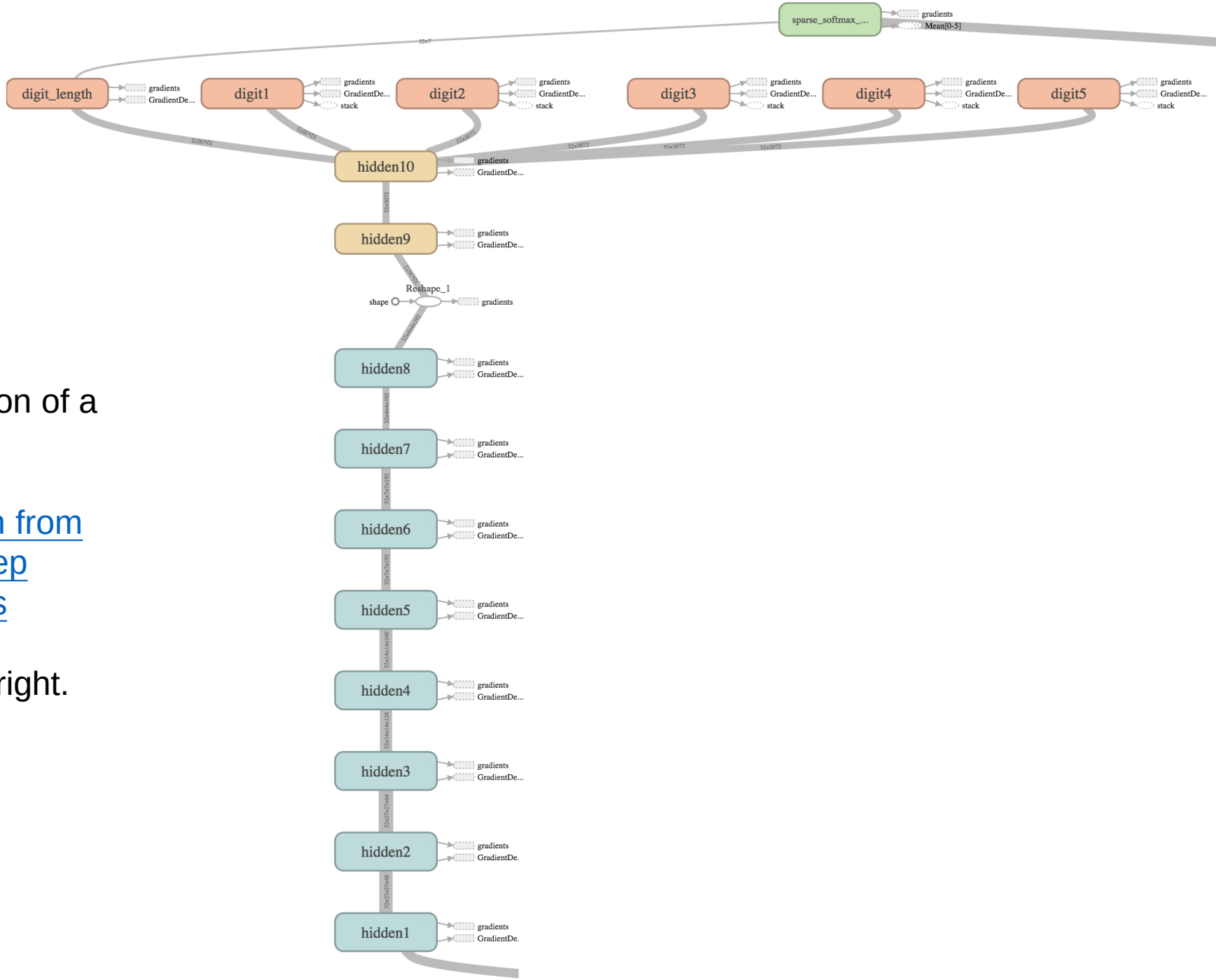
Street View House Number Recognition

Model

This is a Pytorch implementation of a baseline work:

[Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks]

Its network structure is on the right.



Params

Steps	GPU	Batch Size	Learning rate	Patience	Decay Step	Decay Rate
122000	GTX 1080Ti	512	0.01	100	625	0.9

```
self._hidden1 = nn.Sequential(
    nn.Conv2d(in_channels=3, out_channels=48,
              kernel_size=5, padding=2),
    nn.BatchNorm2d(num_features=48),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=2, stride=2, padding=1),
    nn.Dropout(0.2)
)
self._hidden2 = nn.Sequential(
    nn.Conv2d(in_channels=48, out_channels=64,
              kernel_size=5, padding=2),
    nn.BatchNorm2d(num_features=64),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=2, stride=1, padding=1),
    nn.Dropout(0.2)
)
```

```
self._hidden3 = nn.Sequential(
    nn.Conv2d(in_channels=64, out_channels=128,
              kernel_size=5, padding=2),
    nn.BatchNorm2d(num_features=128),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=2, stride=2, padding=1),
    nn.Dropout(0.2)
)
self._hidden4 = nn.Sequential(
    nn.Conv2d(in_channels=128, out_channels=160,
              kernel_size=5, padding=2),
    nn.BatchNorm2d(num_features=160),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=2, stride=1, padding=1),
    nn.Dropout(0.2)
)
```

Params

```
self._hidden5 = nn.Sequential(
    nn.Conv2d(in_channels=160, out_channels=192,
              kernel_size=5, padding=2),
    nn.BatchNorm2d(num_features=192),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=2, stride=2, padding=1),
    nn.Dropout(0.2)
)
self._hidden6 = nn.Sequential(
    nn.Conv2d(in_channels=192, out_channels=192,
              kernel_size=5, padding=2),
    nn.BatchNorm2d(num_features=192),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=2, stride=1, padding=1),
    nn.Dropout(0.2)
)
```

```
self._digit_length = nn.Sequential(nn.Linear(3072, 7))
self._digit1 = nn.Sequential(nn.Linear(3072, 11))
self._digit2 = nn.Sequential(nn.Linear(3072, 11))
self._digit3 = nn.Sequential(nn.Linear(3072, 11))
self._digit4 = nn.Sequential(nn.Linear(3072, 11))
self._digit5 = nn.Sequential(nn.Linear(3072, 11))
```

```
self._hidden7 = nn.Sequential(
    nn.Conv2d(in_channels=192, out_channels=192,
              kernel_size=5, padding=2),
    nn.BatchNorm2d(num_features=192),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=2, stride=2, padding=1),
    nn.Dropout(0.2)
)
self._hidden8 = nn.Sequential(
    nn.Conv2d(in_channels=192, out_channels=192,
              kernel_size=5, padding=2),
    nn.BatchNorm2d(num_features=192),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=2, stride=1, padding=1),
    nn.Dropout(0.2)
)
self._hidden9 = nn.Sequential(
    nn.Linear(192 * 7 * 7, 3072),
    nn.ReLU()
)
self._hidden10 = nn.Sequential(
    nn.Linear(3072, 3072),
    nn.ReLU()
)
```

Training method

- Loss function

Use Backpropagation algorithm on the sum of all digit cross entropies.

That is using loss.backward() where loss is defined as follow:

```
def _loss(length_logits, digit1_logits, digit2_logits, digit3_logits, digit4_logits, digit5_logits, length_labels, digits_labels):
    length_cross_entropy = torch.nn.functional.cross_entropy(
        length_logits, length_labels)
    digit1_cross_entropy = torch.nn.functional.cross_entropy(
        digit1_logits, digits_labels[0])
    digit2_cross_entropy = torch.nn.functional.cross_entropy(
        digit2_logits, digits_labels[1])
    digit3_cross_entropy = torch.nn.functional.cross_entropy(
        digit3_logits, digits_labels[2])
    digit4_cross_entropy = torch.nn.functional.cross_entropy(
        digit4_logits, digits_labels[3])
    digit5_cross_entropy = torch.nn.functional.cross_entropy(
        digit5_logits, digits_labels[4])
    loss = length_cross_entropy + digit1_cross_entropy + digit2_cross_entropy + \
        digit3_cross_entropy + digit4_cross_entropy + digit5_cross_entropy
    return loss
```

Optimization

- Use SGD method and StepLR to optimize the training

- SGD

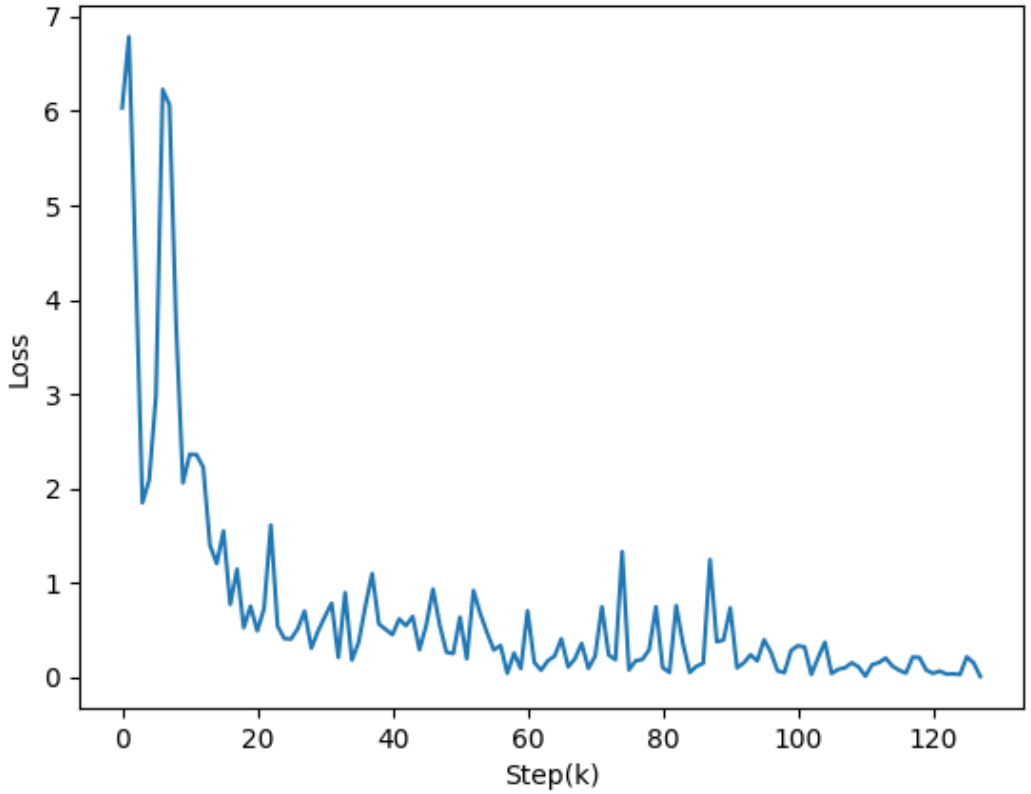
```
optimizer = optim.SGD(model.parameters(
), lr=initial_learning_rate, momentum=0.9, weight_decay=0.0005)
```

```
optimizer.zero_grad()
loss.backward()
optimizer.step()
scheduler.step()
```

StepLR

```
scheduler = StepLR(
    optimizer, step_size=training_options['decay_steps'], gamma=training_options['decay_rate'])
```

Training curve



Accuracy

```
=> 2022-05-07 17:22:54.110391: step 120100, loss = 0.215957, learning_rate = 0.002824 (1435.7 examples/sec)
=> 2022-05-07 17:22:57.400590: step 120200, loss = 0.023837, learning_rate = 0.002824 (1438.2 examples/sec)
=> 2022-05-07 17:23:00.711765: step 120300, loss = 0.042559, learning_rate = 0.002824 (1429.8 examples/sec)
=> 2022-05-07 17:23:04.031853: step 120400, loss = 0.203008, learning_rate = 0.002824 (1421.0 examples/sec)
=> 2022-05-07 17:23:07.309088: step 120500, loss = 0.099059, learning_rate = 0.002824 (1439.5 examples/sec)
=> 2022-05-07 17:23:10.617239: step 120600, loss = 0.223312, learning_rate = 0.002824 (1428.0 examples/sec)
=> 2022-05-07 17:23:13.959300: step 120700, loss = 0.472976, learning_rate = 0.002824 (1401.1 examples/sec)
=> 2022-05-07 17:23:17.287398: step 120800, loss = 0.107261, learning_rate = 0.002824 (1428.0 examples/sec)
=> 2022-05-07 17:23:20.650403: step 120900, loss = 0.134481, learning_rate = 0.002824 (1421.0 examples/sec)
=> 2022-05-07 17:23:24.025376: step 121000, loss = 0.156390, learning_rate = 0.002824 (1395.7 examples/sec)
=> Evaluating on validation dataset...
==> accuracy = 0.882265, best accuracy 0.892101
```

All these house numbers on the right can be well sorted by running the *infer.py*

THANKS