Analisis dan Desain Pipeline Pembuatan Dataset TFRecord untuk HTR

Berdasarkan skrip `create_tfrecord_fixed.py`

October 6, 2025

1 Pendahuluan

Efisiensi dalam pemuatan data merupakan faktor krusial dalam siklus pengembangan model *deep learning*, terutama untuk tugas yang melibatkan data dalam jumlah besar seperti *Handwritten Text Recognition* (HTR). Proses membaca ribuan file gambar individual dari disk pada setiap epoch dapat menjadi hambatan I/O yang signifikan, memperlambat proses training dan mengurangi utilisasi GPU.

Untuk mengatasi masalah ini, format `TFRecord` dari TensorFlow diadopsi. `TFRecord` adalah format biner sederhana yang dirancang untuk menyimpan sekuens data. Format ini memungkinkan data untuk dibaca secara efisien, mendukung paralelisasi, dan dapat diintegrasikan dengan mulus ke dalam pipeline `tf.data`, sehingga mengoptimalkan alur data dari penyimpanan ke unit pemrosesan.

Dokumen ini menguraikan analisis dan desain dari pipeline pembuatan dataset `TFRecord` yang diimplementasikan dalam skrip `create_tfrecord_fixed.py`. Tujuannya adalah untuk mengonversi dataset gambar baris teks dan label transkripsinya menjadi satu file `TFRecord` yang terstruktur, terstandarisasi, dan siap pakai untuk melatih model HTR.

2 Desain Pipeline

Pipeline konversi data dirancang sebagai proses sekuensial yang sistematis, dimulai dari pemuatan data mentah hingga serialisasi ke dalam format `TFRecord`.

2.1 Input Pipeline

Pipeline ini memerlukan tiga jenis input utama:

1. **Direktori Gambar (`images_dir`):** Sebuah direktori yang berisi file-file gambar baris teks dalam format yang dapat dibaca oleh library Pillow (misalnya, PNG, JPG). Setiap gambar merepresentasikan satu sampel data.
2. **File Label (`labels_file`):** Sebuah file teks (`.txt`) di mana setiap baris berisi nama file gambar dan transkripsi teksnya, dipisahkan oleh spasi. Formatnya adalah: `<nama_file> <teks_transkripsi>`.
3. **File Charset (`charset`):** Sebuah file teks yang mendefinisikan kosakata karakter yang akan dikenali oleh model. Setiap baris berisi satu karakter. Karakter yang tidak ada dalam daftar ini akan dipetakan ke token "tidak diketahui" (unknown).

2.2 Tahapan Pemrosesan

Setiap pasangan gambar dan label dari input diproses melalui serangkaian langkah berikut sebelum ditulis ke file `TFRecord`.

2.2.1 Pemuatan Charset dan Pemetaan Karakter

Langkah pertama adalah membangun mekanisme untuk mengonversi label teks (string) menjadi sekuens numerik (integer).

- **Mekanisme:** Sebuah `tf.lookup.StaticHashTable` dibuat. Tabel ini memetakan setiap karakter dari file charset ke sebuah ID integer unik, dimulai dari 1.
- **Penanganan Karakter Tidak Dikenal:** Nilai *default* (0) ditetapkan untuk setiap karakter yang tidak ditemukan dalam charset. Ini memastikan bahwa model dapat menangani karakter di luar kosakata tanpa menyebabkan error.

2.2.2 Pra-pemrosesan Gambar

Setiap gambar menjalani serangkaian transformasi untuk standardisasi dan normalisasi.

1. **Konversi Grayscale:** Gambar dibuka dan dikonversi ke mode *grayscale* ('L') untuk menyederhanakan data dan mengurangi kompleksitas komputasi.
2. **Pengubahan Ukuran (Resizing):** Ukuran setiap gambar diubah secara seragam menjadi dimensi target (misalnya, 1024x128 piksel) menggunakan metode interpolasi `LANCZOS` untuk menjaga kualitas visual.
3. **Normalisasi Nilai Piksel:** Gambar dikonversi menjadi array NumPy dengan tipe data `float32`, dan nilai pikselnya dinormalisasi ke rentang [0.0, 1.0] dengan membaginya dengan 255.0.
4. **Penambahan Dimensi Channel (FIX):** Ini adalah perbaikan krusial. Sebuah dimensi channel ditambahkan secara eksplisit di akhir array gambar (`np.expand_dims(img, axis=-1)`). Ini mengubah shape dari (H, W) menjadi (H, W, 1), sesuai dengan format input yang diharapkan oleh lapisan konvolusi TensorFlow.

2.2.3 Pra-pemrosesan Label

Label teks juga diproses untuk mengubahnya menjadi format tensor.

1. **Pemisahan Karakter:** String teks dipecah menjadi sekuens karakter individual menggunakan `tf.strings.unicode_s` untuk menangani karakter Unicode dengan benar.
2. **Konversi Numerik:** `StaticHashTable` yang telah dibuat sebelumnya digunakan untuk memetakan setiap karakter dalam sekuens ke ID numeriknya, menghasilkan sebuah tensor integer.

2.3 Struktur Data dan Serialisasi

Setelah diproses, data gambar dan label siap untuk diserialisasi.

2.3.1 Struktur Fitur `tf.train.Example`

Setiap sampel data (pasangan gambar-label) dikemas dalam sebuah proto `tf.train.Example`. Struktur fitur yang digunakan adalah sebagai berikut:

- `'image'`: Berisi tensor gambar yang telah diproses dan kemudian diserialisasi menjadi string byte menggunakan `tf.io.serialize_tensor`.
- `'label'`: Berisi tensor label numerik yang juga diserialisasi menjadi string byte.

Penggunaan `tf.io.serialize_tensor` merupakan pilihan desain yang modern dan fleksibel. Metode ini secara otomatis menyimpan informasi tipe data (*dtype*) dan bentuk (*shape*) dari tensor asli, sehingga proses deserialisasi (parsing) di sisi training menjadi lebih sederhana dan tidak rawan kesalahan.

```
def serialize_example(image, label):
    feature = {
        'image': _bytes_feature(tf.io.serialize_tensor(image)),
        'label': _bytes_feature(tf.io.serialize_tensor(label)),
    }
    example_proto = tf.train.Example(
        features=tf.train.Features(feature=feature)
    )
    return example_proto.SerializeToString()
```

Listing 1: Contoh fungsi serialisasi dalam skrip.

2.3.2 Penulisan ke File

Objek `tf.io.TFRecordWriter` digunakan untuk menulis setiap proto `tf.train.Example` yang telah diserialisasi ke dalam satu file output biner. Proses ini dilakukan secara iteratif untuk semua sampel data yang valid. Skrip juga menyertakan penanganan error untuk melewati file gambar yang tidak ditemukan atau data yang korup, sehingga memastikan integritas dataset final.

3 Desain Fungsi Parsing (Sisi Training)

Sebagai pelengkap dari proses pembuatan, fungsi parsing di sisi training dirancang untuk membaca dan merekonstruksi data dari file `TFRecord`.

```
def _parse_function(example_proto):
    feature_description = {
        'image': tf.io.FixedLenFeature([], tf.string),
        'label': tf.io.FixedLenFeature([], tf.string),
    }
    example = tf.io.parse_single_example(
        example_proto, feature_description
    )

    image = tf.io.parse_tensor(example['image'], out_type=tf.float32)
    label = tf.io.parse_tensor(example['label'], out_type=tf.int64)

    # Set shape jika diperlukan untuk optimasi graph
    image.set_shape([128, 1024, 1])

    return image, label
```

Listing 2: Contoh fungsi parsing untuk membaca TFRecord.

Fungsi ini membaca string byte dari fitur `'image'` dan `'label'`, lalu menggunakan `tf.io.parse_tensor` untuk merekonstruksinya kembali menjadi tensor dengan tipe data dan bentuk aslinya. Ini menunjukkan keunggulan dari metode serialisasi yang dipilih.

4 Kesimpulan

Pipeline yang diimplementasikan dalam `create_tfrecord_fixed.py` secara efektif mengubah dataset gambar dan teks mentah menjadi sebuah file `TFRecord` tunggal yang terstruktur dan efisien. Desain ini memberikan beberapa keuntungan kunci:

- **Efisiensi I/O:** Mengurangi waktu pemuatan data secara drastis selama training.
- **Standardisasi Data:** Memastikan semua gambar dan label memiliki format, ukuran, dan normalisasi yang konsisten.
- **Integritas Data:** Penggunaan `tf.io.serialize_tensor` menjaga keutuhan tipe data dan bentuk tensor, menyederhanakan proses parsing.
- **Robustness:** Penanganan error memastikan bahwa hanya data yang valid yang disertakan dalam dataset final.

Dengan demikian, pipeline ini menyediakan fondasi data yang solid dan andal untuk melatih model HTR dengan performa optimal.