ECE 763 Project 03 Report

Jianxun WANG jwang75

0. Highlight

In this project, I explored the approach to widen the structure of convolutional neural network. Specifically, the proposed architecture applies kernels with different sizes instead of single size kernels. The detailed architecture is illustrated in the graph below.

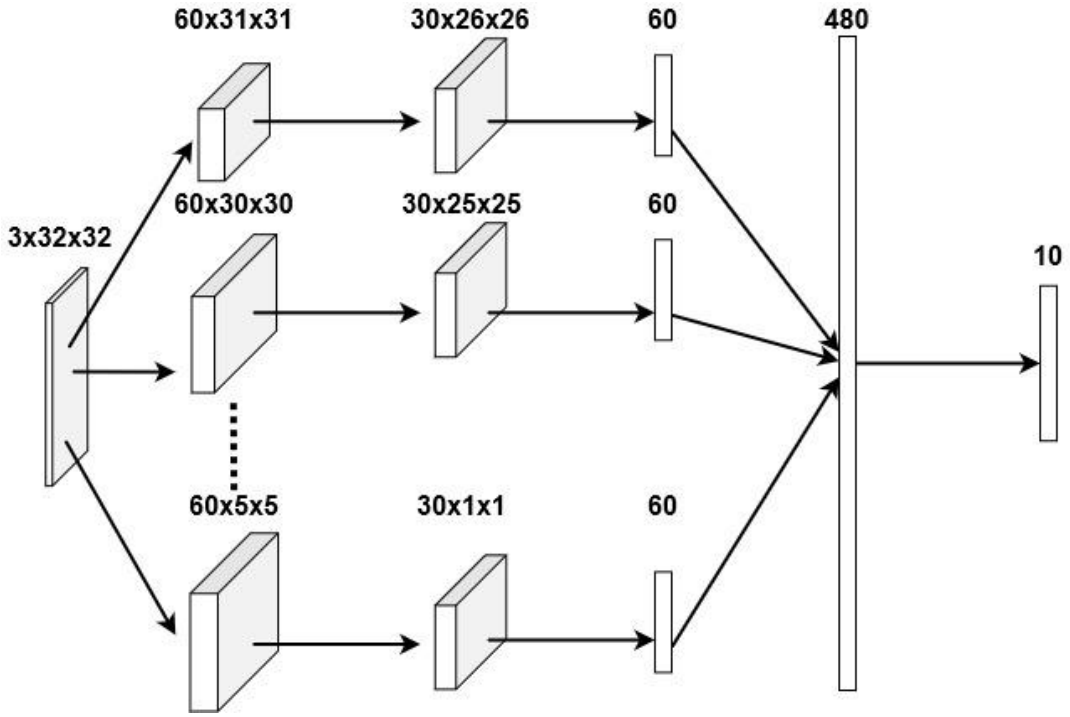


Fig 1. Proposed architecture

The architecture consists of two convolutional layers. In each layer, there are 26 sets of kernels. As a result, the input images are directed into 26 streams before being merged to make final prediction. Kernels in the first convolutional layer have size from 1 to 27 and kernels in the second layer has fixed size as 5. Because of different kernel sizes in the first convolutional layer, the outputs from the second convolutional layer have different sizes. Directly flattening such output can lead to unbalanced influence from different kernel upon final prediction. To avoid this, the outputs from convolutional layers of each stream are flattened and fed to equal size fully connected layers before being merged. Equal size flattened output from each stream are merged and directed to a final fully connected layer and then the output classification layer.

1. Dataset and preprocessing

In this project, I experimented on CIFAR-10 and reference face dataset provided by professor. CIFAR-10 dataset contains RBG images with 3 channels in 32*32 format and pixel value ranges from 0 to 255. The face dataset contains BMP images with 1 channel in 16*16 format. Its pixel value has the same range as CIFAR-10. Images from both sources are scaled into 0 to 1 range before

preprocessing, in other words divide by 255. Images from the face dataset is resized into 32*32 using bilinear interpolation before additional preprocessing.

In this project, I explored different techniques to preprocess the data and examine their effects using CIFAR-10 dataset. I included normalization for each channel and global contrast normalization for each image. I also examined the effect of ZCA whitening after normalization. The matrix and mean for ZCA whitening are computed using entire training dataset.

2. Additional Model Details and Implementation

I experimented two variances of proposed architecture. The first one utilizes the RELU activation function with size 2*2 and stride 1 max pooling following both convolutional layers. The second one utilizes the Maxout activation function (https://arxiv.org/pdf/1302.4389.pdf) along the 1-st dimension of input data. It is kernel dimension for convolutional layer and feature dimension for fully connected layer. Both variances use Softmax as activation function for final model output.

The model, preprocessing and data loader for face data are implemented in python using PyTorch, implementing corresponding interface. Details of each folder and file are listed as follow:

- Folder:
  - *data*: containing CIFAR-10 data and face data.
  - *models*: containing model architecture and maxout implementation.
    - *base_model.py*: implementation of model architectures.
    - *custom_layer.py*: implementation of custom layers and activations such as Maxout.
  - *utils*: utilities such as training, prediction, preprocessing, etc.
    - *data.py*: class to load face dataset for PyTorch's data loader.
    - *loss_function.py*: implementation of loss function, e.g. categorical cross entropy loss for softmax activation function.
    - *model_util.py*: utilities for models, such as training, prediction, hyperparameter search.
    - *performance.py*: performance evaluation functions.
    - *preprocessing.py*: class implementing preprocessing transforms for data set.
- *cifar_experiment.ipynb*: experiment platform for hyperparameter search and final model training using CIFAR-10 dataset.
- *cifar_experiment2.ipynb*: experiment platform for comparing different preprocessing techniqus.
- *face_experiment.ipynb*: experiment platform for face dataset.

3. Training and hyperparameter search

For CIFAR-10 and face data, the training data is split in 8:2 ration where 80% data is used for

training and 20% data is used as validation for hyperparameter search. Training process uses Adam optimizer to update model weights. 4 hyperparameters are included in training search: learning rate, regularization factor, number of steps for learning rate decay and ratio of learning rate decay. However, due to incorrect setting of learning rate decay, the search was not performed in desired ration and there was no sufficient time to perform additional hyperparameter search. As a result, only learning rate and regularization factor is considered.

4. Result

4.1. Model training babysitting

Following graphs show the babysitting process for one of model training. Due to space limit and repeated process, I cannot show the process for all preprocessing and architectures.

```
#split validation set from training set
smallset = Subset(trainset, np.arange(200))

train_length = len(trainset)
train_index = np.arange(int(train_length*0.8))
valid_index = np.arange(int(train_length*0.8), train_length)

validset = Subset(trainset, valid_index)
trainset = Subset(trainset, train_index)

smallloader = DataLoader(smallset, batch_size = 4, shuffle=False, num_workers=4, pin_memory =True)
trainloader = DataLoader(trainset, batch_size=250, shuffle=True, num_workers=4, pin_memory =True)
validloader = DataLoader(validset, batch_size=250, shuffle=False, num_workers=4, pin_memory =True)
testloader = DataLoader(testset, batch_size=250, shuffle=False, num_workers=4)

classes = ('plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck')
```

```
Files already downloaded and verified
Files already downloaded and verified
```

```
[3]: #initialize model, optimizer, etc
model = MagicModel1((3, 32, 32), 10)
loss_function = ProbCatCrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=1e-4, weight_decay=0.0) #, momentum=0.9
lr_scheduler = optim.lr_scheduler.StepLR(optimizer, 10, gamma=1)
```

```
[5]: #training
#history = fit(model, trainLoader, optimizer, loss_function, lr_scheduler, epochs=300, valid_loader = validloader, measure = ['accuracy'], verbose = 1)
history = fit(model, smallloader, optimizer, loss_function, lr_scheduler, epochs=40, measure = ['accuracy'])
```

```
Epoch 27: 100% 50/50 [00:04<00:00, 10.74it/s, training_loss=0.000582, train_accuracy=1]
Epoch 28: 100% 50/50 [00:04<00:00, 11.06it/s, training_loss=0.000538, train_accuracy=1]
Epoch 29: 100% 50/50 [00:04<00:00, 10.98it/s, training_loss=0.000497, train_accuracy=1]
Epoch 30: 100% 50/50 [00:04<00:00, 10.81it/s, training_loss=0.00046, train_accuracy=1]
Epoch 31: 100% 50/50 [00:04<00:00, 10.87it/s, training_loss=0.000426, train_accuracy=1]
Epoch 32: 100% 50/50 [00:04<00:00, 10.79it/s, training_loss=0.000396, train_accuracy=1]
Epoch 33: 100% 50/50 [00:04<00:00, 10.48it/s, training_loss=0.000368, train_accuracy=1]
Epoch 34: 100% 50/50 [00:04<00:00, 10.92it/s, training_loss=0.000343, train_accuracy=1]
Epoch 35: 100% 50/50 [00:04<00:00, 10.81it/s, training_loss=0.000319, train_accuracy=1]
Epoch 36: 100% 50/50 [00:04<00:00, 11.27it/s, training_loss=0.000298, train_accuracy=1]
Epoch 37: 100% 50/50 [00:04<00:00, 10.48it/s, training_loss=0.000278, train_accuracy=1]
Epoch 38: 100% 50/50 [00:04<00:00, 10.52it/s, training_loss=0.00026, train_accuracy=1]
Epoch 39: 100% 50/50 [00:04<00:00, 10.43it/s, training_loss=0.000244, train_accuracy=1]
```

Fig 2. Overfitting a small dataset

```
[3]: #initialize model, optimizer, etc
model = MagicModel1((3, 32, 32), 10)
loss_function = ProbCatCrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=1e-4, weight_decay=0.0) #, momentum=0.9
lr_scheduler = optim.lr_scheduler.StepLR(optimizer, 10, gamma=1)
```

```
[4]: #training
history = fit(model, trainloader, optimizer, loss_function, lr_scheduler, epochs=3, valid_loader = validloader, measure = ['accuracy'], verbose = 1)
#history = fit(model, smallloader, optimizer, loss_function, lr_scheduler, epochs=40, measure = ['accuracy'])
```

```
Epoch 0:   1% 1/160 [00:00<01:59,  1.33it/s, training_loss=2.3, train_accuracy=0.08]
/home/jwang75/tmp/workspace/models/base_model.py:70: UserWarning: Implicit dimension choice for softmax has been deprecated. Change the call to include
  x = F.softmax(self.fc3(x))
Epoch 0: 100% 160/160 [00:28<00:00,  5.63it/s, training_loss=1.37, train_accuracy=0.509, val_loss=1.12, val_accuracy=0.606]
Epoch 1: 100% 160/160 [00:28<00:00,  5.66it/s, training_loss=0.978, train_accuracy=0.658, val_loss=0.964, val_accuracy=0.659]
Epoch 2: 100% 160/160 [00:28<00:00,  5.63it/s, training_loss=0.822, train_accuracy=0.714, val_loss=0.889, val_accuracy=0.692]
```

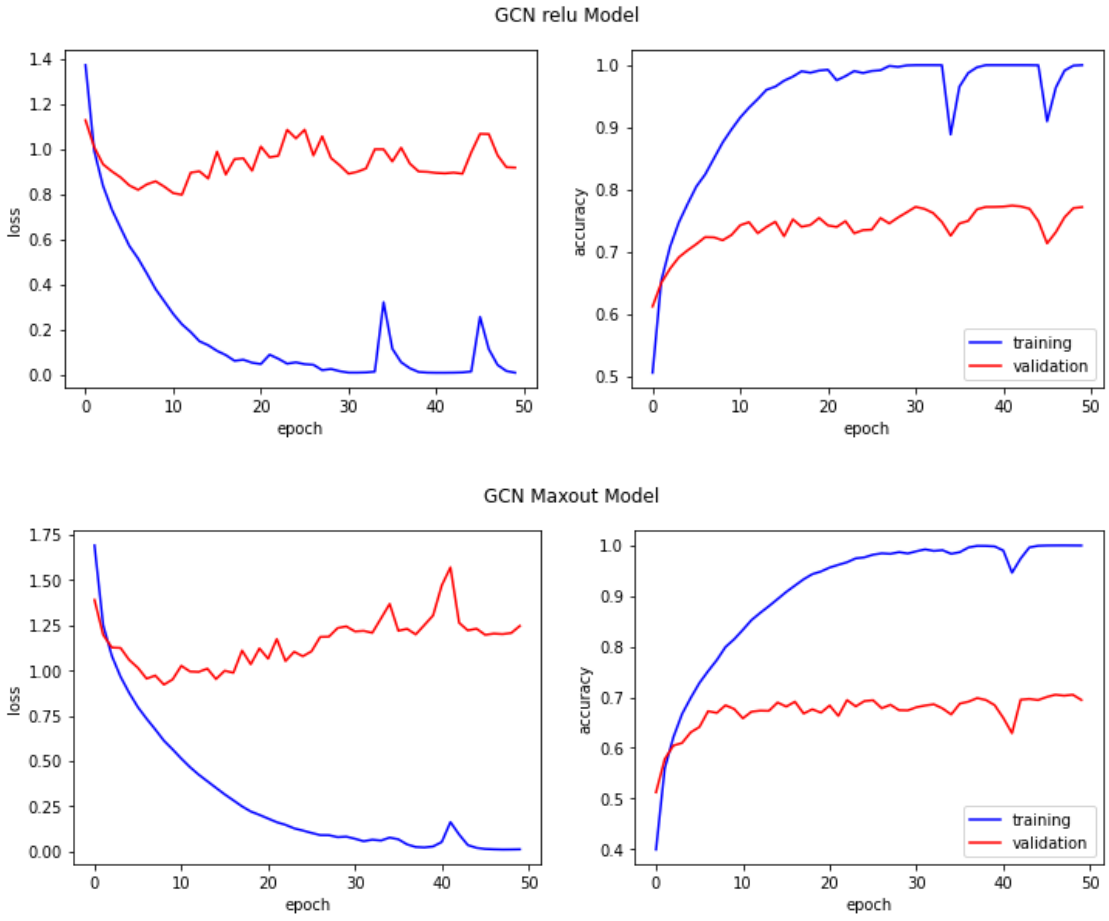
Fig 3. Training with no regularization

```
[3]: #initialize model, optimizer, etc
model = MagicModel1((3, 32, 32), 10)
loss_function = ProbCatCrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=1e-4, weight_decay=1e3) #, momentum=0.9
lr_scheduler = optim.lr_scheduler.StepLR(optimizer, 10, gamma=1)

[4]: #training
history = fit(model, trainloader, optimizer, loss_function, lr_scheduler, epochs=3, valid_loader = validloader, measure = ['accuracy'], verbose = 1)
#history = fit(model, smallloader, optimizer, loss_function, lr_scheduler, epochs=40, measure = ['accuracy'])

Epoch 0:   1% 1/160 [00:00<02:04,  1.28it/s, training_loss=2.31, train_accuracy=0.076]
/home/jwang75/tmp/workspace/models/base_model.py:70: UserWarning: Implicit dimension choice for softmax has been deprecated. Change the call to include d
  x = F.softmax(self.fc3(x))
Epoch 0: 100% 160/160 [00:29<00:00,  5.39it/s, training_loss=2.3, train_accuracy=0.0905, val_loss=2.3, val_accuracy=0.0953]
Epoch 1: 100% 160/160 [00:29<00:00,  5.41it/s, training_loss=2.3, train_accuracy=0.0954, val_loss=2.3, val_accuracy=0.101]
Epoch 2: 100% 160/160 [00:29<00:00,  5.40it/s, training_loss=2.3, train_accuracy=0.0997, val_loss=2.3, val_accuracy=0.101]
```

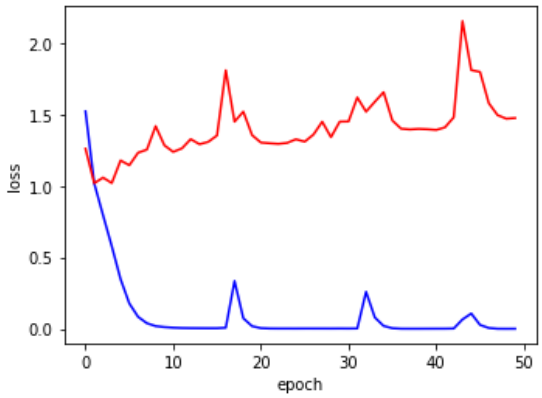
Fig 4. Training with huge regularization, model barely learning

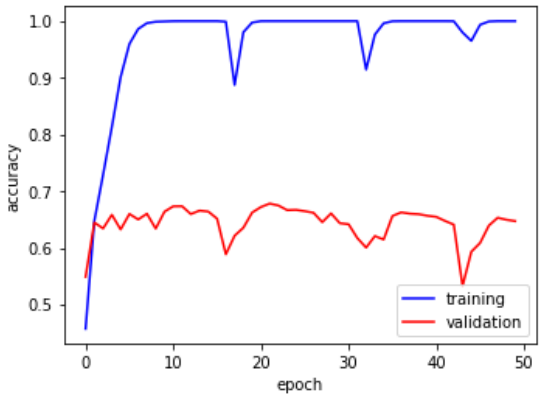
4.2. Preprocessing and Model architecture Result

Following graphs show the learning curve of different preprocessing techniques in CIFAR-10.

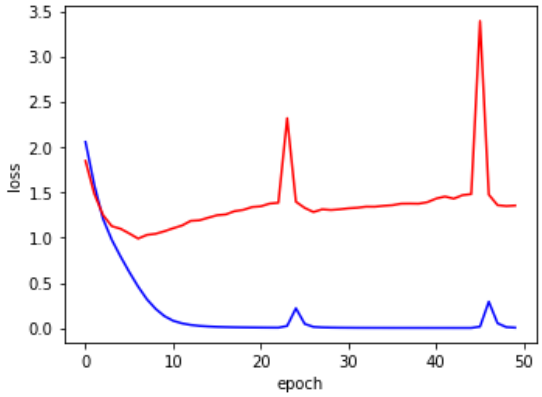


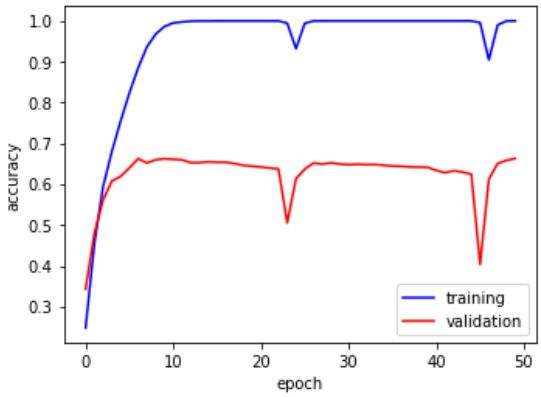
GCN ZCA relu Model



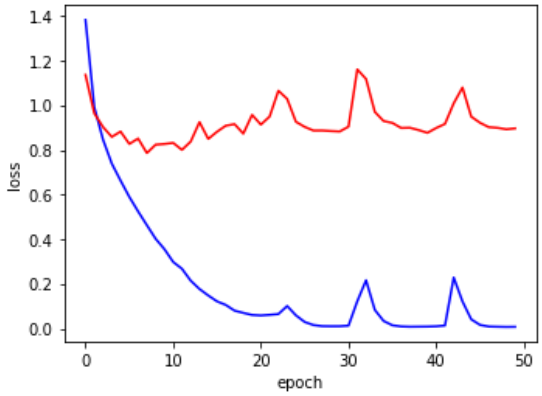


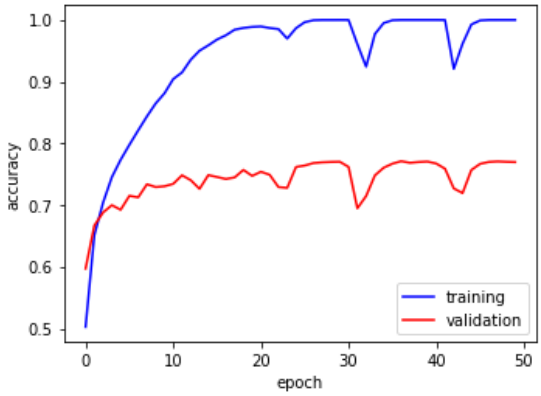
GCN ZCA Maxout Model

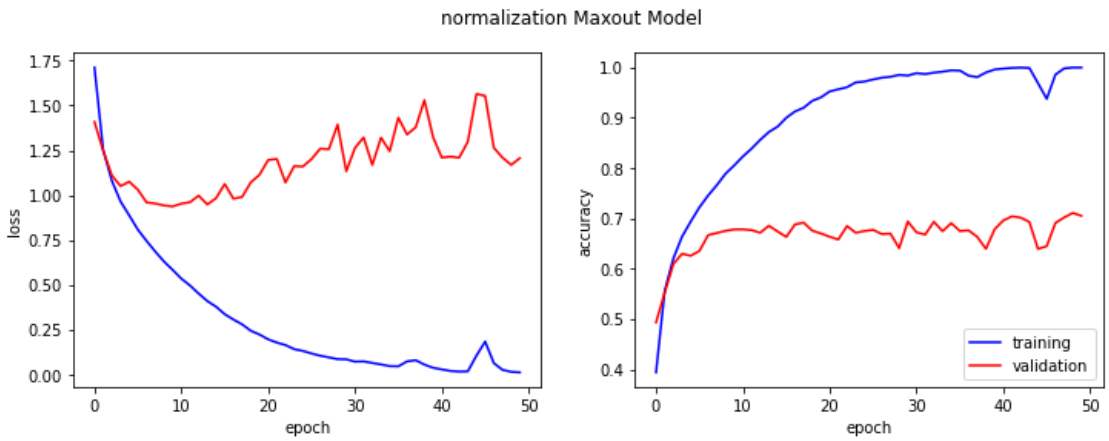


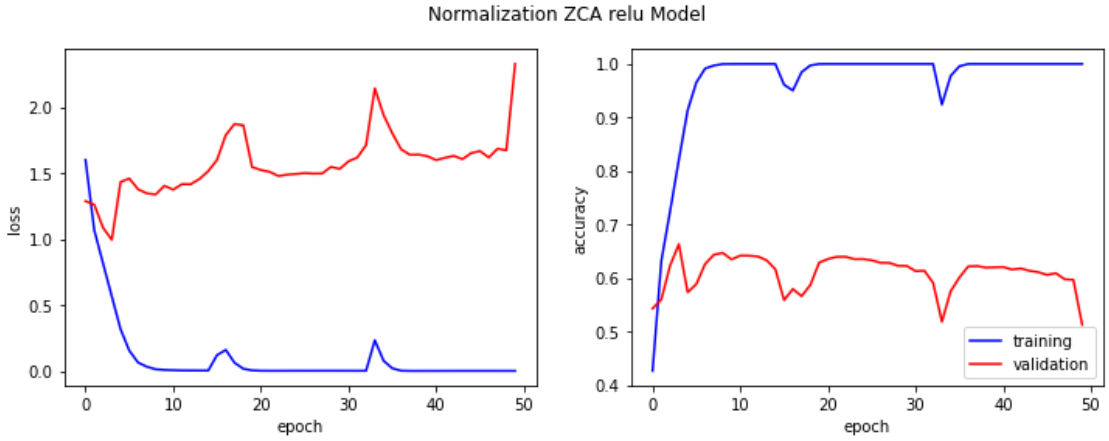


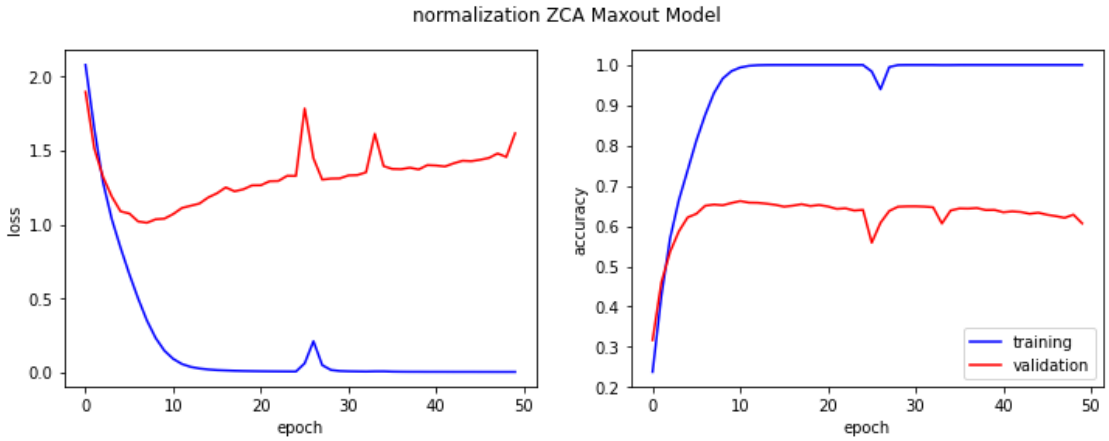
Normalization relu Model











From the learning curve, there are several observations:

1. ZCA whitening following both normalization techniques significantly reduce the model performance, as shown from learning curve.
2. Traditional normalization and Global Contrast Normalization show similar performance. This in fact is desirable since GCN can be done image-base.
3. Maxout layer reduces performance comparing to RELU+maxpooling.

4.3. Hyperparameter Search

Here I show the example of hyperparameter search and result for the best model as traditional normalization with RELU+maxpooling

```
(25: val_acc: 0.77220, lr: 1.34117e-04, reg: 1.47415e-04, lr_steps: 38, lr_gamma: 1.43970e-02)
(26: val_acc: 0.76860, lr: 6.74624e-05, reg: 1.42014e-03, lr_steps: 48, lr_gamma: 1.65726e-02)
(27: val_acc: 0.67480, lr: 2.95777e-06, reg: 6.81093e-02, lr_steps: 44, lr_gamma: 5.10792e-01)
(28: val_acc: 0.51610, lr: 3.97115e-06, reg: 1.90767e-01, lr_steps: 10, lr_gamma: 1.00334e-01)
(29: val_acc: 0.75650, lr: 5.46532e-04, reg: 3.02253e-02, lr_steps: 24, lr_gamma: 9.03501e-02)
(30: val_acc: 0.38460, lr: 1.05878e-06, reg: 4.00001e-04, lr_steps: 3, lr_gamma: 3.58233e-02)
(31: val_acc: 0.72300, lr: 5.61766e-06, reg: 2.29735e-03, lr_steps: 2, lr_gamma: 9.66783e-01)
(32: val_acc: 0.67330, lr: 1.72048e-05, reg: 1.61840e-01, lr_steps: 22, lr_gamma: 9.54779e-02)
(33: val_acc: 0.71120, lr: 4.12747e-06, reg: 3.00052e-04, lr_steps: 25, lr_gamma: 4.85622e-01)
(34: val_acc: 0.34130, lr: 1.18954e-05, reg: 6.03379e-01, lr_steps: 1, lr_gamma: 3.46537e-01)
(35: val_acc: 0.58260, lr: 1.28216e-06, reg: 1.95420e-04, lr_steps: 19, lr_gamma: 2.10231e-02)
(36: val_acc: 0.66460, lr: 5.27071e-06, reg: 1.03606e-03, lr_steps: 9, lr_gamma: 2.78480e-01)
(37: val_acc: 0.65440, lr: 1.19982e-06, reg: 2.57636e-05, lr_steps: 49, lr_gamma: 2.95294e-02)
(38: val_acc: 0.77050, lr: 6.80299e-05, reg: 2.49956e-03, lr_steps: 23, lr_gamma: 1.08357e-02)
(39: val_acc: 0.69670, lr: 2.02526e-04, reg: 3.93852e-03, lr_steps: 1, lr_gamma: 1.91458e-01)
(40: val_acc: 0.77610, lr: 1.14522e-04, reg: 2.64603e-04, lr_steps: 26, lr_gamma: 3.06112e-02)
(41: val_acc: 0.67460, lr: 1.98278e-06, reg: 2.21246e-02, lr_steps: 42, lr_gamma: 3.24996e-01)
(42: val_acc: 0.68530, lr: 2.98522e-06, reg: 3.81403e-02, lr_steps: 44, lr_gamma: 6.81405e-02)
(43: val_acc: 0.35350, lr: 5.66565e-06, reg: 9.03141e-01, lr_steps: 8, lr_gamma: 1.84275e-01)
(44: val_acc: 0.29060, lr: 4.41164e-04, reg: 7.41320e-01, lr_steps: 44, lr_gamma: 3.08155e-02)
(45: val_acc: 0.77330, lr: 1.89575e-04, reg: 1.64972e-05, lr_steps: 5, lr_gamma: 6.52453e-02)
(46: val_acc: 0.73270, lr: 7.70380e-06, reg: 1.60672e-03, lr_steps: 38, lr_gamma: 3.68737e-02)
(47: val_acc: 0.76280, lr: 1.55398e-04, reg: 1.24643e-02, lr_steps: 48, lr_gamma: 8.35699e-01)
(48: val_acc: 0.52260, lr: 2.47380e-06, reg: 1.10864e-05, lr_steps: 5, lr_gamma: 7.65140e-02)
(49: val_acc: 0.68540, lr: 3.45489e-06, reg: 3.32638e-02, lr_steps: 32, lr_gamma: 1.04330e-01)
(50: val_acc: 0.66650, lr: 1.35969e-05, reg: 1.49578e-01, lr_steps: 26, lr_gamma: 1.37802e-02)
(51: val_acc: 0.77670, lr: 5.57660e-04, reg: 3.17040e-04, lr_steps: 13, lr_gamma: 1.22398e-02)
(52: val_acc: 0.48900, lr: 2.58534e-06, reg: 4.27637e-03, lr_steps: 4, lr_gamma: 1.08753e-02)
(53: val_acc: 0.43280, lr: 3.50967e-04, reg: 3.60632e-01, lr_steps: 10, lr_gamma: 9.46335e-01)
(54: val_acc: 0.75180, lr: 4.97771e-05, reg: 5.88852e-05, lr_steps: 8, lr_gamma: 1.64367e-01)
(55: val_acc: 0.71870, lr: 6.43495e-05, reg: 7.43638e-02, lr_steps: 37, lr_gamma: 5.34274e-01)
(56: val_acc: 0.41430, lr: 6.40393e-04, reg: 2.60100e-01, lr_steps: 46, lr_gamma: 1.63243e-02)
(57: val_acc: 0.77050, lr: 4.87089e-04, reg: 6.13752e-05, lr_steps: 36, lr_gamma: 3.67919e-02)
(58: val_acc: 0.69240, lr: 3.73240e-06, reg: 9.97423e-05, lr_steps: 26, lr_gamma: 6.44554e-02)
(59: val_acc: 0.74930, lr: 4.34365e-05, reg: 6.71799e-05, lr_steps: 18, lr_gamma: 1.20951e-02)
```

Fig 5. Part of hyperparameter search output

```
[ ]: #load hyperparameter search result
with open('hyper1.pickle', 'rb') as handle:
    result = pickle.load(handle)
```

```
[29]: #show top 20 hyperparameter search result
for i, item in enumerate(zip(*result.values())):
    print("{} - val_acc: {}, lr: {}, reg: {}, steps: {}, gamma: {}".format(i, *item))
```

```
0 - val_acc: 0.77510003681527 45, lr: 0.0001870958988208472, reg: 0.0011999674755967396, steps: 43.0, gamma: 0.07829314721109162
1 - val_acc: 0.7747000367962755, lr: 0.0001401711817078652 8, reg: 1.686140121257603e-05, steps: 34.0, gamma: 0.10306160920150186
2 - val_acc: 0.7722000366775319, lr: 0.0001341174320613411, reg: 0.0001474154699057607, steps: 38.0, gamma: 0.014396985968001763
3 - val_acc: 0.7497000356088392, lr: 3.4303585106964615e-05, reg: 0.0006977337511121456, steps: 21.0, gamma: 0.8944505364121259
4 - val_acc: 0.7705000365967862, lr: 6.802994294189812e-05, reg: 0.0024995605211921247, steps: 23.0, gamma: 0.01083568908118864
5 - val_acc: 0.7493000355898403, lr: 4.343649952067692e-05, reg: 6.717993909671879e-05, steps: 18.0, gamma: 0.012095104321666767
6 - val_acc: 0.7590000360505655, lr: 0.0002940760588961017, reg: 0.0051610332836317975, steps: 35.0, gamma: 0.8561671024962839
7 - val_acc: 0.7683000364922918, lr: 0.0001039727352859999 9, reg: 2.6564375838058756e-05, steps: 13.0, gamma: 0.022319369265557998
8 - val_acc: 0.7639000362833303, lr: 0.0007495597658619 17, reg: 8.114136396585245e-05, steps: 39.0, gamma: 0.015208543769396104
9 - val_acc: 0.7768000368960202, lr: 0.0001427047698329951 3, reg: 0.005488660275794192, steps: 47.0, gamma: 0.024602848476209565
10 - val_acc: 0.7733000367297791, lr: 0.0001895749361863395, reg: 1.64972170397767e-05, steps: 5.0, gamma: 0.06524526944953266
11 - val_acc: 0.7529000357608311, lr: 0.0007299296350243092, reg: 0.0029948816087141806, steps: 44.0, gamma: 0.7545122451134154
12 - val_acc: 0.7686000365065411, lr: 6.746240209870068e-05, reg: 0.0014201413594279232, steps: 48.0, gamma: 0.016572603130797125
13 - val_acc: 0.7761000368627 72, lr: 0.0001145219160267444 7, reg: 0.0002646031867713716, steps: 26.0, gamma: 0.030611228842793006
14 - val_acc: 0.7761000368627 72, lr: 0.0002046842324517876 2, reg: 6.038360875609636e-05, steps: 25.0, gamma: 0.06630645051108598
15 - val_acc: 0.7565000359318219, lr: 0.0005465319766762126, reg: 0.03022527150889816, steps: 24.0, gamma: 0.09035013299406805
16 - val_acc: 0.7628000362310559, lr: 0.0001553983093319071, reg: 0.012464255989518625, steps: 48.0, gamma: 0.8356991358918261
17 - val_acc: 0.7518000357085839, lr: 4.977707127217288e-05, reg: 5.888515316652843e-05, steps: 8.0, gamma: 0.16436670290126587
18 - val_acc: 0.7767000368912704, lr: 0.0005576599473493 18, reg: 0.0003170399000662958, steps: 13.0, gamma: 0.012239826689084455
19 - val_acc: 0.7705000365967862, lr: 0.0004870891780012 22, reg: 6.137520059830022e-05, steps: 36.0, gamma: 0.03679186047748997
```

Fig 6. Top 20 result from hyperparameter search

From hyperparameter search result, I set learning rate as 1.5e-4 and regularization factor as 1e-3. The following graph is the learning curve for this model. Its test accuracy is 0.7758.

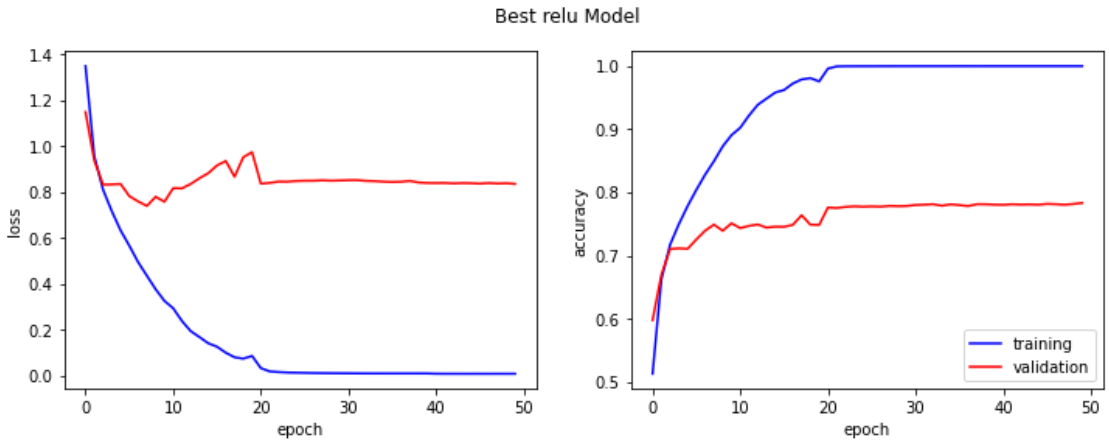


Fig 7. Learning curve of model with traditional normalization and RELU+maxpooling

4.4. Face dataset result

For face dataset, I cannot perform hyperparameter search because the initial attempt shows test accuracy as 0.9978. The model fit both training data and validation data in less than 10 epochs. The potential reason can be the small size of entire dataset comparing to CIFAR-10. In addition, binary classification is much easier comparing to multi-label classification. Finally, it appears the face features are much more salient comparing to non-face data, as suggested from Project 02.