CSIT5210 Project Final Report

Implementation of MVNN Model -Detecting Fake and Tampered Images

Project Type: Implementation

Group No: 8

Members Information

Student ID:20714865 Student Name:Lu, Mingwei

Student ID:20716344 Student Name:Yang, Dian

Student ID:20743139 Student Name:Fan, Boqian

Student ID:20722185 Student Name:Yuan, Yuhao

Student ID:20748309 Student Name:TANG, Yinkai

Declaration Statement:

We claim that this project is done solely within the course but not other scopes.

Abstract -The paper we selected mainly takes the advantage of images from news, fusing frequency domain and pixel domain of them, to detect fake news. The novel framework used for detecting which called Multi-domain Visual Neural Network (MVNN) is divided into three parts. This report primarily covers the research for the novel framework, the details of our implementation for each part and the conclusion drawn from the computational results of integral system.

I Introduction

1.1 Introduction to fake news images

Classification for fake news images：

Fake news pictures can be divided into two categories:

(1) Tampered images: The images tampered by software such as photoshop.

(2) Misleading images: The pictures are not consistent with text. The pictures are real, but the contents are misleading. Such images are usually derived from works of art or images describing previous events.

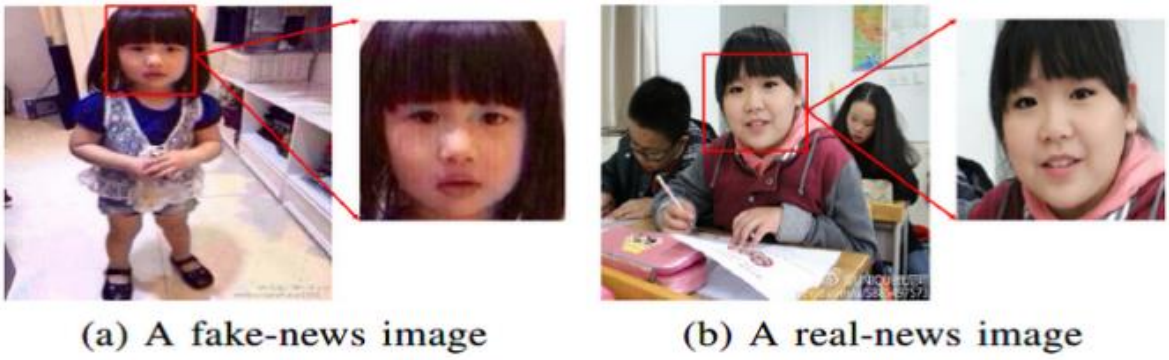
The difference between fake news image and real news image：

Fake news images and real news images may have significant differences in physical and semantic levels.

(1) In physical level:

Tampered images: There are inevitable traces of tampering

Misleading images: Fake news images may be of low quality, which is obviously reflected in the frequency domain. For example, after multiple uploads and downloads on social platforms, misleading images often have more serious re-compression artifacts than real news images, such as the block effect shown in Figure 1.



(a) A fake-news image (b) A real-news image

Figure 1

(2) In semantic level:

Fake news images also have some obvious characteristics in the pixel domain. Fake news publishers tend to use images to attract and mislead readers in order to achieve the rapid dissemination of news. Therefore, fake news images usually have visual impact and emotional provocation, as shown in Figure 2.

Figure 2

1.2 Introduction to Multi-domain Visual Neural Network

The features of fake news image have been proved to be related to many visual factors (from low level to high level); therefore, the author established a multi branch cnn-rnn network to extract features of different semantic levels as shown in Figure 3, and fully capture the features of fake news images in pixel domain.

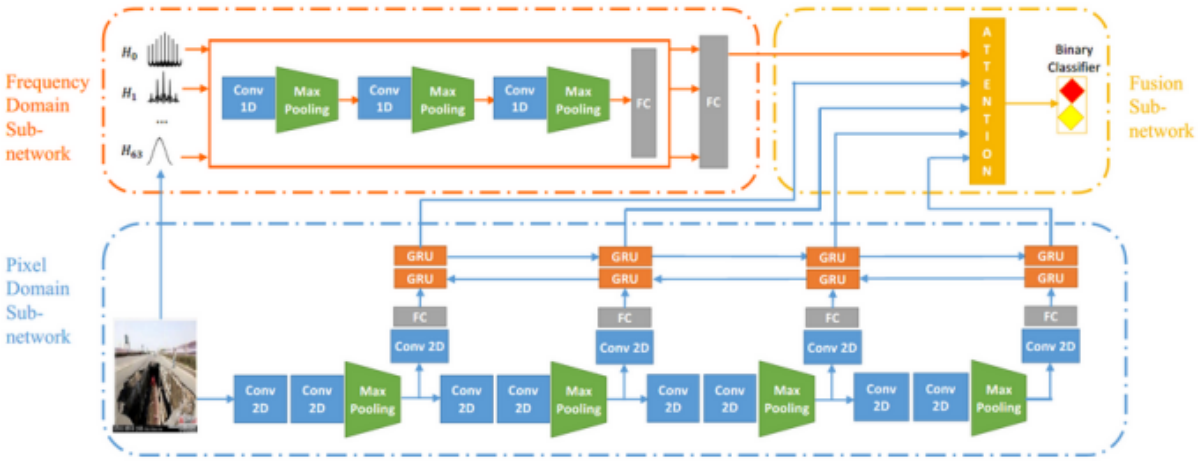


Figure 3

The purpose of MVNN model is to evaluate whether a given image is fake news image or real news image by using visual information in frequency domain and pixel domain. As shown in

Figure 3, mvnn includes three main modules: 1) a frequency domain sub network; 2) a pixel domain sub network; 3) a fusion sub network.

The fusion of visual information in frequency domain and pixel domain can improve the performance of fake news detection model. While, it is not equally important to fuse these features from different news domains. Therefore, attention mechanism is used to dynamically fuse these visual features from different domains.

For an input image, we first input it into the frequency domain and pixel domain subnetworks to obtain the physical and semantic features respectively. Then these features are used as the input of the fusion sub network to obtain the final visual representation of the image, which is used to predict whether it is true news image or fake news image.

II Related Work

Most of the existing work on fake news detection focuses on the text content and social context, and few works use visual information to detect fake news.

There are some works to evaluate the authority of images by extracting features, but these features are mostly designed to detect specific modification traces, which can not be applied to misleading images.

There are also some methods that use pre-trained CNN, such as vgg19, to obtain the overall visual representation. Due to the lack of task-related information, it is difficult to capture the semantic commonness of fake news images.

Therefore, how to effectively use the inherent characteristics of fake news images to achieve fake news detection is a challenge.

III Model and Algorithm

3.1 Frequency domain sub-network

Discrete cosine transform (DCT) is used to transform the input image from pixel domain to frequency domain. Tampered or recompressed images usually have periodic features in the frequency domain, which can be captured by CNN. Therefore, the author designs a network

based on CNN to capture the characteristics of fake news images in frequency domain, as shown in Figure 4.

Figure 4

(1) For the input image, block DCT is used to get 64 histograms of DCT coefficients corresponding to 64 frequencies.

(2) Then 1D Fourier transform is performed on these DCT coefficient histograms to enhance the influence of CNN. Considering that CNN needs input of fixed size, the histogram is sampled and 64 250 dimensional vectors are obtained represented as $\{H0,H1,H63\}$

(3) After preprocessing, each input vector is input into the shared CNN network to obtain the corresponding feature representation $\{w0,w1,...,w63\}$

This CNN network consists of three convolution blocks and a full connection layer. Each convolution block consists of a one-dimensional convolution layer and a maximum pooling layer. In order to accelerate the convergence of the model, the number of filters in the convolution layer is increased.

The existing work on image forensics usually considers only a part of the frequency coefficients. In this paper, the author finds that all the frequencies are helpful to the fake news detection task. Therefore, the feature vectors of all frequencies are fused to obtain the feature representation $l0$, which is used as the input of the fusion sub network.

3.2 Pixel domain sub-network

A pixel domain sub network is designed to extract the visual features of the input image at the semantic level, as shown in Figure 5.

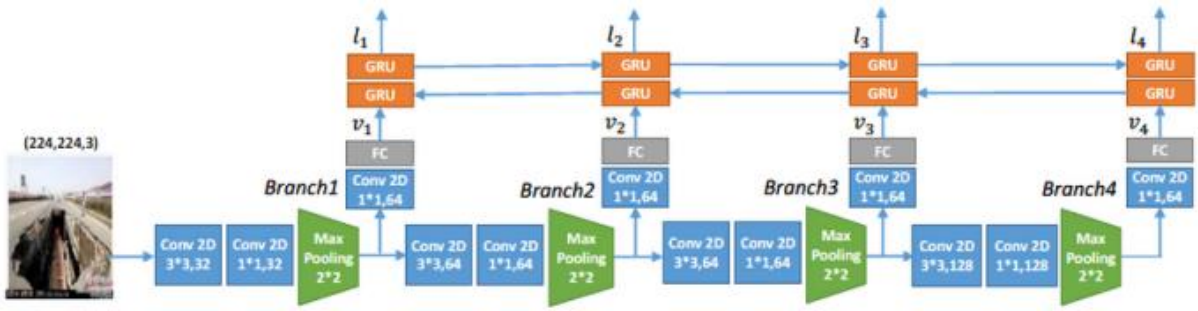


Figure 5

The first convolution layer tends to capture low-level features such as color, line and shape, while the latter tends to object. In the process of abstraction, the underlying features will inevitably lose, which further shows that the bottom layer and middle layer of CNN can provide supplementary information for the top layer.

A lot of work has proved that for some tasks, integrating features of different layers can achieve better performance than only using high-level features. We have also explained that fake news images usually have visual impact and emotional provocation, which are proved to be related to many visual factors from low level to high level.

Therefore, in order to capture the semantic features of fake news images, the author establishes a multi-brach CNN network to capture different levels of features and uses Bi-Gru network to model the sequence dependence between these features.

As shown in Figure 5, the CNN network is mainly composed of four blocks, each of which is composed of a 3*3 convolutional layer, a 1*1 convolutional layer and a maximum pooling layer. The image is input into CNN, and the features extracted from the four branches will pass through a layer of 1*1 convolution and a layer of full connection to obtain the corresponding feature vector $vt, t \in [1,4]$. These features represent different parts of the picture, such as line, color, texture, object.

Inspired by the inception module used in GoogleNet, the author uses 1*1 convolutional layer to reduce the dimension and increase the representational ability of the model, because it increases the nonlinearity of activation functions and promotes the fusion of different channel information.

There is a strong dependence between the features of different levels. For example, the texture features in the middle level are composed of low-level line features and high-level features, such as object. Therefore, Bi-Gru is used to model the dependency between low-level and high-level features.

$$r_t = \sigma(W_r[v_t, h_{t-1}] + b_r) \quad (1)$$

$$z_t = \sigma(W_z[v_t, h_{t-1}] + b_z) \quad (2)$$

$$\tilde{h}_t = \tanh(W_{\tilde{h}}[v_t, r_t \odot h_{t-1}] + b_{\tilde{h}}) \quad (3)$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t \quad (4)$$

$$\overrightarrow{l_t} = \overrightarrow{GRU}(v_t), t \in [1, 4] \quad (5)$$

$$\overleftarrow{l_t} = \overleftarrow{GRU}(v_t), t \in [1, 4] \quad (6)$$

Where *rt*, *zt*, *ht~*, *ht* represent reset gate, update gate, hidden candidate, hidden state, the forward and backward hidden layer states of each time step are spliced together to form the final semantic feature representation $L=\{lt\}, t \in [1,4]$

3.3 Fusing sub-network

The author assumes that the physical features and semantic features of images are complementary, so a fusion sub network is proposed to fuse these features, *l0* and $\{l1, l2, l3, l4\}$. The author uses attention mechanism to fuse these features, and the enhanced image representation is calculated as follows, in which the vector *v* is randomly initialized and learned jointly in the training process.

$$\mathcal{F}(l_i) = v^T \tanh(W_f l_i + b_f), i \in [0, 4] \quad (7)$$

$$\alpha_i = \frac{\exp(\mathcal{F}(l_i))}{\sum_i \exp(\mathcal{F}(l_i))} \quad (8)$$

$$u = \sum_i \alpha_i l_i \quad (9)$$

At present, we model the physical and semantic features of the image, and obtain the high-level of the input image to represent *u*. Then, the probability distribution of fake news image and true news image is obtained by prediction:

$$p = \text{softmax}(W_c u + b_c) \quad (10)$$

The loss Entropy:

$$L = -\sum [y \log p + (1 - y) \log (1 - p)] \qquad (11)$$

IV Implementation

4.1 Data collection and data cleaning

The input of pixel domain model is image data with size 224 * 224 * 3, and the input of frequency domain model is 64 250-dimensions vectors which are obtained by adopting DCT algorithm to the image data. In other words, what we need for our model is fake news images and real news images.

Since the authors do not provide the datasets they used in the paper, we have to collect dataset ourselves. Finally we find a dataset called Fakeddit which is a novel multimodal dataset consisting of over 1 million samples from multiple categories of fake news.

Fakeddit provide multimodal text and image data, metadata, comment data, and fine-grained fake news categorization at the scale and breadth. And the samples are labeled according to 2-way, 3-way, and 6-way classification categories through distant supervision. Our final choice is to use 2-way labels since our goals is to judge whether a image is fake or not, we don't need fine-grained labels like 6-way labels.

The data structure of Fakeddit is a folder with all images and a csv file contains all details of all images, such as 2-way labels, number of comments, author, title and so on.

We write a python script to do the image processing.

We found that we can associate a image file with its information in csv file by file name.

Firstly, we load images data paired with file name, and load the csv file which contains all details of these images into a dictionary with file name as the key.

Secondly, we associate images with its information by using the dictionary we obtained in step one, and then remove images whose information is not found.

Thirdly, we divide the whole dataset into different folders according to the label of the data. This can simplify the reading process when we use torch.ImageFolder to load the dataset into our model.

After that, we obtain 600k labeled images with 20k fake news samples and 580k real news samples. The number of positive samples and negtive samples is unbalanced, and we found

that the authors only use 5k positive samples and 5k negtive samples with 300 epochs training. Based on this fact, we make a conclusion that 20k positive samples and 20k negtive samples is enough for our model. So we finally randomly choose 20k real news from the whole dataset, and keep 20k fake news samples unchanged.

We divide our dataset into training set, validation set, testing set with the percentage of 80%, 10%, 10% respectively. And each dataset with balanced number of positive samples and negtive samples.

4.2 Model architecture implementation

The model consists of three parts, which is frequency domain sub-network, pixel domain sub-network and fusion sub-network. We use the deep learning framework PyTorch to implement the entire model architecture. In addition to building the MVNN model, we also need to preprocess the image data, including reading the image data, and performing DCT on the obtained image. Finally, we need to write a main function to train the model and test the performance of the trained model.

4.2.1 DCT

Before constructing the frequency domain sub-network, we firstly have to apply discrete cosine transform (DCT), which has been widely utilized for capturing the tamperped and re-compressed architects, to each input image in order to explore its physical feartures.

According to the steps in [1], we convert images to grayscale two-dimensional matrixes at first since DCT dose not care about color information. And because DCT is designed to work on pixel values ranging from -128 to 127, we substract 128 from each image matrix. Then, each image is broken into 8*8 blocks of pixels and for each block, we apply DCT algorithm from cv2 module as below:

```
for block in blocks:
    # apply dct
    block_dct = cv2.dct(block)
    # do quantization
    block_dct = (block_dct / q50).astype(np.int16)
    for i in range(8):
        for j in range(8):
            cof_histograms[i * 8 + j].append(block_dct[i][j])
```

After the dct transform, we get 64 DCT coefficients for each block. After that, we also apply

quantization by dividing each element in the DCT coefficients matrix by the corresponding element in the quantization matrix. For a standardization and balanced image quality/compression ratio, we choose the standard quantization matrix with a quality level 50 as below [1].

$$Q_{50} = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

After quantization process, we then obtain 64 histograms of DCT coefficients corresponding to 64 frequencies. Considering the CNN needs an input of fixed size, we randomly sample these histograms and get 64 250-dimensional vectors for each input image at last of the DCT procedure.

4.2.2 Frequency domain sub-network

The frequency domain sub-network only has one sequential blocks of operation, composing of three 1-d convolutional layers, one maximum pooling layer, and two fully connected layers. The main architecture for the sub-network is shown below. It processes the vectors from DCT with a dimension of (batchsize,64, 250). After passing through three convolutional layers with a rectified linear unit and a maximum pool unit, the dimension of the vectors are flattened out from (batchsize,128,29) to (batchsize,128*29). Finally, the two fully connected layers transform the dimension from (batchsize,128*29) to (batchsize,64).

```
class FrequencyNet(nn.Module):
    def __init__(self):
        super(FrequencyNet, self).__init__()
        self.conv1 = nn.Conv1d(in_channels=64, out_channels=32, kernel_size=3)
        self.pool = nn.MaxPool1d(2)
        self.conv2 = nn.Conv1d(in_channels=32, out_channels=64, kernel_size=3)
        self.conv3 = nn.Conv1d(in_channels=64, out_channels=128, kernel_size=3)
        self.fc1 = nn.Linear(128 * 29, 64)
        self.fc2 = nn.Linear(64, 64)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = self.pool(F.relu(self.conv3(x)))
        x = x.view(-1, 128 * 29)
        x = self.fc1(x)
        x = self.fc2(x)
        return x
```

4.2.3 Pixel domain sub-network

The pixel domain sub-network is more complicated than the frequency domain sub-network, because it is no longer a single sequential branch. Every time the vectors processed by a CNN block would be flowed into a GRU block and also the next CNN block. The CNN part is similar to what we have learned from our machine learning course which there has been some assignments introducing the pytorch way of building the CNN. However, the hardest part of this section is to firstly learn what the bidirectional GRU is and then understand how to use the API so that transforms from a GRU to a bidirectional GRU. Even though the PyTorch API provides a parameter “bidirectional” which you can set it as true or false, after I watched a Youtube video which Andrew NG introduced the architecture of the bidirectional GRU, I found out it is more understandable to seperately calculate the output by just reversing the order of the input vectors. This section did involve with many vector operations, for instance, the “stack” and “cat” operations, which concatenates vectors in specified dimensions. The number of hidden units of the GRU and fully connected layer is 32 and 64 respectively.

```
def forward(self, x):
    #初始化 hidden layer
    h0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(device)

    #branch down 1
    x = F.relu(self.conv11(x))
    x = F.relu(self.conv12(x))
    x_b1 = self.pool(x)

    #branch up 1
    v1 = F.relu(self.conv1_b(x_b1))
    v1 = v1.view(-1, 64 * 111 * 111)
    v1 = self.fc1(v1)

    #branch down 2
    x_b2 = F.relu(self.conv21(x_b1))
    x_b2 = F.relu(self.conv22(x_b2))
    x_b2 = self.pool(x_b2)

    #branch up 2
    v2 = F.relu(self.conv22(x_b2))
    v2 = v2.view(-1, 64 * 54 * 54)
    v2 = self.fc2(v2)

    #branch down 3
    x_b3 = F.relu(self.conv31(x_b2))
    x_b3 = F.relu(self.conv32(x_b3))
    x_b3 = self.pool(x_b3)
```

```
#branch up 3
v3 = F.relu(self.conv22(x_b3))
v3 = v3.view(-1, 64 * 26 * 26)
v3 = self.fc3(v3)

#branch down 4
x_b4 = F.relu(self.conv41(x_b3))
x_b4 = F.relu(self.conv42(x_b4))
x_b4 = self.pool(x_b4)

#branch up 4
v4 = F.relu(self.conv4_b(x_b4))
v4 = v4.view(-1, 64 * 12 * 12)
v4 = self.fc4(v4)

# Bidirectional GRU
# forward GRU
# Merge the 64 batches of 64 vectors on dimension 1: (64,64) -> (64,4,64)
v_forward = torch.stack([v1,v2,v3,v4],dim=1)
l_forward, hidden = self.gru(v_forward, h0)

# backward GRU
v_backward = torch.stack([v4, v3, v2, v1], dim=1)
l_backward, hidden = self.gru(v_backward, h0)

# Merge the forward and backward output on dimension 2: (64,4,32) -> (64,4,64)
res = torch.cat([l_forward, l_backward], dim=2)
# (64,4,64) -> (64,4,1)
# return self.fc_gru(res)
# (64,4,64)
return res
```

4.2.4 Fusion sub-network

The fusion sub-network plays a crucial role as merging both the results from pixel and the frequency domain sub-networks, and then applies the attention mechanism to assign weights on the vectors. First of all, I tried to search through the API of PyTorch, but found no attention module. Then, I read through several articles introducing the concept and PyTorch code for the attention module. I finally chose to follow the code template from this website [2].

```
#(64,64)
frequency_output = self.frequency_net(dct_imgs)
#(64,1,64)
frequency_output = frequency_output.unsqueeze(1)
#(64,4,64)
pixel_output = self.pixel_net(x)
#(64,1,64)+(64,4,64)=(64,5,64)
l = torch.cat([frequency_output,pixel_output],dim=1)

#实现attention
# l (64,5,64) x weight (64,64) + bias (64,1,1) = u (64,5,64)
temp = torch.tanh(torch.add(torch.matmul(l,self.weight),self.bias))
# temp (64,5,64)  x  v (64,1) = F score (64,5,1)
f = torch.matmul(temp, self.v)
f_score = F.softmax(f, dim=1)
scored_x = l*f_score
# (64,5,64) -> (64,64)
u = torch.sum(scored_x, dim=1)
# (64,64) -> (64,2)
y = self.fc(u)
p = F.softmax(y)
return p
```

4.3 Optimization

4.3.1 DCT preprocessing

The calculation of DCT is time-consuming. After the model is built, the existence of bugs is inevitable. Many bugs need to be discovered after the model starts training. But the data loading must always be completed before the model starts training, and the DCT transformation and the image data loading are performed at the same time, so even a small bug will cause us to adopt DCT transformation to all the images once. This will cause our debugging efficiency to be extremely low, which is obviously not a wise way.

The result of DCT transformation of a certain image is constant. Though we do sampling to the result to make sure the number of dimensions of frequency features is fixed, we can still store the DCT transformation results of each image, as for whether or not sampling is done makes little difference on the performance of the model. The solution we finally adopted is to sample the DCT transform result of each image and store it in the json file using the file name as a key. This allows us to speed up data loading by nearly 10 times, greatly improving the efficiency of debugging.

4.3.2 Code optimization

There are many similar modules in this model architecture, and there are only individual parameter differences between different modules, such as the number of filters and the shape of the input tensor. Based on these observations, we have used a lot of syntactic sugar to optimize the code of the model to reduce redundancy, which greatly enhances the readability, facilitates maintenance, and improves the efficiency of development.

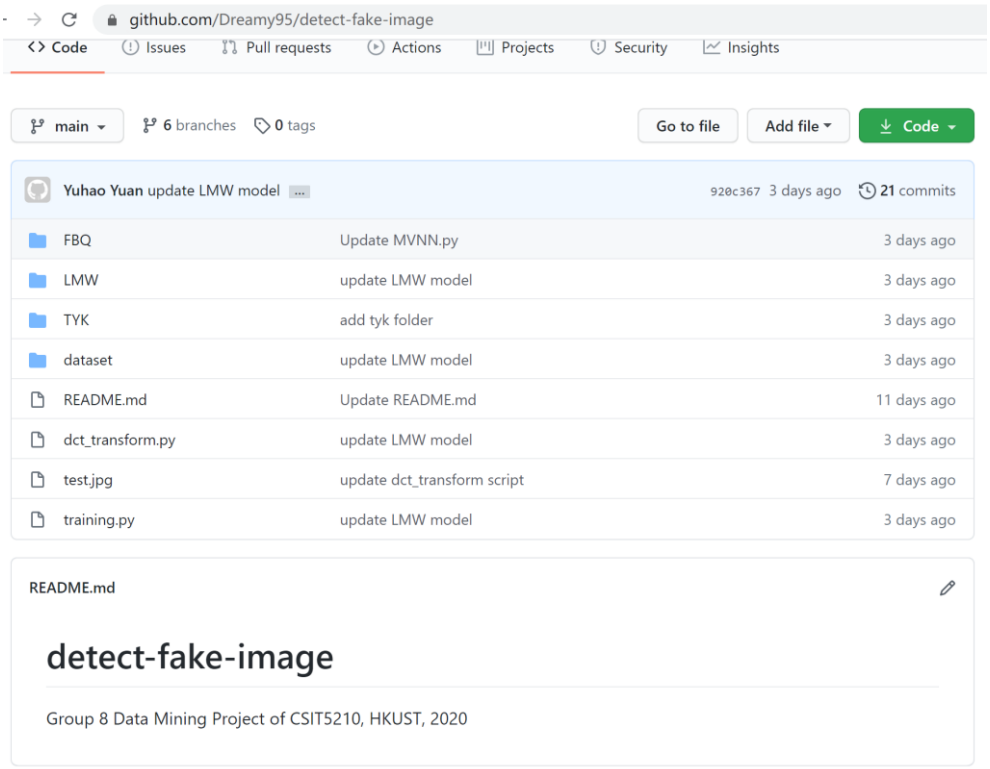
```python
class FrequencyModel(nn.Module):
    def __init__(self):
        super(FrequencyModel, self).__init__()
        self.conv = nn.Sequential(
            # (64, 250) -> (32, 248)
            self.conv_block(64, 32, 3),
            # (32, 124) -> (64, 122)
            self.conv_block(32, 64, 3),
            # (64, 61) -> (128, 29)
            self.conv_block(64, 128, 3)
        )
        self.linear = nn.Sequential(
              nn.Linear(128 * 29, 16),
              nn.Linear(16, 64),
              nn.Dropout(0.4)
        )

    def conv_block(self, in_channels, out_channels, kernel_size):
        return nn.Sequential(
            nn.Conv1d(in_channels, out_channels, kernel_size),
            nn.BatchNorm1d(out_channels),
            nn.ReLU(),
            nn.MaxPool1d(2)
        )
```

This is an example of syntactic sugar. In the frequency domain model, we define a function named conv_block to define a module which contains a convolution layer, a batch normalization layer, a relu activation function and a max pooling layer in turn. We can define the frequency model easily by using this function because convolutional part in the frequency domain sub-network all have the same structure with small difference. The sequential function can be used to define a module which only contains one way from the input to the output, so it's an efficient way to define a sub-module of a big model by using this function.

4.4 Github

In this project, we created a private github repository to coordinate and commit programs. The screenshot below is the main page of our group's repository which there are three versions of the MVNN model named in each teammate's name abbreviation and one dataset folder. The original idea to use github is that we can share our implementation to other teammates and also other teammates can comment on where your program can optimize. Another benefit of using the github is to take the advantage of creating branches based on each important milestone so that we can experiment without disturbing others. The development flow is quite smooth and we are satisified with this website for improving our development experience.



V Result and Conclusion

After we finish building the entire MVNN model, we also train the model and evaluate the model using the dataset mentioned above. Considering the limitation of devices (laptops from group member) and time, instead of the 64 batch size and 300 epochs that original paper conducted, we actually set the batch size to 20 and do 10 epochs for each training. But generally, we get very impressive results at below.

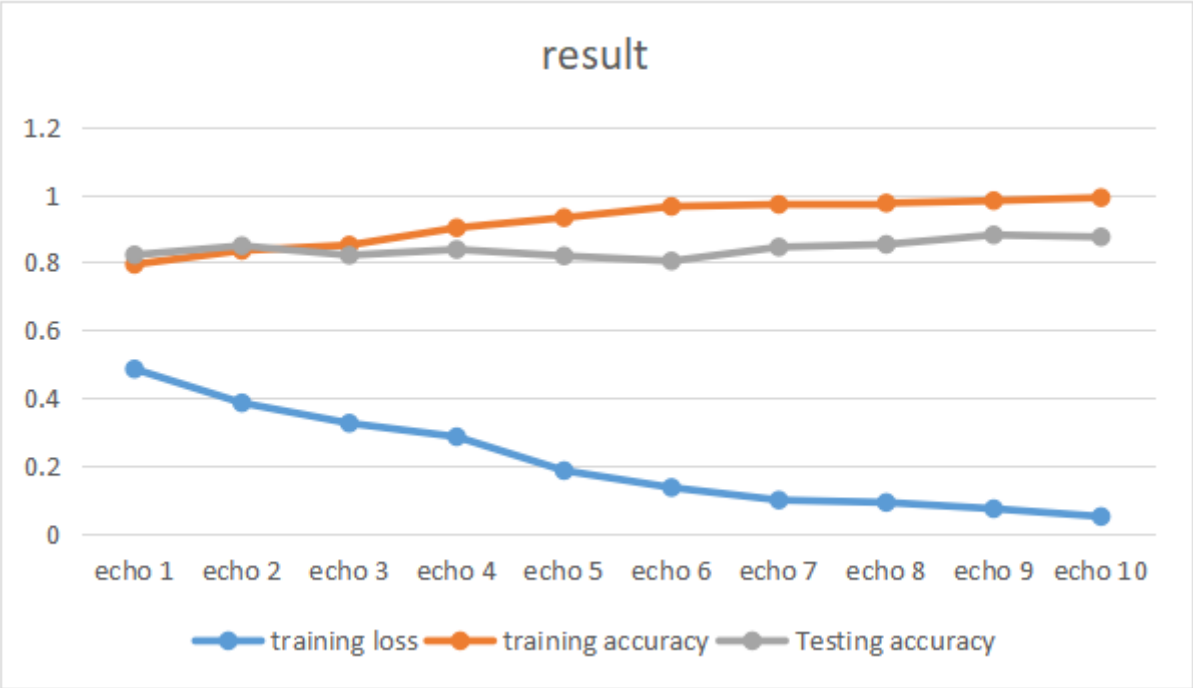
5.1 Accuracy table of testing set

Accuracy of Testing Set	Epoch2	Epoch4	Epoch6	Epoch8	Epoch10
Training 1	91.4%	88.8%	88.3%	89.1%	90.5%
Training 2	84.9%	83.8%	80.4%	85.3%	87.5%

Training 3	90.9%	91.2%	89.2%	90.3%	80.6%
Training 4	92.5%	92.7%	91.1%	92.4%	88.0%

The table above shows the accuracy results of testing set of four (we actually trained and tested the model many times, those four examples are the representation of a consistent result) training and evaluating experiments of our MVNN model. The average accuracy of classifying the testing set after finishing the training is 86.65%.

5.2 Line chart of training accuracy and testing accuracy



The line chart above shows the changes of training loss, training accuracy and testing accuracy during one of the model trainings and evaluating experiments.

5.3 Conclusion

After many times of training and debugging of the model, we got results similar to the original paper. The most important reason for obtaining such results is that the author gave a complete model implementation. At the same time, the model details, including the selection of various hyperparameters, optimization methods and other details are fully listed, so that we can reconstruct the entire model architecture easily. Secondly, although we did not get the

dataset used by the author, we found a similar high-quality dataset, which laid the foundation for our model to reconstruct.

VI Summary

Our project started on September 8, 2020 and ended on November 23, 2020. In the nearly three months, we held a meeting every two weeks in our spare time to discuss the phased work.

Our detailed work cycle is as follows:

- From 8 September to 30 September:

Read suggested topics and papers, determine the topic and paper

- From 1 October to 16 October:

Research the selected paper, determine the required knowledge, collect relevant information, determine the learning plan

- From 17 October to 31 October:

Learn the required knowledge through paper, video and related websites

- From 1 November to 7 November:

Collect, research, organize data sets and make implementation plan

- From 8 November to 19 November:

Implement mvnn model, run the program and adjust the parameters, collect and count the running results

- From 20 November to 22 November:

Summarize the work and write the final report.

In conclusion, duiring this period we are all actively communicating with each other. Everyone has conscientiously completed the content they are responsible for and contributed a lot for the completion of this project. Even if we have faced some minor problems, we worked them out smoothly with the strength of whole group. Overall, we believe we have successfully completed this project (mainly the implentation of the MVNN model) since we have finished building the model and got a reasonable evaluating results that similar to the

original paper.

VII References

[1] Ken Cabeen and Peter Gent, "Image Compression and the Discrete Cosine Transform"

[2] Dazhuanlan.com. 2020. 文本分类中的 attention 注意力机制及 pytorch 实现 | 大专栏. [online] Available at: [Accessed 22 November 2020].

[3] Y.-L. Chen and C.-T. Hsu, *"Detecting recompression of jpeg images via periodicity analysis of compression artifacts for tampering detection,"* IEEE Transactions on Information Forensics and Security, vol. 6, no. 2, pp. 396–406, 2011.

[4] A. Persson, *Pytorch Seq2Seq Tutorial for Machine Translation*. https://www.youtube.com/watch?v=EoGUlvhRYpk&t=1678s&ab_channel=AladdinPersson: Aladdin Persson, 2020.

[5] A. Persson, *Pytorch Seq2Seq with Attention for Machine Translation*. https://www.youtube.com/watch?v=sQUqQddQtB4&t=502s&ab_channel=AladdinPersson: Aladdin Persson, 2020.

VIII Contribution

- Lu, Mingwei

In this project, I made correspnding contributions. Firstly, I discussed the DCT preprocessing scheme with other group members, iterated several versions of the scheme and finally wrote the code and tested it to make sure it can work well. Secondly, I spent a lot of time on the collection, cleaning and analysis of the dataset, and then built a data reading pipeline to simplify the data loading process. This laid the foundation for the subsequent model construction of other group members. Finally, I optimized the code structure of our project by using multiple syntactic sugars make the code more readable and maintainable.

- Yang, Dian

I am mainly responsible for sorting out the structure of the paper, figuring out the main idea of the paper and proposing ideas for the architecture of proposal and final report, collecting the

running result of the ultimate version of our program to generate line diagram for the result. And summarize the work done by us. In the writing section, I write the background and motivation section for proposal and the abstract, introduction, related work, model and algorithm, and summary section in the final report.

- Fan, Boqian

In this project, I am mainly in charge of developing my version of MVNN model with PyTorch framework and also writing the training script for our models. In addition, during development, I point out a issue, for example, how to tackle with the complex number result of the DCT after Fast Fourier Transform (PyTorch does not support complex number calculations). I also helped other teammates solve some problems during discuss and reported my progress in time on our wechat group. Except for the program development, I also participated in writing the final report for sections 4.2.2, 4.2.3, 4.2.4, and 4.4. Finally, I proofread our final report and fix some grammar or spelling mistakes.

- Yuan, Yuhao

In this project, I participated in almost all parts and made corresponding contributions. Firstly, for the project proposal, I wrote the project introduction and scope section. For the group cooperation, I arranged all the zoom meetings so that the group members could share information, communicate with each other and work together in a timely manner. And as for the model implentation, I wrote the initial version of dct transform script and made some updates on Mingwei's model. And I also trained the model and evaluated the model many times in order to get a consistent result. Finally, for the project report. I wrote the DCT part and finished the results part in cooperating with group members.

- TANG，Yinkai

In this project, firstly I have crawled some data sets related to this project from Weibo, but the data sets I crawled are not used in this project in the end. Then I have read the paper about DCT, and developed a DCT function. Next, I developed my vertion of MVNN model. In the process of completing this model, I searched a lot of information on the Internet and watched many videos related to the model to help me implement the MVNN model. Finally, I finished the implementation part of the report with my teammates.