Machine Learning Final Project

Team#4

Members: 徐福臨(0416074), 吳承翰(0316084), 劉哲豪(0416032)

Topic – Naruto Boruto Noruto Baruto

- Task:

  To classify comic graphs between Naruto and Boruto
- Key words:

  Feature extracting, Classification, Model design, Optimization
- About Naruto and Boruto:

  Naruto is a famous comic created by Kishimoto Masashi and the story is ended in 2014. Boruto is the comic which continue the story and is created by Ikemoto Mikio, the assistant of Masashi for 15 years.

Implement

- Data Acquisition:

  We take screenshot from the online comic website.

  We expect our model can focus on the "style" (features) to do classification. Therefore, the data we collect should have clear outline, not be landscape picture and not include dialogue, since landscapes in Naruto are drawn by Mikio and dialogues are nothing to do with "style". In addition, we do not collect pictures focusing on specific characters, since we do not want our models to do classification by "remembering" which characters belongs to which comics.
- Data Preprocessing:
  - For training data

    There are 590 data for training (about 50% - 50% for Naruto and Boruto). We divide the RGB value by 255 to make the initialization of the model easier. The size of screenshot is (length = 210, width = 210), we rescale it to (192, 192) by interpolation. Lastly, every data is flipped horizontally and then add to the training set so there are 1180 data in total.
  - For testing data

    We choose 50 data for testing (50% - 50% for Naruto and Boruto) based on our rules. The rules for choosing testing data:

    1. Characters should rarely or never appear in training set
    2. It is the different angle which do not appear in training set of the character
    3. It is the different costume which do not appear in training set of the character
    4. It has noisy effects added to the picture

The preprocessing method is the same as training data except horizontal flipping.

- Model#1 (ResNet):

The first model we implemented is ResNet, since we desire its powerful ability to extract features. When it comes to neural network, we prefer deep and narrow architecture. However, there are some problems when the network become deeper – the gradient vanishing problem, and some hidden layers are just mapping identically from the input.

The solution:

The hidden layer in the original network is:

Input: X, Output: H(X)

We do not know what H(X) will become when the hidden layer is converge, so we cannot ensure what the hidden layer will learn, may be identical mapping or usually worse.

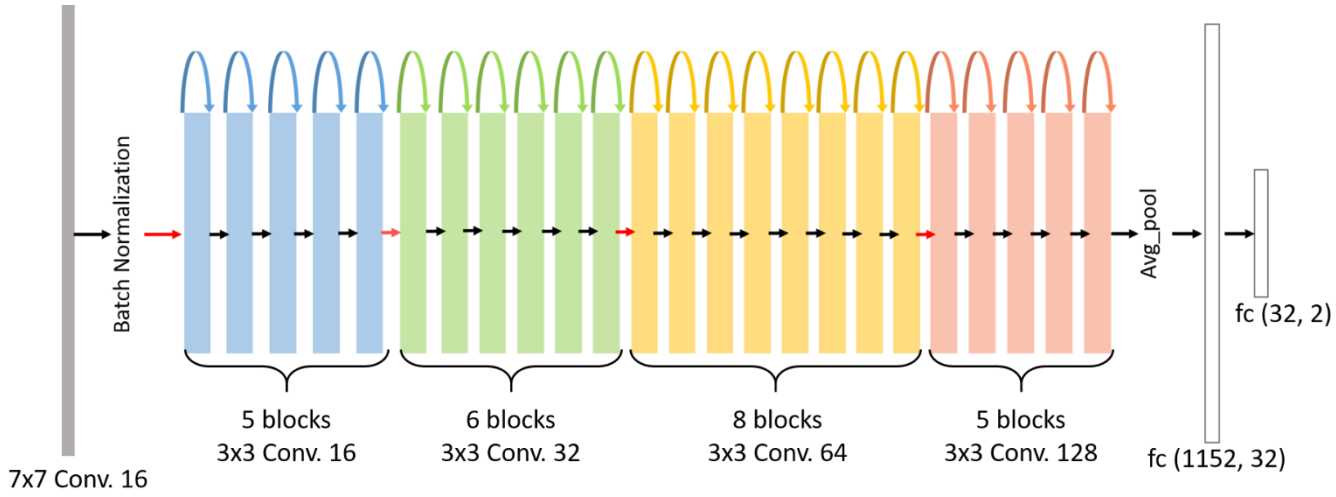
The hidden layer in ResNet is:

Input: X, Output: Res(X) + X

We want the hidden layer learn the residual Res(X), so we use a shortcut to add to the output to ensure what the hidden layer learn is H(X) = Res(X) + X – X = Res(X)

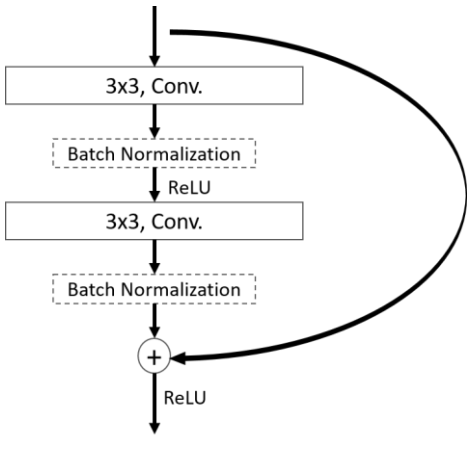
In this way, the worst case of what the network will learn is identical mapping (Res(X) = 0), it will not get worse than it.

The model architecture is:



The red arrow means the stride of there is 2 and the size of data will be divided by 2.

Every color rectangle is a "Basic Block":



In order not to miss features, when the size of the model is divided by 2, the number of channels in convolutional layers will become twice of the previous.
Every Basic Block contains 2 convolutional layers, and there are 5+6+8+5 = 24 Basic Blocks, so there are 24*2 + 1 + 1 + 1 = 48 + 3 = 51 layers in total.

The hyper parameters we set is

Batch size: 32

Learning rate: 0.01 (divided by 10 at epoch 40, and divided by 100 at epoch 70)

Total Epoch: 200

For optimization:

Optimizer: SGD, momentum=0.9

Loss Function: Cross Entropy

Network Initialization: He initialization

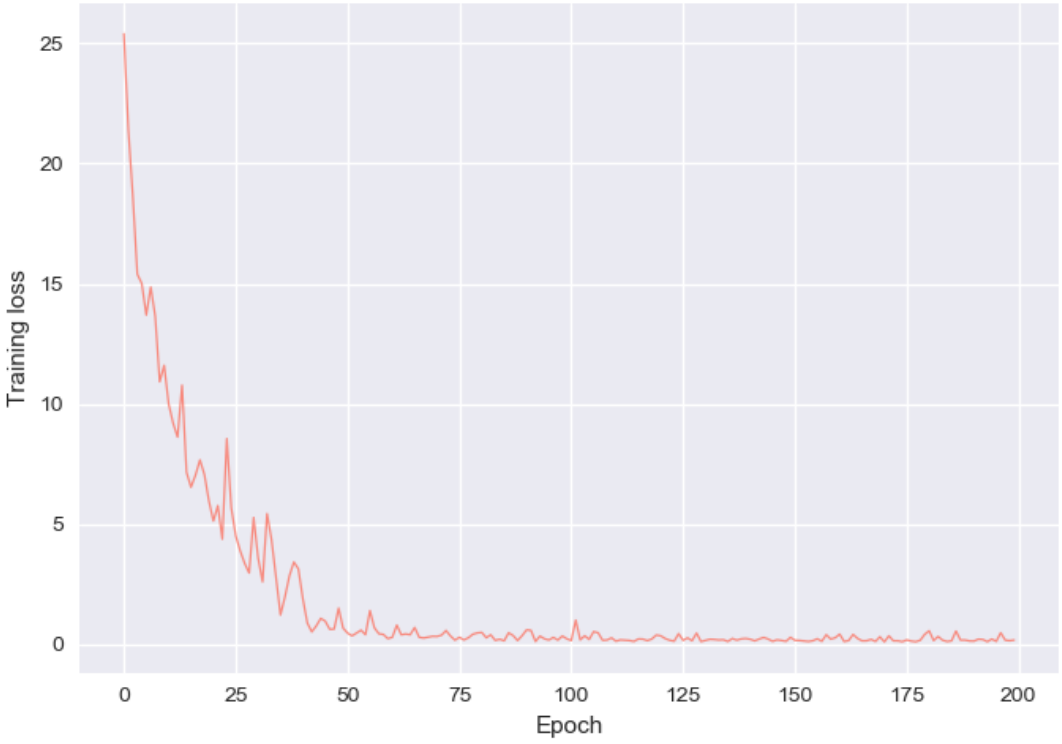
Performance:

```
mingxu1067@cgi23022-52:~/0416074/pj$ python3 main.py -m ResNet -t on -e 1
Test Mode:  True
Model: ResNet
Learning Rate: 0.010000
Epoch: 1

Building Dataset...
Building Completed

Start Running
[Testing] Epoch: 0      | Accuracy: 0.9200
```

Accuracy in testing set: 92%

Accuracy in training set: 99.03%

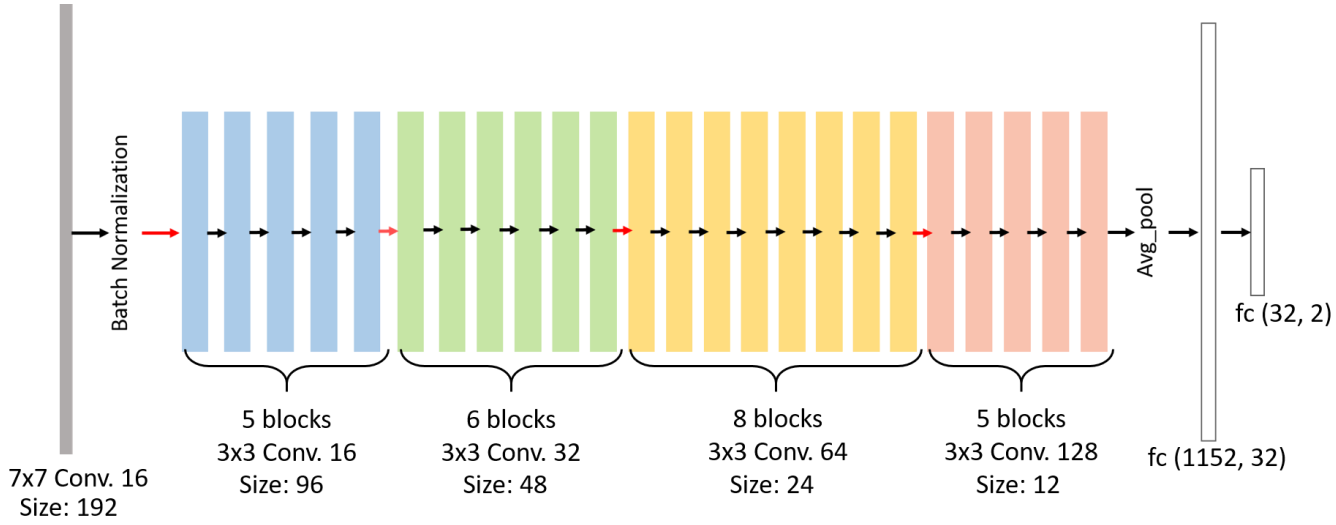
Training Loss Curve:



- Model#2 (Uniform CNN):

  This is an experimental implement, we use uniform CNN (remove all shortcut of ResNet) to do classification.

  The model architecture is:



  The shortcut has been removed, and others are the same. In order to compare it with ResNet, we set them on the same benchmark (Hyper parameters and optimization are the same)

  Performance:

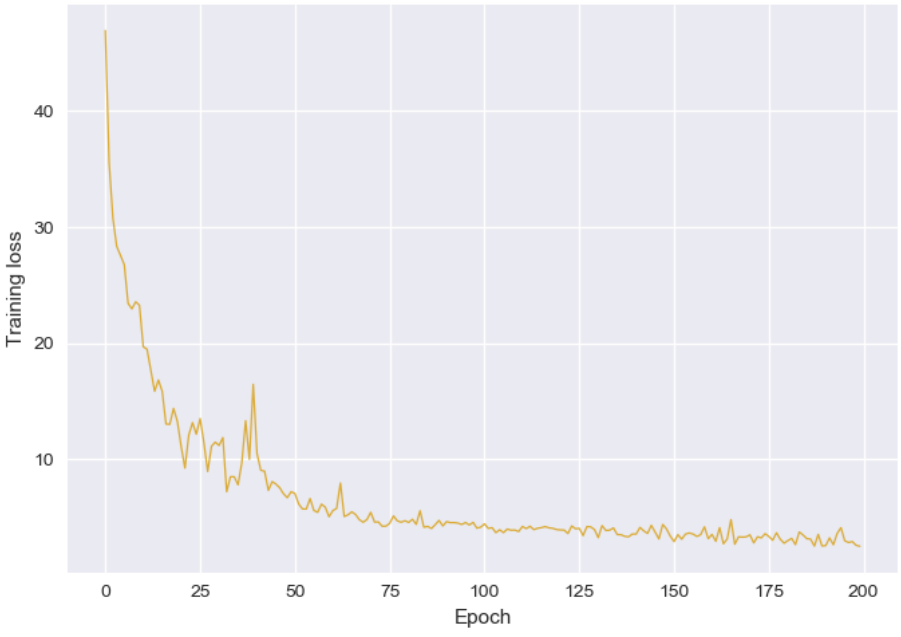
```
mingxu1067@cgi23022-52:~/0416074/pj$ python3 main.py -m VallinaCNN -t on -e 1
Test Mode:  True
Model: VallinaCNN
Learning Rate: 0.010000
Epoch: 1

Building Dataset...
Building Completed

Start Running
[Testing] Epoch: 0      | Accuracy: 0.7800
```

  Accuracy in testing set: 78%

  Accuracy in training set: 96.53%

  We can see that uniform CNN fails on deep network architecture.

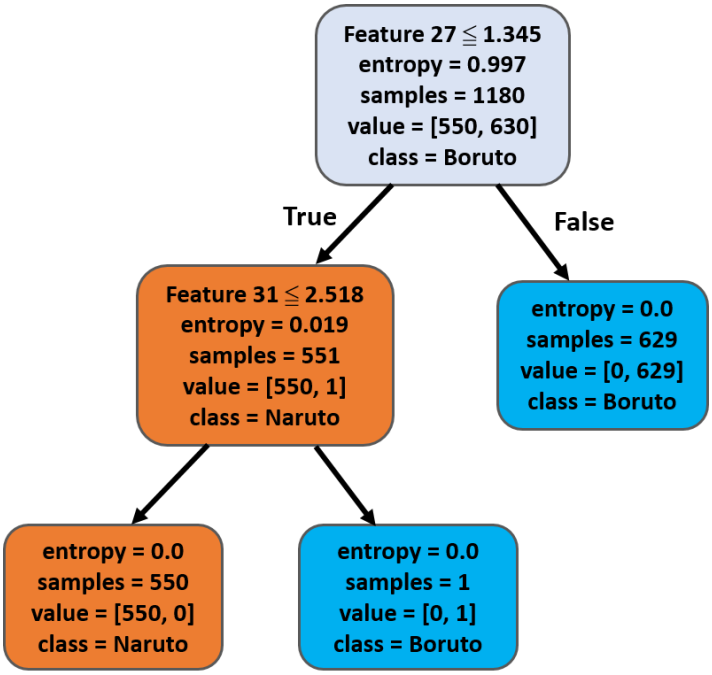
  Training Loss curve:



- Model#3 (Decision Tree):

  We use our ResNet model as encoder, remove the last fully connected layer, and obtain the 32-dimension vector as encoded feature. We encode all training data to this kind of vectors and use these vectors to build a decision tree as a classifier. The method to compute the purity of data and get information-gain is entropy.

  The decision tree we built:



  Performance:

```
mingxu1067@cgi23022-52:~/0416074/pj$ python3 main.py -m ResNet_DT -t on -e 1
Test Mode:  True
Model: ResNet_DT
Learning Rate: 0.010000
Epoch: 1

Building Dataset...
Building Completed

Start Running
[Testing] Epoch: 0      | Accuracy: 0.9400
```

  Accuracy in testing set: 94%

  Accuracy in training set: 100%

- Model#4 (Support Vector Machine):

  Again, we use our ResNet model as encoder and encode all training data to 32-dimension vectors as data to build SVM. We have implemented 2 SVM with different kernels.

  The first one is with Radial Basis Function Kernel (RBF), gamma = 1/3.

  Performance:

```
mingxu1067@cgi23022-52:~/0416074/pj$ python3 main.py -m ResNet_SVM -t on -e 1
Test Mode:  True
Model: ResNet_SVM
Learning Rate: 0.010000
Epoch: 1

Building Dataset...
Building Completed

Start Running
[Testing] Epoch: 0      | Accuracy: 0.9200
```

  Accuracy in testing set: 92%

  Accuracy in training set: 100%

The second one is with Sigmoid Kernel

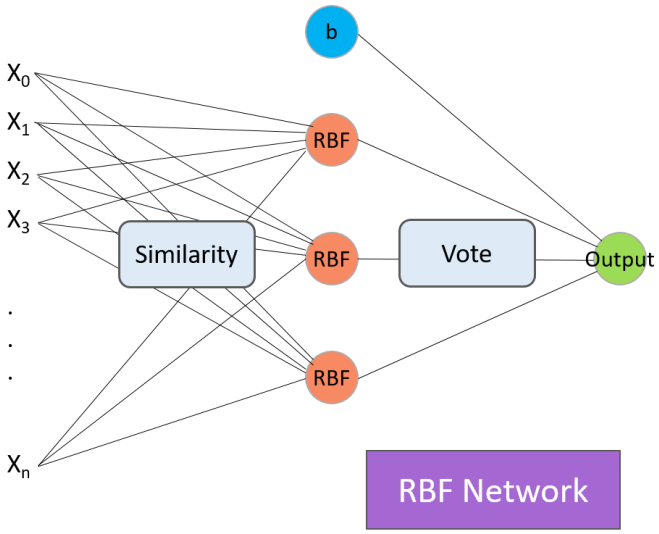
Performance:

Accuracy in testing set: 90%

Accuracy in training set: 99.15%

We can find that SVM with RBF kernel has the same accuracy as our ResNet model. It' s because we can view it as a neural network with one hidden layer.

$F = \text{Output}[\ \Sigma_m \beta_m RBF(X, \mu_m)\ ] + b$



The RBF compute the similarity between the data and support vectors. Then use this value multiplied with β to vote for the class of the output (β is the sign, +1 or -1) This architecture is like the last fully connected layer in the original ResNet we have built, so we can expect that their performance will be almost the same.

All of them are Implemented with

Python 3.5

Pytorch 0.4.1

CUDA 8.0

Scikit-learn 0.20.0

Result

	Accuracy (Testing)
ResNet	92%
Uniform CNN	78%
ResNet + Decision Tree	94%
ResNet + SVM (RBF)	92%

Conclusion:

- ResNet is quite good at extracting features
- Uniform CNN fails on deep network architecture
- Decision Tree outperforms others in classification
- SVM with RBF kernel is similar to fully connected layer architecture

Contribution Table

徐福臨 (0416074)	Data Preprocessing Implement ResNet Implement Uniform CNN Implement ResNet + Decision Tree Implement ResNet + SVM Models Analysis Make PPT Speech delivering Write Report
吳承翰 (0316084)	Data collecting Data Preprocessing Make PPT Speech delivering
劉哲豪 (0416032)	Data collecting Label tagging Make PPT Speech delivering