Contents

1 Introduction

In the past, rapid development has been seen in the area of multi-class classification for computer vision tasks. With the help of convolutional neural networks (CNN), many groundbreaking models have been discovered, including AlexNet, VGGNet and ResNet. At the same time, natural language processing (NLP) also achieved drastic improvement in areas such as machine translation, entity recognition etc. with the help of GRU, LSTM, transformer and BERT etc. However, in the real world, multi-label multi-class classification problems are even more ubiquitous than single label classification. The aim of the study is to integrate the knowledge from both computer vision and NLP to solve a multi-label multi-class problem with limited model size of less than 100Mb. In this paper, we propose a model that uses the ResNet with LSTM model to capture both the vision information and image information in the dataset and then use the combined information to make multi-label multi-class prediction. With our work, we provide one simple model to solve a relatively complex problem with limited computation power needed.

2 Related works

2.1 CNN

The architecture of VGG takes the advantage of ReLU function to solve the vanish gradient problem. Comparing to the traditional activation function sigmoid, which will have very small gradient at the saturating regions, ReLU function will have gradient equal to 1 where the output is bigger than 0 thus will speed up the training and the converge of the model and has less vanish gradient problem. Another character of the VGG model is that it use a very small kernel size (3,3), which is the smallest kernel size to capture the spacial information (left, right, up, down and center). The VGG model shows that with smaller kernel size and much deeper channels can achieve results better than large kernel size with shallow channels (Simonyan & Zisserman, 2014).

Before the publishment of ResNet, researchers have reach a common agreement that by increase the depth of the computer vision model will increase the performance of the model. However, with the increase of depth, comes with vanishing/exploding gradient problem. This problem can be largely address through normalization at the initialization and intermediate steps. However, as the layers increase even more, the degradation problem start to occur. The degradation described by the He etc in 2016 Deep residual learning for image recognition as when layers increased, the accuracy becomes saturated and start to degrading. What is striding is that this phenomenon is not due to overfitting. The ground breaking technique proposed by ResNet is the shortcut connection mechanism, which significantly improve the performance of the model. The structure of ResNet is derived from VGG with addition of shortcut mechanism (He, Zhang, Ren, & Sun, 2016) .

2.2 LSTM

Long Short Term Memory (LSTM) model improve the traditional RNN model by adding three gates, which are forgetting gate to control how much information the current cell state should keep, the input gate that control the inflow of the information from inputs, and the output gate to control the output. As the information propagate through the hidden state, LSTM can be effective for extracting the useful information and keep the important information in the hidden state(Wang et al., 2016).

2.3 YOLOv5

We also tried the YOLOv5 algorithm, which frames object detection as a regression problem to spatially separated bounding boxes and associated class probabilities. The neural network is able to predicts bounding boxes and class probabilities directly from images in one evaluation. The performance can be optimized due to the end-to-end detection pipeline(Redmon, Divvala, Girshick, & Farhadi, 2016).
The YOLO detection network includes 24 convolutional layers and 2 fully connected layers, as shown in the following figure:

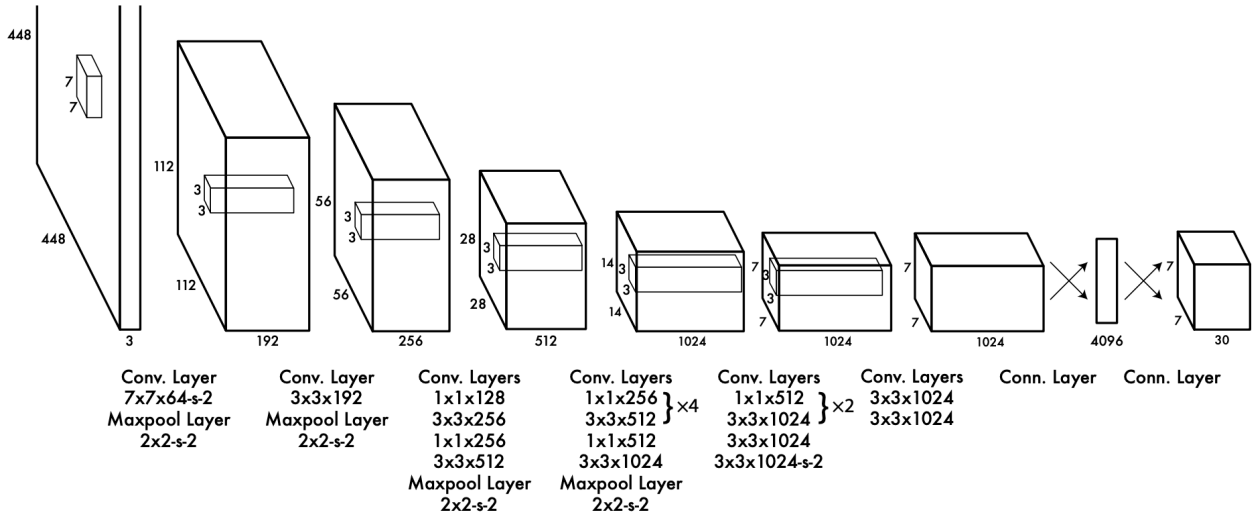

Figure 1: The architecture of YOLO
(Redmon et al., 2016)

Among them, the convolutional layer is used to extract image features, and the fully connected layer is used to predict the image location and class probability value. YOLO divides the input image into SxS grids, and each grid is responsible for detecting objects that 'fall into' the grid. If the coordinates of the center position of an object fall into a certain grid, then this grid is responsible for detecting the object. As shown in the figure below, the center point (red origin) of the object dog in the figure falls into the grid in the fifth row and the second column, so this grid is responsible for predicting the object dog in the image.

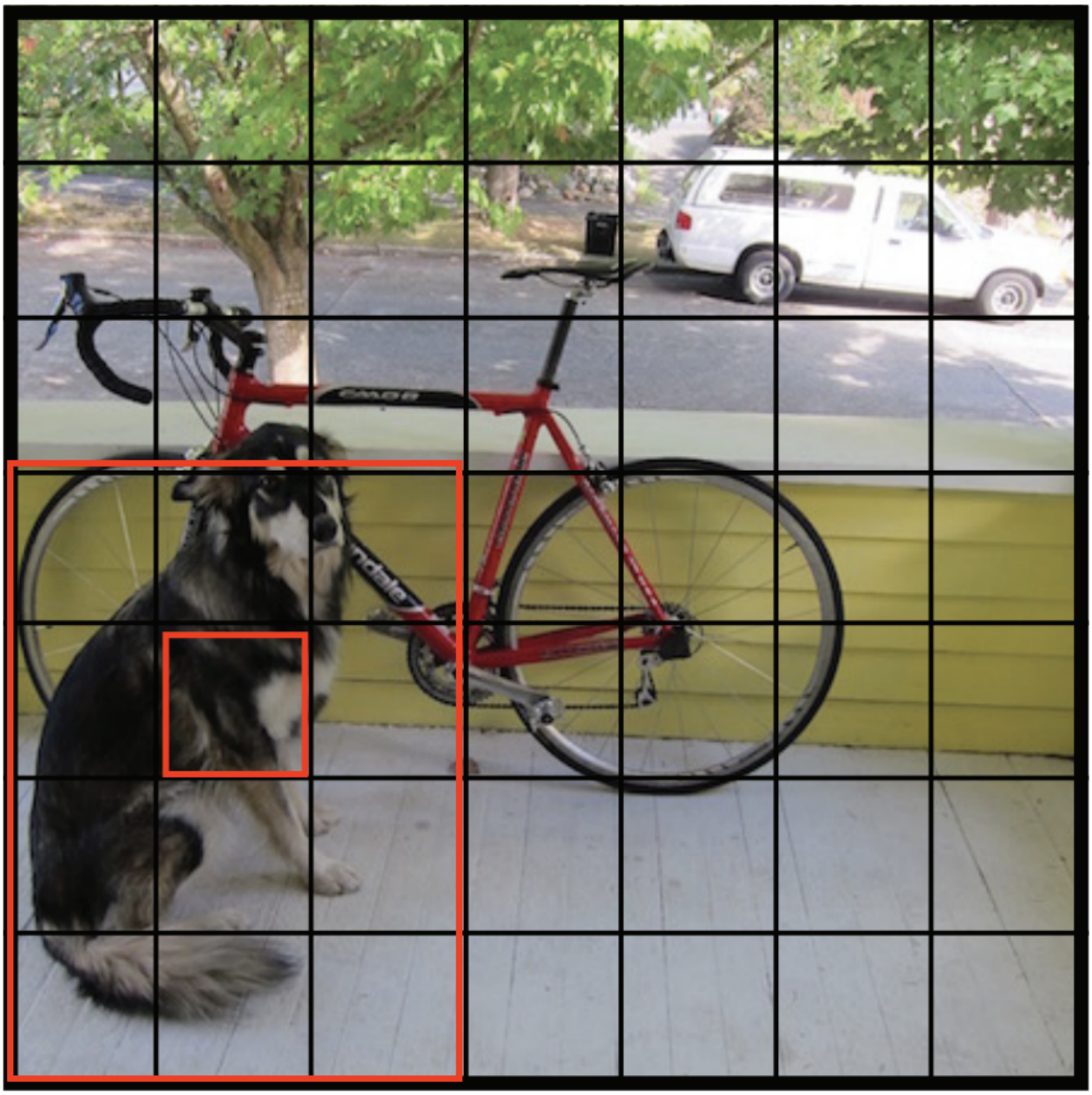

Figure 2: A image of dog for illustration
(Redmon et al., 2016)

Each grid outputs B bounding boxes (rectangular areas containing objects) information, and C objects belong to a certain category of probability information.

The Bounding box information contains 5 data values, namely x, y, w, h, and confidence. Where x, y refer to the coordinates of the center position of the bounding box of the object predicted by the current grid. w, h are the width and height of the bounding box. Note: In the actual training process, the values of w and h are normalized to the interval [0,1] using the width and height of the image; x, y are the offset values of the center position of the bounding box relative to the current grid position, and Is normalized to [0,1]. Confidence reflects whether the current bounding box contains an object and the accuracy of the position of the object. The calculation method is as follows:
Confidence = $P(object)$, if the bounding box contains an object, then $P(object) = 1$; otherwise, $P(object) = 0$.

IOU (intersection over union) is the intersection area between the predicted bounding box and the real area of the object (in pixels, use The pixel area of the real area is normalized to the interval [0,1]).

Therefore, the output dimension of the final fully connected layer of the YOLO network is $S * S * (B * 5 + C)$. In the YOLO paper, the input image resolution used by the author for training is $448 * 448, S = 7, B = 2$; VOC 20-type labeled objects are used as training data, C=20. Therefore, the output vector is $7 * 7 * (20 + 2 * 5) = 1470$ dimensions.

3 Techniques

3.1 The principle of method

3.1.1 The architecture of network

The CNN-RNN model is a combination of CNN model and RNN model, which is used to solve multi-label classification problems. The architecture of our CNN-RNN model is as follows:

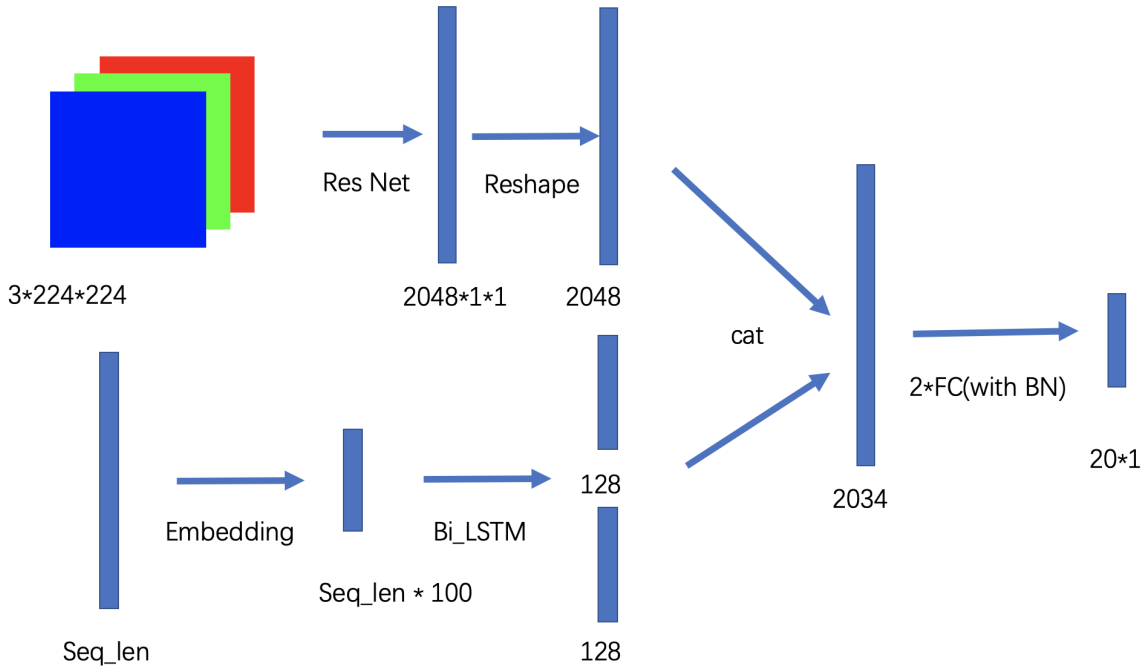

Figure 3: The architecture of CNN-RNN

and below is the code version

```
resnet_model = models.resnet50(pretrained=True)
self.resnet = nn.Sequential(*list(resnet_model.children())[:-1]) # (batch,2048,1,1)
for param in self.resnet.parameters():
    param.require_grad = False
```

```
self.embed = nn.Embedding(vocab_size, emb_dim)
self.embed.weight.data.copy_(torch.from_numpy(emb_table))
self.embed.weight.requires_grad = False
self.lstm = nn.LSTM(input_size=emb_dim, hidden_size=lstm_hidden,
    num_layers=lstm_layers, bidirectional=True, batch_first=True)
self.fc1 = nn.Linear(2048 + lstm_hidden * 2, num_class)
```

3.1.2 Convolutional Neural Network for image data

Taking into account the accuracy and running speed of the required results, the CNN network cannot be too large, but the structure cannot be too simple, so we chose the relatively good performance of pre-training model of ResNet-50 and the relatively simple structure of the pre-training model of VGG-16 from torchvision to do feature extraction.

Since the input of VGG and Resnet in torchvision is required to be 224*224*3, so we first transform the image into reasonable size.

```
train_transform = transforms.Compose([
    transforms.Resize(224),
    transforms.RandomHorizontalFlip(),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ])

test_transform = transforms.Compose([
    transforms.Resize(224),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ])
```

VGG-16

The architechture of VGG is really simple and below is the detail:

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figure 4: The architecture of VGG
(Simonyan & Zisserman, 2014)

here we utilize the VGG-16 model from torchvision for extracting the feature from image data. The output dimension of last layer is 1000 which is also the number of features.

Resnet

The architecture of ResNet is below:

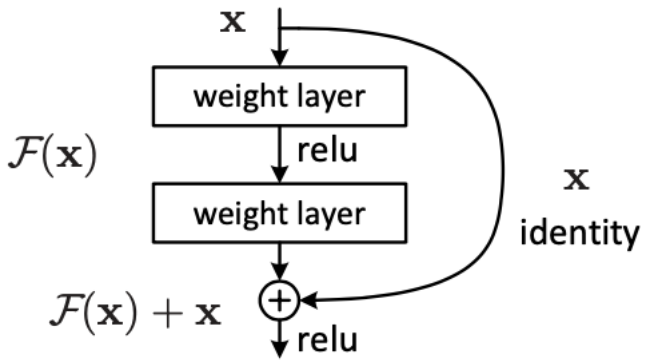

Figure 5: The architecture of ResNet
(He et al., 2016)

The key idea of ResNet is to insert shortcut connections to make the plain network into its residual version(He et al., 2016). And what that means is that it doesn't really hurt the plain neural network ability to do as well as the simpler network without insertion, because it is quite easy for it to learn the identity function. This residual trick allow us to add more layer to the network to improve the performance.

In addition, since the model of ResNet on torchvision performed quite good, so we only extracting features from the ResNet model and the output of ResNet50 on torchvision is 1000 dimensions, so we first delete the last layer of model then reshape the output of last layer from 2048*1*1 to 2048*1, which is convenient for the concatenating operation next.

RNN: Embedding and LSTM for caption data

Traditional Artificial Neural Networks simply accept a real-valued vector and output a real number, whose function is limited to a static mapping between input and output and cannot model prediction tasks that consider time. The model that take time into consider is called dynamic classifier (Staudemeyer & Morris, 2019).

To consider time into the model, signals from previous timesteps should be able to feed into the current timestep. Recurrent Neural Networks solve the problem but leave a flaw: They can only trace back at most ten timesteps. That is because previous signals are prior to either vanish or explode during backpropagation. Long Short-Term Memory Recurrent Neural Networks solve the problem and can trace back to more than 1000 timesteps (Staudemeyer & Morris, 2019).

The reason that vanilla RNN can not utilize long information is that the backpropagated error signal either explodes or vanishes, which can cause weight oscillation or unacceptable slow learning time. LSTM uses an idea called constant error carousels (CECs), which forces an error constantly flow inside some special cells, which are controlled by gate units to retain information (Staudemeyer & Morris, 2019). Bi-LSTM is a more advanced form of LSTM, which stacks two LSTM modes in reverse order together. This model can learn information from a varied time sequence.

The dataset provides a caption that describe each image, which is part of information for prediction. Using the pytorch embedded layer, captions are embedded to a sequence length * 100 matrix. One word is fed into a Bi_LSTM at one timestep, the last hidden state of the forwards LSTM and backwards LSTM is extracted, which are represented by two 128 dimensional vectors. These two vectors are concatenated with the 2048 dimensional resnet image vector and produce a 2304 dimensional vector. After passing through two full connected layer with two batch norm layer, the output is a 20 dimensional vector. To enable learning, Adam optimizer and BCElogit loss function is used, and will be introduced in later sections. The difference of using VGG16 is that the output of VGG16 is a 1000 dimensional vector, so the reshape process is not required. Then this vector is concatenated with two language vectors as a 1256 vector. The rest of model is the same, and finally a 20 dimensional vector is received. The loss function used here is BCElogit loss, while the chosen optimizer is Adadelta, which will be discussed later.

3.1.3 One-hot

The representation of labels of a image should be introduced. Labels of an image are numbers ranging from 1 to 19. Using a 20-dimensional vector, labels of an image can be represented, i.e., the position of the labels are 1, and the rest positions are 0. For example, suppose the labels of an image is 1 and 3, then the vector should be (0, 1, 0, 1, ..., 0). The ellipses omits 15 zeros.

3.1.4 Optimizer

Adam

In this project, Adam is used when resnet is used as the CNN layer. Adam is an batch optimization method that computes adaptive learning rate using first-order gradients (Kingma & Ba, 2014). To illustrate how the optimizer works, several definitions should be made:

$f(\theta)$ is a stochastic noisy objective function, which is differentiable with respect to parameters θ. $f(\theta)_1, \ldots, f(\theta)_T$ are objective functions at timestep $1 \ldots T$. g_t is the gradient of $f(\theta)_t$. There are two moving averages value that required to be updated: the exponential moving averages of the gradient at timestep t (m_t) and the squared gradient v_t. Both values are controlled by two hyperparameters $\beta_0, \beta_1 \in [0, 1)$. In timestep 0, m_0 and v_0 are initialized as 0. During timestep 1 with small β_0 and β_1 values, the bias of m_t and v_t are easy to become 0. To correct the bias, two values are used: $\hat{m}_t = m_t/\beta_1^t$ and $\hat{v}_t = v_t/\beta_2^t$. Here, β_1^t and β_2^t are β_1 and β_2 to the power of timestep t (Kingma & Ba, 2014).

During an optimization process, before iterating for convergence, step size α, exponetial decay rate β_1 and β_2, and a very small ϵ used to prevent zero division are manually set. The parameters θ are initialized either randomly or by zero vector. Parameters m and v are initialized by zero vector. During the iteration for convergence, parameters m_t, v_t, $\hat{m}_t$, $\hat{v}_t$, and

θ_t are updated using the following rules (Kingma & Ba, 2014):

$$
\begin{aligned}
m_t &\leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t \\
v_t &\leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2 \\
\hat{m}_t &\leftarrow m_t / (1 - \beta_1^t) \\
\hat{v}_t &\leftarrow v_t / (1 - \beta_2^t) \\
\theta_t &\leftarrow \theta_{t-1} - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)
\end{aligned}
$$

which is referenced from (Kingma & Ba, 2014). The θ_t when convergence will return from the optimizer.

An Adam optimizer has several advantages: First, gradient rescaling is not affected by the size of parameter updates. Second, the hyperparameter of stepsize controls stepsize. Third, an Adam does not need a fixed aim. Forth, sparse gradients can be used in an Adam optimizer. Finally, the stepsize naturally fade in time (Kingma & Ba, 2014).

Adadelta

In this project, an Adadelta optimizer is used when VGG is used as the CNN, which is a method that uses only first-order information to get per-dimension learning rate and perform grandeint descent (Zeiler, 2012).

Several definitions are required to continue: t is the timestep. g_t is the gradient of timestep t. $E[g^2]_t$ is the average of squared of gradient at time t, which will decay in time exponentially. It is also called the accumulation of gradient. ρ is the decaying rate that is used to the decay of $E[g^2]_t$. The square root of $E[g^2]_t$ is actually needed for parameter updating, and the problem of $E[g^2]_t = 0$ should be prevented, So we define $RMS[g]_t = \sqrt{E[g^2]_t + \epsilon}$, where ϵ is a small constant (Zeiler, 2012).

Before updating, the decay rate ρ and the small constant ϵ should be manually choosen, and parameters x_1 should be initialized either by random initialization or by zero vector. The accumulation variable $E[g^2]_0$ and $E[\Delta x^2]_0$ are initialized as 0. During timestep 1 to timestep T (largest timestep), the gradient g_t should be computed first. Other variables are computed as the following (Zeiler, 2012):

$$
\begin{aligned}
&E[g^2]_t = \rho E[g^2]_{t-1} + (1 - \rho) g_t^2 \\
&RMS[\Delta x]_{t-1} = \sqrt{E[\Delta x^2]_{t-1} + \epsilon} \\
&RMS[g]_t = \sqrt{E[g^2]_t + \epsilon} \\
&\Delta x = -\frac{RMS[\Delta x]_{t-1}}{RMS[g]_t} g_t \\
&E[\Delta x^2]_t = \rho E[\Delta x^2]_{t-1} + (1 - \rho) \Delta x_t^2 \\
&x_{t+1} = x_t + \Delta x_t
\end{aligned}
$$

which is referenced from (Zeiler, 2012). x_T will return from the optimizer.

An Adadelta optimizer has the following benefits: First, operator don't need to manually set learning rate. Second, the reacts of an Adadelta optimizer towards hyperparamers is insensitive. Third, the dynamic learning rate of each dimension is separated. Fourth, the computation of

gradient descent is minimal. Fifth, the performance of this optimizer is very stable towards large gradient, a dataset with a lot of noise, and different architectures. Finally, it is adaptable to local and distributed environments (Zeiler, 2012).

3.1.5 BCElogit loss function

BCElogit loss function is a combination of binary cross entropy loss function and sigmoid function. For an input x the binary cross entropy loss function will produce:

$$l(x, y) = L = \{l_1, \ldots, l_N\}^T,\ l_n = w_n[y \cdot \log x_n + (1 - y_n) \cdot \log(1 - x_n)]$$

referenced from (Paszke et al., 2019). Input x to the sigmoid function will product:

$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$

referenced from (Paszke et al., 2019). The BCElogit loss function uses the sigmoid function result as the input to binary cross entropy loss function:

$$l(x, y) = L = \{l_1, \ldots, l_N\}^T,\ l_n = w_n[y \cdot \log \sigma(x_n) + (1 - y_n) \cdot \log(1 - \sigma(x_n))]$$

referenced from (Paszke et al., 2019).

The reason that BCElogit is used as the loss function is the way multi-labels are handled. To represent the multi-labels of an image, a vector is used, whose length is the total number of distinct labels. Only labels that are contained in the image, those vector positions are set to 1. For example, suppose there are 8 distinct labels, and label 0 and label 6 are involved. The label vector is represented as [1, 0, 0, 0, 0, 0, 1, 0]. This representation is similar to one-hot embedding but cannot use softmax to deal with, since the highest predicted probability cannot be used in predicting multi-labels. The correct solution is to predict every label position using binary-classification, which is suitable to use BCElogit loss function that combines sigmoid function and binary cross entropy function.

3.1.6 Pos Weight Calculation

One of the parameter for BCElogit loss in pytorch is pos weight. Adding pos weight is an effective approach to force the model to treat each label equally. When the unevenly distributed data is provided, the loss will be calculated differently based on the frequency of appearance of the label. The label that is rarely seen will have large penalty comparing to the label that is frequently seen. The pos weight is calculated by dividing negative sample by positive sample.

3.2 The Reasonability and Benefit of The Method

The reason CNN-RNN is used is that it combines the benefits of CNN in processing images and RNN models in processing natural languages. A fully-connected neural network has to process too many parameters for an image, even if computing sources are unlimited, considering all the details of an image is prior to overfitting. The solution CNN used is weight sharing, that

is, using convolution technique to break image matrix into several smaller parts, which share the same weight matrix(LeCun et al., 1990). The following pooling layer can extract essential features. As a result, CNN is a good method in single-label image classification(Wang et al., 2016). However, in the real world, an image often has multiple labels, and these labels can have some internal relationships. Multiple CNN models can be applied to predicted each label independently. However, this kind of model repeatedly invoke the same CNN models, which cost large amount of resources, and it ignores the possible co-appearance of individual labels. One possible solution is using RNN to process labels and to memorize the relationship between single labels (Wang et al., 2016). As for the project, besides labels, a short description of an image (which is called "caption" in the dataset) is provided. Since a caption contains more information than labels, captions are fed into the model instead of labels. Before concatenating with processed image data, captions are handled by LSTM. Since RNN and its advance version LSTM has internal memeory, which can save information from previous timesteps, it can work on arbitrary sequences. As a result, it can work on natural language processing tasks such as processing the description of an image (Tarwani & Edem, 2017). The reason to choose LSTM instead of vanilla RNN is that captions are long paragraphs, LSTM can contain information longer than vanilla RNN.

3.3 Any advantage or novelty of the proposed method

The advantage of our model is that it incorporate information from both images and captions. Choosing Bi-LSTM to further extract caption information so that both the positional information as well as semantic meaning has been extracted. Additionally, our model size only 97MB, which is relatively small and still can achieve a desirable performance of 84.7% F1 score on the test set.

4 Experiments and Result

4.1 Best Model Experiment Data Pre-processing

The input image file will be resize to 3*224*244 and been normalized. The pre-processing of captions includes removing the punctuation then tokenizing the sentence into an array of vocabularies. The vocabularies then will be lemmatized. The Lemmatization process will convert the vocabularies to the dictionary form. The array of vocabularies will be converted to one hot encoding. Each caption, which is represented by an array of one hot encoding, will be padded to the target length and go through a pre-trained w2v model. In our case, we use Genism API to obtain the glove-wiki-gigword-100 model, which will convert a one hot encoding to a vector with size of 100. After the preprocessing, the shape for image input will be (batch size, 3, 224,224) and the shape for captions is (batch, target sequence length, 100).

4.2 Experiment with Positional Weight Parameter

We start by fixing the pretrain CNN model to ResNet and optimizer to Adadelta with learning rate set to 5. We choose binary cross entropy loss(BCEWithLogitsLoss) with logistics as the loss function. BCEWithLogitsLoss has the position weight parameter. The purpose of the using pos weight is to penalize the model much more when it fail to predict a class that has very low frequency of occurrence. Since our training dataset highly unevenly distributed, pos weight will be helpful to make the model treat each class equally.

Setting	With Pos Weight	Without Pos Weight
Starting Loss	15.53	4.07
Loss After 5 Epoch	11.34	2.41
Best F1	0.633	**0.847**
Running Time	3850s	3900s

Table 1: Positional Weight Experiment

After 5 epoch, the loss decreased only 27% with positional weight added. But when without pos weigh add, the loss decreased by 41%. The loss decrease faster without the pos weight added comparing to the BCEWithLogitsLoss with the pos weight parameter given. Additionally, the pos weight parameter will limit the performance of the model. This is highly likely due to the distribution of the test set is very similar to the training set and pos weight will force the model to treat each label equally when in fact their are not.

4.3 Experiment with different LSTM layer

By setting the loss function to BCEWithLogitsLoss without pos weight given, optimizer set to Adadelta, and only changing the LSTM layers, the result is the following:

Layers	1	2	4	8
F1 Score	**0.831**	0.810	0.770	0.630

Table 2: LSTM layers experiment

As the number of the layers of LSTM model increase, the performance of the model decrease. The reason could due to the fact the caption is relative short, thus with one layer will be enough to extract the information needed. Adding more layers will cause vanish gradient problem and make the model hard to converge.

4.4 Experiment with different CNN model

When we changing the pretrained CNN model from ResNet to VGG and set other settings unchanged, the performance changes by about 1% with VGG slightly better than ResNet.The difference is so small that it is reasonable to believe the change is due to the discrepancy during

training the model rather than conclude that VGG is better than ResNet as a better image feature extrator.

4.5 Other Attempt besides Best Model

4.5.1 YOLOv5 result

We utilized pre-trained models of YOLOv5, which includes the small size yolov5s, the middle size yolov5m, the large size yolov5l and the largest size yolov5x. The speed and accuracy of them are showed in the official document. The larger the model size, the higher the accuracy, but the slower the running speed; conversely, the smaller the model size, the lower the accuracy will be, but the running speed is faster. And below is the result for our data set:

Model	yolov5s	yolov5m	yolov5l	yolov5x	yolov5x6
F1 score	0.766	0.777	0.787	**0.790**	0.778
Running Time	**438s**	492s	601s	920s	10830s

Table 3: F1 score on different YOLO model

The result of YOLO is not ideal and below is the reason we considered:

- **Reason 1** The YOLO method model training relies on object recognition and annotation data. Therefore, the detection effect of YOLO is not ideal for unconventional object shapes or proportions. Like the detection Figure6 below, it is difficult for YOLO to detect the hand holding the cup, but the caption of the image describes that there is a person in that picture.

(a) After detection

(b) Original image

Figure 6: Result of YOLO detection

- **Reason 2** In the YOLO loss function, the IOU error of the large object and the IOU error of the small object have close contribution to the loss in the network training (although the

square root method is used, it does not fundamentally solve the problem). Therefore, for small objects, a small IOU error will also have a great impact on the network optimization process, thereby reducing the positioning accuracy of object detection.

4.5.2 CNN Encoder and RNN Decoder

When we try to use the structure similar to image captioning to solve the problem, the result is undesired. The labels are treated as captions, which will append SOS(start of sentence) in the beginning and EOS (end of sentence) token at the end. The SOS is represented by using number 20 and EOS is represented by using number 21. The number is chosen since we only have 20 labels range from 0 to 19. For instance, a given label [1,3] will be converted to [20, 1, 3, 21]. The image captioning structure normal is an encoder-decoder structure. We use ResNet as the encoder to extract image information. The image feature and the previous prediction of label is used as input for the decoder model. Decoder will convert the labels to embedding and concat with the image feature. Then the concated matrix is feed into LSTM. The result of LSTM is connected to Full Connected Layer (FC). The advantage of using RNN to multi-label tasks is the RNN will incorporate the semantic dependency and label dependency. The semantic dependency is when label has similar meaning. For instance, label cat is semantically connect to label kitten. Label dependency is when two labels has very high co-occurrence rate. For instance, label boat is highly likely to co-occur with label sea(Wang et al., 2016). Even the logic is sounding, our implement fail to achieve a desirable outcome. The model will keep predicting label 1 as the correct answers. We believe this could be the incorrect implementation of our model or due to the dataset is heavily concentrated to label one.

4.6 Hardware and Software

We use colab for all our experiments and the details of hardware and software is below:

```
OS info:
Linux 8797690000d4 5.4.109+ #1 SMP Tue Apr 20 19:55:43 PDT 2021 x86_64 x86_64 x86_64 GNU/Linux

Python version:
Python 3.7.10

Cuda version:
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2020 NVIDIA Corporation
Built on Wed_Jul_22_19:09:09_PDT_2020
Cuda compilation tools, release 11.0, V11.0.221
Build cuda_11.0_bu.TC445_37.28845127_0

GPU info:
Sun May 30 11:46:08 2021
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 460.32.03    Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla K80           Off  | 00000000:00:04.0 Off |                    0 |
| N/A   73C    P8    35W / 149W |      0MiB / 11441MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

Figure 7: Hardware and Software Infomation

5 Conclusion and Discussion

We purposed a rather simple but effective model which uses pre-trained CNN model to extract image feature and use pre-train W2V model with Bi-LSTM model to extract caption features. By concating the image feature with caption feature and feed into fully connected layer, the model is able to incorporate both information and let the model decide which information is important in unsupervised manner. Additionally, the result of YOLO model shows that only using image data is not sufficient to have a good performance since many images contains small objects that is extremely hard for conputer vision model to pick up, thus captions data is necessary to achieve a better performance.

A Appendix

A.1 Instruction for running the code

- To start with running the cells in the 'Loading data from the google drive" to download the data.

- Run the cell in the 'Define fix parameter' to define the data path.
- The cells in 'Define Helper Functions' section need to be executed to make sure data preprocessing and training process run smoothly.
- The 'Define hyper parameters part' define the lstm layers, lstm hidden units, pretrain cnn model used to extract image, epoch to run and learning rate. User can adjust these paramters to achieve different performance.
- To experiment with the effect of pos weight for the loss function, go to 'Training process' section and 'Define model, loss function and optimizer' section, uncomment the code with pos_weight added.
- After adjusting the hyper parameters, sequentially execute the following sections to train the model and make predictions on the test dataset

A.2 Hyper-parameters for best model

```
lstm_layers = 1
lstm_hidden = 128
num_class = 20
batch_size = 32
num_epoch = 30
learning_rate = 5
num_epochs = 10
pretrain_cnn='Resnet' # 'VGG' for VGG-16 model
```

References

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. , 770–778.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

LeCun, Y., Boser, B. E., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. E., & Jackel, L. D. (1990). Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems* (pp. 396–404).

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in neural information processing systems 32* (pp. 8024–8035). Curran Associates, Inc. Retrieved from http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-de

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection.

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. , 1409.1556.

Staudemeyer, R. C., & Morris, E. R. (2019). Understanding lstm–a tutorial into long short-term memory recurrent neural networks. *arXiv preprint arXiv:1909.09586*.

Tarwani, K. M., & Edem, S. (2017). Survey on recurrent neural network in natural language processing. *Int. J. Eng. Trends Technol*, *48*, 301–304.

Wang, J., Yang, Y., Mao, J., Huang, Z., Huang, C., & Xu, W. (2016). Cnn-rnn: A unified framework for multi-label image classification. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 2285–2294).

Zeiler, M. D. (2012). Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.