In [1]:

```
# import libraries
import pandas as pd
import numpy as np
from skimage.io import imread
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from tqdm import tqdm

import torch
from torch.autograd import Variable
from torch.nn import Linear, ReLU, CrossEntropyLoss, Sequential, Conv2d, MaxPool2d, Module
, Softmax, BatchNorm2d, Dropout
from torch.optim import Adam, SGD
```

In [31]:

```
# loading dataset
from skimage.color import rgb2gray
from sklearn.preprocessing import LabelEncoder

train = rgb2gray(np.load('train_data.npy'))
train = train/255
label = np.load('train_labels.npy')
le = LabelEncoder()
le.fit(label)
target = le.transform(label)
label = target+1

train_x, val_x, train_y, val_y = train_test_split(train, label, test_size=0.2)
train_x.shape, val_x.shape, train_y.shape, val_y.shape, train.shape
```

Out[31]:

```
((1475, 100, 100), (369, 100, 100), (1475,), (369,), (1844, 100, 100))
```

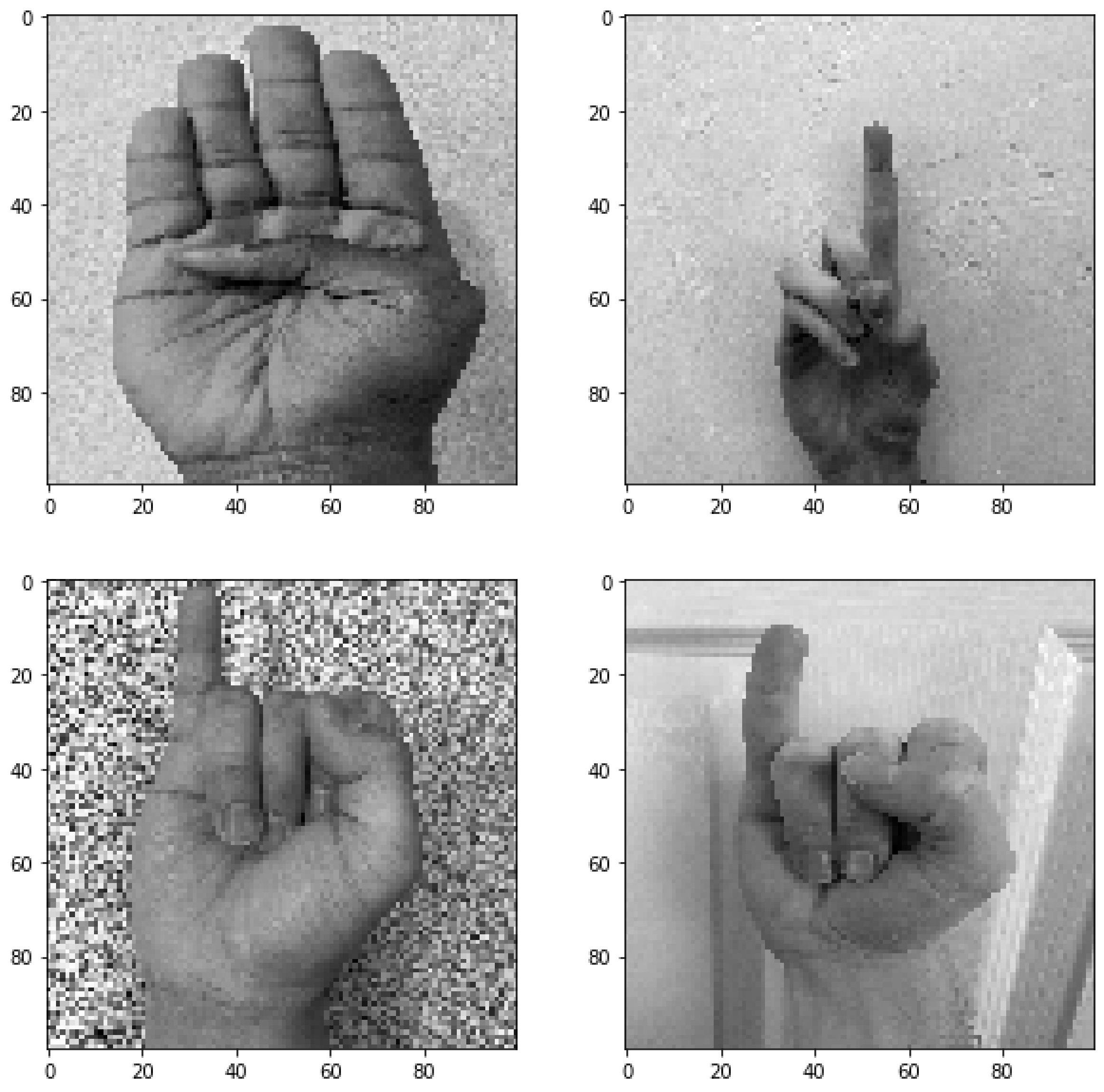
In [ ]:

In [32]:

```
# Show images
i = 0
plt.figure(figsize=(10,10))
plt.subplot(221), plt.imshow(train_x[i], cmap='gray')
plt.subplot(222), plt.imshow(train_x[i+25], cmap='gray')
plt.subplot(223), plt.imshow(train_x[i+50], cmap='gray')
plt.subplot(224), plt.imshow(train_x[i+75], cmap='gray')
```

Out[32]:

```
(<matplotlib.axes._subplots.AxesSubplot at 0x2884b0ac5c8>,
 <matplotlib.image.AxesImage at 0x2884b0eba08>)
```



In [33]:

```
# convert to tensor
train_x = train_x.reshape(1475, 1, 100, 100)
train_x  = torch.from_numpy(train_x)

train_y = train_y.astype(int);
train_y = torch.from_numpy(train_y)

train_x.shape, train_y.shape
```

Out[33]:

```
(torch.Size([1475, 1, 100, 100]), torch.Size([1475]))
```

In [6]:

```
# Convert to tensor
val_x = val_x.reshape(369, 1, 100, 100)
val_x  = torch.from_numpy(val_x)

val_y = val_y.astype(int);
val_y = torch.from_numpy(val_y)

val_x.shape, val_y.shape
```

Out[6]:

```
(torch.Size([369, 1, 100, 100]), torch.Size([369]))
```

In [27]:

```
class Net(Module):
    def __init__(self):
        super(Net, self).__init__()

        self.cnn_layers = Sequential(
            # First 2D convolution layer
            Conv2d(1, 4, kernel_size=3, stride=1, padding=1),
            BatchNorm2d(4),
            ReLU(inplace=True),
            MaxPool2d(kernel_size=2, stride=2),
            # Second 2D convolution layer
            Conv2d(4, 4, kernel_size=3, stride=1, padding=1),
            BatchNorm2d(4),
            ReLU(inplace=True),
            MaxPool2d(kernel_size=2, stride=2),
        )

        self.linear_layers = Sequential(
            Linear(1475, 2500)
        )

    # Forward pass
    def forward(self, x):
        x = self.cnn_layers(x)
        x = x.view(x.size(0), -1)
        x = self.linear_layers(x)
        return x
```

In [28]:

```
model = Net()
optimizer = Adam(model.parameters(), lr=0.07)
criterion = CrossEntropyLoss()

print(model)
```

```
Net(
  (cnn_layers): Sequential(
    (0): Conv2d(1, 4, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(4, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
    (2): ReLU(inplace=True)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=
False)
    (4): Conv2d(4, 4, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (5): BatchNorm2d(4, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
    (6): ReLU(inplace=True)
    (7): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=
False)
  )
  (linear_layers): Sequential(
    (0): Linear(in_features=1475, out_features=2500, bias=True)
  )
)
```

In [29]:

```
def train(epoch):
    model.train()
    tr_loss = 0
    # training set
    x_train, y_train = Variable(train_x), Variable(train_y)
    # validation set
    x_val, y_val = Variable(val_x), Variable(val_y)


    # clear Gradients of the model parameters
    optimizer.zero_grad()

    # prediction
    output_train = model(x_train)
    output_val = model(x_val)

    # training and validation loss
    loss_train = criterion(output_train, y_train)
    loss_val = criterion(output_val, y_val)
    train_losses.append(loss_train)
    val_losses.append(loss_val)

    # updated weights
    loss_train.backward()
    optimizer.step()
    tr_loss = loss_train.item()
    if epoch%2 == 0:

        print('Epoch : ',epoch+1, '\t', 'loss :', loss_val)
```

In [30]:

```
# number of epochs
n_epochs = 25
# empty list to store training losses
train_losses = []
# empty list to store validation losses
val_losses = []
# training
for epoch in range(n_epochs):
    train(epoch)
```

```
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
<ipython-input-30-6618bd6c6ef8> in <module>
      7 # training the model
      8 for epoch in range(n_epochs):
----> 9     train(epoch)

<ipython-input-29-5ab780eb6f11> in train(epoch)
     12 
     13     # prediction for training and validation set
---> 14     output_train = model(x_train)
     15     output_val = model(x_val)
     16 

~\Anaconda3\lib\site-packages\torch\nn\modules\module.py in __call__(self, *i
nput, **kwargs)
    530             result = self._slow_forward(*input, **kwargs)
    531         else:
--> 532             result = self.forward(*input, **kwargs)
    533         for hook in self._forward_hooks.values():
    534             hook_result = hook(self, input, result)

<ipython-input-27-12d59159c08b> in forward(self, x)
     24         x = self.cnn_layers(x)
     25         x = x.view(x.size(0), -1)
---> 26         x = self.linear_layers(x)
     27         return x

~\Anaconda3\lib\site-packages\torch\nn\modules\module.py in __call__(self, *i
nput, **kwargs)
    530             result = self._slow_forward(*input, **kwargs)
    531         else:
--> 532             result = self.forward(*input, **kwargs)
    533         for hook in self._forward_hooks.values():
    534             hook_result = hook(self, input, result)

~\Anaconda3\lib\site-packages\torch\nn\modules\container.py in forward(self,
 input)
     98     def forward(self, input):
     99         for module in self:
--> 100             input = module(input)
    101         return input
    102 

~\Anaconda3\lib\site-packages\torch\nn\modules\module.py in __call__(self, *i
nput, **kwargs)
    530             result = self._slow_forward(*input, **kwargs)
    531         else:
--> 532             result = self.forward(*input, **kwargs)
    533         for hook in self._forward_hooks.values():
    534             hook_result = hook(self, input, result)

~\Anaconda3\lib\site-packages\torch\nn\modules\linear.py in forward(self, inp
ut)
     85 
     86     def forward(self, input):
---> 87         return F.linear(input, self.weight, self.bias)
```

```
     88
     89     def extra_repr(self):

~\Anaconda3\lib\site-packages\torch\nn\functional.py in linear(input, weight,
bias)
   1368     if input.dim() == 2 and bias is not None:
   1369         # fused op is marginally faster
-> 1370         ret = torch.addmm(bias, input, weight.t())
   1371     else:
   1372         output = input.matmul(weight.t())

RuntimeError: size mismatch, m1: [1475 x 2500], m2: [1475 x 2500] at C:\w\1\s
\tmp_conda_3.7_075911\conda\conda-bld\pytorch_1579075223148\work\aten\src\TH/
generic/THTensorMath.cpp:136
```

In [ ]:

```python
# plot training and validation loss
plt.plot(train_losses, label='Training loss')
plt.plot(val_losses, label='Validation loss')
plt.legend()
plt.show()
```

In [ ]:

```python
# prediction for training set
with torch.no_grad():
    output = model(train_x.cuda())

softmax = torch.exp(output).cpu()
prob = list(softmax.numpy())
predictions = np.argmax(prob, axis=1)

# accuracy on training set
accuracy_score(train_y, predictions)
```

In [ ]:

```python
```

In [ ]:

```python
```

In [ ]:

```python
```