Raken Putra Athallah
1103204186

PyTorch Computer Vision

Computer Vision

Computer Vision (penglihatan komputer) adalah disiplin ilmu yang berkaitan dengan pengembangan sistem yang memungkinkan mesin untuk menginterpretasikan dan memahami informasi visual dari dunia nyata. Tujuan utama dari computer vision adalah untuk memberikan kemampuan komputer untuk melihat dan memahami dunia seperti yang dilakukan manusia.

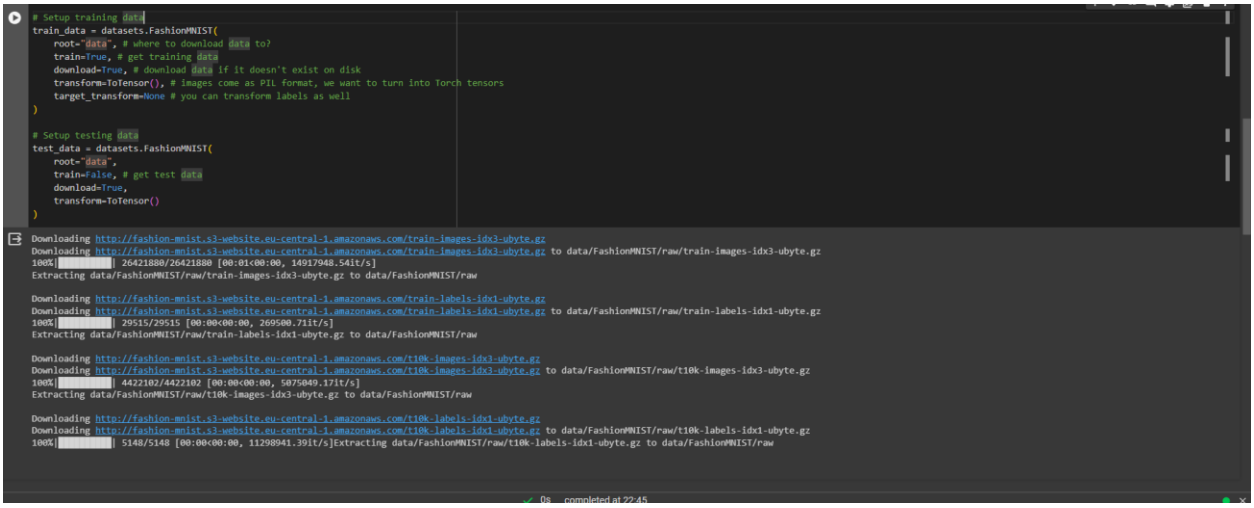
Dalam konteks computer vision, mesin atau komputer diharapkan dapat melakukan tugas-tugas berikut:

1. **Pengenalan Objek**: Mengidentifikasi dan mengklasifikasikan objek-objek yang ada dalam gambar atau video.
2. **Deteksi dan Pelacakan Objek:** Menemukan keberadaan dan melacak pergerakan objek tertentu dalam ruang dan waktu.
3. **Segmentasi Gambar**: Memisahkan gambar menjadi beberapa bagian atau region untuk analisis yang lebih mendalam.
4. **Rekognisi Wajah**: Mengenali dan mengidentifikasi wajah manusia dalam gambar atau video.
5. **Analisis Citra Medis**: Menggunakan teknologi visi komputer untuk mendiagnosis dan menganalisis gambar medis seperti hasil pemindaian CT atau MRI.
6. **Rekognisi Aksi**: Mengidentifikasi dan mengklasifikasikan tindakan atau aktivitas yang dilakukan oleh objek atau orang dalam video.
7. **Pemahaman Scene**: Memahami konteks atau makna dari suatu scene berdasarkan informasi visual yang diberikan.

Dalam pencapaian tujuan-tujuan tersebut, teknologi computer vision menggunakan algoritma dan model-machine learning, terutama Convolutional Neural Networks (CNNs), untuk memproses dan menganalisis data visual. Hal ini membuka potensi penerapan luas, mulai dari aplikasi keamanan, otomasi industri, pengenalan wajah, hingga pengembangan mobil otonom dan lebih banyak lagi.

Memuat Dataset

Untuk melakukan computer vision hal pertama yang perlu kita lakukan adalah mendapatkan computer vision dataset. Kali ini kita akan menggunakan dataset dari FashionMNIST (Modified National Institute of Standards and Technology).



Prepare DataLoader

DataLoader dalam PyTorch adalah komponen kritis yang bertanggung jawab untuk memproses dan memuat dataset ke dalam model. Dalam konteks pelatihan dan inferensi, DataLoader membantu mengorganisir data dengan cara-cara berikut:

1. Batching:

- DataLoader mengorganisir data menjadi batch atau mini-batch, yang merupakan subset kecil dari dataset keseluruhan.
- Batch size, yang dapat dikonfigurasi, menentukan jumlah sampel dalam setiap batch.
- Pemecahan dataset menjadi batch membantu dalam efisiensi komputasi dan penanganan dataset besar.

2. Iterabilitas:

- DataLoader mengubah dataset menjadi Python iterable yang dapat diakses dalam iterasi.
- Setiap iterasi memberikan satu batch data untuk diproses oleh model.

3. Shuffling (Opsional):

- DataLoader dapat mengacak dataset sebelum membaginya menjadi batch.
- Ini membantu model untuk mengakses contoh-contoh data dalam urutan yang acak, mencegah model untuk mengingat pola urutan tertentu.

4. Paralelisasi Loading (Opsional):

- Jika diperlukan, DataLoader dapat dikonfigurasi untuk memuat batch secara paralel, meningkatkan efisiensi pemuatan data.

Pada dasarnya, DataLoader adalah jembatan antara dataset dan model, memfasilitasi proses pembelajaran mesin atau pembelajaran mendalam. Dengan menerapkan batching dan iterasi, DataLoader memungkinkan model untuk secara efisien memproses dan memperbarui parameter pada setiap epoch, memfasilitasi konvergensi yang cepat dan efisien.

Membuat Model Dan Training Model

Setelah mendapatkan dataset yang akan digunakan kita dapat membuat model computer vision yang akan digunakan. Kali ini kita membuat tiga model berbeda yaitu model baseline, model dengan menerapkan non linearity dan model convolutional neural network (CNN). Kemudian kita melakukan training pada setiap model tersebut yang kemudian akan dilihat performa dari model masing-masing

```python
from torch import nn
class FashionMNISTModelV0(nn.Module):
    def __init__(self, input_shape: int, hidden_units: int, output_shape: int):
        super().__init__()
        self.layer_stack = nn.Sequential(
            nn.Flatten(), # neural networks like their inputs in vector form
            nn.Linear(in_features=input_shape, out_features=hidden_units), # in_features = number of features in a data sample (784 pixels)
            nn.Linear(in_features=hidden_units, out_features=output_shape)
        )

    def forward(self, x):
        return self.layer_stack(x)
```

```python
# Create a model with non-linear and linear layers
class FashionMNISTModelV1(nn.Module):
    def __init__(self, input_shape: int, hidden_units: int, output_shape: int):
        super().__init__()
        self.layer_stack = nn.Sequential(
            nn.Flatten(), # flatten inputs into single vector
            nn.Linear(in_features=input_shape, out_features=hidden_units),
            nn.ReLU(),
            nn.Linear(in_features=hidden_units, out_features=output_shape),
            nn.ReLU()
        )

    def forward(self, x: torch.Tensor):
        return self.layer_stack(x)
```

```python
# Create a convolutional neural network
class FashionMNISTModelV2(nn.Module):
    """
    Model architecture copying TinyVGG from:
    https://poloclub.github.io/cnn-explainer/
    """
    def __init__(self, input_shape: int, hidden_units: int, output_shape: int):
        super().__init__()
        self.block_1 = nn.Sequential(
            nn.Conv2d(in_channels=input_shape,
                      out_channels=hidden_units,
                      kernel_size=3, # how big is the square that's going over the image?
                      stride=1, # default
                      padding=1),# options = "valid" (no padding) or "same" (output has same shape as input) or int for specific number
            nn.ReLU(),
            nn.Conv2d(in_channels=hidden_units,
                      out_channels=hidden_units,
                      kernel_size=3,
                      stride=1,
                      padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2,
                         stride=2) # default stride value is same as kernel_size
        )
        self.block_2 = nn.Sequential(
            nn.Conv2d(hidden_units, hidden_units, 3, padding=1),
            nn.ReLU(),
            nn.Conv2d(hidden_units, hidden_units, 3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2)
        )
        self.classifier = nn.Sequential(
            nn.Flatten(),
            # Where did this in_features shape come from?
            # It's because each layer of our network compresses and changes the shape of our inputs data.
            nn.Linear(in_features=hidden_units*7*7,
                      out_features=output_shape)
        )

    def forward(self, x: torch.Tensor):
        x = self.block_1(x)
        # print(x.shape)
        x = self.block_2(x)
        # print(x.shape)
        x = self.classifier(x)
        # print(x.shape)
        return x

torch.manual_seed(42)
model_2 = FashionMNISTModelV2(input_shape=1,
    hidden_units=10,
    output_shape=len(class_names)).to(device)
model_2
```

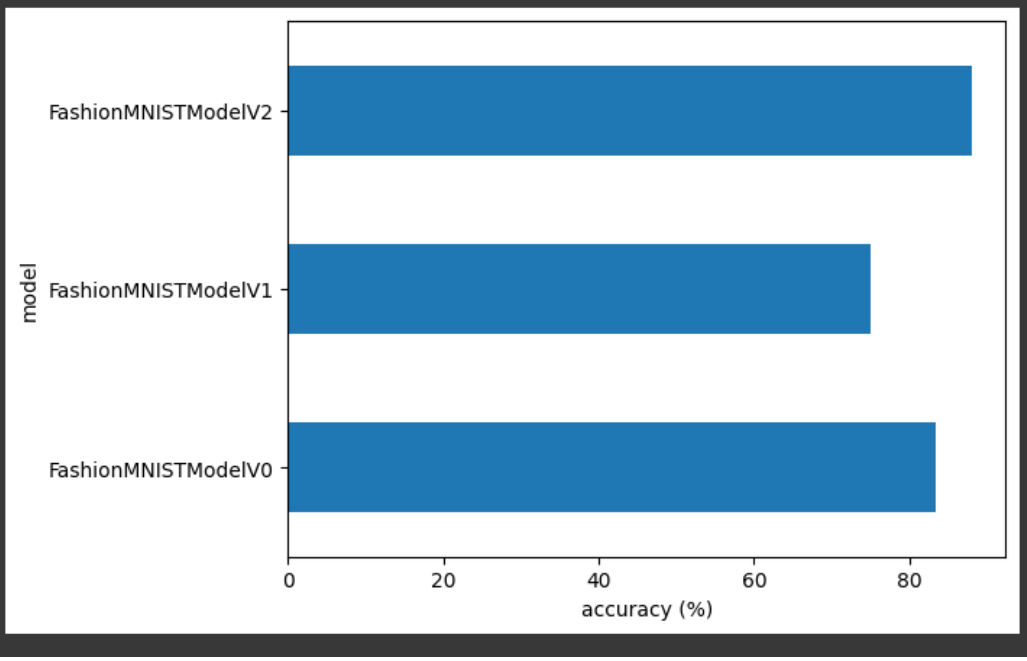
Evaluasi Model

Setelah melakukan training dengan ketiga model yang berbeda, kita melakukan evaluasi dengan membandingkan performa dari ketiga model tersebut

```
# Add training times to results comparison
compare_results["training_time"] = [total_train_time_model_0,
                                    total_train_time_model_1,
                                    total_train_time_model_2]

compare_results
```

	model_name	model_loss	model_acc	training_time
0	FashionMNISTModelV0	0.476639	83.426518	28.710748
1	FashionMNISTModelV1	0.685001	75.019968	34.721093
2	FashionMNISTModelV2	0.328518	88.039137	39.927219

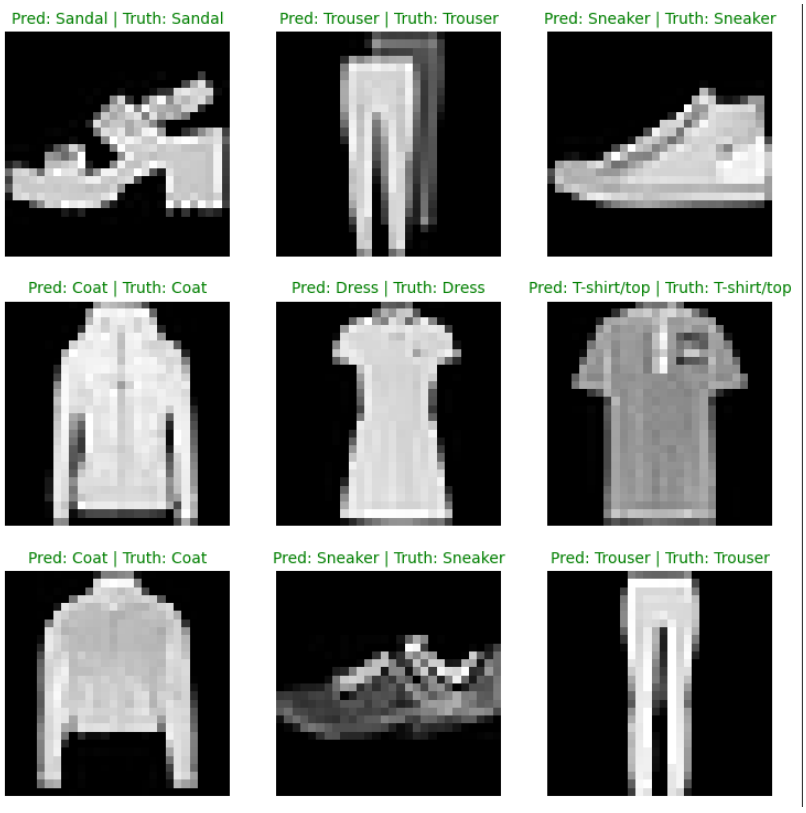
```
# Visualize our model results
compare_results.set_index("model_name")["model_acc"].plot(kind="barh")
plt.xlabel("accuracy (%)")
plt.ylabel("model");
```



Random Prediction On Best Model

Setelah mengetahui model dengan performa terbaik, kita mencoba untuk melakukan random prediction pada model_2 dikarenakan model tersebut merupakan model terbaik dari tiga model yang ada

```
# Plot predictions
plt.figure(figsize=(9, 9))
nrows = 3
ncols = 3
for i, sample in enumerate(test_samples):
  # Create a subplot
  plt.subplot(nrows, ncols, i+1)

  # Plot the target image
  plt.imshow(sample.squeeze(), cmap="gray")

  # Find the prediction label (in text form, e.g. "Sandal")
  pred_label = class_names[pred_classes[i]]

  # Get the truth label (in text form, e.g. "T-shirt")
  truth_label = class_names[test_labels[i]]

  # Create the title text of the plot
  title_text = f"Pred: {pred_label} | Truth: {truth_label}"

  # Check for equality and change title colour accordingly
  if pred_label == truth_label:
      plt.title(title_text, fontsize=10, c="g") # green text if correct
  else:
      plt.title(title_text, fontsize=10, c="r") # red text if wrong
  plt.axis(False);
```



Saving Best Model

Setelah menyelasikan seluruh proses kita menyimpan model yang telah kita buat

```python
from pathlib import Path

# Create models directory (if it doesn't already exist), see: https://docs.python.org/3/library/pathlib.html#pathlib.Path.mkdir
MODEL_PATH = Path("models")
MODEL_PATH.mkdir(parents=True, # create parent directories if needed
                 exist_ok=True # if models directory already exists, don't error
)

# Create model save path
MODEL_NAME = "03_pytorch_computer_vision_model_2.pth"
MODEL_SAVE_PATH = MODEL_PATH / MODEL_NAME

# Save the model state dict
print(f"Saving model to: {MODEL_SAVE_PATH}")
torch.save(obj=model_2.state_dict(), # only saving the state_dict() only saves the learned parameters
           f=MODEL_SAVE_PATH)
```