**UJIAN AKHIR SEMESTER**

|  |  |  |
| --- | --- | --- |
| Nama Mahasiswa | : | Muhammad Ilham Ihsanudin |
| NPM | : | 3332180047 |
| Nama Mata Kuliah | : | Pengolahan Citra |
| Semester | : | Ganjil 2021/2022 |

**CLASSIFYING TRAFFIC SIGNS USING A DEEP LEARNING MODEL (WITH PYTORCH)**

**BAB I**

**PENDAHULUAN**

Rambu lalu lintas diatur menurut Peraturan Menteri Perhubungan Nomor 13 Tahun 2014 dan merupakan bagian dari perlengkapan lalu lintas jalan yang meliputi angka, huruf, frasa, lambang, dan/atau kombinasinya. Rambu lalu lintas digunakan sebagai larangan, peringatan, perintah dan petunjuk bagi setiap orang dalam lalu lintas. Menurut data Kementerian Perhubungan, di Indonesia terdapat lebih dari 300 rambu lalu lintas, termasuk berbagai jenis rambu seperti rambu peringatan, rambu larangan, rambu kontrol, dan rambu petunjuk.

Traffic Sign Recognition (TSR) merupakan teknologi yang dapat mengenali rambu lalu lintas menggunakan pengolahan citra. TSR telah diterapkan pada sistem bantuan pengemudi (DAS), sistem bantuan pengemudi canggih (ADAS), sistem mengemudi otonom (ADS), pemahaman perkotaan. atmosfer, keselamatan jalan dan pemantauan rambu-rambu untuk tujuan pemeliharaan. Output dari TSR dapat menjadi input untuk DAS, ADAS, ADS dan lain-lain. Keakuratan keluaran TSR akan sangat mempengaruhi keputusan yang dibuat oleh sistem tersebut. Pengambilan keputusan yang buruk dapat berdampak fatal dan paling buruk, kecelakaan lalu lintas yang fatal.

Beberapa pendekatan dan metode telah menggunakan perhitungan matriks komputasi yang rumit untuk memecahkan masalah deteksi dan pengenalan secara bersamaan. Awalnya, algoritma komputasi sederhana, seperti ambang warna, digunakan untuk deteksi. Klasifikasi kemudian dilakukan pada kandidat hasil deteksi dengan algoritme yang lebih rumit secara komputasi dan lebih akurat.

**BAB II**

**METODE**

Metode yang digunakan pada tugas akhir ini adalah Mengklasifikasikan Rambu Lalu Lintas menggunakan komputasi Deep Learning menggunakan pemograman python dengan library Pytorch. Pytorch menekankan fleksibilitas dan membangun model DeepLearning untuk mengekspresikan sintaks bahasa pemrograman Python. Pendekatan dan kemudahan penggunaan ini dimanfaatkan oleh pengguna awal komunitas riset, dan pada tahun-tahun sejak dirilis, library ini telah menjadi salah satu alat yang paling penting untuk menangani teori Deep Learning yang diterapkan secara luas.

Pytorch menyediakan struktur data dasar, yang disebut Tensor, sebuah array mutidimensional yang mirip dengan array yang disediakan di pustaka NumPy. Atas dasar ini, banyak fitur telah dirancang untuk memfasilitasi pengaturan dan implementasi proyek, atau untuk merancang dan melatih eksperimen pada arsitektur baru Neural Network (Jaringan Sarah Tiruan). Tensor dapat mempercepat operasi matematika (dengan asumsi perangkat keras dan perangkat lunak yang digunakan adalah teknologi terbaru) dan Pytorch memiliki alat untuk melakukan training secara terpisah (distributed training), process pekerja (worker process, untuk menggunakan beberapa sumber daya pada satu mesin) untuk efisiensi pemuatan data, dan library tambahan dari fungsi Deep Learning pada umumnya.

Pytorch, yang digunakan bersama dengan library Torchvision, memiliki lebih banyak kemampuan untuk melakukan perhitungan pembelajaran mendalam pada objek digital seperti gambar (2D) dan video (3D). Salah satu metode yang umum digunakan adalah Convolutional Neural Network (CNN) yang menggunakan Cortical Neuron dalam analisis citra. Torchvision sangat handal digunakan untuk pemodelan menggunakan konsep Convolutional Neural Network (CNN), dikarenakan dukungan pada arsitektur-arsitektur Convolutional Neural Network (CNN).

**BAB III**

**PEMBAHASAN**

Pertama-tama menentukan library perlu yang diimport agar listing berjalan tanpa adanya error. Setelah itu perlu untuk mendownload dataset yang digunakan, yakni Set Data Rambu Lalu Lintas Jerman (GTSRB). Kami akan menggunakan sekitar 34.800 gambar untuk dataset pelatihan, 12630 gambar untuk dataset pengujian dan 4410 gambar untuk dataset validasi. Dengan menjalankan perintah berikut.

**!**wget https://sid.erda.dk/public/archives/daaeac0d7ce1152aea9b61d9f1e19370/GTSRB\_Final\_Training\_Images.zip

**!**unzip -qq GTSRB\_Final\_Training\_Images.zip

Database tersebut akan secara otomatis masuk ke dalam library Google Drive yang nantinya dapat dipanggil kembali melalui Google Collab. Sebelum running perlu untuk mengganti runtime menggunakan GPU agar proses berjalan lebih cepat.

Lalu dengan menggunakan function tambahan seperti OpenCV dan Torchvision untuk memuat dan menampilkan gambar,

**def** load\_image(img\_path, resize**=True**):

img **=** cv2**.**cvtColor(cv2**.**imread(img\_path), cv2**.**COLOR\_BGR2RGB)

**if** resize:

img **=** cv2**.**resize(img, (64, 64), interpolation **=** cv2**.**INTER\_AREA)

**return** img

**def** show\_image(img\_path):

img **=** load\_image(img\_path)

plt**.**imshow(img)

plt**.**axis('off')

**def** show\_sign\_grid(image\_paths):

images **=** [load\_image(img) **for** img **in** image\_paths]

images **=** torch**.**as\_tensor(images)

images **=** images**.**permute(0, 3, 1, 2)

grid\_img **=** torchvision**.**utils**.**make\_grid(images, nrow**=**11)

plt**.**figure(figsize**=**(24, 12))

plt**.**imshow(grid\_img**.**permute(1, 2, 0))

plt**.**axis('off');

Lalu selanjutnya dilakukan membangun database dengan fokus pada klasifikasi beberapa rambu lalu lintas yang paling sering digunakan, dengan tiga class:

class\_names **=** ['priority\_road', 'give\_way', 'stop', 'no\_entry']

class\_indices **=** [12, 13, 14, 17]

Lalu menyalin file gambar ke direktori baru, sehingga mempermudah dataset Torchvision, dengan membagi direktornya sesuai dengan class masing-masing.

**!**rm -rf data

DATA\_DIR **=** Path('data')

DATASETS **=** ['train', 'val', 'test']

**for** ds **in** DATASETS:

**for** cls **in** class\_names:

(DATA\_DIR **/** ds **/** cls)**.**mkdir(parents**=True**, exist\_ok**=True**)

Lalu menerapkan beberapa teknik augmentasi gambar untuk meningkatkan ukuran dataset. Serta beberapa pengubahan ukuran acak, rotasi, dan pembalikan horizontal. Dan menormalkan tensor menggunakan nilai preset untuk setiap saluran

mean\_nums **=** [0.485, 0.456, 0.406]

std\_nums **=** [0.229, 0.224, 0.225]

transforms **=** {'train': T**.**Compose([

T**.**RandomResizedCrop(size**=**256),

T**.**RandomRotation(degrees**=**15),

T**.**RandomHorizontalFlip(),

T**.**ToTensor(),

T**.**Normalize(mean\_nums, std\_nums)

]), 'val': T**.**Compose([

T**.**Resize(size**=**256),

T**.**CenterCrop(size**=**224),

T**.**ToTensor(),

T**.**Normalize(mean\_nums, std\_nums)

]), 'test': T**.**Compose([

T**.**Resize(size**=**256),

T**.**CenterCrop(size**=**224),

T**.**ToTensor(),

T**.**Normalize(mean\_nums, std\_nums)

]),

}

Lalu membuat bebereapa data PyTorch untuk setiap kumpulan data dan pemuat data dengan menjalankan perintah berikut.

image\_datasets **=** {

d: ImageFolder(f'{DATA\_DIR}/{d}', transforms[d]) **for** d **in** DATASETS

}

data\_loaders **=** {

d: DataLoader(image\_datasets[d], batch\_size**=**4, shuffle**=True**, num\_workers**=**4)

**for** d **in** DATASETS

}

**BAB III**

**HASIL**

Untuk melihat hasil dari pengklasifikasian, perlu untuk menjalankan 3 fungsi pembantu untuk merangkum logika pelatihan dan evaluasi. Dengan menjalankan

**def** train\_epoch(

model,

data\_loader,

loss\_fn,

optimizer,

device,

scheduler,

n\_examples

):

model **=** model**.**train()

losses **=** []

correct\_predictions **=** 0

**for** inputs, labels **in** data\_loader:

inputs **=** inputs**.**to(device)

labels **=** labels**.**to(device)

outputs **=** model(inputs)

\_, preds **=** torch**.**max(outputs, dim**=**1)

loss **=** loss\_fn(outputs, labels)

correct\_predictions **+=** torch**.**sum(preds **==** labels)

losses**.**append(loss**.**item())

loss**.**backward()

optimizer**.**step()

optimizer**.**zero\_grad()

scheduler**.**step()

**return** correct\_predictions**.**double() **/** n\_examples, np**.**mean(losses)

Lalu mengubah model menjadi train mode dan memeriksa datanya. Setelah mendapatkan prediksi, kita mendapatkan kelas dengan probabilitas maksimum bersama dengan kerugiannya, sehingga kita dapat menghitung kerugian dan akurasi epoch.

Perhatikan bahwa kami juga mengenai learning rate scheduler.

**def** eval\_model(model, data\_loader, loss\_fn, device, n\_examples):

model **=** model**.**eval()

losses **=** []

correct\_predictions **=** 0

**with** torch**.**no\_grad():

**for** inputs, labels **in** data\_loader:

inputs **=** inputs**.**to(device)

labels **=** labels**.**to(device)

outputs **=** model(inputs)

\_, preds **=** torch**.**max(outputs, dim**=**1)

loss **=** loss\_fn(outputs, labels)

correct\_predictions **+=** torch**.**sum(preds **==** labels)

losses**.**append(loss**.**item())

**return** correct\_predictions**.**double() **/** n\_examples, np**.**mean(losses)

Dengan melakukan banyak pemformatan string dan perekaman riwayat training sistem. Hal-hal rumit didelegasikan ke fungsi pembantu sebelumnya. Sehingga bobot model yang paling akurat disimpan selama pelatihan. Lalu mengujinya dengan menjalankan program berikut.

**%%time**

base\_model, history **=** train\_model(base\_model, data\_loaders, dataset\_sizes, device)

Lalu function tambahan dari hasil riwayat training divisualisasikan dalam listing berikut

plot\_training\_history(history)

**def** plot\_training\_history(history):

fig, (ax1, ax2) **=** plt**.**subplots(1, 2, figsize**=**(18, 6))

ax1**.**plot(history['train\_loss'], label**=**'train loss')

ax1**.**plot(history['val\_loss'], label**=**'validation loss')

ax1**.**xaxis**.**set\_major\_locator(MaxNLocator(integer**=True**))

ax1**.**set\_ylim([**-**0.05, 1.05])

ax1**.**legend()

ax1**.**set\_ylabel('Loss')

ax1**.**set\_xlabel('Epoch')

ax2**.**plot(history['train\_acc'], label**=**'train accuracy')

ax2**.**plot(history['val\_acc'], label**=**'validation accuracy')

ax2**.**xaxis**.**set\_major\_locator(MaxNLocator(integer**=True**))

ax2**.**set\_ylim([**-**0.05, 1.05])

ax2**.**legend()

ax2**.**set\_ylabel('Accuracy')

ax2**.**set\_xlabel('Epoch')

fig**.**suptitle('Training history')

Lalu diplotkan untuk menvisualisasikannya

plot\_training\_history(history)

Sehingga hasilnya menjadi sebagai berikut



Dilihat pada gambar diatas bahwa Model pra-pelatihan sesuai sehingga mendapatkan akurasi yang sangat tinggi dan loss yang rendah setelah 3 epoch. Sayangnya, set validasi kami terlalu kecil untuk mendapatkan beberapa metrik yang dapat diterima.

Lalu hasil dari prediksi rambu lalu lintas dijalankan dengan program berikut.

**def** show\_predictions(model, class\_names, n\_images**=**6):

model **=** model**.**eval()

images\_handeled **=** 0

plt**.**figure()

**with** torch**.**no\_grad():

**for** i, (inputs, labels) **in** enumerate(data\_loaders['test']):

inputs **=** inputs**.**to(device)

labels **=** labels**.**to(device)

outputs **=** model(inputs)

\_, preds **=** torch**.**max(outputs, 1)

**for** j **in** range(inputs**.**shape[0]):

images\_handeled **+=** 1

ax **=** plt**.**subplot(2, n\_images**//**2, images\_handeled)

ax**.**set\_title(f'predicted: {class\_names[preds[j]]}')

imshow(inputs**.**cpu()**.**data[j])

ax**.**axis('off')

**if** images\_handeled **==** n\_images:

**return**

Lalu divisualisasikan dengan menjalankan listing berikut.

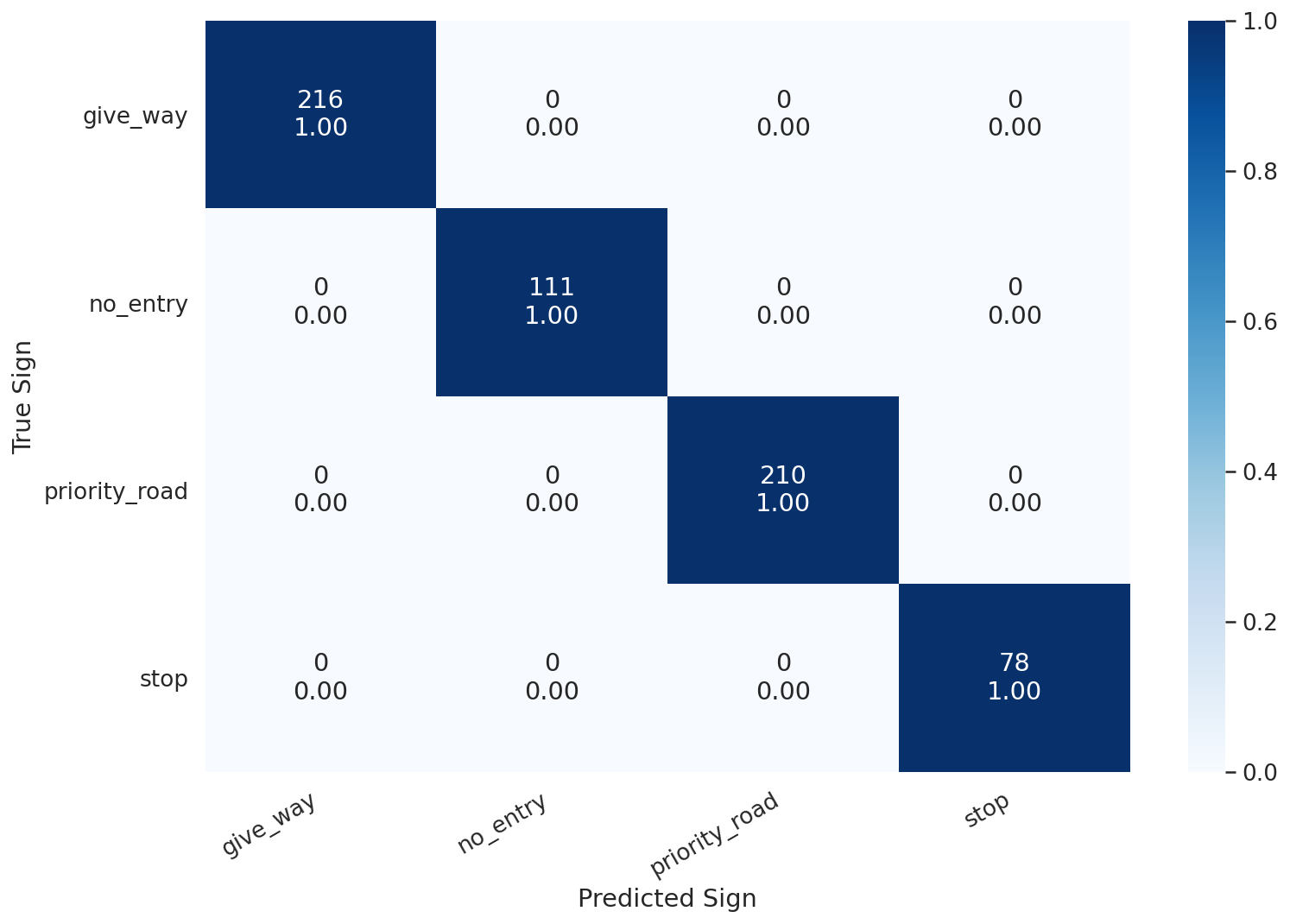
show\_predictions(base\_model, class\_names, n\_images**=**8)

Berikut adalah hasil yang didapatkan.



Dari gambar diatas dapat disimpulkan bahwa program berjalan dengan sempurna karena output sesuai dengan prediksi dari dalam model.

Lalu untuk mengevaluasi dari kinerja program, dapar diidentifikasikan dengan keakuratan dan berapa loss yang didapatkan dengan menggunakan listing program berikut yang akan menampilkan Confusion Matrix dari proses pengklasifikasian,



Di sini, Confusion Matrix digunakan untuk mengevaluasi kualitas keluaran pengklasifikasi pada kumpulan data. Dalam plot ini, elemen diagonal menunjukkan jumlah titik yang label prediksi sama dengan label sebenarnya, sedangkan elemen diagonal adalah yang salah diberi label oleh pengklasifikasi. Semakin tinggi nilai diagonal matriks konfusi semakin baik, yang menunjukkan banyak prediksi yang benar. Label yang benar berada di sepanjang sumbu y sedangkan label yang diprediksi berada di sumbu x. Dari hasil Confusion Matrix Ke menunjukkan bahwa citra rambu lalu lintas yang diinputkan dapat diklasifikasikan secara akurat.