wasla_digit_recognizer - output

May 12, 2025

#

WASLA - Deaf and Dumb Community

[ ]:
```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import os
import warnings
```

[ ]:
```
# Dataset Path
dataset_path = "ASL - Digits Dataset/Dataset"

if not os.path.exists(dataset_path):
    raise FileNotFoundError(f"Dataset path {dataset_path} does not exist.␣
 ↪Please check the directory.")
```

[ ]:
```
# Loading, Splitting and Preprocessing data.
train_dataset = tf.keras.preprocessing.image_dataset_from_directory(
    dataset_path,
    validation_split=0.3,
    subset="training",
    seed=42,
    image_size=(64, 64),
    batch_size=32
)

test_dataset = tf.keras.preprocessing.image_dataset_from_directory(
    dataset_path,
    validation_split=0.3,
    subset="validation",
    seed=42,
    image_size=(64, 64),
    batch_size=32
)

# Normalize pixel values to 0-1
train_dataset = train_dataset.map(lambda x, y: (x / 255.0, y))
```

```
test_dataset = test_dataset.map(lambda x, y: (x / 255.0, y))
```

```
Found 2062 files belonging to 10 classes.
Using 1444 files for training.
Found 2062 files belonging to 10 classes.
Using 618 files for validation.
```

[4]:
```
# Create the CNN model
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(64, 64,␣
 ↪3)),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(10, activation='softmax')
])
warnings.filterwarnings(action='ignore', category=UserWarning)
```

```
c:\Users\umarh\AppData\Local\Programs\Python\Python310\lib\site-
packages\keras\src\layers\convolutional\base_conv.py:107: UserWarning: Do not
pass an `input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in the model
instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

[5]:
```
# Compile the model
model.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)
```

[6]:
```
# Train the model
history = model.fit(
    train_dataset,
    validation_data=test_dataset,
    epochs=30,
    verbose=1
)
```

```
Epoch 1/30

46/46              30s 520ms/step -
accuracy: 0.1441 - loss: 2.3069 - val_accuracy: 0.3900 - val_loss: 2.1178
```

```
Epoch 2/30
46/46             28s 251ms/step -
accuracy: 0.3425 - loss: 1.8568 - val_accuracy: 0.6861 - val_loss: 0.9223
Epoch 3/30
46/46             11s 232ms/step -
accuracy: 0.5871 - loss: 1.2029 - val_accuracy: 0.8123 - val_loss: 0.5869
Epoch 4/30
46/46             14s 293ms/step -
accuracy: 0.7227 - loss: 0.7936 - val_accuracy: 0.8495 - val_loss: 0.4849
Epoch 5/30
46/46             10s 208ms/step -
accuracy: 0.7954 - loss: 0.6418 - val_accuracy: 0.9094 - val_loss: 0.3329
Epoch 6/30
46/46             9s 194ms/step -
accuracy: 0.8191 - loss: 0.5221 - val_accuracy: 0.8981 - val_loss: 0.2886
Epoch 7/30
46/46             10s 224ms/step -
accuracy: 0.8752 - loss: 0.3943 - val_accuracy: 0.9304 - val_loss: 0.2405
Epoch 8/30
46/46             13s 277ms/step -
accuracy: 0.8933 - loss: 0.3236 - val_accuracy: 0.9239 - val_loss: 0.2256
Epoch 9/30
46/46             18s 222ms/step -
accuracy: 0.9046 - loss: 0.2835 - val_accuracy: 0.9256 - val_loss: 0.2107
Epoch 10/30
46/46             19s 187ms/step -
accuracy: 0.9053 - loss: 0.2836 - val_accuracy: 0.9353 - val_loss: 0.1909
Epoch 11/30
46/46             10s 209ms/step -
accuracy: 0.9333 - loss: 0.2091 - val_accuracy: 0.9515 - val_loss: 0.1724
Epoch 12/30
46/46             9s 201ms/step -
accuracy: 0.9379 - loss: 0.1842 - val_accuracy: 0.9531 - val_loss: 0.1858
Epoch 13/30
46/46             9s 202ms/step -
accuracy: 0.9509 - loss: 0.1675 - val_accuracy: 0.9385 - val_loss: 0.1973
Epoch 14/30
46/46             13s 247ms/step -
accuracy: 0.9499 - loss: 0.1636 - val_accuracy: 0.9547 - val_loss: 0.1658
Epoch 15/30
46/46             11s 243ms/step -
accuracy: 0.9324 - loss: 0.2053 - val_accuracy: 0.9547 - val_loss: 0.1485
Epoch 16/30
46/46             11s 228ms/step -
accuracy: 0.9426 - loss: 0.1819 - val_accuracy: 0.9434 - val_loss: 0.2006
Epoch 17/30
46/46             10s 208ms/step -
accuracy: 0.9528 - loss: 0.1362 - val_accuracy: 0.9466 - val_loss: 0.1749
```

```
Epoch 18/30
46/46             12s 231ms/step -
accuracy: 0.9687 - loss: 0.0852 - val_accuracy: 0.9498 - val_loss: 0.1945
Epoch 19/30
46/46             10s 222ms/step -
accuracy: 0.9736 - loss: 0.0857 - val_accuracy: 0.9450 - val_loss: 0.2035
Epoch 20/30
46/46             9s 185ms/step -
accuracy: 0.9724 - loss: 0.0945 - val_accuracy: 0.9482 - val_loss: 0.1954
Epoch 21/30
46/46             11s 226ms/step -
accuracy: 0.9557 - loss: 0.1149 - val_accuracy: 0.9595 - val_loss: 0.1620
Epoch 22/30
46/46             9s 198ms/step -
accuracy: 0.9791 - loss: 0.0667 - val_accuracy: 0.9450 - val_loss: 0.2371
Epoch 23/30
46/46             10s 219ms/step -
accuracy: 0.9760 - loss: 0.0640 - val_accuracy: 0.9482 - val_loss: 0.1920
Epoch 24/30
46/46             9s 198ms/step -
accuracy: 0.9758 - loss: 0.0752 - val_accuracy: 0.9612 - val_loss: 0.1584
Epoch 25/30
46/46             11s 233ms/step -
accuracy: 0.9735 - loss: 0.0731 - val_accuracy: 0.9628 - val_loss: 0.1746
Epoch 26/30
46/46             23s 275ms/step -
accuracy: 0.9745 - loss: 0.0641 - val_accuracy: 0.9563 - val_loss: 0.1737
Epoch 27/30
46/46             19s 241ms/step -
accuracy: 0.9792 - loss: 0.0653 - val_accuracy: 0.9628 - val_loss: 0.1702
Epoch 28/30
46/46             9s 188ms/step -
accuracy: 0.9796 - loss: 0.0700 - val_accuracy: 0.9498 - val_loss: 0.1877
Epoch 29/30
46/46             10s 186ms/step -
accuracy: 0.9681 - loss: 0.0852 - val_accuracy: 0.9563 - val_loss: 0.1792
Epoch 30/30
46/46             9s 185ms/step -
accuracy: 0.9822 - loss: 0.0606 - val_accuracy: 0.9515 - val_loss: 0.2235
```

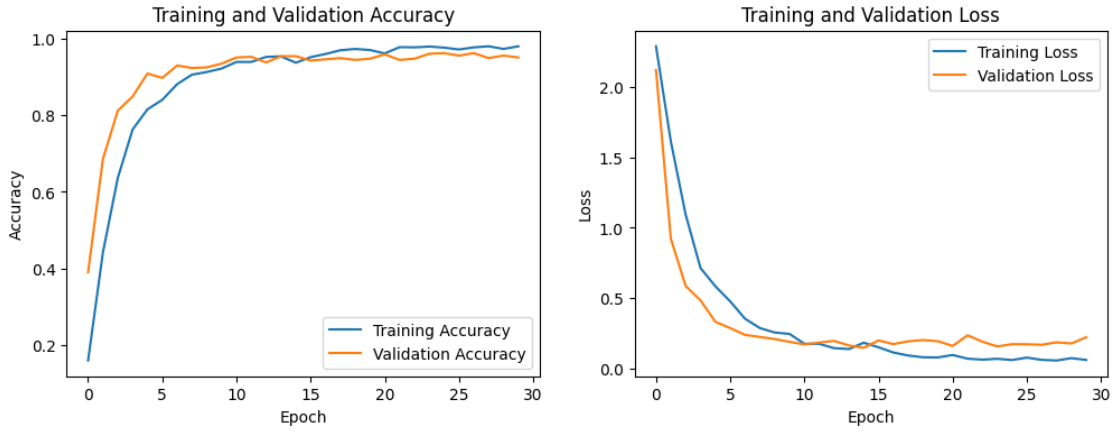
[7]:
```
# Evaluate the model on the test set
test_loss, test_accuracy = model.evaluate(test_dataset, verbose=0)
print(f"Test Accuracy: {test_accuracy * 100:.2f}%")
```

```
Test Accuracy: 95.15%
```

[ ]:
```
# Plot training and validation accuracy
plt.figure(figsize=(12, 4))
```

```
# Accuracy plot
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

# Loss plot
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



[ ]:
```
# Save the model
model.save("models/sign_language_model.h5")
```

#

Visualizing the Results on Test Images

[ ]:
```
# Initiailizing Empty Lists
all_images = []
all_labels = []
all_predictions = []
```

```
# Iteration over entire dataset.
for images, labels in test_dataset:
    all_images.append(images.numpy())
    all_labels.append(labels.numpy())
    # Predict for the current batch
    predictions = model.predict(images, verbose=0)
    predicted_labels = np.argmax(predictions, axis=1)
    all_predictions.append(predicted_labels)

# Joining all batches
all_images = np.concatenate(all_images, axis=0)
all_labels = np.concatenate(all_labels, axis=0)
all_predictions = np.concatenate(all_predictions, axis=0)

# Showing Stats
print(f"Total images processed: {len(all_labels)}")
print(f"First 10 true labels: {all_labels[:10]}")
print(f"First 10 predicted labels: {all_predictions[:10]}")
```

```
Total images processed: 618
First 10 true labels: [8 6 9 2 3 0 0 2 7 8]
First 10 predicted labels: [8 6 9 2 3 0 0 2 7 8]
```

[ ]:
```
# Visualize 25 images out of 32
num_images_to_show = 25
plt.figure(figsize=(10, 10))
for i in range(min(num_images_to_show, len(all_labels))):
    plt.subplot(5, 5, i + 1)
    plt.imshow(all_images[i])
    plt.title(f"Label: {all_labels[i]}\nPred: {all_predictions[i]}")
    plt.axis('off')
plt.tight_layout()
plt.show()
```

#

"The best way to predict the future is to invent it." - Alan Kay