Project ASSIGNMENT 1:

Name: Akash Verma

Aim

We are well familiar with the “fill in the blanks” homework where we choose the most suitable word from the given words and fill the blanks to complete the sentence. Suppose you have an image of the homework (Please refer attached images). You want to complete the homework (i.e. fill those blanks with a proper word from the box) using image processing and Machine Learning/ Deep Learning.

Libraries

- pytesseract
- pathlib
- numpy
- torch
- overrides
- transformers
- deeppavlov
- PIL
- cv2
- typing

Language

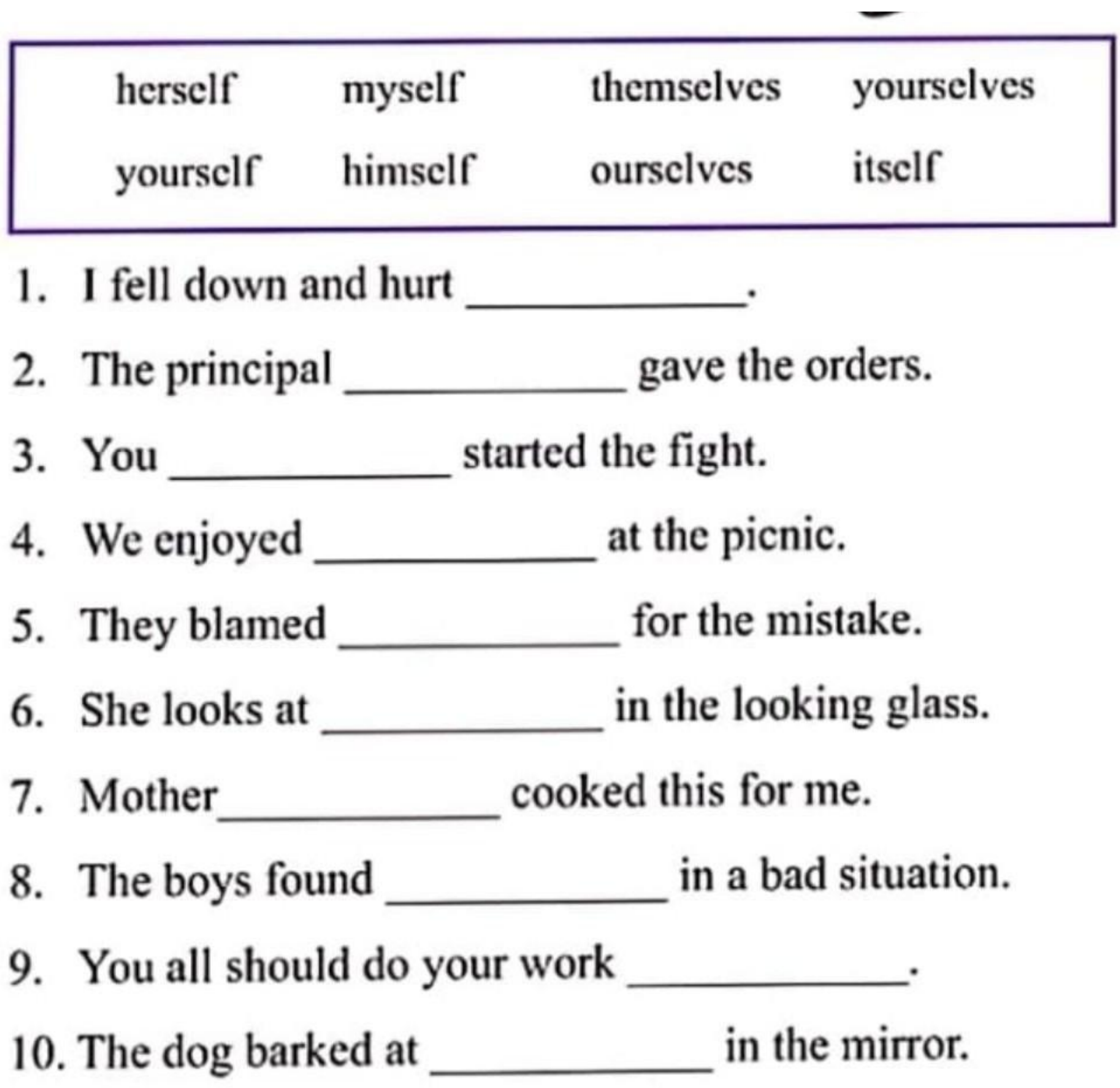
- Python3

How to run

1. Execute the following command:

   > python3 main.py

2. Then a selected image(eg. 2,jpg)window will open,

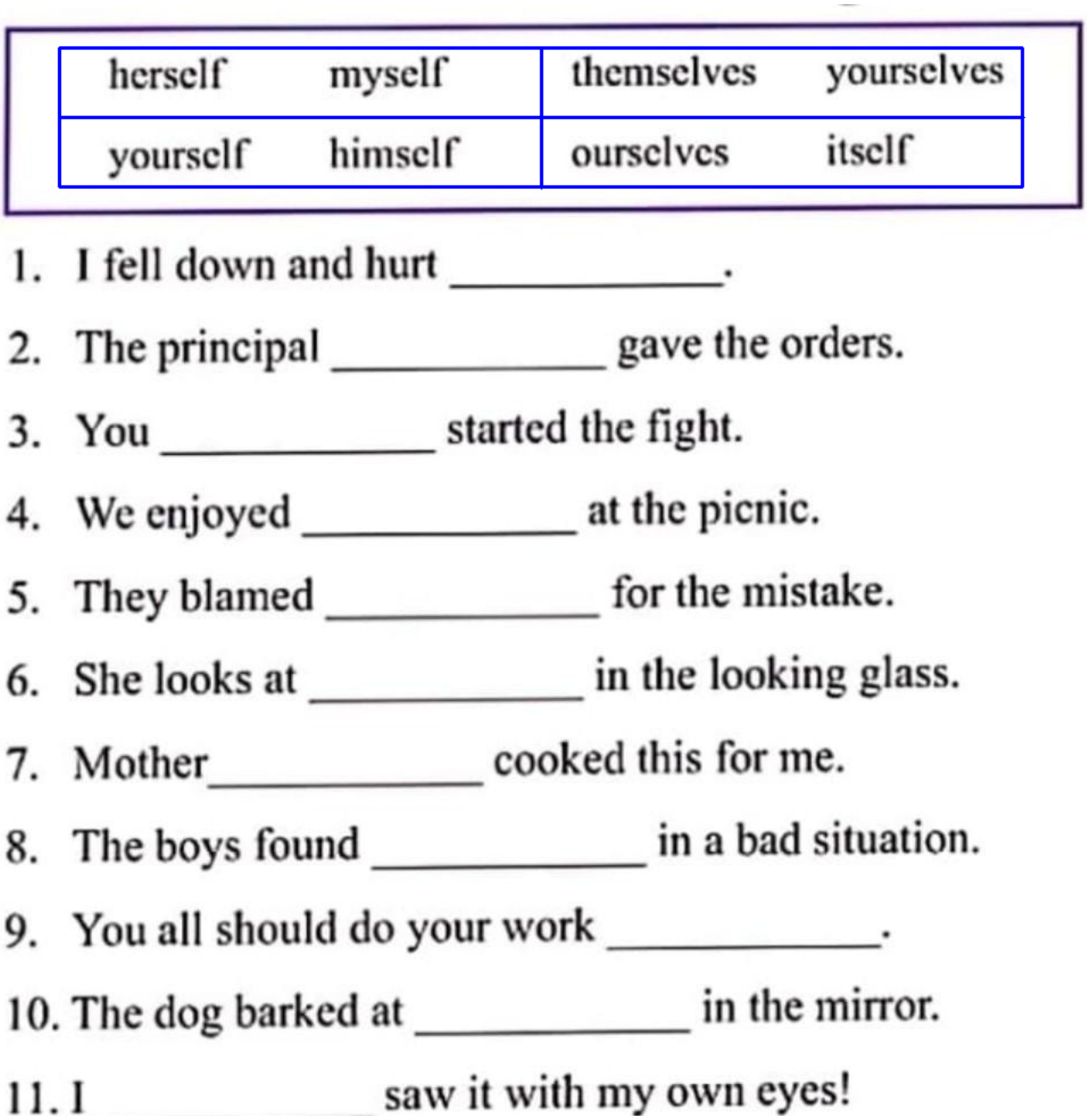


herself myself themselves yourselves

yourself himself ourselves itself

1. I fell down and hurt ____________.
2. The principal ____________ gave the orders.
3. You ____________ started the fight.
4. We enjoyed ____________ at the picnic.
5. They blamed ____________ for the mistake.
6. She looks at ____________ in the looking glass.
7. Mother____________ cooked this for me.
8. The boys found ____________ in a bad situation.
9. You all should do your work ____________.
10. The dog barked at ____________ in the mirror.

3. First select the options in the given box,

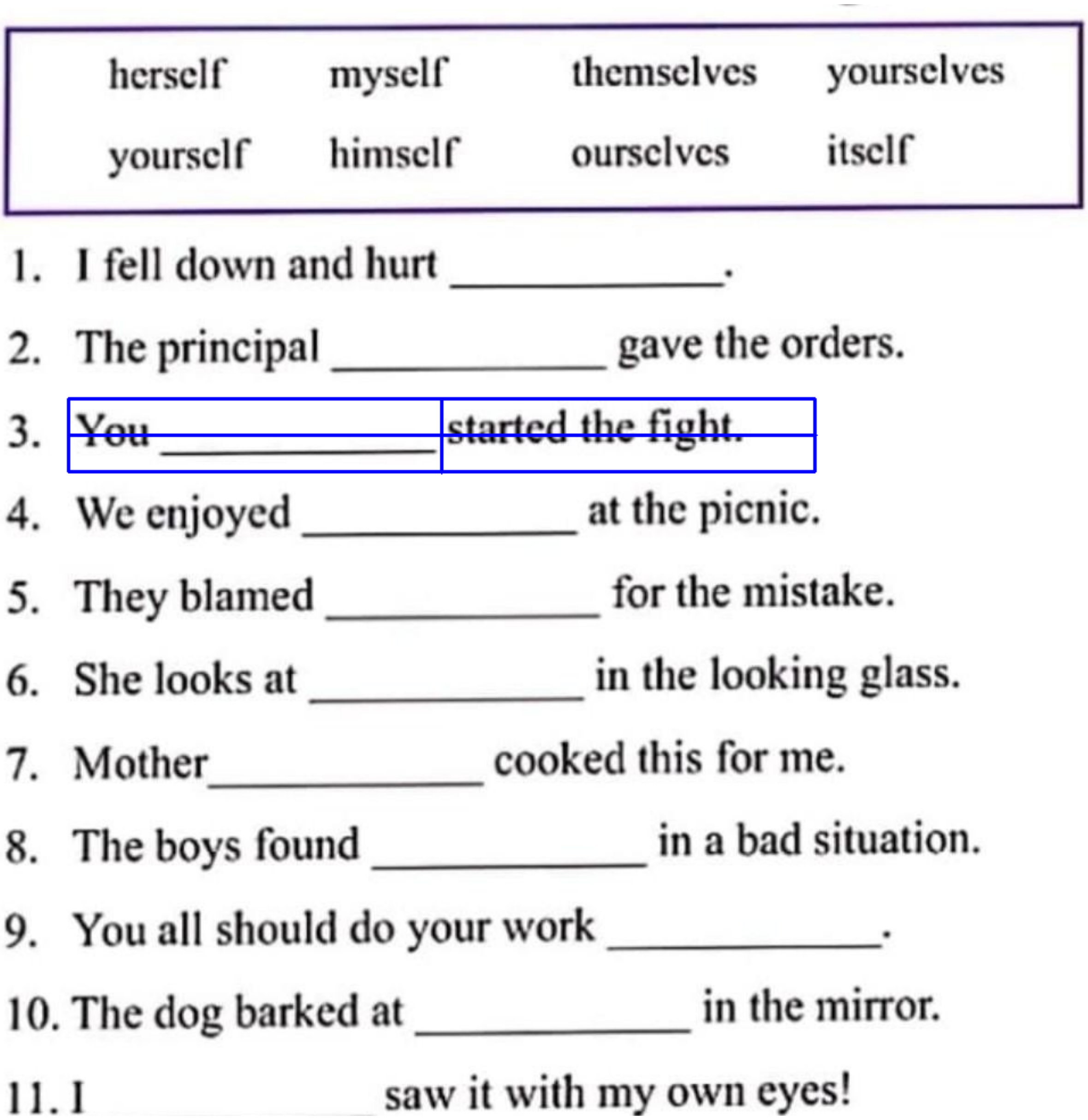


herself	myself	themselves	yourselves
yourself	himself	ourselves	itself

1. I fell down and hurt ____________.
2. The principal ____________ gave the orders.
3. You ____________ started the fight.
4. We enjoyed ____________ at the picnic.
5. They blamed ____________ for the mistake.
6. She looks at ____________ in the looking glass.
7. Mother____________ cooked this for me.
8. The boys found ____________ in a bad situation.
9. You all should do your work ____________.
10. The dog barked at ____________ in the mirror.
11. I ____________ saw it with my own eyes!

4. Press ENTER

5. Again new window will appear, this time select the sentence,

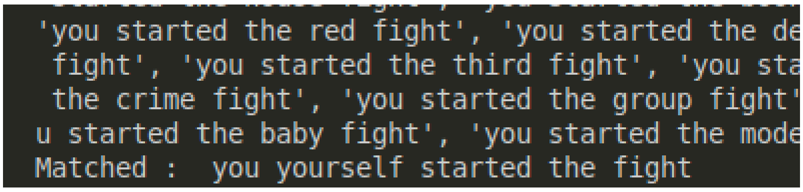


herself myself themselves yourselves

yourself himself ourselves itself

1. I fell down and hurt ____________.
2. The principal ____________ gave the orders.
3. You ____________ started the fight.
4. We enjoyed ____________ at the picnic.
5. They blamed ____________ for the mistake.
6. She looks at ____________ in the looking glass.
7. Mother____________ cooked this for me.
8. The boys found ____________ in a bad situation.
9. You all should do your work ____________.
10. The dog barked at ____________ in the mirror.
11. I saw it with my own eyes!

6. The selected options and sentences obtained are stored in the following files:
   - `crp_opt.txt` contains OPTIONS.
   - `crp_qs.txt` contains SENTENCES.
7. Output: the resultant answer for fill in the blanks will be denoted by "**Matched :** "



```
 'you started the red fight', 'you started the de
  fight', 'you started the third fight', 'you sta
  the crime fight', 'you started the group fight'
 u started the baby fight', 'you started the mode
 Matched :  you yourself started the fight
```

File structure

```
.
├── main.py
├── crp_opt.txt
├── crp_qs.txt
├── 1.jpg
├── 2.jpg
├── 3.jpg
├── 4.png
├── 5.jpg
├── 6.jpg
├── readme.pdf
└── readme.md
```

Code explanation

crp_opt.txt

Contains the options of the selected image.

crp_qs.txt

Contains the sentences of the selected image.

main.py

In this file I followed the following steps:

1. Using Image processing(openCV, tesseract)

```
from PIL import Image
import pytesseract
import cv2
import numpy as np

# print(pytesseract.get_tesseract_version())

# quit(0)

 # Read image
im = cv2.imread("2.jpg")

# Select ROI
r = cv2.selectROI(im)

# Crop image
imCrop = im[int(r[1]):int(r[1]+r[3]), int(r[0]):int(r[0]+r[2])]
a = pytesseract.image_to_string(imCrop)
imag = open("crp_opt.txt","w+")
imag.write(a)
imag.close()
```

```
my_file = open("crp_opt.txt", "r")
content = my_file.read()

content_list = [ line.split(' ') for line in content.split("\n")]
content_list.pop()

content_list1 = []
for line in content_list:
#     new_line = list()
    for word in line:
        new_word = ''.join([  x for x in word if ord(x.lower())<=ord('z') and
ord(x.lower())>=ord('a')])
        if len(new_word)>0:
            content_list1.append(new_word)
#         print(new_word)
#     if len(new_line)>0:
#         content_list2.append(new_line)

my_file.close()

print(content_list1)

options = content_list1

#################################################################################

r = cv2.selectROI(im)

imCrop = im[int(r[1]):int(r[1]+r[3]), int(r[0]):int(r[0]+r[2])]
a = pytesseract.image_to_string(imCrop,config='--oem 1 --psm 6')
imag = open("crp_qs.txt","w+")
imag.write(a)
imag.close()

my_file = open("crp_qs.txt", "r")
content = my_file.read()

content_list = [ line.split(' ') for line in content.split("\n")]
content_list.pop()
print(content_list)

content_list2 = []
for line in content_list:
    new_line = list()
    for word in line:
        new_word = ''.join([  x for x in word if (ord(x.lower())<=ord('z') and
ord(x.lower())>=ord('a')) or (x=='-' or x=='_' or x=='[' or x == ']')])
        if len(new_word)>0 and np.array([ 1 if x=='-' or x=='_' else 0 for x in
new_word]).all():
            new_word='[MASK]'
        if len(new_word)>0:
            new_line.append(new_word)

#         print(new_word)
```

```
    if len(new_line)>0:
        content_list2.append(new_line)

my_file.close()

print(content_list2)
```

- selected the image using computer vision and image processing.
- As there was not any information regading the position of the options box I choosed to select the options manually from the image and after cleaning converted into text converted using python tesseract library and appended it into the list(content_list_1).
- Similarly, again selected region of interest from the image i.e sentences and after cleaning the data appended into another list(content_list2).

2. NLP Model(Machine Learning)

```
class TorchTransformersMLMPreprocessor(Component):
    def __init__(self,
                 vocab_file: str,
                 do_lower_case: bool = True,
                 max_seq_length: int = 512,
                 return_tokens: bool = False,
                 **kwargs):
        self.max_seq_length = max_seq_length
        self.return_tokens = return_tokens
        if Path(vocab_file).is_file():
            vocab_file = str(expand_path(vocab_file))
            self.tokenizer = AutoTokenizer(vocab_file=vocab_file,
                                           do_lower_case=do_lower_case)
        else:
            self.tokenizer = AutoTokenizer.from_pretrained(vocab_file,
do_lower_case=do_lower_case)
    def __call__(self, texts_a: List[str]):
        input_features = []
        tokens = []
        mask_idxs = []
        for text_a in texts_a:
            encoded_dict = self.tokenizer.encode_plus(
                text=text_a, add_special_tokens=True, max_length=self.max_seq_length,
                pad_to_max_length=True, return_attention_mask=True, return_tensors='pt')
            curr_features = InputFeatures(input_ids=encoded_dict['input_ids'],
                                          attention_mask=encoded_dict['attention_mask'],
                                          token_type_ids=encoded_dict['token_type_ids'],
                                          label=None)
            input_features.append(curr_features)
            if self.return_tokens:

tokens.append(self.tokenizer.convert_ids_to_tokens(encoded_dict['input_ids'][0]))
            tokens = self.tokenizer.convert_ids_to_tokens(encoded_dict['input_ids'][0])
            mask_idx = 0
            for i in range(len(tokens)):

                if tokens[i] == '[MASK]':
                    mask_idx = i
            mask_idxs.append(mask_idx)
        if self.return_tokens:
            return input_features, tokens, mask_idxs
        else:
            return input_features, mask_idxs

class TorchTransformersMLMModel(TorchModel):
    def __init__(self,
                 pretrained_bert,
                 preprocessor,
                 optimizer: str = "AdamW",
                 optimizer_parameters: dict = {"lr": 1e-3, "weight_decay": 0.01,
"betas": (0.9, 0.999), "eps": 1e-6},
                 clip_norm: Optional[float] = None,
                 bert_config_file: Optional[str] = None,
                 **kwargs) -> None:
        self.preprocessor = preprocessor
        self.pretrained_bert = pretrained_bert
        self.bert_config_file = bert_config_file
        self.clip_norm = clip_norm
        super().__init__(optimizer=optimizer,
                         optimizer_parameters=optimizer_parameters,
                         **kwargs)

    def load(self, fname=None):

        if self.pretrained_bert:
            log.info(f"From pretrained {self.pretrained_bert}.")
            config = AutoConfig.from_pretrained(self.pretrained_bert,
                                                output_attentions=False,
output_hidden_states=False)

            self.model = AutoModelForMaskedLM.from_pretrained(self.pretrained_bert,
config=config)
        else:
            print("Invalid bert model")
    def __call__(self, sentences, options):
        features, mask_idxs = self.preprocessor(sentences)
        _input = {}
        for elem in ['input_ids', 'attention_mask', 'token_type_ids']:
            _input[elem] = [getattr(f, elem) for f in features]
        for elem in ['input_ids', 'attention_mask', 'token_type_ids']:
            _input[elem] = torch.cat(_input[elem], dim=0).to(self.device)

        with torch.no_grad():
            tokenized = {key:value for (key,value) in _input.items() if key in
self.model.forward.__code__.co_varnames}
            logits = self.model(**tokenized)['logits']
            # return logits
        # options = 5
        output = []

        for i in range(len(sentences)):
            idx = logits[i][mask_idxs[i]].topk(options).indices
            top_mask_tokens = idx
            predicted = []
            idx_copy = _input['input_ids'][i]
            for j in range(options):
                idx_copy[mask_idxs[i]] = top_mask_tokens[j]
                tmp =
self.preprocessor.tokenizer.convert_tokens_to_string(self.preprocessor.tokenizer.convert
_ids_to_tokens(idx_copy))
                predicted.append(re.sub(r"\[CLS\]|\[SEP]|\[PAD]", "", tmp).strip())
            output.append(predicted)
        return output

    def train_on_batch():
        pass

torch_preprocesor = TorchTransformersMLMPreprocessor('bert-base-uncased',
max_seq_length=64)

# features, mask_idxs = torch_preprocesor()
model = TorchTransformersMLMModel(pretrained_bert = 'bert-base-uncased',preprocessor
=torch_preprocesor,  save_path = './sample_data') #'/content/sample_data'
```

Used pre-trained Torch Transformers models

3. Searching for perfect sentence

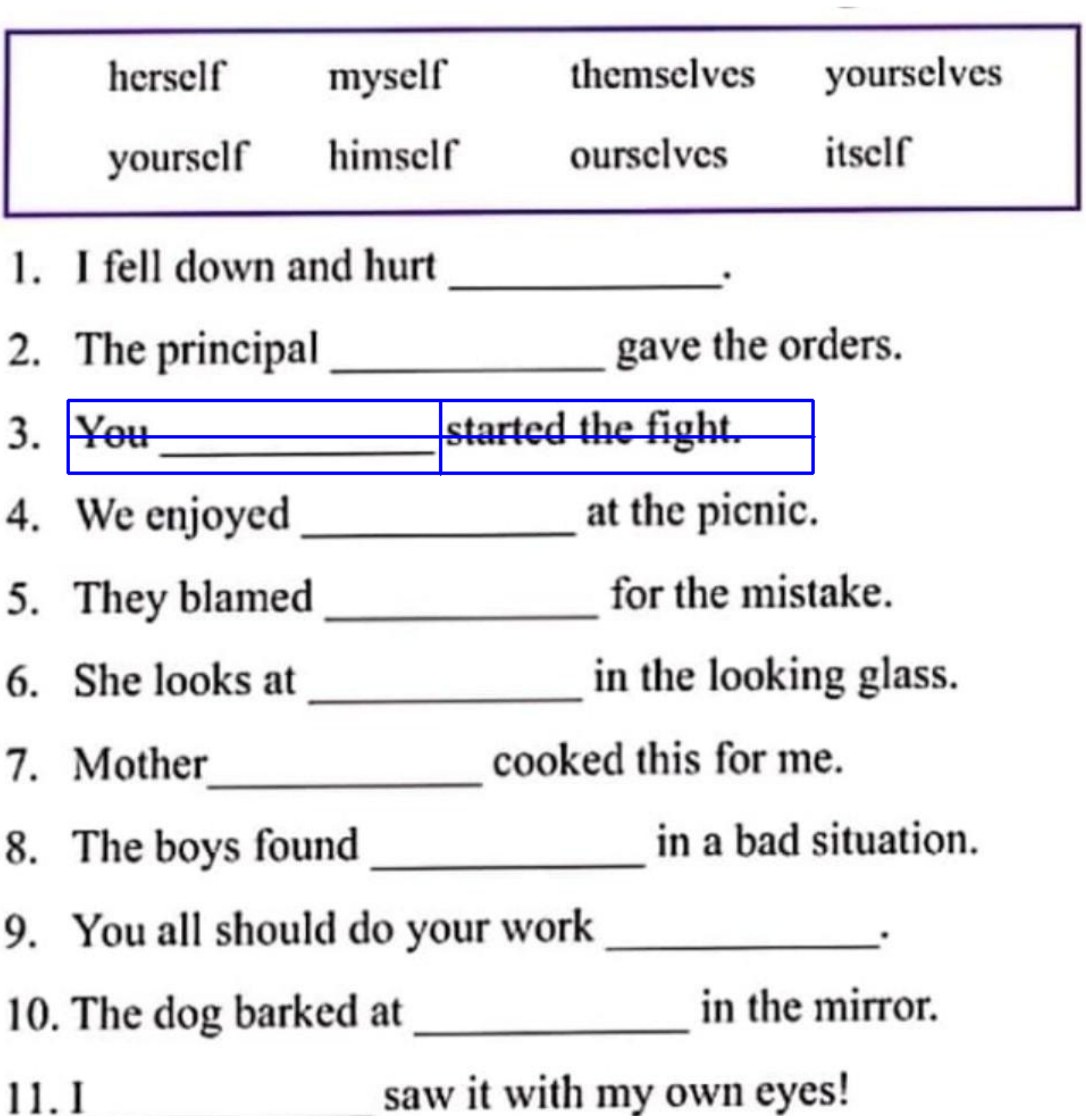
Since I was unable to find the position of blank, I used the brute force algorithm to compare the sentences across all the matches. Our algorithm is as follows. Suppose the example is ‘The boy __ the cake.’ We make all the possible sentences with the words that we can fill in the blank. ‘The boy eats the cake’ ‘The boy drinks the cake’ ’The boy itself the cake’ ….

We compare each of these sentence to the sentence produces by the NLP model. Our intuition is that the NLP model will definitely produce a matching results, since the number of words that makes sense at those position are finite. Hence using matching, we are able to find the required sentence.

```
sen = content_list2
ll = content_list1
for a in content_list2:
    sen = [x.lower() for x in a]
    print(sen)
    all_sentences = []
    for i in range(len(sen)):
        for k in range(len(ll)):
            newsen = ''
            for j in range(len(sen)):
                if j == i:
                    newsen+= ll[k] + ' '
                if j!= len(sen) -1:
                    newsen+= sen[j] + ' '
                else :
                    newsen+= sen[j]
            all_sentences.append(newsen)

    print(all_sentences)

    predicted_sentences = []
    for i in range(len(sen)):
        newsen = ''
        for j in range(len(sen)):
            if j == i:
                newsen+= '[MASK] '

            newsen+= sen[j] + ' '

        sentences = [newsen]
        predicted_sentences.append(model(sentences, 100))


# print(predicted_sentences)
    print(predicted_sentences)
    flag = 0
    for i in all_sentences:
        for j in predicted_sentences:
            for k in j:
                for l in k:
                    # print(i)
                    if i == l :
                    # print(i)
                    # print("Here")
                        print("Matched : " , i)
                        flag = 1
                        break

                if flag == 1:
                    break
            if flag == 1:
                break
        if flag == 1 :
            break
    if flag == 1:
        break
```

OUTPUT EXPLAINED

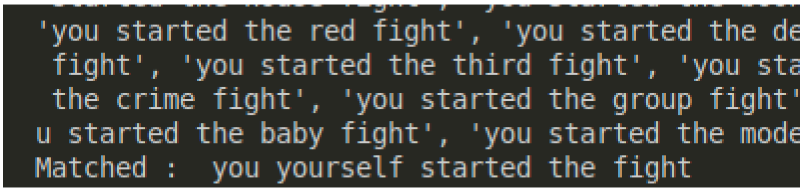
Selected fill in the blank:



herself	myself	themselves	yourselves
yourself	himself	ourselves	itself

1. I fell down and hurt _____________.
2. The principal _____________ gave the orders.
3. You _____________ started the fight.
4. We enjoyed _____________ at the picnic.
5. They blamed _____________ for the mistake.
6. She looks at _____________ in the looking glass.
7. Mother_____________ cooked this for me.
8. The boys found _____________ in a bad situation.
9. You all should do your work _____________.
10. The dog barked at _____________ in the mirror.
11. I _____________ saw it with my own eyes!

Blank filled by the suitable option from the given options:



```
'you started the red fight', 'you started the de
 fight', 'you started the third fight', 'you sta
 the crime fight', 'you started the group fight'
u started the baby fight', 'you started the mode
Matched :  you yourself started the fight
```

OPTIMIZATION FURTHER

- ***We can optimise further by using a better image processing solution. Currently, whenever our system fails, most of the time it is the due to the failing of the image processing model.***

- ***We can automate the process of seperating the option words with the fill the blanks sentence by using a targeted dataset for this problem.***