CS512: Advanced Machine Learning.

Assignment 2: Conditional Random Fields and Convolutions

Garima Gupta: ggupta22@uic.edu

Sai Teja Karnati: skarna3@uic.edu

Shubham Singh: ssing57@uic.edu

Wangfei Wang: wwang75@uic.edu

March 14, 2020

1 (20 points) Convolution

(3a) **(20 points)** We implemented Conv layer and the get_conv_features(x) function in the starter code, with accommodation of different strides and an option of zero padding. The implementation was tested.

For the following X and K with unit stride and zero padding:

$$X = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{bmatrix}; \quad K = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

the result of convolving the X with K is:

$$\hat{X} = \begin{bmatrix} 2 & 2 & 3 & 1 & 1 \\ 1 & 4 & 3 & 4 & 1 \\ 1 & 2 & 4 & 3 & 3 \\ 1 & 2 & 3 & 4 & 1 \\ 0 & 2 & 2 & 1 & 1 \end{bmatrix};$$

See `code` folder for `conv_test.py`.

2 (50 points) CRF

(4a) We implemented the forward, backward pass and loss inside `crf.py`.
See code.

(4b) **(20 points)**

Parameters we are using:

Batch size = 64

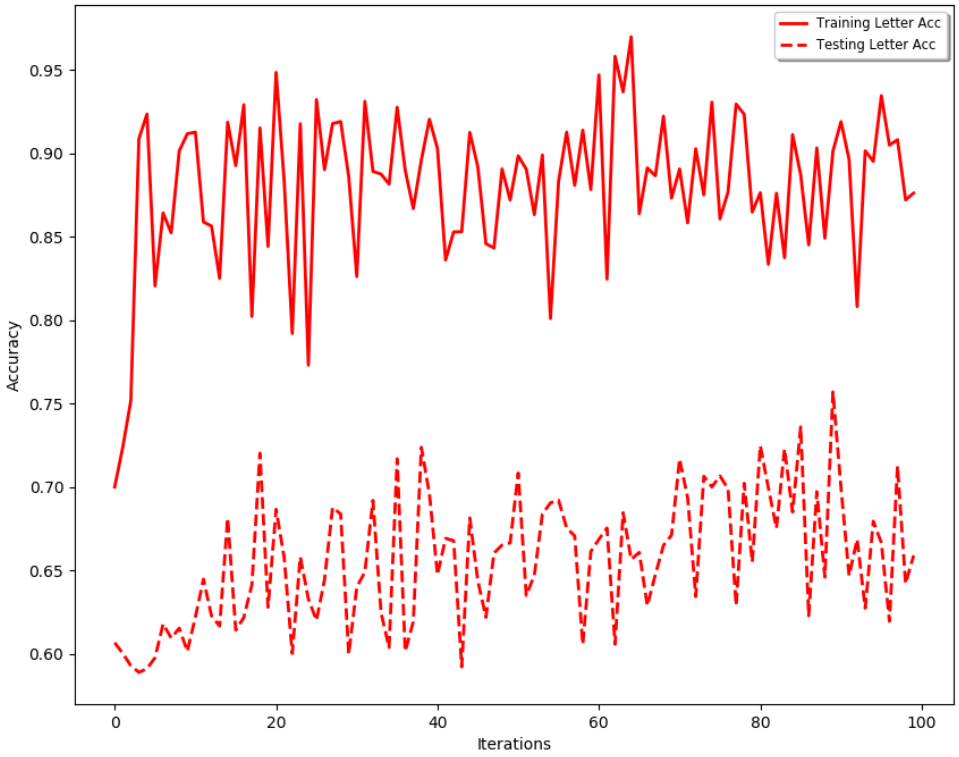
Number of iterations = 100

C = 1000 for CRF

LBFGS lr = 0.1

Zero-padded

(1) **letter-wise prediction accuracy**.

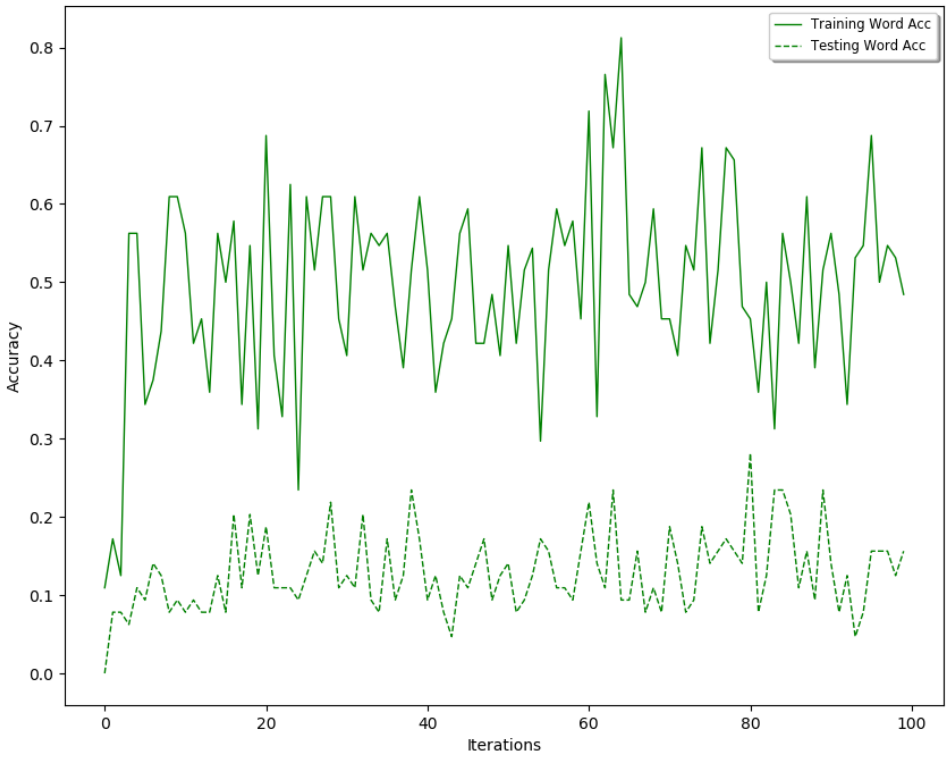


(2) **word-wise prediction accuracy**.

Average letter accuracy for training set is 88.0%

Average letter accuracy for test set is 65.7%.

Average word accuracy for training set is 49.4%



Average word accuracy for test set is 12.9%

(4c) **(20 points)** Repeat experiments in (4b) with the following convolution layers. Set stride and zero/no padding to optimize the test performance.

The other parameters used in this case are:

Batch size 64

Iterations 100

C = 1000 for CRF

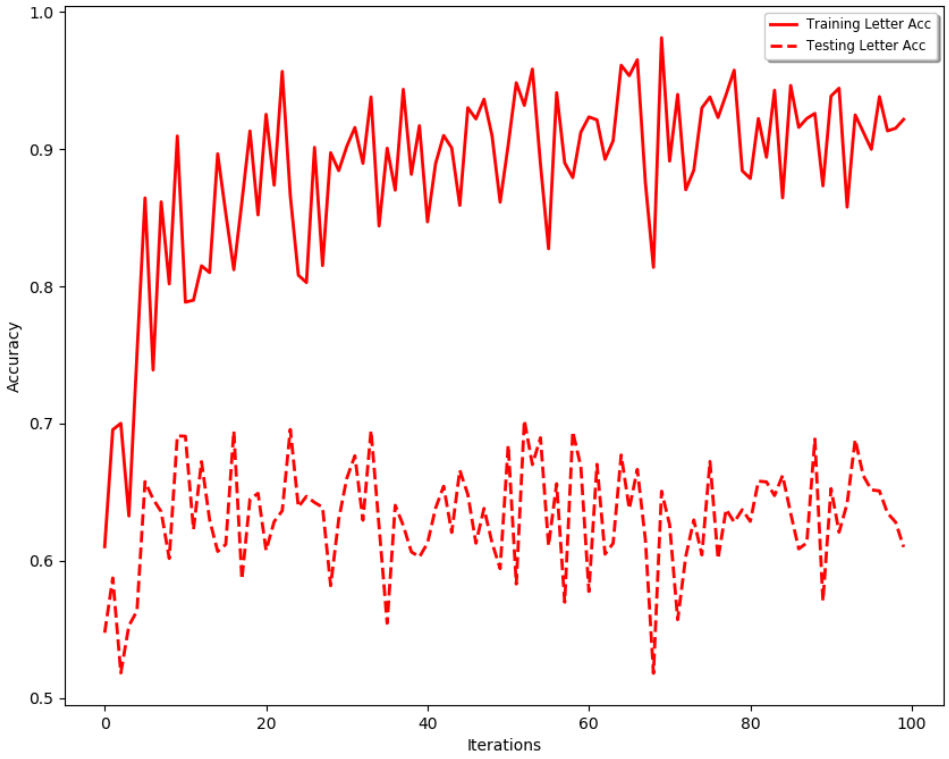
LBFGS learning rate = 0.1

Zero-padded

2-CNN CRF:

1. A Layer with 5×5 filter matrix
2. A Layer with 3×3 filter matrix

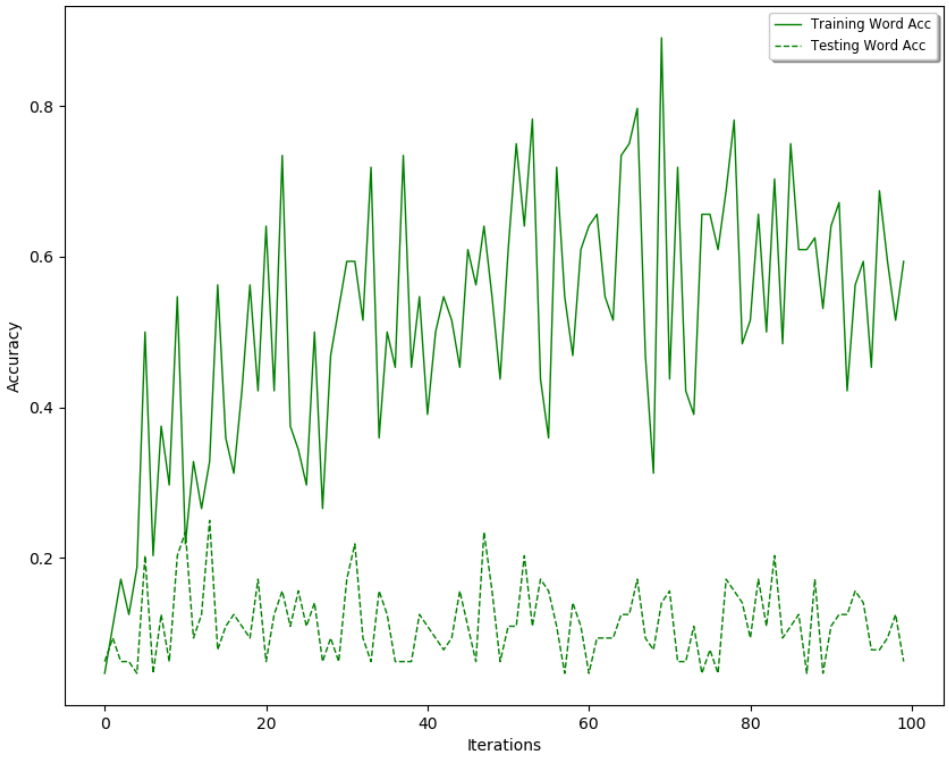
(1) **letter-wise prediction accuracy**,



(2) **word-wise prediction accuracy**.

Average letter accuracy for training set is 88.3%

Average letter accuracy for test set is 63.1%



Average word accuracy for training set is 51.4%

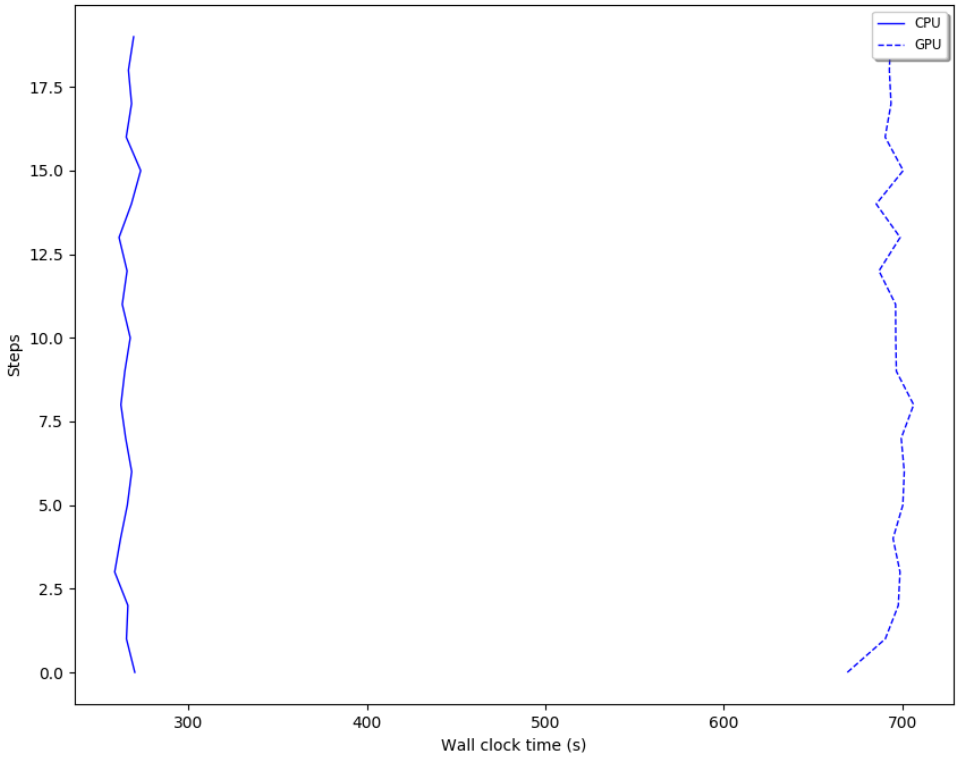
Average word accuracy for test set is 11.4%

Our accuracies fluctuates a little bit between iterations.

1. This could mean that our model is subjective to small noises. And on test set, if the accuaracy fluctuates, it usually means that our model is overfitting. We've played with hyper-parameters (e.g., learning rate and value of C) to try to increase the regularization. However, the result still fluctuates.
2. We've also noticed that the dataset we were given include some words that were repetitive. That could also cause the unstableness of our model.

Comparing 4b and 4c, the accuracies are pretty similar. It's very like both of the models in 4b and 4c are overfitting and therefore, adding an extra layer in 4c did not improve accuracies a lot.

(4d) **(10 points)** Enable GPU in your implementation. Does it lead to significant speedup? You can test on the network in 4c. Make sure your plot uses wallclock time as the horizontal axis.



The GPU isn't much faster than CPU. In our case, GPU is even slower than CPU. We think it's first because that function dp_infer() in inference.py is quite slow and not optimal for GPU. Second, we think somewhere in our code is using CPU that is not compatible with the GPU we were using, which caused communication problem between CPU and GPU.

We also noticed that using matrix operations instead of for loop in dp_infer() could speed up the code.

Q5) Comparisons with Deep learning Models

a) Designed simple custom LeNet, and also used Resnet50:

LeNet model: We implemented a custom LeNet with 2 convolution layers and Max pooling with 3*3 filters and 2 fully connected layers (6 layers) while maintaining the 3 channels.

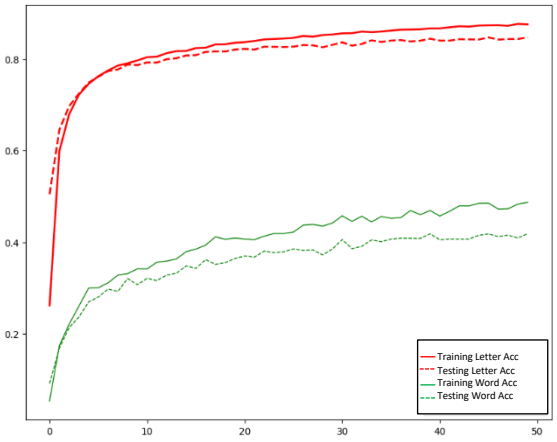
ResNet model: The main reason to use ResNet architecture is for the skip connections which help with vanishing gradient problem. Further information provided in the last section.

b) Plotting Letter wise and Word wise accuracies:

Plot of Letter wise and Word wise accuracies for ResNet and LeNet respectively:

LeNet Plots:

Adam



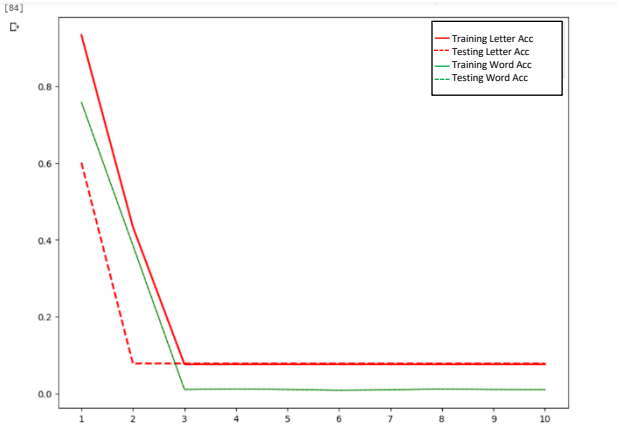
Training Letter Accuracy Peak: 87.67%

Testing Letter Accuracy Peak: 84.73%

Training Word Accuracy Peak: 48.7%

Testing Word Accuracy Peak: 41.8%

LBGFS



Training Letter Accuracy Peak: 96.78%

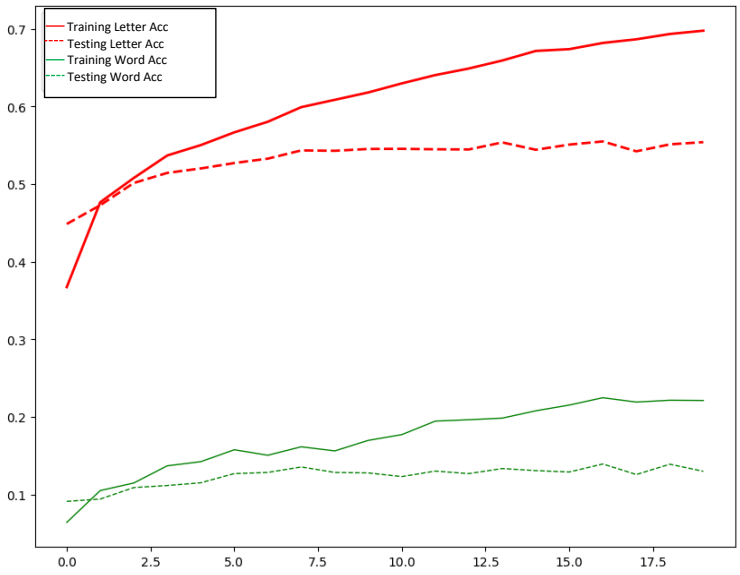
Testing Letter Accuracy Peak: 69.4%

Training Word Accuracy Peak: 78.9%

Testing Word Accuracy Peak: 37.4%

ResNet Plots:

Adam



Structure of results in notebook (LBFGS)

```
  Processing epoch 0
Batch= 0
Letter accuracy = tensor(0.4492, dtype=torch.float64)
Batch= 25
Letter accuracy = tensor(0.9922, dtype=torch.float64)
Batch= 50
Letter accuracy = tensor(0.9883, dtype=torch.float64)
Batch= 75
Letter accuracy = tensor(0.2969, dtype=torch.float64)
Batch= 100
Letter accuracy = tensor(0.9961, dtype=torch.float64)
Training acc = : tensor(0.9607, dtype=torch.float64)
Testing acc = : tensor(0.7010, dtype=torch.float64)
Batch= 0
```

Training Letter Accuracy Peak: 70.4%

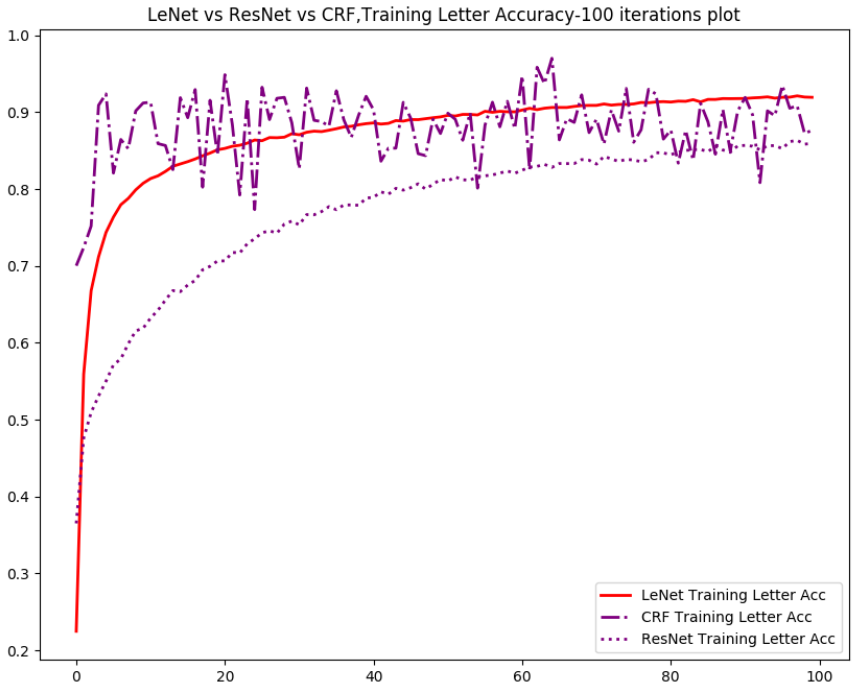
Testing Letter Accuracy Peak: 55.01%

Training Word Accuracy Peak: 23.5%

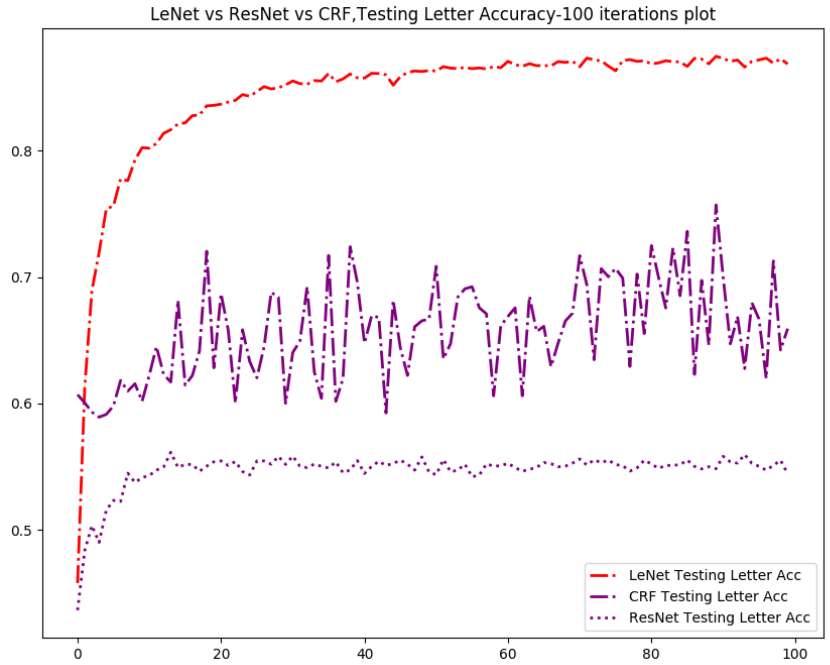
Testing Word Accuracy Peak: 13.5%

Some Comparison sub Plots with our LeNet, ResNet CRF model (100 iterations):

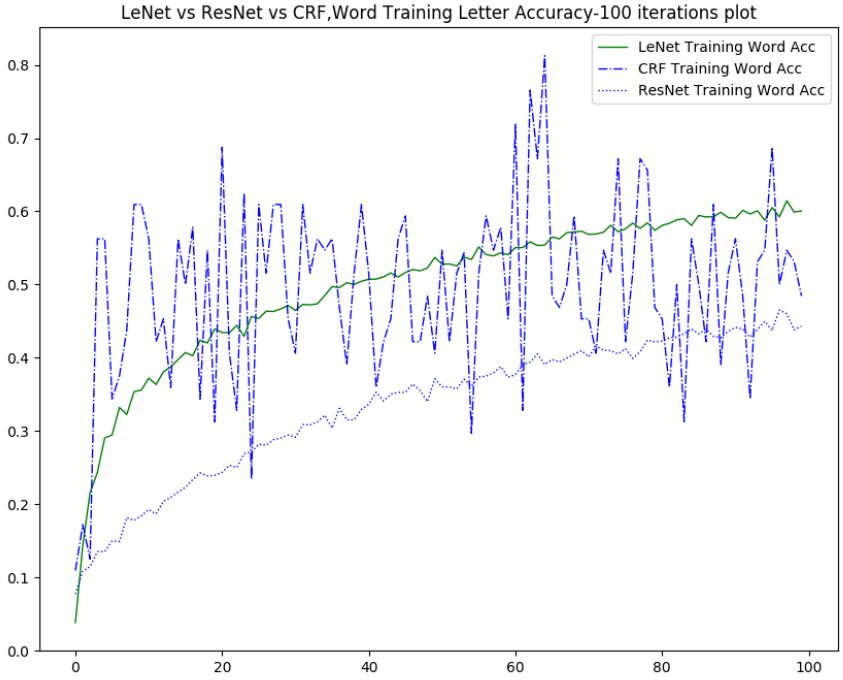
Training Letter wise comparison:



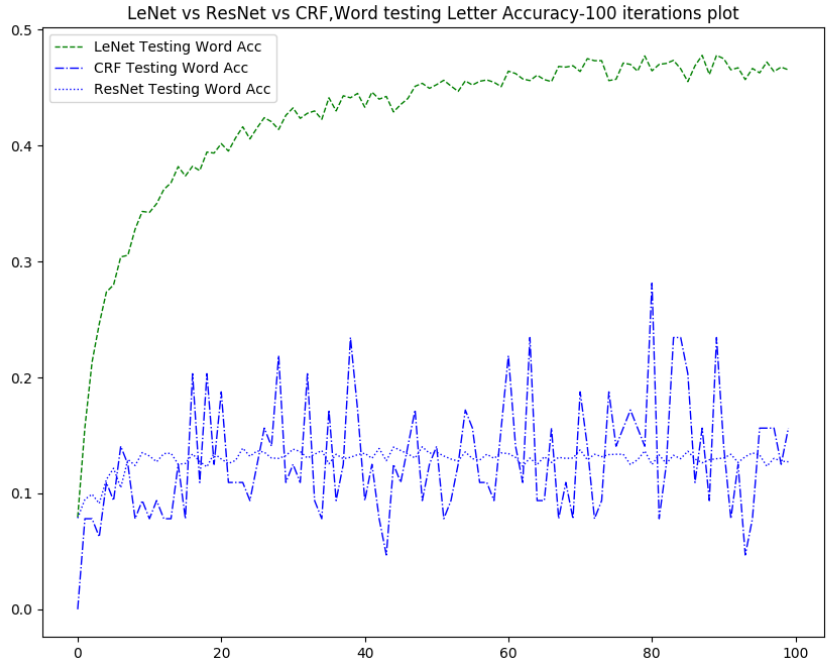
Testing Letter wise comparison:



Training Word wise comparison:



Testing Word wise comparison:



Observations: Our Custom LeNet performed better compared to ResNet since there is no extra padding, or upscaling to fit the 224*224 requirement of input size ResNet model.

Surprisingly LeNet outperforms CRF as well with word accuracies getting close to 50%.

CRF has some fluctuations mainly due to the small noises changing model heavily. Changing hyper parameters, (learning rate, c) and some regularizations didn't smoothen the results enough.

At around epoch 65-70, we have around 97% training letter accuracy and around 80% testing letter accuracy for CRF model. Overfitting can be avoiding in future.

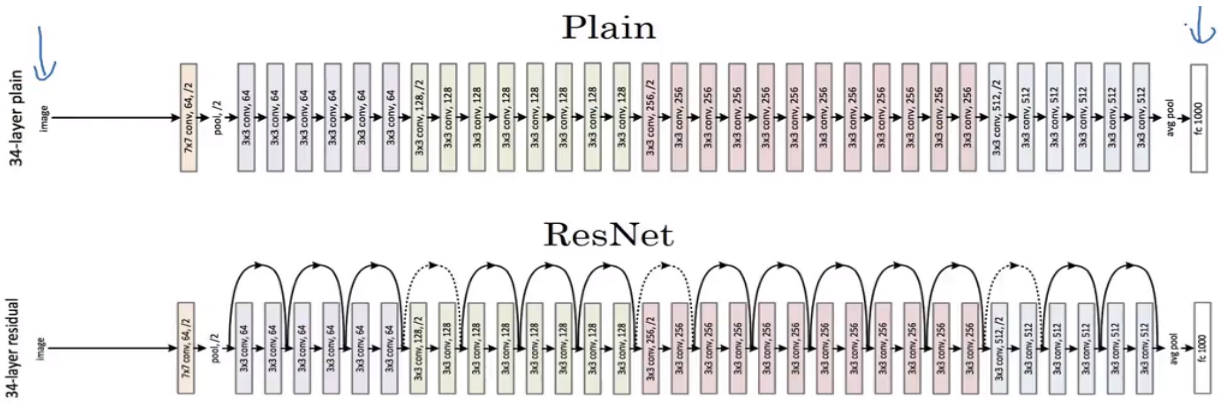
c) Comparing Letter wise and Word wise accuracies with ADAM and LBFGS:

Plots of Letter wise and Word wise accuracies for ResNet and LeNet respectively were shows above. ResNet plot of LBGFS was taking lot of time for each iteration. Adam finds better solution way faster and also converges faster. LBFGS solver is a true quasi-Newton method in that it estimates the curvature of the parameter space via an approximation of the Hessian. With larger input space, the second order calculations of LBFGS get pretty computationally heavy. That's the reason for the slow iteration speed.

d) Why ResNet and LeNet? Design and Implementation of models ResNet and LeNet:

ResNet:

We chose ResNet mainly to understand the skip connections and how it deals with Vanishing gradient. With networks going deeper, the issue of vanishing gradient becomes much more crucial. Therefore it is important to dig into the issues and how models like ResNet solve those issues.



Compared to normal plain network, ResNet uses skip connections. They help is two things:
Mitigating Vanishing Gradient
Helps model learn identity function which ensures that higher layer will perform at least as good as the lower layer, and not worse.
The ResNet-50 model consists of 5 stages each with a convolution and Identity block. Each convolution block has 3 convolution layers and each identity block also has 3 convolution layers. The ResNet-50 has over 23 million trainable parameters. Drop used in ResNet- 50 is 0.7x, 70% drop chance.

LeNet: We used a simple LeNet-5 which has 2 convolutions and 2 maxpool layers. Followed by 3 fullyconnected layers. We maintain same number of channels and pad it accordingly. Even though LeNet is simplest deep learning model out there, it does a great job for dataset with small features like out 16*8 input.