Developing a Convolutional Neural Network (CNN) Model for Accurate House Number Classification in the SVHN Dataset: Importance and Applications

Project Description

The main goal of this project is to develop a CNN model capable of accurately classifying house numbers in the Street View House Numbers (SVHN) dataset. The SVHN dataset consists of images containing house numbers from real-world scenarios, such as street addresses, storefronts, and building exteriors.

Data Description

Key Features:

1. Image Format: The dataset consists of RGB images in a variety of sizes and aspect ratios. Each image represents a real-world scene containing one or more house numbers.
2. Labeling: The dataset provides accurate bounding box information and label annotations for the digits present in each image. This information allows for precise digit localization and classification.
3. Multi-digit Numbers: The SVHN dataset includes images with single-digit numbers as well as those with multiple digits. The dataset provides information about the location, order, and labeling of each digit within a multi-digit number.
4. Large-Scale Dataset: The SVHN dataset is relatively large, containing tens of thousands of labeled images for training, validation, and testing. It offers a diverse range of real-world scenarios, capturing variations in fonts, colors, backgrounds, and digit appearance.
5. Training, Validation, and Testing Sets: The dataset is typically divided into three separate sets: a training set used to train the model, a validation set used for hyperparameter tuning and model selection, and a testing set for evaluating the model's performance on unseen data.
6. Class Distribution: The SVHN dataset is typically labeled with the digits 0-9 as the classes. The dataset's class distribution ensures a balance of examples for each digit, providing equal representation across the different digits.

The SVHN dataset serves as a valuable resource for developing and evaluating models for digit recognition, multi-digit classification, and address-related tasks. It offers a realistic and challenging dataset that reflects the complexity and variations encountered in real-world house number recognition scenarios.

```
import tensorflow as tf
import tensorflow_datasets as tfds

# Load the SVHN dataset
svhn_dataset, svhn_info = tfds.load('svhn_cropped', split='train', with_info=True)

# Split the dataset into train and test sets
train_percentage = 0.8
num_train_examples = int(train_percentage * svhn_info.splits['train'].num_examples)

train_dataset = svhn_dataset.take(num_train_examples)
test_dataset = svhn_dataset.skip(num_train_examples)

# Extract x_train, y_train, x_test, y_test
x_train, y_train = [], []
x_test, y_test = [], []

for example in train_dataset:
    x_train.append(example['image'])
    y_train.append(example['label'])

for example in test_dataset:
    x_test.append(example['image'])
    y_test.append(example['label'])
```

```
Downloading and preparing dataset 1.47 GiB (download: 1.47 GiB, generated: Unknown size, total: 1.47 GiB) to /root/tenso
Dl Completed...: 100%     3/3 [00:26<00:00, 10.46s/ url]

Dl Size...: 100%     1501/1501 [00:25<00:00, 50.81 MiB/s]
Dataset svhn_cropped downloaded and prepared to /root/tensorflow_datasets/svhn_cropped/3.0.0. Subsequent calls will reus
```

Visualizing the images

```
import matplotlib.pyplot as plt
# Extract a sample image and its label
# Extract 10 sample images and labels
sample_dataset = svhn_dataset.take(100)

# Display the images
fig, axes = plt.subplots(10, 10, figsize=(12, 6))
axes = axes.flatten()

for i, sample in enumerate(sample_dataset):
    image = sample['image']
    label = sample['label']

    # Convert image to numpy array
    image = image.numpy()

    # Display the image
    axes[i].imshow(image)
    axes[i].set_title(f"Label: {label}")
    axes[i].axis('off')

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

Split the dataset to train, test data and preprocessing the data

```
# Convert the lists to TensorFlow tensors
x_train = tf.stack(x_train)
y_train = tf.stack(y_train)
x_test = tf.stack(x_test)
y_test = tf.stack(y_test)

# Convert x_train and x_test to float32
x_train = tf.cast(x_train, tf.float32)
x_test = tf.cast(x_test, tf.float32)

# Normalize pixel values
x_train = x_train / 255.0
x_test = x_test / 255.0
```

```
x_train.shape

    TensorShape([58605, 32, 32, 3])
```

```
x_test.shape

    TensorShape([14652, 32, 32, 3])
```

```
import numpy as np
unique_values = np.unique(y_train.numpy())
num_unique_values = len(unique_values)
print(num_unique_values)

    10
```

```
from tensorflow.keras.utils import to_categorical

# Convert y_train to one-hot encoded format
num_classes = 10
y_train_encoded =to_categorical(y_train, num_classes)
```

```
y_test_encoded = to_categorical(y_test, num_classes)
```

```
y_train_encoded.shape

    (58605, 10)
```

```
import tensorflow.keras as keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Flatten, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import TensorBoard
```

```
cnn_model = Sequential()

# Convolutional layers
cnn_model.add(Conv2D(filters=32, kernel_size=(3, 3), activation='relu', input_shape=(32, 32, 3)))
cnn_model.add(Conv2D(filters=32, kernel_size=(3, 3), activation='relu'))
cnn_model.add(MaxPooling2D(pool_size=(2, 2)))
cnn_model.add(Dropout(0.3))

cnn_model.add(Conv2D(filters=32, kernel_size=(3, 3), activation='relu'))
cnn_model.add(Conv2D(filters=32, kernel_size=(3, 3), activation='relu'))
cnn_model.add(MaxPooling2D(pool_size=(2, 2)))
cnn_model.add(Dropout(0.3))

# Flatten the output from the previous layer
cnn_model.add(Flatten())

# Dense (fully connected) layers
cnn_model.add(Dense(512, activation='relu'))
cnn_model.add(Dense(512, activation='relu'))

# Output layer
cnn_model.add(Dense(10, activation='softmax'))
```

Double-click (or enter) to edit

```
cnn_model.compile(loss="categorical_crossentropy", optimizer='adam', metrics=["accuracy"])
```

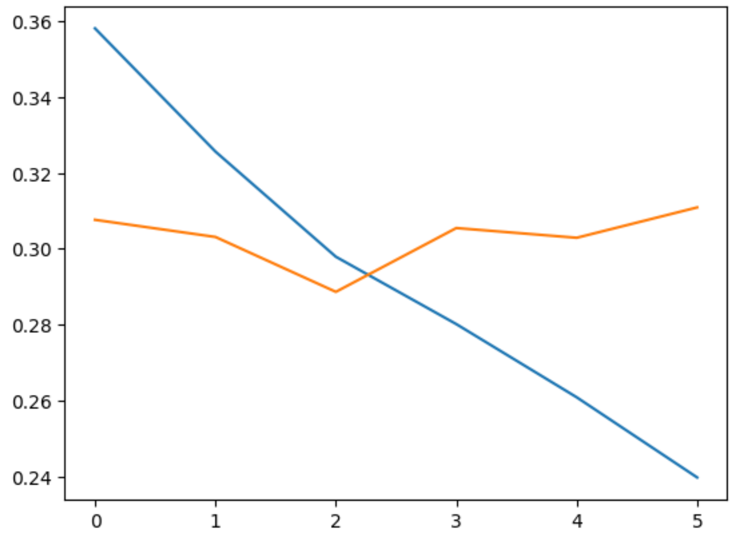
```
history = cnn_model.fit(x_train, y_train_encoded, epochs = 6, batch_size = 32, shuffle = True, validation_data=[x_test, y_test_enc

    Epoch 1/6
    1832/1832 [==============================] - 216s 118ms/step - loss: 0.3581 - accuracy: 0.8888 - val_loss: 0.3077 - val_accu
    Epoch 2/6
    1832/1832 [==============================] - 216s 118ms/step - loss: 0.3257 - accuracy: 0.8984 - val_loss: 0.3032 - val_accu
    Epoch 3/6
    1832/1832 [==============================] - 222s 121ms/step - loss: 0.2979 - accuracy: 0.9069 - val_loss: 0.2887 - val_accu
    Epoch 4/6
    1832/1832 [==============================] - 212s 116ms/step - loss: 0.2802 - accuracy: 0.9125 - val_loss: 0.3055 - val_accu
    Epoch 5/6
    1832/1832 [==============================] - 230s 126ms/step - loss: 0.2609 - accuracy: 0.9192 - val_loss: 0.3029 - val_accu
    Epoch 6/6
    1832/1832 [==============================] - 213s 117ms/step - loss: 0.2398 - accuracy: 0.9250 - val_loss: 0.3109 - val_accu
```

```
prediction = cnn_model.predict(x_test)
prediction = np.argmax(prediction, axis =1)
```

```
    458/458 [==============================] - 12s 25ms/step
```

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.ylabel("")
```

```
    Text(0, 0.5, '')
```

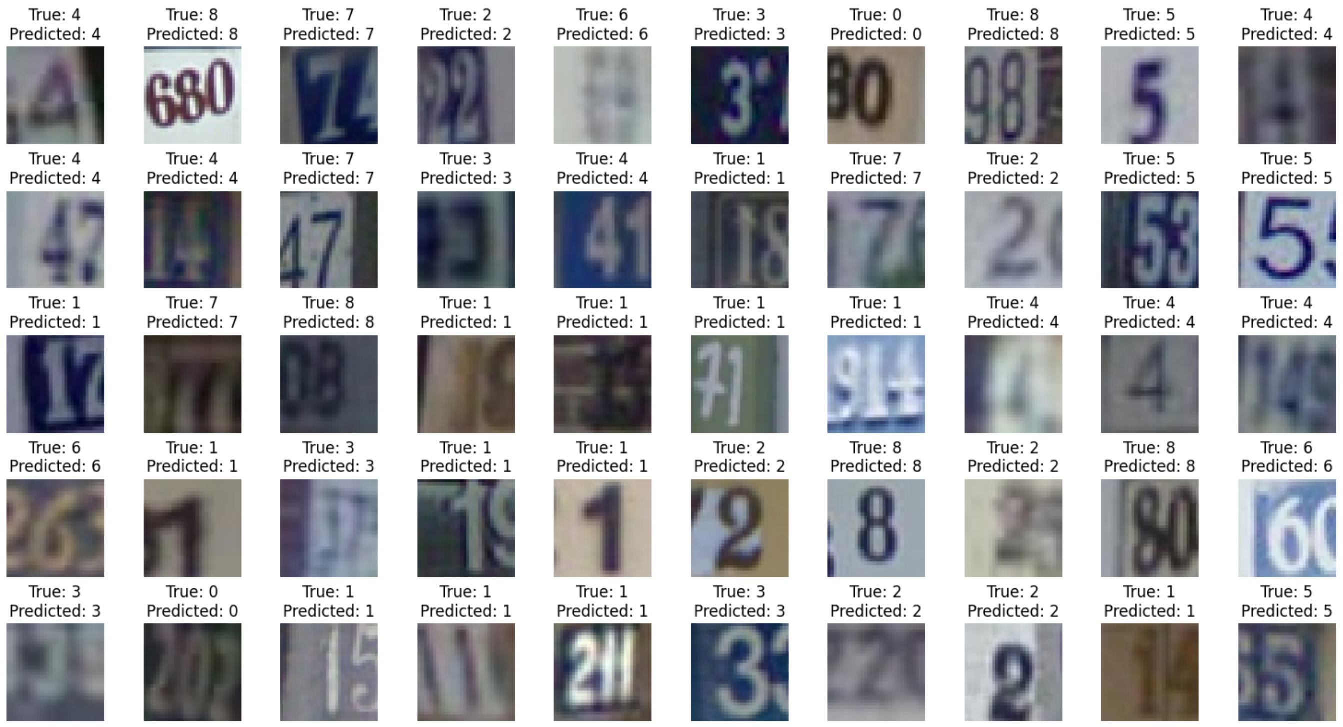


```
from sklearn.metrics import classification_report
print(classification_report(prediction, y_test))
```

```
                  precision    recall  f1-score   support

               0       0.91      0.90      0.91      1024
               1       0.93      0.94      0.93      2766
               2       0.93      0.94      0.94      2137
               3       0.91      0.88      0.90      1710
               4       0.92      0.92      0.92      1471
               5       0.90      0.93      0.91      1370
               6       0.93      0.86      0.89      1263
               7       0.92      0.89      0.91      1095
               8       0.87      0.89      0.88       984
               9       0.86      0.94      0.90       832

        accuracy                           0.91     14652
       macro avg       0.91      0.91      0.91     14652
    weighted avg       0.91      0.91      0.91     14652
```

```
# Extract a sample of 50 images and labels
sample_dataset = svhn_dataset.take(50)

# Make predictions on the sample dataset
predictions = cnn_model.predict(x_train[:50])
predicted_labels = [np.argmax(pred) for pred in predictions]

# Display the images with their true labels and predicted labels
fig, axes = plt.subplots(5, 10, figsize=(15, 8))

for i, sample in enumerate(sample_dataset):
    image = sample['image']
    true_label = sample['label']
    predicted_label = predicted_labels[i]
    # Convert image to numpy array
    image = image.numpy()

    # Display the image with true label and predicted label
    axes[i // 10, i % 10].imshow(image)
    axes[i // 10, i % 10].set_title(f"True: {true_label}\nPredicted: {predicted_label}")
    axes[i // 10, i % 10].axis('off')

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

```
2/2 [==============================] - 0s 17ms/step
```



✓ 7s completed at 10:50 AM