Nhung Nguyen
Individual Report - Final Project
12/09/2019
Machine Learning II, Fall 2019

I. Introduction

1) Overview of the project

This project is to classify sign language images. There have been multiple development for incorporating machine learning with technology to assist people with disability. The goal of this project is to create an effective algorithm that can translate sign language images into text. The dataset, called Sign language MNIST, was acquired from Kaggle. The dataset has around 30,000 images that are classified into 24 classes. The letter J and Z require motion so they were excluded from the dataset.

2) Outline of shared work

Both:

- Work on visualizing the result
- Re-check each other's code to provide feedback and questions

Nhung:

- Experiment with Keras on Keras framework
- Experiment with Pytorch

Tina:

- Storing and retrieving data
- Experiment with Tensorflow for Keras subclass

II. Description of individual work

I experiment with 2 frameworks: Keras using Keras framework and Pytorch. I use sequential API for Keras and CNN(nn.Module) for Pytorch. With images of multiple hand patterns, I use conv2D since it works well in detecting particular pattern in images.

III. Detailed description of work

For both frameworks, I use numpy package to reshape the data. I also split the train data into training and validation dataset with 7:3 ratio.

```
test_imgs = test.iloc[:,1:].values.reshape((test.iloc[:,1:].shape[0],28,28))
test_imgs = np.expand_dims(test_imgs, axis=-1)
```

```
x_train, x_test2, y_train, y_test2 = train_test_split(train_imgs, train_labels, test_size=0.30)
```

1) Pytorch

For Pytorch, since the framework requires tensor, I use Variable from torch.autograd to change the type of variable. For labels, I used 'LongTensor' since BinaryCrossEntropy loss requires 'long type value'.

```
x_train1 = Variable(torch.FloatTensor(x_train), requires_grad = True).view(len(x_train), 1,28,28)
y_train1 = Variable(torch.LongTensor(y_train), requires_grad=False)
```

To transform the data, I first use transforms from torchvision. The process includes transform the data into PILImages and then randomly do some transformation with rotation and translation. However, it did not work well.

```
def test_transform(tensor):
    all_tranform = transforms.Compose([
                    transforms.ToPILImage(),
                    transforms.RandomAffine(20, translate=(0.2, 0.4), shear = (0.1, 0.3)),
                    transforms.ToTensor()
                ])

    tensor = all_tranform(tensor)
    return tensor
```

As a result, I used a different approach and follow code suggestion to transform custom dataset. I have 2 transformation: horizontally or vertically flip the images. I decided to use random probability to decide which images to flip and what not. I think this will help the model to have more flexibility for future use.

```
def hv_flip(tensor):
    """Flips tensor horizontally or vertically"""

    if np.random.rand() <0.3:
        tensor = tensor.flip(2)
    if np.random.rand() > 0.5:
        tensor = tensor.flip(1)
    return tensor
```

For structure of the model, I follow the design from professor Jafari's github code. For optimizer, I used Adam since it is relatively fast and requires low memory. I chose CrossEntropyLoss since it works Ill for classifying multiple classes.

While training model, to compare the labels, I have to use label.squeeze_() first or I will run into the following error. This error was because I make the label type as Float tensor when I put labels into TensorDataset.

```
  File "/home/ubuntu/gpu_noon/pytorch2.py", line 151, in <module>
    loss = criterion(y_test_pred, y_test2)
  File "/usr/local/lib/python3.6/dist-packages/torch/nn/modules/module.py", line 547, in __call__
    result = self.forward(*input, **kwargs)
  File "/usr/local/lib/python3.6/dist-packages/torch/nn/modules/loss.py", line 916, in forward
    ignore_index=self.ignore_index, reduction=self.reduction)
  File "/usr/local/lib/python3.6/dist-packages/torch/nn/functional.py", line 1995, in cross_entropy
    return nll_loss(log_softmax(input, 1), target, weight, None, ignore_index, None, reduction)
  File "/usr/local/lib/python3.6/dist-packages/torch/nn/functional.py", line 1824, in nll_loss
    ret = torch._C._nn.nll_loss(input, target, weight, _Reduction.get_enum(reduction), ignore_index)
RuntimeError: multi-target not supported at /pytorch/aten/src/THNN/generic/ClassNLLCriterion.c:22
```

For each epoch, I train the model and print out loss and accuracy of the train data, the validation data, and the test data.

```
model.eval()
with torch.no_grad():
    x_test_split = Variable(x_test2)
    y_test_pred_split = model(x_test_split)
    y_test_split = y_test2.squeeze_()
    loss = criterion(y_test_pred_split, y_test2)
    loss_test_split = loss.item()

    x_test_ori = Variable(x_test)
    y_test_pred_ori = model(x_test_ori)
    y_test_ori = y_test.squeeze_()
    loss_ori = criterion(y_test_pred_ori, y_test_ori)
    loss_test_ori = loss_ori.item()

print("Epoch {} | Valid Loss: {:.5f}, Valid Acc: {:.2f}, Test Loss: {:.5f}, Test A
    epoch, loss_test_split, acc(x_test2, y_test2), loss_test_ori, acc(x_test, y_te
```

2) Keras

I use the sequential API Keras. I first experimented with 2 layers and continue to increase until 4 layers, when the results stop significantly improve. As a result, I use 4 layers of conv2D.

```
# initilize a model
model = Sequential()

# add the first layer
model.add(Conv2D(32, (2, 2), padding= 'same', strides =  (1, 1), act
```

- `kernel_initializer=weight_init))`

Within each layer, I experiment with multiple parameters inside layer conv2D. For the first layer, I initialized the weight using kernel_initlizer. For ‘padding’ parameter, I use ‘same’ instead of ‘valid’ to maintain that the input and output length are the same.

For ‘stride’ parameter, I use (1,1) as increasing the strides do not affect my model performance in any significant way . For ‘activation’ parameter, I use ‘relu’ for all layers as ‘relu’ has been discussed as a good activation layer to adjust to nonlinearity. Furthermore, ‘relu’ allows layers to train faster (reference). I also use ‘maxpool2d’ in the layers to reduce output size and allow the model to focus on specific features within features.

To avoid overfitting and ensure the model is more flexible with varied data, I use ImageDataGenerator. To transform the data, I used rotation, shift and flip. Since some of the hand patterns are only slightly different from each other, it would be effective to make model see the images in different ways so it can learn the unique pattern for each class.

```
# construct the image generator for data augmentation
img_aug = ImageDataGenerator(rotation_range=30, width_shift_range=0.1,
                             height_shift_range=0.1, horizontal_flip=True, shear_range=0.2,
                             vertical_flip=True)
```

To compile the model, I used Adam optimizer as it is fast and requires low memory. The chosen loss criterion was 'categorical_crossentropy' since the labels was formatted using 'to_categorical'.

```
# compile model
model.compile(optimizer=Adam(learning_rate), loss=categorical_crossentropy, metrics=[categorical_accuracy])
```

To save the best model, I used ModelCheckpoint and using loss validation as a quantity to monitor.

```
# create checkpoint and choose to save the best model
model_c = ModelCheckpoint('model_group3.hdf5', monitor='val_loss', mode='auto', verbose=1, save_best_only=True)
```

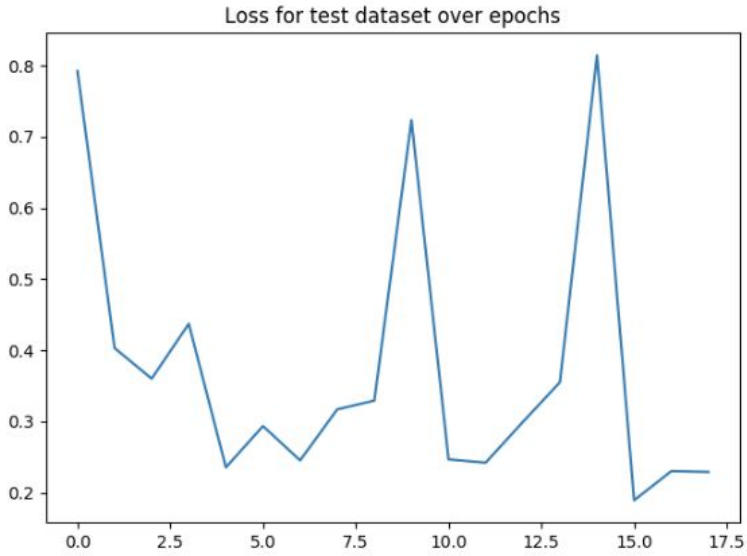
To fit the model, I used fit_generator. The batch size was 200 to reduce memory and allow the model to train faster.

```
# using fit_generator for model
model.fit_generator(img_aug.flow(x_train, y_train, batch_size=200),
    validation_data=(x_test2, y_test2), steps_per_epoch=len(x_train) // 200, epochs=n_epochs,
     verbose=1, shuffle=True, callbacks=[model_c])
```

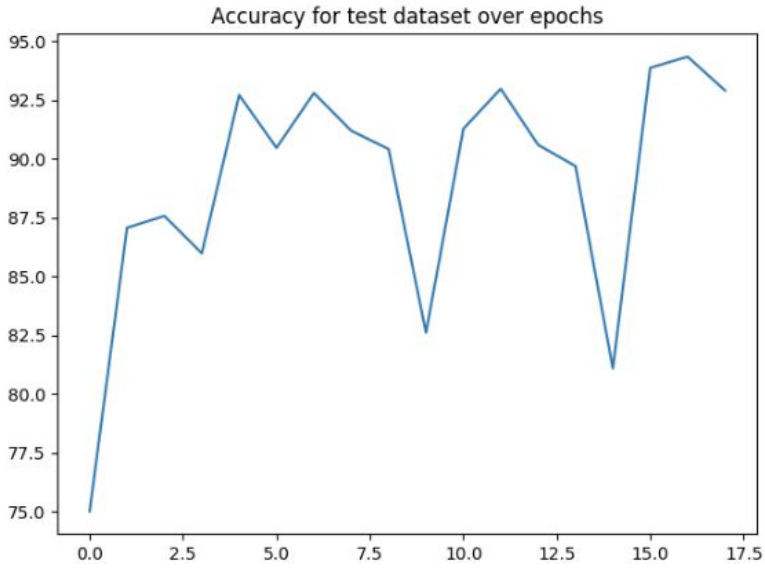
IV. Result

1) Pytorch result

The loss of the test dataset over epochs decreases first and then increase again. It decreases significantly after first 5 epochs but spikes up right after. This happens as I indicates 'shuffle=True' in the train_loader so that the batch sequences affect how the model was train in each epoch.

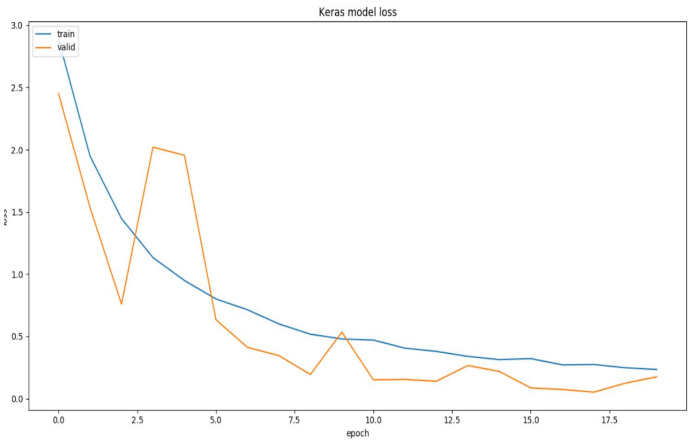


The accuracy for the test dataset, over the epochs, increases. Similar to the loss pattern, at epoch 8th and 13th, the accuracy drastically decreased. However, there might be problem presenting as the trend was changed significantly.

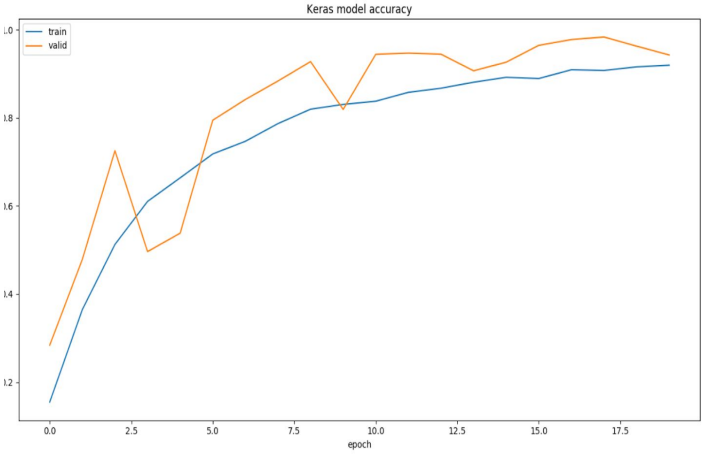


2) Keras result

The loss for the valid and train set over the epochs gradually decrease. Although the loss was spiked up for test set in first few epochs, the overall decrease pattern shows that the learning rate was probably chosen.



The final accuracy on test set, which was set aside, is 92%. This accuracy is around 3% lower than the average accuracy for the validation and train dataset (figure below). As a result, the model is not likely to be overfitted.



V. Summary and conclusions

Keras performs more stable. Next, the model was trained with ImageDataGenerator that has a variety of transformation compared to Pytorch's transformation.

The performance of Pytorch is not as stable as Keras. Within the Pytorch code, when I create the transformation code, I use np.random.randn() to flip the data so it can contribute to the irregular behavior of Pytorch. In the future, I need to check how I transform the data to improve the stability of the model's behavior.

VI. Percentage of the code copied from other sources

1) Keras
(10/100)*100 = 10%
The source includes reference from link for data-preprocessing.

2) Pytorch
(50/100) *100 = 50%
The source includes professor Github and stackoverflow.

VII. References

Dataset Source:

https://www.kaggle.com/datamunge/sign-language-mnist

Code reference:

https://stackoverflow.com/questions/55588201/pytorch-transforms-on-tensordataset

https://github.com/amir-jafari/Deep-Learning/blob/master/Pytorch_/CNN/1_ImageClassification/example_MNIST.py

https://www.kaggle.com/ranjeetjain3/deep-learning-using-sign-langugage

Article reference:

https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks-Part-2/

Appendix

The code document is available at *Nhung-Nguyen-final-project* folder in the below link:
https://github.com/nhungnoon/Final-Project-Group3

For Pytorch version:
The file is called **mywork2.py**. To access, click here.

For Keras version:
The file is called **mywork.py**. To access, click here.