American Sign Language Fingerspelling Recognition

Rajkumar Conjeevaram Mohan (25th April 2023)

1. Introduction

American Sign Language (ASL) serves as the primary means of communication for deaf and hard of hearing individuals in the United States. Fingerspelling, a significant aspect of ASL, utilizes 26 unique hand configurations to denote the letters of the alphabet. I have contributed code that does file handling, loading of the data, pre-processing (augmentation), custom Dataset, and CNN model.

2. Description of individual work

Created the base Convolutional model that initially did not have a Batch Normalization, and when other optimizations were completed, Batch Normalization was included. With BatchNorm2D and larger learning rate, the 4-layer (incl. linear) CNN model was created. This produced an f1-score of ~0.988 on the test set.

3. Contribution

   a. Folder restructuring

   The data was sourced from Kaggle webpage, and is comprised of approximately 131,000 images that is produced by 5 non-native signers. Since the sign images were produced by 5 different subjects, images produced by each individual subject was originally found to be in its respective folder. For the Pytorch dataset to be able to efficiently load and process the images, I wrote a snippet of code that collates images corresponding to each alphabet from all signers and placing them in a separate folder.

```python
#%%
import os
from os.path import join
import numpy as np
from shutil import *

#%%
# DATA_DIR = r"C:\Users\Rajkumar\Downloads\ASL\dataset5"
DATA_DIR = r"D:/GWU/DATS-6303/project/archive/dataset5"
COLLATED_DIR = join(DATA_DIR, "collated")
#%%
if "collated" in os.listdir(DATA_DIR):
    rmtree(COLLATED_DIR)

os.mkdir(COLLATED_DIR)

subjects = os.listdir(DATA_DIR)
subjects = list(filter(lambda x: x != "collated", subjects))
max_len = 100000
for subject in subjects:
    subject_folder = f"{DATA_DIR}{os.path.sep}{subject}"
    char_folders = os.listdir(subject_folder)
    for char_ in char_folders:
        char_path = join(subject_folder, char_)
        collated_char_path = join(COLLATED_DIR, char_)
        if not char_ in os.listdir(COLLATED_DIR):
            os.mkdir(collated_char_path)
        img_fnames = os.listdir(char_path)
        for img_fname in img_fnames:
            fname = img_fname.split(".")[0]
            new_fname = fname + f"_{np.random.randint(0, max_len)}.png"
            copyfile(src=join(char_path, f"{fname}.png"), dst=join(collated_char_path, new_fname))
```

b. Data loading and Image Dataset

```python
Rajkumar Conjeevaram Mohan *
def load_filenames_df(self, data_dir, le):
    total_n_imgs = 0
    # join_path = lambda x: join(self.data_dir, join(x[0], x[1]))

    total_imgs = []
    total_labels = []
    for char_ in listdir(data_dir):
        fnames = listdir(join(data_dir, char_))
        color_filter_mask = [len(re.findall("color", fname)) > 0 for fname in fnames]
        fnames = np.array(fnames)[color_filter_mask]
        fnames = [join(data_dir, join(char_, fname)) for fname in fnames]
        total_imgs = np.r_[total_imgs, fnames]
        total_labels = np.r_[total_labels, [char_] * len(fnames)]
        total_n_imgs += len(fnames)
    total_n_imgs = total_n_imgs
    total_labels = le.fit_transform(total_labels)
    indices = list(range(total_n_imgs))
    total_labels = total_labels[indices]
    total_imgs = total_imgs[indices]
    df = pd.DataFrame({"img": total_imgs, "label": total_labels})
    tr_data, ts_data, tr_labels, ts_labels = train_test_split(df, df.label, test_size=0.3,
                                                              stratify=total_labels, random_state=self.random_seed)
    tr_data, vl_data, tr_labels, vl_labels = train_test_split(tr_data, tr_data.label,
                                                              test_size=0.3, stratify=tr_labels,
                                                              random_state=self.random_seed)
    return tr_data, vl_data, ts_data, tr_labels, vl_labels, ts_labels
```

Once the images from subject A – E are placed into a single folder, but distinguished by folders corresponding to the English alphabets, load_filenames_df routine creates a Pandas dataframe with img as the file path to the image, and label denoting the character. This was then consumed by the ImageDataset and the DataLoader.

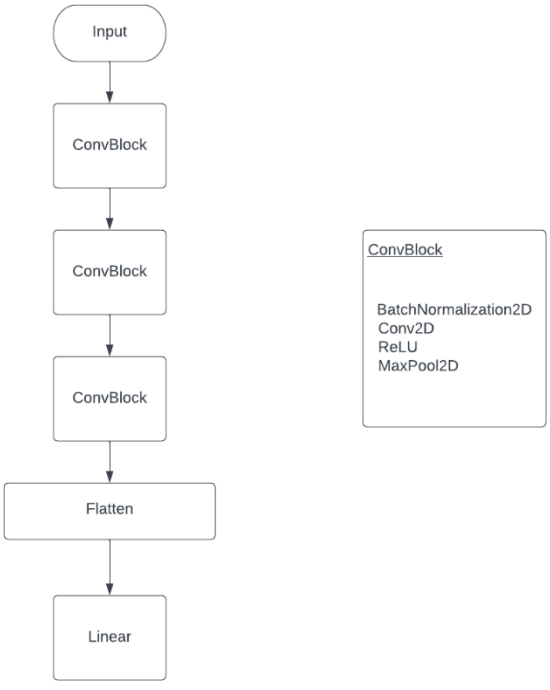
```
Rajkumar Conjeevaram Mohan +1
class ImageDataset(Dataset):
    Rajkumar Conjeevaram Mohan
    def __init__(self, data, transforms):
        self.data = data.loc[:, ["img", "label"]].reset_index()
        self.transforms = transforms

    Rajkumar Conjeevaram Mohan
    def __len__(self):
        return self.data.shape[0]

    Rajkumar Conjeevaram Mohan
    def __getitem__(self, idx):
        img_fname, label = self.data.iloc[idx]["img"], self.data.iloc[idx]["label"]
        image = Image.open(img_fname)
        image = torchvision.transforms.PILToTensor()(image)
        image = torchvision.transforms.ConvertImageDtype(dtype=torch.float32)(image)
        image = torchvision.transforms.ToPILImage()(image)
        image = self.transforms(image)
        return image, label
```

```
tr_data, vl_data, ts_data, tr_labels, vl_labels, ts_labels = self.load_filenames_df(DATA_DIR, self.le)
tr_dset = ImageDataset(tr_data, train_transforms)
vl_dset = ImageDataset(vl_data, val_test_transforms)
ts_dset = ImageDataset(ts_data, val_test_transforms)

self.N_CLASSES = len(tr_data.label.unique())
self.tr_loader = DataLoader(tr_dset, batch_size=self.BATCH_SIZE, num_workers=NUM_WORKERS,
                            prefetch_factor=PREFETCH_FACTOR, worker_init_fn=seed_worker, generator=g)
self.vl_loader = DataLoader(vl_dset, batch_size=self.BATCH_SIZE, num_workers=NUM_WORKERS,
                            prefetch_factor=PREFETCH_FACTOR, worker_init_fn=seed_worker, generator=g)
# for i, (X_tmp, y_tmp) in enumerate(vl_dset):
#     print(i)
self.ts_loader = DataLoader(ts_dset, batch_size=self.BATCH_SIZE, num_workers=NUM_WORKERS,
                            prefetch_factor=PREFETCH_FACTOR, worker_init_fn=seed_worker, generator=g)
```

c. Model

```
def model_def(self) -> torch.nn.Module:
    final_layer_img_size = int(((self.IMG_SIZE / 2) / 2) / 2)
    last_input_shape = final_layer_img_size * final_layer_img_size * 64
    # new_test = DataLoader(test_dataset, batch_size=BATCH_SIZE, num_workers=NUM_WO
    model = torch.nn.Sequential(
        ConvBlock(3, 16, (5, 5), groups=1),  # 100x100x3 -> 96x96x16 -> 48x48x16
        ConvBlock(16, 32),  # 48x48x16 -> 46x46x32 -> 23x23x32
        ConvBlock(32, 64),  # 23x23x32 -> 21x21x64 -> 10x10x64
        torch.nn.Flatten(),  # Nx6400
        torch.nn.Linear(6400, self.N_CLASSES)  # Nx24
    )

    model.cuda()
    print(summary(model.cuda(), (3, 100, 100)))
    return model
```

It could be a common practice to use padding – “same” while applying convolution to preserve the spatial resolution so that more layers can be included. However, it is my understanding that even a simpler model can outperform complex models given correct optimization, and tuning. Hence, I decided not to use padding i.e., (used padding – “valid”) and at each layer the spatial resolution was decreased thereby reducing the computational burden to efficiently process the mini-batches for optimization given there is over 131K images in the dataset.



```
Sequential(
  (0): ConvBlock(
    (bn): BatchNorm2d(3, eps=1e-05, momentum=0.1, affine=True,
     track_running_stats=True)
    (conv): Conv2d(3, 16, kernel_size=(5, 5), stride=(1, 1), padding=valid)
    (mp): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0, dilation=1,
     ceil_mode=False)
    (h_act): ReLU()
  )
  (1): ConvBlock(
    (bn): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True,
     track_running_stats=True)
    (conv): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1), padding=valid)
    (mp): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0, dilation=1,
     ceil_mode=False)
    (h_act): ReLU()
  )
  (2): ConvBlock(
    (bn): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True,
     track_running_stats=True)
    (conv): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=valid)
    (mp): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0, dilation=1,
     ceil_mode=False)
    (h_act): ReLU()
  )
  (3): Flatten(start_dim=1, end_dim=-1)
  (4): Linear(in_features=6400, out_features=24, bias=True)
)
```

I also thought it would be a neat coding practice to use modules instead of stacking layers inside the Sequential. My understanding is the use of modules improves readability, and paves way for easy debugging. Instead of having to make a common change in every single layer, the common change can be applied in the module that will be replicated while used inside the Sequential.

4. Results

Hyperparameter for Raj's model	
Learning rate	0.001
minibatch size	500
prefetch_factor	30
Kernel size	5x5, 3x3, 3x3
N of layers	4
Epoch Number	30
Optimizer	ADAM

Results for Raj's model	
Best N of Epoch	25
Train accuracy	1.0
Validation loss	0.09125
Validation accuracy	1.0
F1 score for test set	0.988856

The F1-score was almost near 99% on the test set and I believe it is great to see a simpler model achieving this score.

As part of the experiments, I also tried using Depthwise Separable Convolution to minimize the overall number of parameters used. However, the results were not as promising as the current model. Hence, Depthwise Separable Convolution was ignored.

5. Summary and Conclusion.

I have always thought about using many layers, and neurons to improve the accuracy of the model overlooking the potential of a hyperparameter tuning of a simpler CNN model. From the experiments, I have learnt that adequate and appropriate hyperparameter tuning can yield optimal results, although I do not want to generalize it for all the cases, it at least has been the case with this dataset.

In future, I wish to implement an end-to-end architecture that does real-time palm sign capturing and produces a sequence of text. Object Detection -> CNN -> Transformer hybrid architecture that would not only be interesting, but also serve the otherwise challenged people, and I believe would be a ground-breaking research project.

6. No external code was reused from my side.

7. References

Geislinger, V. (2020). ASL Fingerspelling Images (RGB & Depth) (Version 2)[Data Set]. Kaggle. https://www.kaggle.com/datasets/mrgeislinger/asl-rgb-depth-fingerspelling-spelling-it-out