Sign Language Recognition using CNN Model

Rachit Bhalla (21BAI1869)

Natanya Modi (21BAI1405)

Dataset Used: https://www.kaggle.com/datasets/datamunge/sign-language-mnist

Code

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam, RMSprop
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Loading the MNIST Sign Language Dataset from Kaggle
train_data = pd.read_csv('/kaggle/input/sign-language-mnist/sign_mnist_train/sign_mnist_train.csv')
test_data = pd.read_csv('/kaggle/input/sign-language-mnist/sign_mnist_test/sign_mnist_test.csv')

# Reshaping the image data into the appropriate format
train_images = np.array(train_data.drop(['label'], axis=1)).reshape(-1, 28, 28, 1)
train_labels = np.array(train_data['label'])
test_images = np.array(test_data.drop(['label'], axis=1)).reshape(-1, 28, 28, 1)
test_labels = np.array(test_data['label'])

# Normalizing the pixel values
train_images = train_images / 255.0
```

```
test_images = test_images / 255.0

# Defining the CNN model architecture
model1 = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(26, activation='softmax')
])
model2 = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(26, activation='softmax')
])

# Compiling the model using 2 optimizers
```

```
model1.compile(optimizer=Adam(learning_rate=0.001), loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

history_adam = model1.fit(train_images, train_labels, epochs=10, validation_data=(test_images,
test_labels))


model2.compile(optimizer=RMSprop(learning_rate=0.001),
loss='sparse_categorical_crossentropy', metrics=['accuracy'])

history_rmsprop = model2.fit(train_images, train_labels, epochs=10,
validation_data=(test_images, test_labels))


# Plot the accuracy and loss curves for all 2 optimizers

plt.plot(history_adam.history['accuracy'], label='Adam')

plt.plot(history_rmsprop.history['accuracy'], label='RMSprop')

plt.title('Accuracy vs Epoch')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.legend()

plt.show()


plt.plot(history_adam.history['loss'], label='Adam')

plt.plot(history_rmsprop.history['loss'], label='RMSprop')

plt.title('Loss vs Epoch')

plt.xlabel('Epoch')

plt.ylabel('Loss')

plt.legend()

plt.show()
```

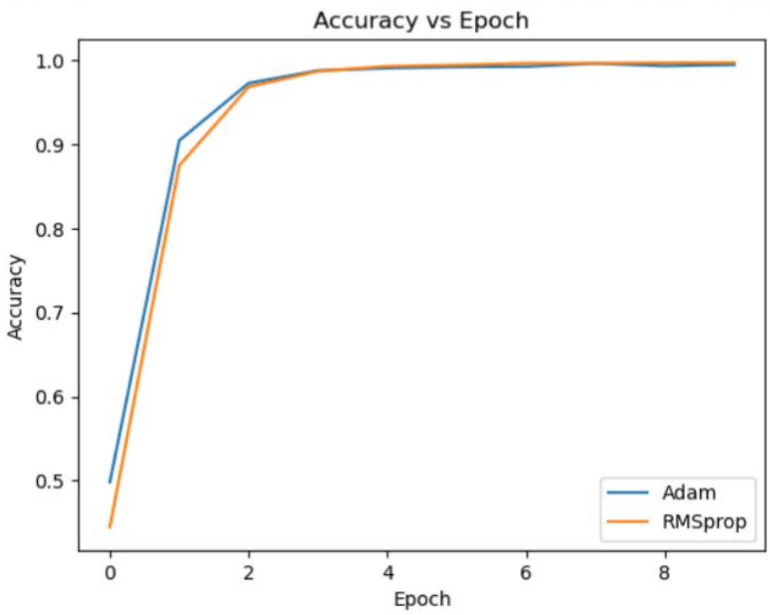
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam, RMSprop
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Load the ASL Alphabet dataset from Kaggle
train_data = pd.read_csv('/kaggle/input/sign-language-mnist/sign_mnist_train/sign_mnist_train.csv')
test_data = pd.read_csv('/kaggle/input/sign-language-mnist/sign_mnist_test/sign_mnist_test.csv')

# Reshape the image data into the appropriate format
train_images = np.array(train_data.drop(['label'], axis=1)).reshape(-1, 28, 28, 1)
train_labels = np.array(train_data['label'])
test_images = np.array(test_data.drop(['label'], axis=1)).reshape(-1, 28, 28, 1)
test_labels = np.array(test_data['label'])

# Normalize the pixel values
train_images = train_images / 255.0
test_images = test_images / 255.0

# Define the CNN model architecture
model1 = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D((2, 2)),
```

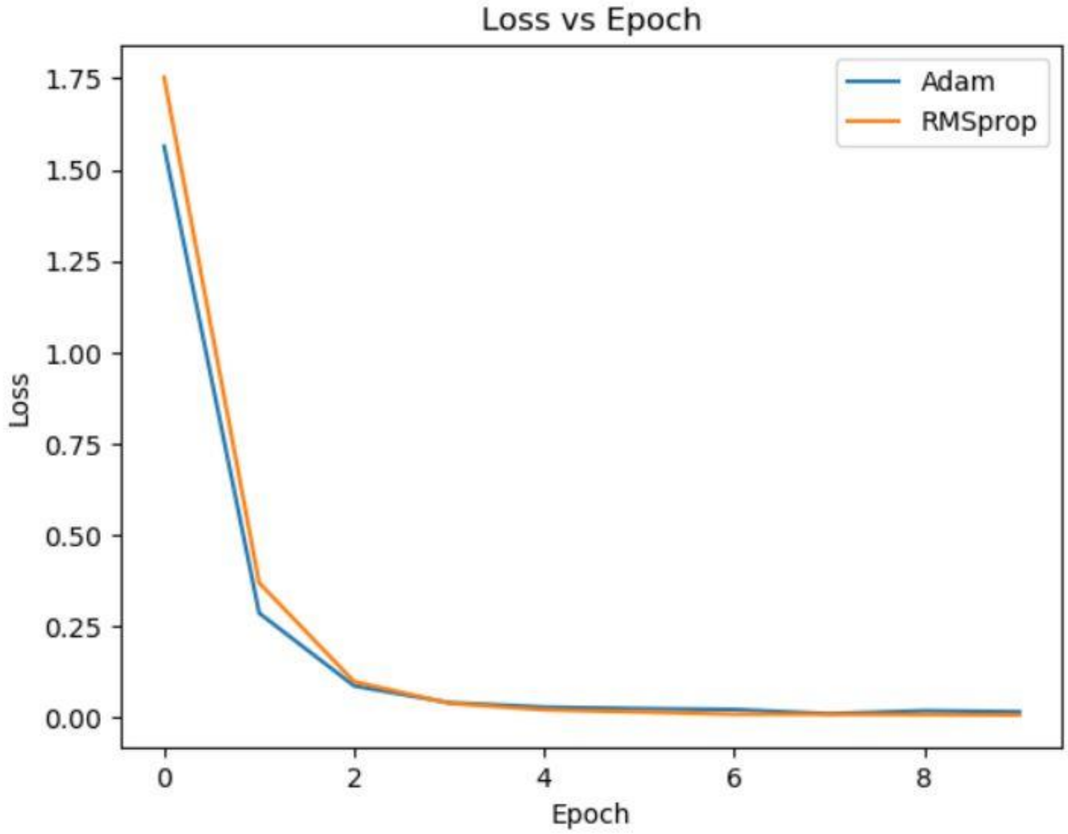
```python
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(26, activation='softmax')
])
model2 = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(26, activation='softmax')
])

# Compile the model using three different optimizers

model1.compile(optimizer=Adam(learning_rate=0.001), loss='sparse_categorical_crossentropy', metrics=['accuracy'])
history_adam = model1.fit(train_images, train_labels, epochs=10, validation_data=(test_images, test_labels))

model2.compile(optimizer=RMSprop(learning_rate=0.001), loss='sparse_categorical_crossentropy', metrics=['accuracy'])
history_rmsprop = model2.fit(train_images, train_labels, epochs=10, validation_data=(test_images, test_labels))
```

```python
# Plot the accuracy and loss curves for all three optimizers
plt.plot(history_adam.history['accuracy'], label='Adam')
plt.plot(history_rmsprop.history['accuracy'], label='RMSprop')
plt.title('Accuracy vs Epoch')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

plt.plot(history_adam.history['loss'], label='Adam')
plt.plot(history_rmsprop.history['loss'], label='RMSprop')
plt.title('Loss vs Epoch')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

```
Epoch 1/10
858/858 [==============================] - 6s 6ms/step - loss: 1.5638 - accuracy: 0.4981 - val_loss: 0.6506 - val_accuracy: 0.7670
Epoch 2/10
858/858 [==============================] - 5s 5ms/step - loss: 0.2866 - accuracy: 0.9048 - val_loss: 0.4458 - val_accuracy: 0.8564
Epoch 3/10
858/858 [==============================] - 5s 6ms/step - loss: 0.0873 - accuracy: 0.9731 - val_loss: 0.3838 - val_accuracy: 0.8872
Epoch 4/10
858/858 [==============================] - 5s 5ms/step - loss: 0.0416 - accuracy: 0.9881 - val_loss: 0.2806 - val_accuracy: 0.9177
Epoch 5/10
858/858 [==============================] - 5s 5ms/step - loss: 0.0294 - accuracy: 0.9911 - val_loss: 0.3003 - val_accuracy: 0.9130
Epoch 6/10
858/858 [==============================] - 4s 5ms/step - loss: 0.0254 - accuracy: 0.9925 - val_loss: 0.2936 - val_accuracy: 0.9304
Epoch 7/10
858/858 [==============================] - 4s 5ms/step - loss: 0.0228 - accuracy: 0.9929 - val_loss: 0.5172 - val_accuracy: 0.8995
Epoch 8/10
858/858 [==============================] - 4s 5ms/step - loss: 0.0112 - accuracy: 0.9966 - val_loss: 0.3946 - val_accuracy: 0.9194
Epoch 9/10
858/858 [==============================] - 4s 5ms/step - loss: 0.0194 - accuracy: 0.9937 - val_loss: 0.2885 - val_accuracy: 0.9293
Epoch 10/10
858/858 [==============================] - 5s 6ms/step - loss: 0.0167 - accuracy: 0.9951 - val_loss: 0.3381 - val_accuracy: 0.9254
Epoch 1/10
858/858 [==============================] - 6s 6ms/step - loss: 1.7526 - accuracy: 0.4447 - val_loss: 0.7122 - val_accuracy: 0.7782
Epoch 2/10
858/858 [==============================] - 4s 5ms/step - loss: 0.3694 - accuracy: 0.8751 - val_loss: 0.4870 - val_accuracy: 0.8554
Epoch 3/10
858/858 [==============================] - 5s 5ms/step - loss: 0.0990 - accuracy: 0.9687 - val_loss: 0.3945 - val_accuracy: 0.8783
Epoch 4/10
858/858 [==============================] - 4s 5ms/step - loss: 0.0399 - accuracy: 0.9874 - val_loss: 0.4164 - val_accuracy: 0.9071
Epoch 5/10
858/858 [==============================] - 4s 5ms/step - loss: 0.0221 - accuracy: 0.9932 - val_loss: 0.3323 - val_accuracy: 0.9232
Epoch 6/10
858/858 [==============================] - 5s 6ms/step - loss: 0.0162 - accuracy: 0.9946 - val_loss: 0.5121 - val_accuracy: 0.8829
Epoch 7/10
858/858 [==============================] - 4s 5ms/step - loss: 0.0100 - accuracy: 0.9968 - val_loss: 0.3644 - val_accuracy: 0.9237
Epoch 8/10
```

```
Epoch 8/10
858/858 [==============================] - 4s 5ms/step - loss: 0.0098 - accuracy: 0.9968 - val_loss: 0.4517 - val_accuracy: 0.9346
Epoch 9/10
858/858 [==============================] - 4s 5ms/step - loss: 0.0088 - accuracy: 0.9973 - val_loss: 0.3817 - val_accuracy: 0.9304
Epoch 10/10
858/858 [==============================] - 4s 5ms/step - loss: 0.0078 - accuracy: 0.9976 - val_loss: 0.4883 - val_accuracy: 0.9221
```




Loss vs Epoch
Adam
RMSprop
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Loss
0
2
4
6
8
Epoch