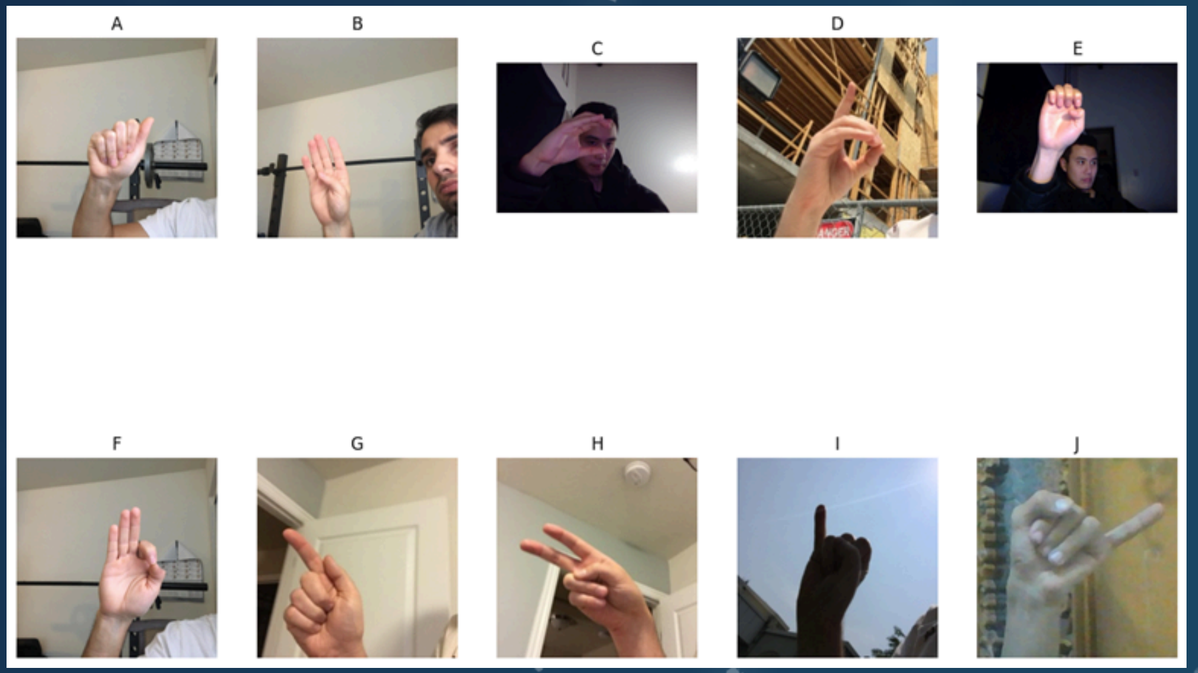
ASL GESTURE RECOGNITION

USING CNNS FOR REAL-TIME TRANSLATION

Abatov M., BDA-2206,
Darmenov Z., BDA-2205

INTRODUCTION

Project Objective:
To classify ASL alphabet signs using deep learning for improved communication accessibility.
Approach:
Leveraged transfer learning with MobileNetV2 for high-accuracy sign recognition.



DATASET DETAILS

- Source: ASL Alphabet Dataset from Kaggle.
- Classes: 29 categories including A-Z, "space," "nothing," and "del".
- Data Distribution:
- Training: 178,447 images.
- Validation: 44,627 images.
- Testing: 28 images (1 per class for validation simplicity).
- Preprocessing:
- Resized to 224x224 pixels.
- Normalized pixel values.

MODEL SELECTION AND DESIGN

Base Model: MobileNetV2 pretrained on ImageNet.
Custom Layers: Added dense layers for classifying 29 ASL categories.
Optimization:

- Loss function: Sparse categorical cross-entropy.
- Optimizer: Adam with initial learning rate of 0.0001.

Build the Model

```
base_model = MobileNetV2(weights='imagenet', include_top=False, input_shape=(IMG_SIZE, IMG_SIZE, 3))
base_model.trainable = False  # Freeze base model layers

# Add custom layers
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(128, activation='relu')(x)
x = Dropout(0.5)(x)
predictions = Dense(len(classes), activation='softmax')(x)

model = Model(inputs=base_model.input, outputs=predictions)

# Compile the model
model.compile(optimizer=Adam(learning_rate=1e-4), loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

TRAINING PROCESS
AND IMPROVEMENTS

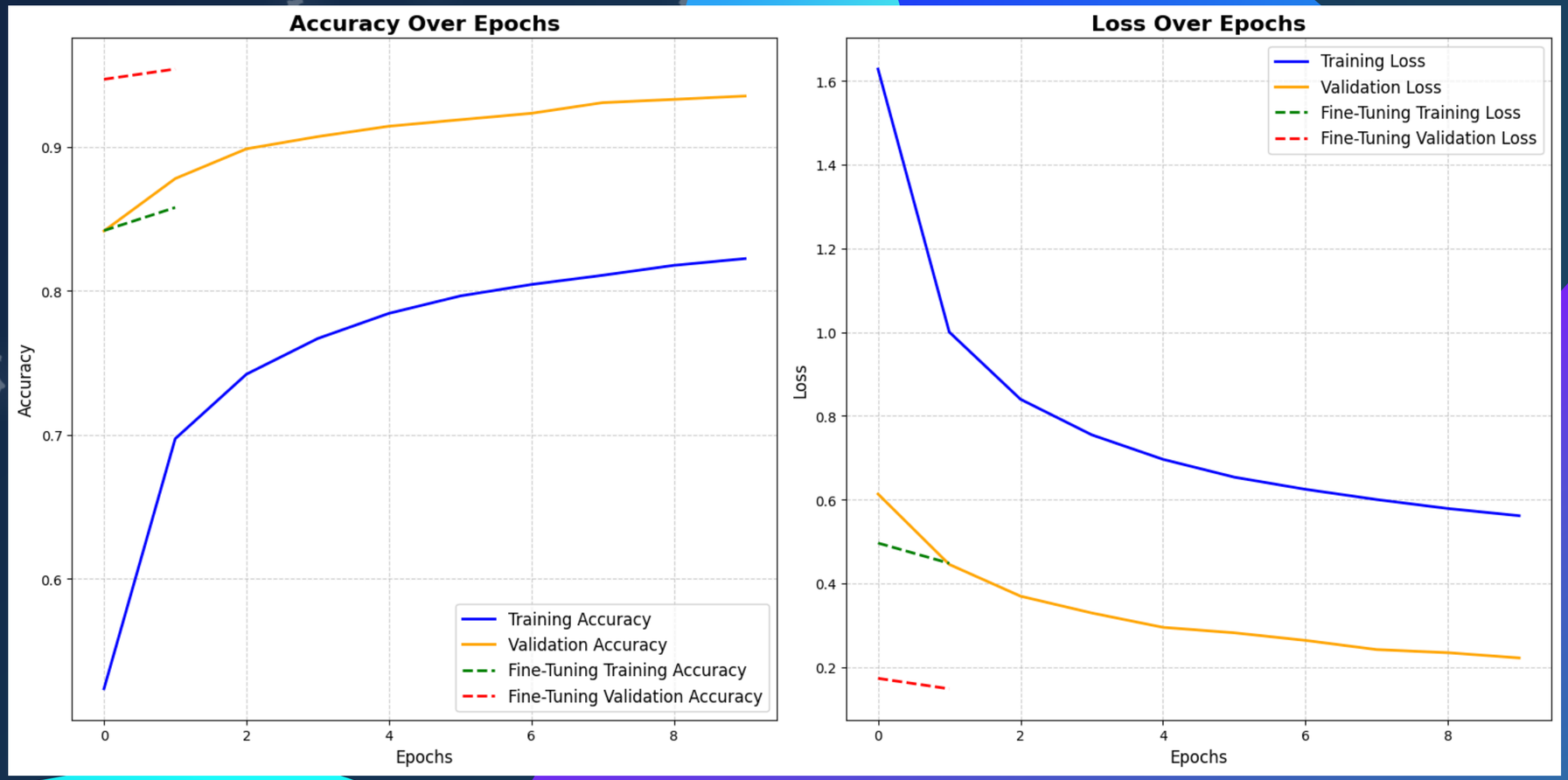
Initial Training:
10 epochs, achieving 82.16% training accuracy and 93.53% validation accuracy.
Fine-Tuning:
Unfroze the last 10 layers and trained for 2 additional epochs.
Improved validation accuracy to 95.40%.
Challenges Addressed:
Class imbalances and overfitting managed with learning rate scheduling and data augmentation.

```
Epoch 1/10
5577/5577 ━━━━━━━━━━━━━━━━━━━━ 3087s 553ms/step - accuracy: 0.3989 - loss: 2.0898 - val_accuracy: 0.8415 - val_loss: 0.6131 - learning_rate: 1.0000e-04
Epoch 2/10
5577/5577 ━━━━━━━━━━━━━━━━━━━━ 3067s 550ms/step - accuracy: 0.6812 - loss: 1.0628 - val_accuracy: 0.8779 - val_loss: 0.4449 - learning_rate: 1.0000e-04
Epoch 3/10
5577/5577 ━━━━━━━━━━━━━━━━━━━━ 3165s 568ms/step - accuracy: 0.7336 - loss: 0.8698 - val_accuracy: 0.8986 - val_loss: 0.3690 - learning_rate: 1.0000e-04
Epoch 4/10
5577/5577 ━━━━━━━━━━━━━━━━━━━━ 3064s 549ms/step - accuracy: 0.7634 - loss: 0.7688 - val_accuracy: 0.9071 - val_loss: 0.3289 - learning_rate: 1.0000e-04
Epoch 5/10
5577/5577 ━━━━━━━━━━━━━━━━━━━━ 3068s 550ms/step - accuracy: 0.7823 - loss: 0.7046 - val_accuracy: 0.9143 - val_loss: 0.2945 - learning_rate: 1.0000e-04
Epoch 6/10
5577/5577 ━━━━━━━━━━━━━━━━━━━━ 3074s 551ms/step - accuracy: 0.7938 - loss: 0.6623 - val_accuracy: 0.9188 - val_loss: 0.2815 - learning_rate: 1.0000e-04
Epoch 7/10
5577/5577 ━━━━━━━━━━━━━━━━━━━━ 3062s 549ms/step - accuracy: 0.8020 - loss: 0.6308 - val_accuracy: 0.9233 - val_loss: 0.2634 - learning_rate: 1.0000e-04
Epoch 8/10
5577/5577 ━━━━━━━━━━━━━━━━━━━━ 3032s 544ms/step - accuracy: 0.8093 - loss: 0.6036 - val_accuracy: 0.9307 - val_loss: 0.2414 - learning_rate: 1.0000e-04
Epoch 9/10
5577/5577 ━━━━━━━━━━━━━━━━━━━━ 3086s 553ms/step - accuracy: 0.8157 - loss: 0.5851 - val_accuracy: 0.9329 - val_loss: 0.2340 - learning_rate: 1.0000e-04
Epoch 10/10
5577/5577 ━━━━━━━━━━━━━━━━━━━━ 3240s 581ms/step - accuracy: 0.8216 - loss: 0.5647 - val_accuracy: 0.9353 - val_loss: 0.2215 - learning_rate: 1.0000e-04
```

```
Epoch 3/4
5577/5577 ━━━━━━━━━━━━━━━━━━━━ 3524s 631ms/step - accuracy: 0.8368 - loss: 0.5114 - val_accuracy: 0.9469 - val_loss: 0.1727 - learning_rate: 1.0000e-05
Epoch 4/4
5577/5577 ━━━━━━━━━━━━━━━━━━━━ 4037s 724ms/step - accuracy: 0.8557 - loss: 0.4544 - val_accuracy: 0.9540 - val_loss: 0.1477 - learning_rate: 1.0000e-05
```

EVALUATION
RESULTS

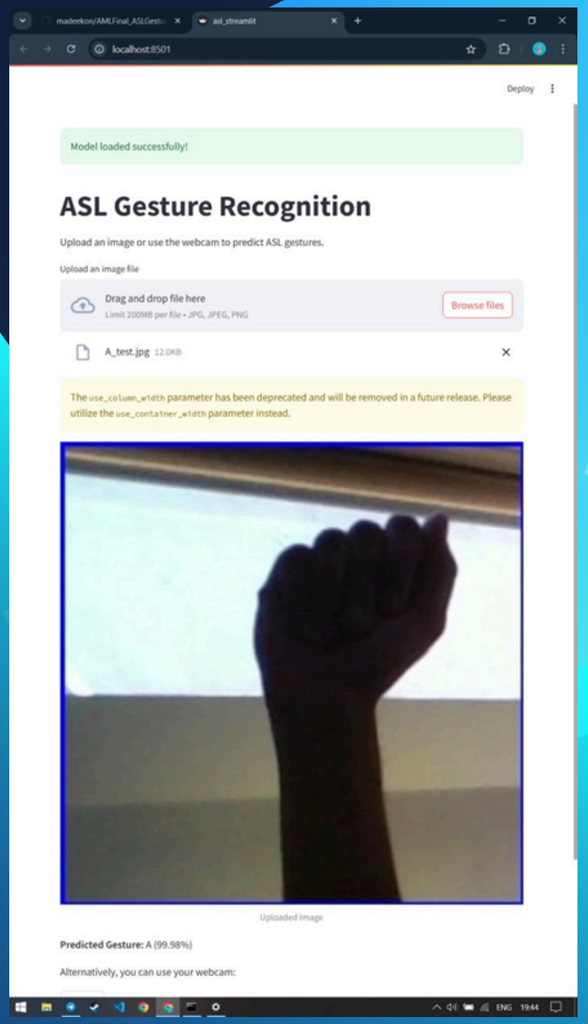
Metrics:
Final Test Accuracy: 92.86%.
Validation Loss Reduction: From 0.61 to 0.22 during training.

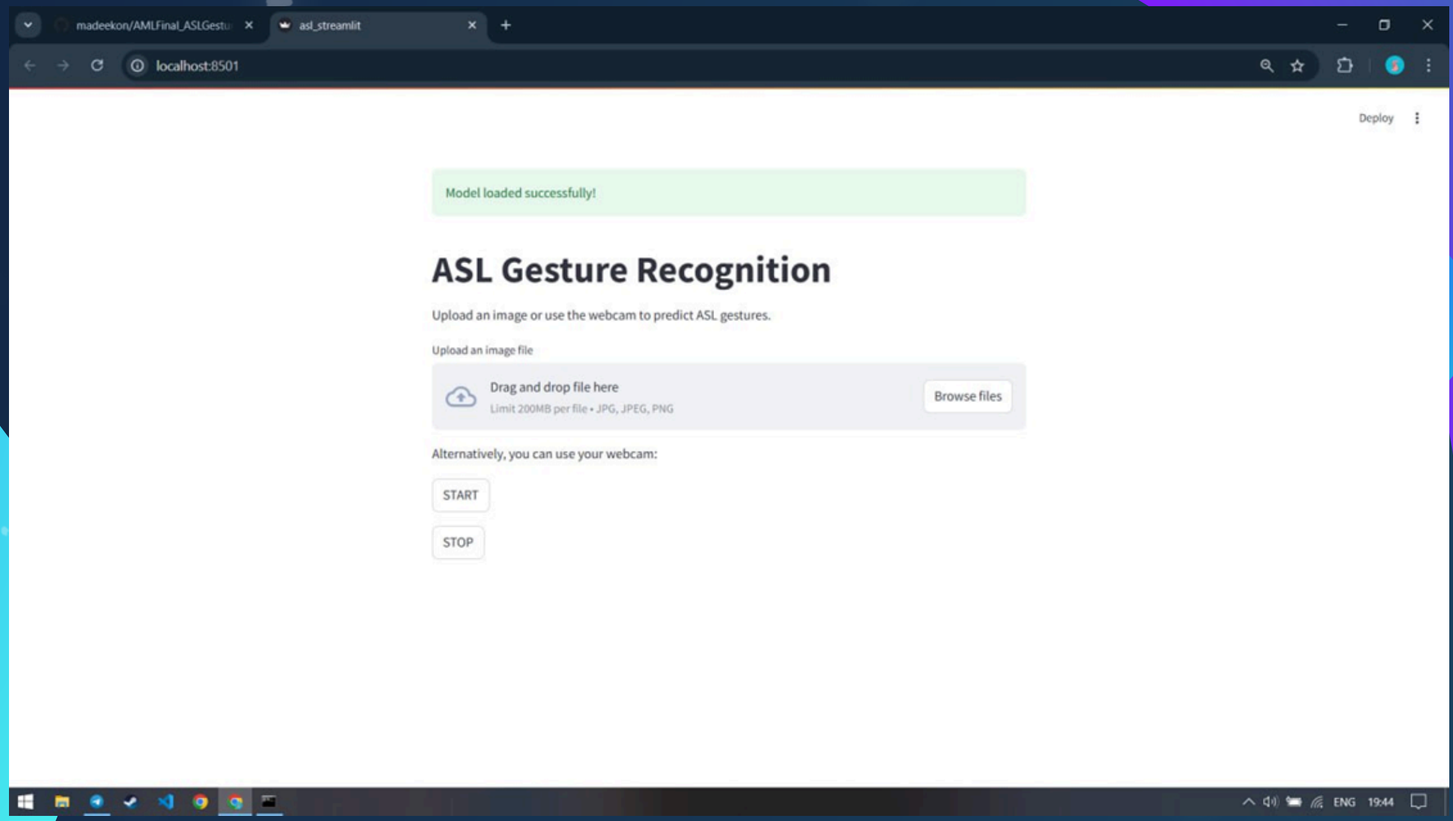


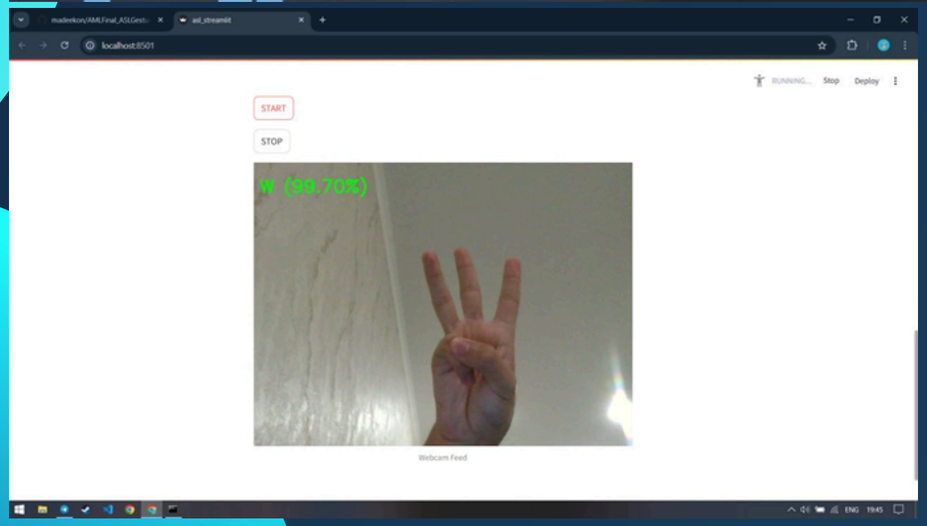
VISUALIZE
PREDICTIONS

```python
# Make predictions on the test data
predictions = model.predict(test_generator, verbose=1)  # Predict using the test generator
predicted_labels = np.argmax(predictions, axis=1)

# Map the class indices from the data generator
index_to_class = {v: k for k, v in train_generator.class_indices.items()}

# Extract the test images and true labels from the generator
test_images, test_labels = next(test_generator)  # Get all data in a single batch

# Visualize predictions
plt.figure(figsize=(12, 12))
num_images = min(9, len(test_images))  # Show up to 9 images or fewer if not available

for i in range(num_images):
    image = test_images[i]
    true_label = test_labels[i]
    predicted_label = predicted_labels[i]

    # Plot the image
    plt.subplot(3, 3, i + 1)
    plt.imshow(image)
    plt.title(f"True: {index_to_class.get(true_label, 'Unknown')}, Pred: {index_to_class.get(predicted_label, 'Unknown')}")
    plt.axis('off')

plt.tight_layout()
plt.show()
```

STREAMLIT
DEPLOYMENT







CONCLUSION

https://github.com/madeekon/AMLFinal_ASLGesture.git