Chanatip Kowsurat 6530075721

Notebook #1 Image Classification with Vision Transformer

Input: Images of healthy and unhealthy bean leaves from Hugging Face
Output: Predicted class labels for input images

Hardware requirement: NVIDIA GPUs

Data Statistics

- Dataset: Images of healthy and unhealthy bean leaves
- Size: 1,300 images
- Classes: angular_leaf_spot, bean_rust and healthy
- Splits: Training 80%, Validation 10% and Testing 10%

Process

1. Preprocess
   1.1. Install necessary libraries
   1.2. Load Beans dataset from Hugging Face
   1.3. Split datasets into 1034 train images, 133 validation images and 128 test images
   1.4. Load pretrained Vision Transformers feature extractor

2. Training
   2.1. Define data collator
   2.2. Define evaluation metric
   2.3. Define training configurations
   2.4. Load model
   2.5. Train model

3. Evaluation

4. Prediction
   - Predict images and print report
   - Display tensor board

Learning curve & Metrics

- Training uses loss function monitoring.
- Evaluation metrics: Accuracy and Loss reduction over epochs

Fine-tuning techniques

- Feature extraction: Pre-trained ViT model is adapted to the new dataset
- Training approach: Freezes some model layers, optimizes classification head and uses standard training loop with Trainer API

Important commands

```
%pip install ...                                          # install packages
ds = load_dataset("beans")                                # load the dataset
ViTImageProcessor.from_pretrained(...)                    # load feature extractor
def collate_fn(batch):                                    # define data collator
def compute_metrics(...)                                  # define evaluation metrics
model = ViTForImageClassification.from_pretrained(...)    # load model
training_args = TrainingArguments(...)                    # set training config
trainer.train()                                           # train model
trainer.evaluate()                                        # evaluate model
trainer.predict(...)                                      # make predictions
print(classification_report(...))                         # print classification report
%load_ext tensorboard                                     # load tensorboard
```

Results

[650/650 08:31, Epoch 10/10]

Epoch	Training Loss	Validation Loss	Model Preparation Time	Accuracy
1	0.589400	0.220962	0.005200	0.939850
2	0.133900	0.077834	0.005200	0.977444
3	0.035000	0.056739	0.005200	0.977444
4	0.010600	0.035540	0.005200	0.992481
5	0.004600	0.038429	0.005200	0.984962
6	0.003100	0.031750	0.005200	0.992481
7	0.002400	0.031819	0.005200	0.992481
8	0.002000	0.031823	0.005200	0.992481
9	0.001800	0.031759	0.005200	0.992481
10	0.001700	0.031705	0.005200	0.992481

```
***** train metrics *****
  epoch                    =        10.0
  total_flos               = 746244894GF
  train_loss               =      0.0785
  train_runtime            =  0:08:32.85
  train_samples_per_second =      20.162
  train_steps_per_second   =       1.267
```

[9/9 00:02]

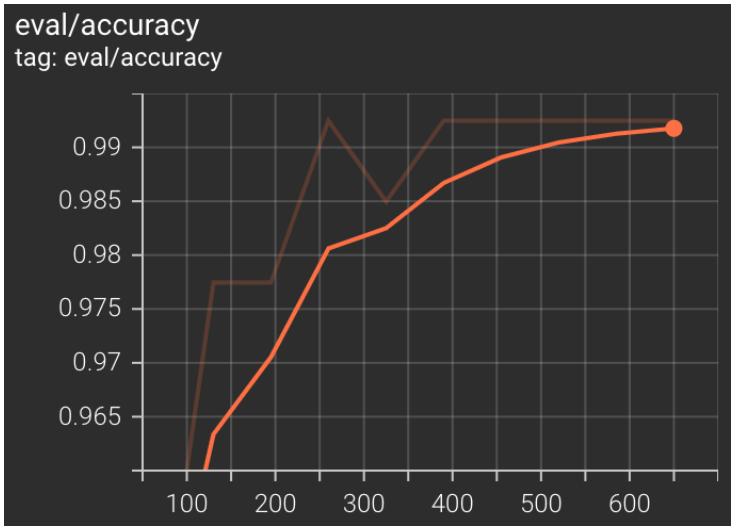
```
***** eval metrics *****
  epoch                   =        10.0
  eval_accuracy           =      0.9925
  eval_loss               =      0.0317
  eval_model_preparation_time =      0.0052
  eval_runtime            = 0:00:03.29
  eval_samples_per_second =       40.41
  eval_steps_per_second   =       2.734
```

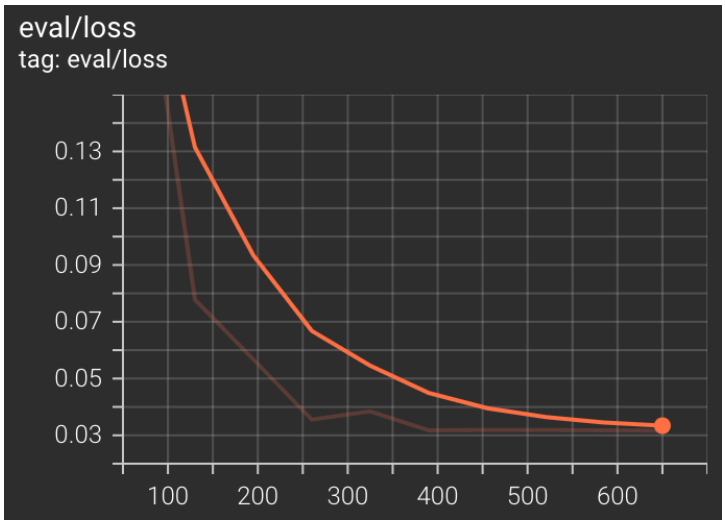
Before fine-tuning

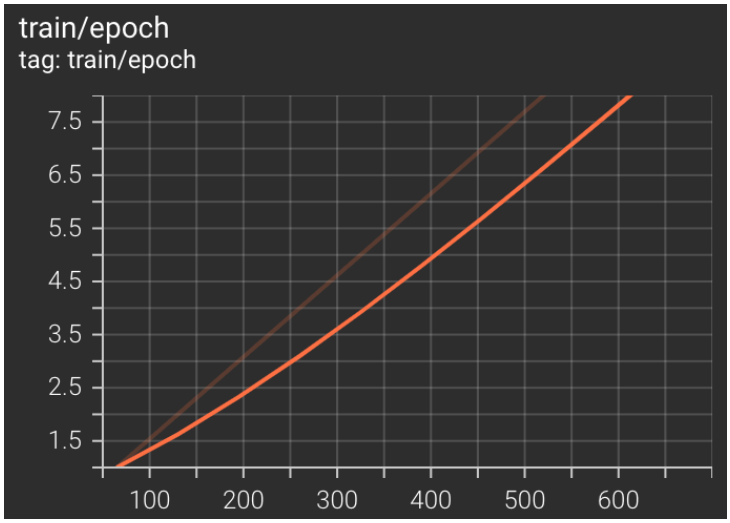
	precision	recall	f1-score	support
angular_leaf_spot	0.3617	0.3953	0.3778	43
bean_rust	0.5000	0.0465	0.0851	43
healthy	0.4026	0.7381	0.5210	42
accuracy			0.3906	128
macro avg	0.4214	0.3933	0.3280	128
weighted avg	0.4216	0.3906	0.3265	128

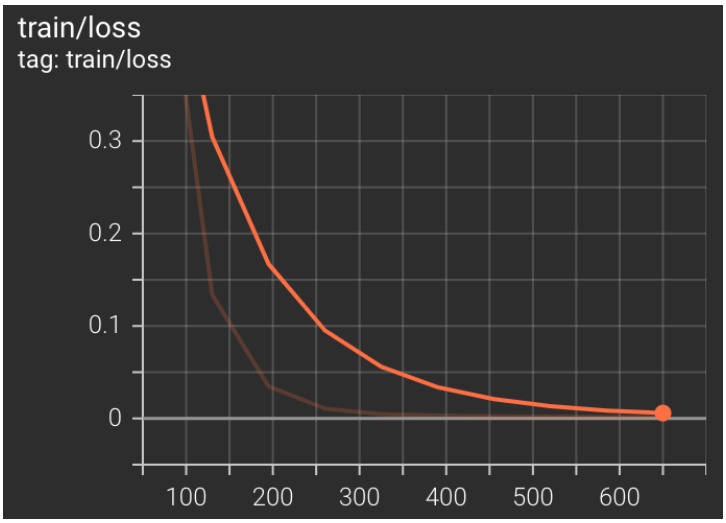
After fine-tuning

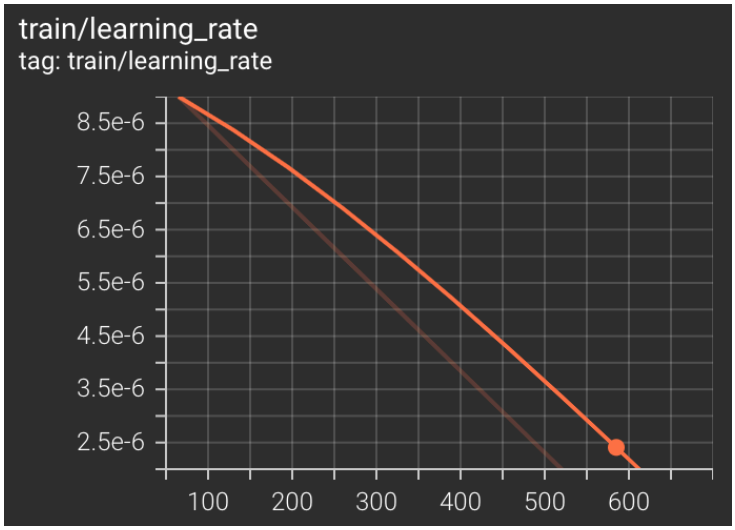
	precision	recall	f1-score	support
angular_leaf_spot	0.9762	0.9535	0.9647	43
bean_rust	0.9333	0.9767	0.9545	43
healthy	1.0000	0.9762	0.9880	42
accuracy			0.9688	128
macro avg	0.9698	0.9688	0.9691	128
weighted avg	0.9696	0.9688	0.9689	128


eval/accuracy
tag: eval/accuracy
0.99
0.985
0.98
0.975
0.97
0.965
100
200
300
400
500
600


eval/loss
tag: eval/loss
0.13
0.11
0.09
0.07
0.05
0.03
100
200
300
400
500
600


train/epoch
tag: train/epoch
7.5
6.5
5.5
4.5
3.5
2.5
1.5
100
200
300
400
500
600


train/loss
tag: train/loss
0.3
0.2
0.1
0
100
200
300
400
500
600


train/learning_rate
tag: train/learning_rate
8.5e-6
7.5e-6
6.5e-6
5.5e-6
4.5e-6
3.5e-6
2.5e-6
100
200
300
400
500
600

Notebook #2 Text classification for Sentiment Analysis

Input: Amazon Cell Phone Reviews (binary labeled, 0 represents negative reviews and 1 represents positive reviews)
Output: Predictions (positive/negative labels)

Hardware requirement: NVIDIA GPUs

Data Statistics

- Dataset: Amazon Cell Phone Reviews
- Size: 1,000 records
- Classes: positive and negative
- Splits: Training 60%, validation 20% and testing 20%

Process

1. Setup
   1.1. Load necessary libraries
   1.2. Download dataset from GitHub
   1.3. Split dataset
   1.4. Setup GPU for training

2. Preprocess
   2.1. Create text preprocess function
   2.2. Apply TF-IDF to vectorize text data

3. Train Naive Bayes Classifier
   3.1. Use cross-validation and AUC score to tune hyperparameters
   3.2. Use MultinominalNB class to find the alpha value that gives the highest CV AUC score.

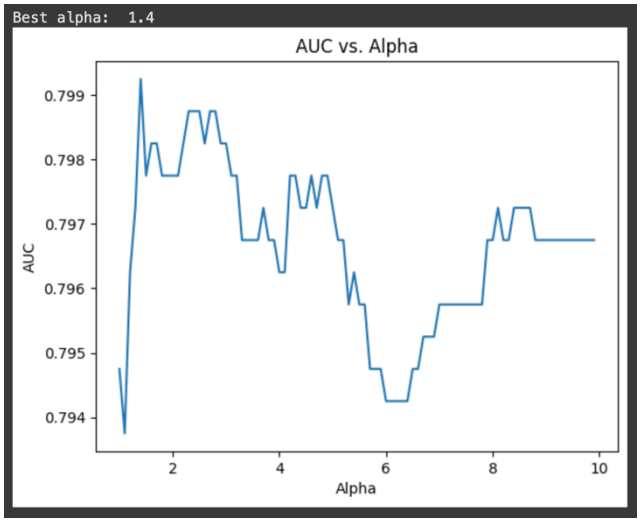
4. Evaluation
   4.1. Define ROC evaluation function
   4.2. Evaluate model with defined function

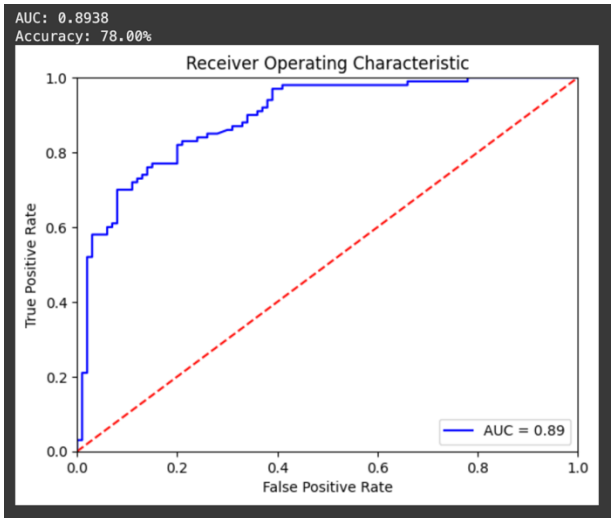
Learning curve & Metrics
Metrics: Accuracy

Important Commands

```
%pip install ...                        # install libraries
amz_review = pd.read_csv(...)           # read CSV file
amz_review.info()                       # get dataset info
train_test_split(...)                   # split dataset
def text_preprocessing(s):              # define text preprocessing function
tf_idf =    TfidfVectorizer(...)        # initialize vectorizer
tf_idf.fit_transform(...)               # apply tf-idf transformation
def get_auc_CV(model):                  # define cross-validation function
get_auc_CV(MultinomialNB(...))          # find the alpha value giving highest CV AUC score
def evaluate_roc(probs, y_true):        # define ROC evaluation function
nb_model = MultinomialNB(alpha=1.4)     # initialize NB model
nb_model.fit(...)                       # fit the model
nb_model.predict_proba(....)            # make predictions
```

Results





Notebook #3 NLP Transfer Learning for Text Classification

Input: Amazon Cell Phone Reviews (binary labeled, 0 represents negative reviews and 1 represents positive reviews)
Output: Predictions (positive/negative labels) and saved fine-tuned model

Data Statistics

- Dataset: Amazon Cell Phone Reviews
- Size: 1,000 records
- Classes: positive and negative
- Splits: Training 60%, validation 20% and testing 20%

Hardware requirement: NVIDIA GPUs

Process

1. Install and import libraries
2. Download dataset from GitHub
3. Split dataset
4. Convert Pandas data frame to Hugging Face dataset
5. Load BERT, text tokenizer
6. Load pretrained BERT model
7. Set training arguments
8. Set evaluation metrics
9. Train model
10. Make predictions
11. Evaluate model performance
12. Save and load model

Learning curve & Metrics

- Learning Curve is visualized by plotting performance metrics.
- Metrics: Accuracy, precision, recall, and F1-score

Fine-tuning techniques

- TF-IDF Parameters: Adjusting options like n-gram range, stop word removal, and maximum features.
- Model Hyperparameters: Tuning classifier settings using cross-validation or grid search.

Important commands

```
%pip install ...                                          # install libraries
amz_review = pd.read_csv(...)                             # read CSV file
amz_review.info()                                         # get dataset info
train_test_split(...)                                     # split dataset
Dataset.from_pandas(...)                                  # create PyTorchx dataset
AutoTokenizer.from_pretrained("bert-base-cased")          # load tokenizer
def tokenize_dataset(data):                               # define tokenization function
AutoModelForSequenceClassification.from_pretrained(...)   # load model
training_args = TrainingArguments(...)                    # set training config
evaluate.list_evaluation_modules()                        # list evaluation modules
evaluate.load("accuracy")                                 # load accuracy module
trainer = Trainer(...)                                    # initialize trainer
trainer.train()                                           # train the model
trainer.predict(dataset_test)                             # make predictions
y_test_predict.predictions                                # get predictions
tf.nn.softmax(y_test_logits)                              # apply softmax function
trainer.evaluate(dataset_test)                            # evaluate the model
f1_score(actual, pred)                                    # get F1 score
recall_score(actual, ped)                                 # get recall scores
tokenizer.save_pretrained(path)                           # save tokenizer
trainer.save_model(path)                                  # save model
AutoTokenizer.from_pretrained(path)                       # load saved tokenizer
AutoModelForSequenceClassification.from_pretrained(path)  # load saved model
```

Results

[300/300 01:03, Epoch 2/2]

Epoch	Training Loss	Validation Loss	Accuracy
1	0.561700	0.404928	0.880000
2	0.299100	0.303143	0.910000

Downloading builder script: 100% 4.20k/4.20k [00:00<00:00, 329kB/s]

```
TrainOutput(global_step=300, training_loss=0.43040327707926435, metrics={'train_runtime': 200.4165,
'train_samples_per_second': 5.988, 'train_steps_per_second': 1.497, 'total_flos': 19733329152000.0,
'train_loss': 0.43040327707926435, 'epoch': 2.0})
```

[50/50 00:00]

```
{'eval_loss': 0.32168495655059814,
 'eval_accuracy': 0.905,
 'eval_runtime': 1.0251,
 'eval_samples_per_second': 195.099,
 'eval_steps_per_second': 48.775,
 'epoch': 2.0}
```

```
f1 score is 0.9015544041450777
recall score is 0.87
```

Notebook #4 Text Classification with PhayaThaiBERT

Input: Review ratings from Wongnai
Output: Star rating predictions and saved fine-tuned model

Data Statistics

- Dataset: Review ratings from Wongnai
- Size: 40,000 records
- Classes: Star rating 5 classes
- Splits: Training 90% and testing 10%

Hardware requirement: NVIDIA GPUs

Process

1. Install and import libraries
2. Load dataset
3. Load PhayaThaiBERT model
4. Split dataset
5. Create Hugging Face dataset
6. Create class for Pytorch Lightning module
7. Fine-tune model
8. Visualize loss graph
9. Evaluate model
10. Save and load fine-tuned model

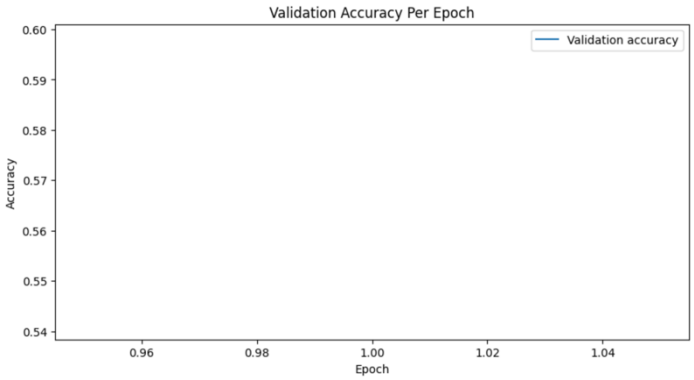
Learning curve & Metrics
Metrics: Accuracy, precision, recall, and F1-score

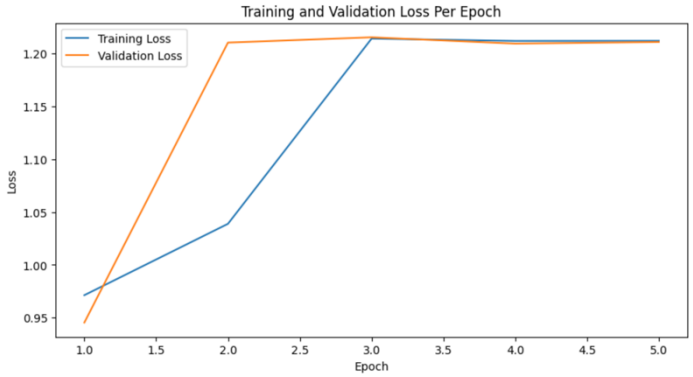
Fine-tuning techniques
Model Hyperparameters: Tuning classifier settings using cross-validation or grid search.

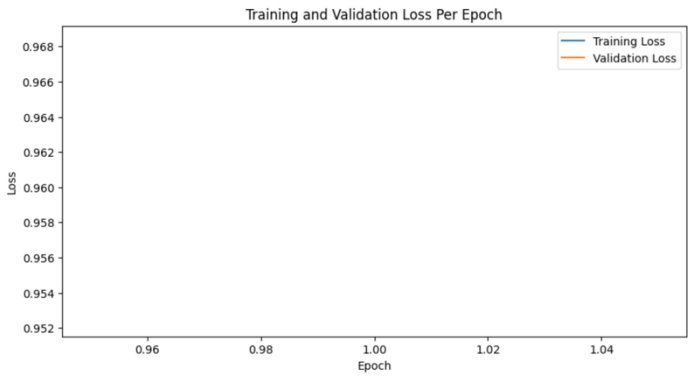
Important commands

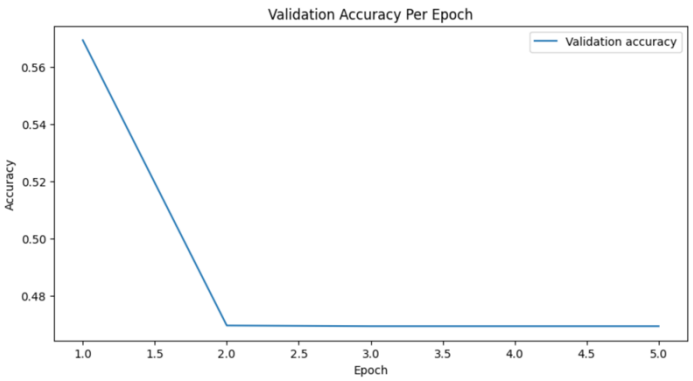
```
%pip install ...                                    # install libraries
load_dataset(...)                                   # load dataset
df = pd.DataFrame(...)                              # create DataFrame from dataset
AutoTokenizer.from_pretrained(...)                  # load tokenizer
def tokenize_dataset(dataset):                      # define tokenization function
train_test_split(...)                               # split dataset
Dataset.from_pandas(...)                            # create PyTorch dataset
train_dataset.map(..., batched=True)                # tokenize dataset
HuggingFaceDataset(...)                             # create Hugging Face dataset
DataLoader(...)                                     # create DataLoader
BertClassifier(...)                                 # define model
model = BertClassifier(...)                         # initialize model
trainer = pl.Trainer(...)                           # initialize trainer
trainer.fit()                                       # train model
model.eval()                                        # make predictions
f1_score(actual, pred)                              # get F1 score
recall_score(actual, ped)                           # get recall scores
tokenizer.save_pretrained(path)                     # save tokenizer
trainer.save_model(path)                            # save model
AutoTokenizer.from_pretrained(path)                 # load saved tokenizer
AutoModelForSequenceClassification.from_pretrained(path)    # load saved model
```

Results









```
F1-score: 0.5225581400897893

Accuracy: 0.574

Classification Report:
              precision    recall  f1-score   support

           1       0.50      0.20      0.29         5
           2       0.43      0.16      0.23        19
           3       0.65      0.33      0.44       141
           4       0.56      0.91      0.70       243
           5       0.55      0.17      0.26        92

    accuracy                           0.57       500
   macro avg       0.54      0.35      0.38       500
weighted avg       0.58      0.57      0.52       500
```

```
Text: เจ้าถิ่นอุดรพามา บอกต้องกินนะ จัดไปอย่างเต็ม
หมูยอหนัง ตำไทย ปอเปี๊ยะทอด ลาบ แกงเห็ด
คือว่าหิวมากหรืออร่อยมาก หรือทั้งสองอย่าง
ปรากฏอาหารทั้งหมดหายวับในพริบตา จนเจ้าบ้านตกใจมาก
ก็เลยบอกเจ้าถิ่นคราวหน้ามา อย่าลืมพามาร้านนี้อีกนะ
Predicted Rating: 3
```

```
Text: เป็นร้านกาแฟอยู่ในร้านอาหารเสน่ห์ดอยหลวง
ร้านสวย ตกแต่งด้วยไม้ดอกไม้ประดับ และเฟอร์นิเจอร์ไม้เก่า มีที่นั่งริมห้วยฟังเสียงน้ำเพลินๆ
สั่งคาปูชิโน่ร้อน และเค้กช็อคโกแลต
กาแฟไม่อร่อย คั่วเข้มจนขม ส่วนเค้กพอใช้ได้
Predicted Rating: 2
```

```
Text: ครัววงเดือน

หิวดึกๆ ตระเวนหาร้านทาน มาเจอร้านริมถนนพุทธมณฑลสาย 1 หน้าปากซอยพุทธมณฑลสาย 1 ซอย 10ครับ ร้า


1. ยำมะเขือเปราะ 100 บาท จัดจ้านมากๆ

2. กบทอดกระเทียม 100 บาท กบตัวเล็กไปหน่อยครับ ทอดมากรอบนอกนุ่มในดีครับ อร่อยๆๆ

3. ลาบปลาช่อน 100 บาท ทำมาเป็นชิ้นๆทอดมาแล้ว ทานง่าย รสชาติลาบ หอมข้าวคั่ว ลาบมาอร่อยเลยครับ

4. เนื้อย่าง 100 บาท เมนูนี้ตกม้าตาย เนื้อไม่ได้หมักอะไรเลย มาจืดๆ เพราะปกติผมทานแบบไม่จิ้มน้ำจิ้มเลย

5. เฉาก๊วยชากังราว 25 บาท

*** ร้านนี้ไม่มีข้าวเหนียวขายนะครับ เพราะเค้าเพิ่งเอาไลน์ส้มตำ จิ้มจุ่มมาขายเมื่อ 2-3 เดือนมานี้เอง ***

เวลาเปิดร้าน : 10:00 - 23:00
โทร : 092 345 4318

Predicted Rating: 3
```