

DSDE-Homework-5_Time_series_forecasting_DataInGD

ใช้ LSTM และ GRU ในการ predict ราคาหุ้น
โดย input เป็นราคา GOOG ใน csv file

โหลด ข้อมูลจาก CSV

```
# Load data
df = pd.read_csv('GOOG.csv', index_col="Date")
df = df.drop(['Adj Close'], axis=1)
```

ดู row

```
df.loc['2019-02-26']
```

```
>>
Open      1.105750e+03
High      1.119510e+03
Low       1.099920e+03
Close     1.115130e+03
Volume    1.471300e+06
Name: 2019-02-26, dtype: float64
```

ใช้ plotly ในการสร้างกราฟแสดงข้อมูล

```
import plotly.express as px
import plotly.graph_objects as go
import plotly.io as pio
pio.templates.default = "plotly_white"

plot_template = dict(
    layout=go.Layout({
        "font_size": 18,
        "xaxis_title_font_size": 24,
        "yaxis_title_font_size": 24})
)

fig = px.line(df['Open'], labels=dict(
    created_at="Date", value="Open", variable="Sensor"
))
fig.update_layout(
    template=plot_template, legend=dict(orientation='h', y=1.02, title_text="")
)
fig.show()
```

สร้าง target value

```
target_col = "Open"
features = list(df.columns.difference([target_col]))

forecast_lead = 1
target = f"{target_col}_lead{forecast_lead}"

df[target] = df[target_col].shift(-forecast_lead)
df = df.iloc[:-forecast_lead]
```

แบ่ง test val test

```
test_start = "2019-01-01"
val_start = "2018-01-01"

df_train = df.loc[:val_start].copy()
```

```
df_val = df.loc[val_start:test_start].copy()
df_test = df.loc[test_start:].copy()

print("Test set fraction:", len(df_test) / len(df))
```

```
>> Test set fraction: 0.050157563025210086
```

ทำ standardize

```
target_mean = df_train[target].mean()
target_stddev = df_train[target].std()

for c in df_train.columns:
    mean = df_train[c].mean()
    stddev = df_train[c].std()

    df_train[c] = (df_train[c] - mean) / stddev
    df_val[c] = (df_val[c] - mean) / stddev
    df_test[c] = (df_test[c] - mean) / stddev
```

สร้าง DataLoader class สำหรับ PyTorch

```
import torch
from torch.utils.data import Dataset

class SequenceDataset(Dataset):
    def __init__(self, dataframe, target, features, sequence_length=5):
        self.features = features
        self.target = target
        self.sequence_length = sequence_length
        self.y = torch.tensor(dataframe[self.target].values).float()
        self.X = torch.tensor(dataframe[self.features].values).float()

    def __len__(self):
        return self.X.shape[0]

    def __getitem__(self, i):
        if i >= self.sequence_length - 1:
            i_start = i - self.sequence_length + 1
            x = self.X[i_start:(i + 1), :]
        else:
            padding = self.X[0].repeat(self.sequence_length - i - 1, 1)
            x = self.X[0:(i + 1), :]
            x = torch.cat((padding, x), 0)

        return x, self.y[i]
```

นำ data loader ที่สร้างมาใช้กับ train val test

```
torch.manual_seed(101)

batch_size = 32
sequence_length = 60

train_dataset = SequenceDataset(
    df_train,
    target=target,
    features=features,
    sequence_length=sequence_length
)
val_dataset = SequenceDataset(
    df_val,
    target=target,
    features=features,
    sequence_length=sequence_length
)
test_dataset = SequenceDataset(
    df_test,
```

```

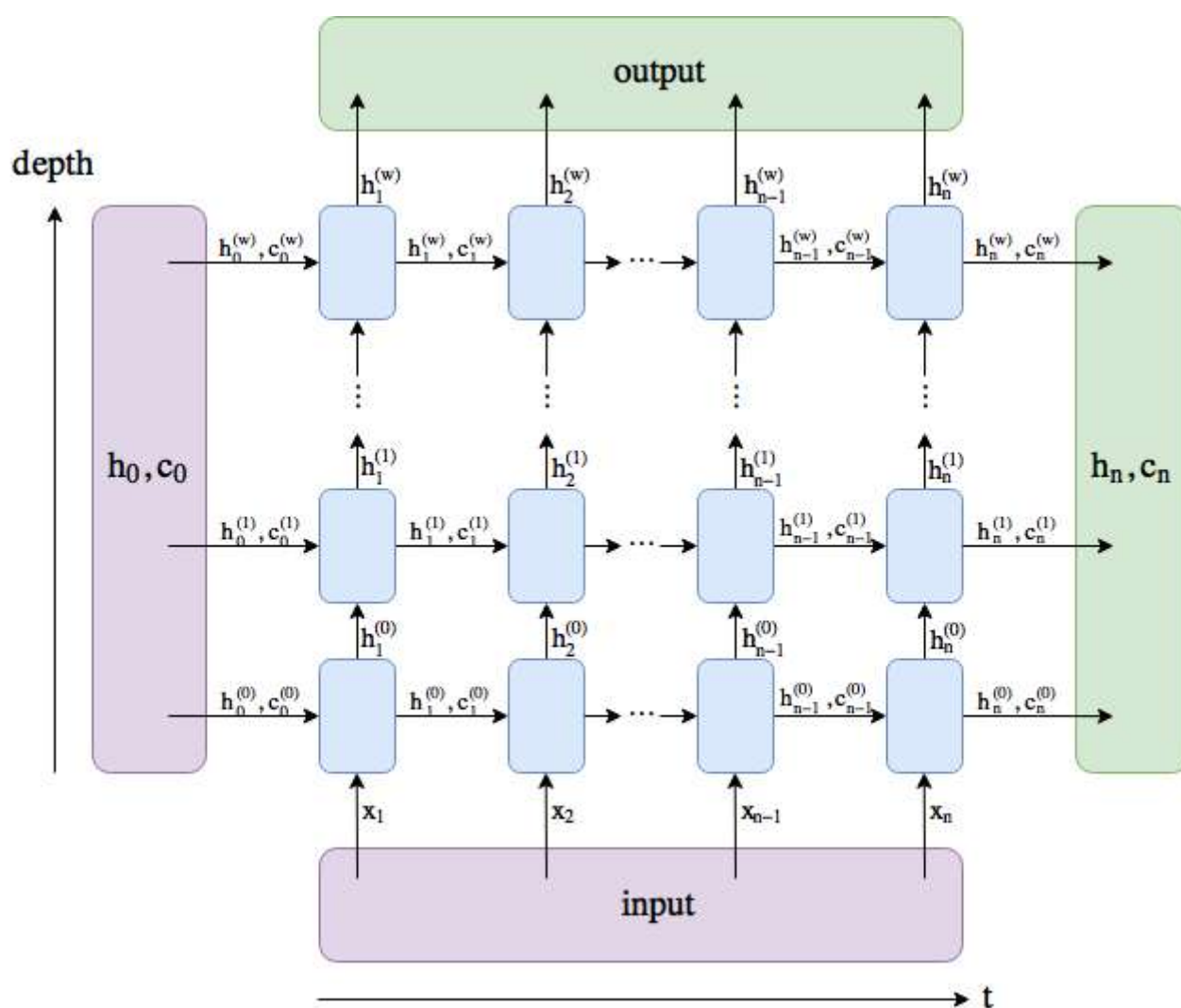
target=target,
features=features,
sequence_length=sequence_length
)

train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)

X, y = next(iter(train_loader))

print("Features shape:", X.shape)
print("Target shape:", y.shape)

```



เลือก device เป็น GPU

```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

สร้าง ShallowRegressionLSTM

```

from torch import nn

class ShallowRegressionLSTM(nn.Module):
    def __init__(self, num_features, hidden_units):
        super().__init__()
        self.num_features = num_features # this is the number of features
        self.hidden_units = hidden_units
        self.num_layers = 4

        self.lstm = nn.LSTM(
            input_size=num_features,
            hidden_size=hidden_units,
            batch_first=True,
            num_layers=self.num_layers
        )

        self.linear = nn.Linear(in_features=self.hidden_units, out_features=1)

    def forward(self, x):
        batch_size = x.shape[0]

        # initialize the hidden and cell state of the LSTM layer
        h0 = torch.zeros(self.num_layers, batch_size, self.hidden_units).to(device).requires_grad_()

```

```

c0 = torch.zeros(self.num_layers, batch_size, self.hidden_units).to(device).requires_grad_()

_, (hn, _) = self.lstm(x, (h0, c0))
out = self.linear(hn[-1]).flatten() # get the output of the last hidden layer
return out

```

กำหนด parameter

```

learning_rate = 5e-4

num_hidden_units = 60

model = ShallowRegressionLSTM(num_features=len(features), hidden_units=num_hidden_units)

model.to(device)

loss_function = nn.MSELoss()

optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

```

print summary

```

from torchinfo import summary
summary(model, input_size=(32, 60, 4))

```

```

=====
Layer (type:depth-idx)                   Output Shape          Param #
=====
ShallowRegressionLSTM                   [32]                  --
├─LSTM: 1-1                             [32, 60, 60]         103,680
├─Linear: 1-2                           [32, 1]              61
=====
Total params: 103,741
Trainable params: 103,741
Non-trainable params: 0
Total mult-adds (M): 199.07
=====
Input size (MB): 0.03
Forward/backward pass size (MB): 0.92
Params size (MB): 0.41
Estimated Total Size (MB): 1.37
=====

```

เริ่ม train model

```

from tqdm.notebook import tqdm
def train_model(data_loader, model, loss_function, optimizer):
    num_batches = len(data_loader)
    total_loss = 0
    model.train()

    for X, y in data_loader:
        X = X.to(device)
        y = y.to(device)
        output = model(X)
        loss = loss_function(output, y)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        total_loss += loss.item()

    avg_loss = total_loss / num_batches
    print(f"Train loss: {avg_loss}")

```

```
def test_model(data_loader, model, loss_function, best_val_loss):

    num_batches = len(data_loader)
    total_loss = 0

    model.eval()
    with torch.no_grad():
        for X, y in data_loader:
            X = X.to(device)
            y = y.to(device)
            output = model(X)
            total_loss += loss_function(output, y).item()

    avg_loss = total_loss / num_batches
    print(f"Test loss: {avg_loss}")
    if avg_loss < best_val_loss:
        best_val_loss = avg_loss
        torch.save(model.state_dict(), 'model.pth')
        print('Save new best model')
    return best_val_loss
```

Evaluation

```
def predict(data_loader, model):

    """Just like `test_loop` function but keep track of the outputs instead of the loss
    function.

    """

    output = torch.tensor([])

    model.eval()

    with torch.no_grad():

        for X, _ in data_loader:

            X = X.to(device)

            y_star = model(X)

            output = torch.cat((output, y_star.detach().cpu()), 0)

    return output
```

โหลด model ที่ train เสร็จแล้ว

```
PATH = './model.pth'
model.load_state_dict(torch.load(PATH))
```

predict ผลลัพธ์

```
train_eval_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=False)

ystar_col = "Model forecast"
df_train[ystar_col] = predict(train_eval_loader, model).numpy()
df_val[ystar_col] = predict(val_loader, model).numpy()
df_test[ystar_col] = predict(test_loader, model).numpy()

df_out = pd.concat((df_train, df_val, df_test))[[target, ystar_col]]

for c in df_out.columns:
    df_out[c] = df_out[c] * target_stddev + target_mean
```

```
print(df_out)
```

Date	Open_lead1	Model forecast
2004-08-19	50.316402	52.646271
2004-08-20	55.168217	52.799896
2004-08-23	55.412300	54.844269
2004-08-24	52.284027	53.112549
2004-08-25	52.279045	52.587830
...
2019-09-27	1220.969971	1203.750732
2019-09-30	1219.000000	1200.191284
2019-10-01	1196.979980	1195.909912
2019-10-02	1180.000000	1181.464844
2019-10-03	1191.890015	1176.163696

print evaluation metrics

```
import numpy as np

import math

from sklearn.metrics import mean_squared_error

def MAPE(Y_actual,Y_Predicted):

    mape = np.mean(np.abs((Y_actual - Y_Predicted)/Y_actual))*100

    return mape

print( 'MPAE =', MAPE(df_test['Open_lead1'], df_test['Model forecast']) )

print( 'RMSE =', math.sqrt(mean_squared_error(df_test['Open_lead1'], df_test['Model forecast']))) )
```

```
>> MPAE = 1.9260324553098662 RMSE = 0.08593983988067332
```

plot กราฟ forecase

```
fig = px.line(df_out, labels={'value': "Open", 'created_at': 'Date'})

fig.add_vline(x=val_start, line_width=4, line_dash="dash")

fig.add_vline(x=test_start, line_width=4, line_dash="dash")

# fig.add_annotation(xref="paper", x=0.75, yref="paper", y=0.8, text="Test set start", showarrow=False)

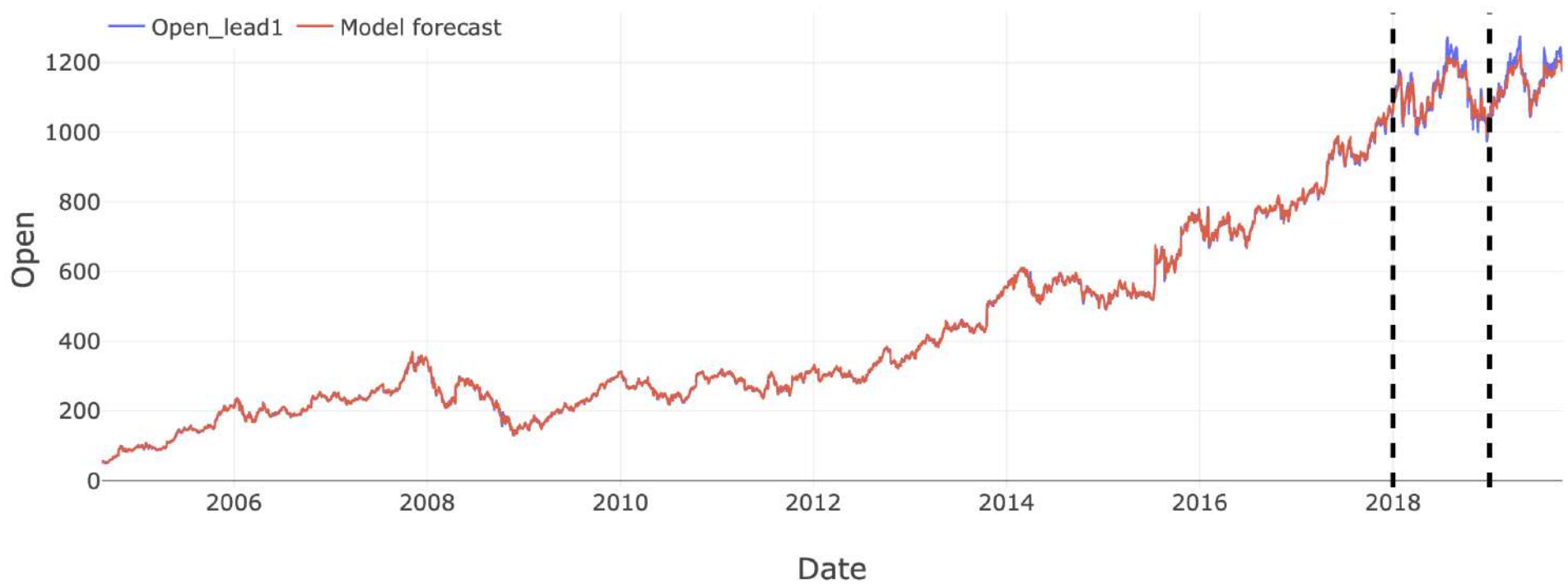
fig.update_layout(

    template=plot_template, legend=dict(orientation='h', y=1.02, title_text="")

)
```



```
fig.show()
```



train GRU

```
from torch import nn

class ShallowRegressionGRU(nn.Module):
    def __init__(self, num_features, hidden_units):
        super().__init__()
        self.num_features = num_features # this is the number of features
        self.hidden_units = hidden_units
        self.num_layers = 4

        self.gru = nn.GRU(
            input_size=num_features,
            hidden_size=hidden_units,
            batch_first=True,
            num_layers=self.num_layers
        )

        self.linear = nn.Linear(in_features=self.hidden_units, out_features=1)

    def forward(self, x):
        batch_size = x.shape[0]

        # initialize the hidden and cell state of the LSTM layer
        h0 = torch.zeros(self.num_layers, batch_size, self.hidden_units).to(device).requires_grad_()

        _, hn = self.gru(x, h0)
        out = self.linear(hn[-1]).flatten() # get the output of the last hidden layer
        return out
```

ประกาศ parameters

```
learning_rate = 5e-4

num_hidden_units = 60

model = ShallowRegressionGRU(num_features=len(features), hidden_units=num_hidden_units)

model.to(device)

loss_function = nn.MSELoss()

optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
```

summary model

```
from torchinfo import summary

summary(model, input_size=(32, 60, 4))
```

Layer (type:depth-idx)	Output Shape	Param #
ShallowRegressionGRU	[32]	--
└GRU: 1-1	[32, 60, 60]	77,760
└Linear: 1-2	[32, 1]	61
Total params: 77,821		
Trainable params: 77,821		
Non-trainable params: 0		
Total mult-adds (M): 149.30		
Input size (MB): 0.03		
Forward/backward pass size (MB): 0.92		
Params size (MB): 0.31		
Estimated Total Size (MB): 1.26		

train GRU

```
def train_model(data_loader, model, loss_function, optimizer):
    num_batches = len(data_loader)
    total_loss = 0
    model.train()

    for X, y in data_loader:
        X = X.to(device)
        y = y.to(device)
        output = model(X)
        loss = loss_function(output, y)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        total_loss += loss.item()

    avg_loss = total_loss / num_batches
    print(f"Train loss: {avg_loss}")

def test_model(data_loader, model, loss_function, best_val_loss):

    num_batches = len(data_loader)
    total_loss = 0

    model.eval()
    with torch.no_grad():
        for X, y in data_loader:
            X = X.to(device)
            y = y.to(device)
            output = model(X)
            total_loss += loss_function(output, y).item()

    avg_loss = total_loss / num_batches
    print(f"Test loss: {avg_loss}")
    if avg_loss < best_val_loss:
        best_val_loss = avg_loss
        torch.save(model.state_dict(), 'model_gru.pth')
        print('Save new best model')
    return best_val_loss
```

ประกาศ function predict


```
def predict(data_loader, model):

    """Just like `test_loop` function but keep track of the outputs instead of the loss
    function.

    """

    output = torch.tensor([])

    model.eval()

    with torch.no_grad():

        for X, _ in data_loader:

            X = X.to(device)

            y_star = model(X)

            output = torch.cat((output, y_star.detach().cpu()), 0)

    return output
```

โหลด GRU ที่ train แล้ว

```
PATH = './model_gru.pth'

model.load_state_dict(torch.load(PATH))
```

predict

```
train_eval_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=False)

ystar_col = "Model forecast"
df_train[ystar_col] = predict(train_eval_loader, model).numpy()
df_val[ystar_col] = predict(val_loader, model).numpy()
df_test[ystar_col] = predict(test_loader, model).numpy()

df_out = pd.concat((df_train, df_val, df_test))[[target, ystar_col]]

for c in df_out.columns:
    df_out[c] = df_out[c] * target_stdev + target_mean

print(df_out)
```

print evaluation metrics

```
import numpy as np

import math

from sklearn.metrics import mean_squared_error

def MAPE(Y_actual, Y_Predicted):

    mape = np.mean(np.abs((Y_actual - Y_Predicted)/Y_actual))*100

    return mape

print( 'MPAE =', MAPE(df_test['Open_lead1'], df_test['Model forecast']) )
```

```
print( 'RMSE =', math.sqrt(mean_squared_error(df_val['Open_lead1'], df_val['Model forecast'])) )
```

```
>> MPAE = 1.9992090562917835 RMSE = 0.08901432835864326
```

plot graph

```
fig = px.line(df_out, labels={'value': "Open", 'created_at': 'Date'})

fig.add_vline(x=val_start, line_width=4, line_dash="dash")

fig.add_vline(x=test_start, line_width=4, line_dash="dash")

# fig.add_annotation(xref="paper", x=0.75, yref="paper", y=0.8, text="Test set start", showarrow=False)

fig.update_layout(

template=plot_template, legend=dict(orientation='h', y=1.02, title_text="")

)

fig.show()
```



DSDE-Homework-1_Huggingface_image_classification

fine-tune ViT สำหรับทำ classification input เป็นใบของต้นถั่ว beans data set ด้วย Hugging Face library

โหลด datasets

```
from datasets import load_dataset
ds = load_dataset('beans',)
ds
```

```
image = ds['train'][400]['image']
image
```



function สำหรับดู data ใน dataset

```
from transformers.utils.dummy_vision_objects import ImageGPTFeatureExtractor
import random
from PIL import ImageDraw, ImageFont, Image

def show_examples(ds, seed: int = 1234, examples_per_class: int = 3, size=(350, 350)):

    w, h = size
    labels = ds['train'].features['labels'].names
    grid = Image.new('RGB', size=(examples_per_class * w, len(labels) * h))
    draw = ImageDraw.Draw(grid)
    font = ImageFont.truetype("/usr/share/fonts/truetype/liberation/LiberationMono-Bold.ttf", 24)

    for label_id, label in enumerate(labels):

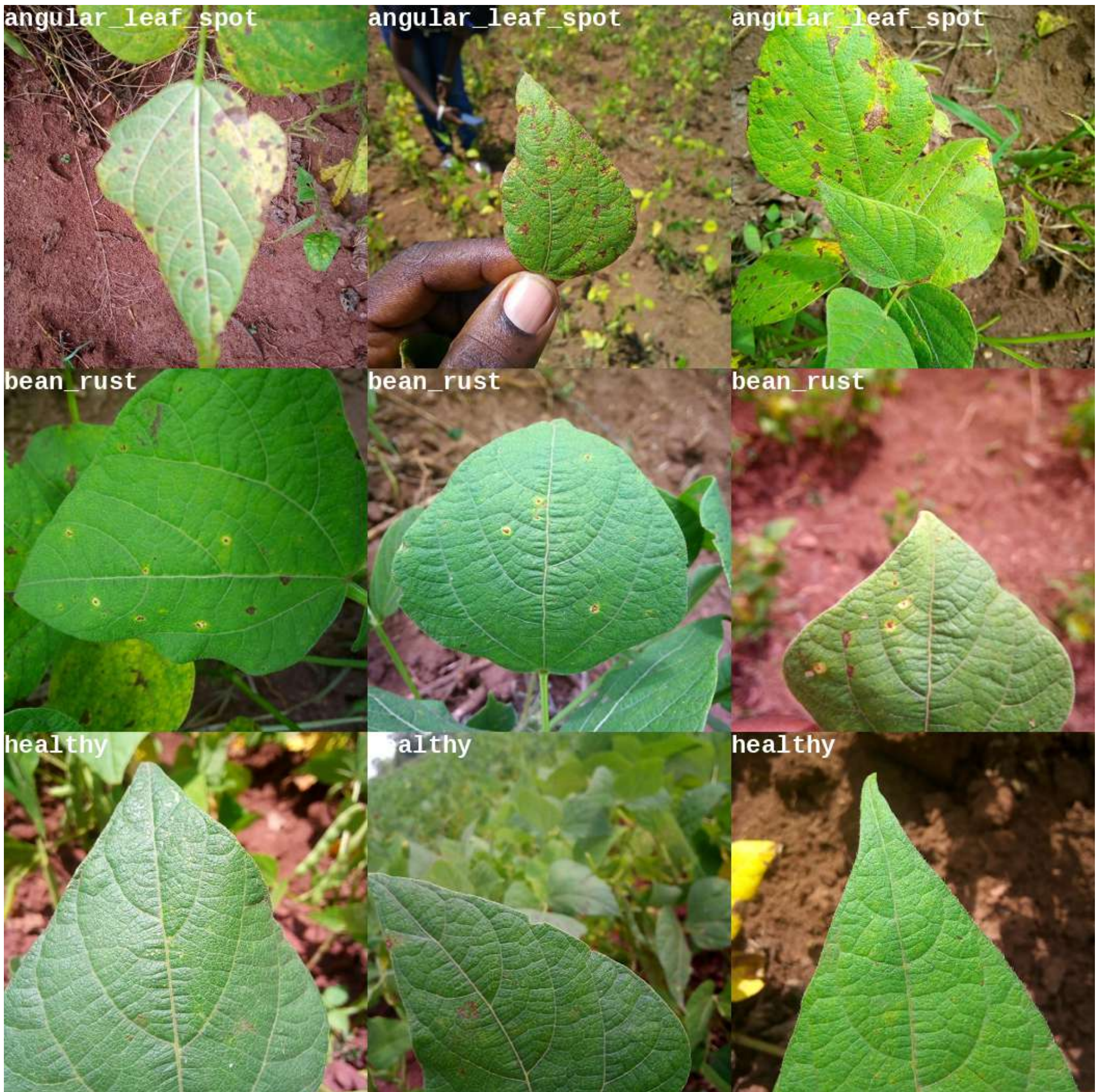
        # Filter the dataset by a single label, shuffle it, and grab a few samples
        ds_slice = ds['train'].filter(lambda ex: ex['labels'] ==
label_id).shuffle(seed).select(range(examples_per_class))

        # Plot this label's examples along a row
        for i, example in enumerate(ds_slice):
            image = example['image']
            idx = examples_per_class * label_id + i
            box = (idx % examples_per_class * w, idx // examples_per_class * h)
            grid.paste(image.resize(size), box=box)
            draw.text(box, label, (255, 255, 255), font=font)

    return grid
```



```
show_examples(ds, seed=random.randint(0, 1337), examples_per_class=3)
```



โหลด pretrained model

```
from transformers import ViTImageProcessor
model_name_or_path = 'google/vit-base-patch16-224'
feature_extractor = ViTImageProcessor.from_pretrained(model_name_or_path)
feature_extractor
```

```
ViTImageProcessor {
  "do_normalize": true,
  "do_rescale": true,
  "do_resize": true,
  "image_mean": [
    0.5,
    0.5,
    0.5
  ],
  "image_processor_type": "ViTImageProcessor",
```



```

"image_std": [
    0.5,
    0.5,
    0.5
],
"resample": 2,
"rescale_factor": 0.00392156862745098,
"size": {
    "height": 224,
    "width": 224
}
}

```

function สำหรับ transform

```

def transform(example_batch):

# Take a list of PIL images and turn them to pixel values

inputs = feature_extractor([x for x in example_batch['image']], return_tensors='pt')

# Don't forget to include the labels!

inputs['labels'] = example_batch['labels']

return inputs

prepared_ds = ds.with_transform(transform)

```

```

import torch

def collate_fn(batch):

return { 'pixel_values': torch.stack([x['pixel_values'] for x in batch]),

'labels': torch.tensor([x['labels'] for x in batch])}

```

load accuracy metric สำหรับ evaluation

```

import numpy as np

from datasets import load_metric

metric = load_metric("accuracy")

def compute_metrics(p):

return metric.compute(predictions=np.argmax(p.predictions, axis=1), references=p.label_ids)

```

โหลด pretrain สำหรับ classification

```

from transformers import ViTForImageClassification

labels = ds['train'].features['labels'].names

model = ViTForImageClassification.from_pretrained('google/vit-base-patch16-224',

num_labels=len(labels),

```

```
ignore_mismatched_sizes=True)
```

ประกาศ Argument การ train

```
from transformers import TrainingArguments

training_args = TrainingArguments(
    output_dir="vit-base-beans-demo-v5",
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    evaluation_strategy="epoch",
    save_strategy="epoch",
    logging_strategy='epoch',
    logging_steps=1,
    num_train_epochs=10,
    optim='adamw_torch',
    learning_rate=1e-5,
    remove_unused_columns=False,
    report_to='tensorboard',
    load_best_model_at_end=True,
    metric_for_best_model="loss",
    seed=0
)
```

ประกาศ trainer

```
from transformers import Trainer

trainer = Trainer(
    model=model,
    args=training_args,
    data_collator=collate_fn,
    compute_metrics=compute_metrics,
    train_dataset=prepared_ds["train"],
    eval_dataset=prepared_ds["validation"],
    tokenizer=feature_extractor,
)
```

ทดลอง evaluate ก่อน fine tune

```
from sklearn.metrics import classification_report

test_predictions = trainer.predict(prepared_ds['test'])

# For each prediction, create the label with argmax
predictions = np.argmax(test_predictions[0], axis=1)

# Retrieve reference labels from test set
test_labels = np.array(prepared_ds["test"][:]["labels"])

print(classification_report(test_labels, predictions, target_names=['angular_leaf_spot', 'bean_rust', 'healthy'],
digits=4))
```


	precision	recall	f1-score	support
angular_leaf_spot	0.0000	0.0000	0.0000	43
bean_rust	0.3097	0.8140	0.4487	43
healthy	0.3333	0.0714	0.1176	42
accuracy			0.2969	128
macro avg	0.2144	0.2951	0.1888	128
weighted avg	0.2134	0.2969	0.1893	128

save result

```
train_results = trainer.train()

trainer.save_model()

trainer.log_metrics("train", train_results.metrics)

trainer.save_metrics("train", train_results.metrics)

trainer.save_state()
```

```
metrics = trainer.evaluate(prepared_ds['validation'])
trainer.log_metrics("eval", metrics)
trainer.save_metrics("eval", metrics)
```

```
***** eval metrics *****
epoch = 10.0
eval_accuracy = 0.9925
eval_loss = 0.0242
eval_runtime = 0:00:03.01
eval_samples_per_second = 44.081
eval_steps_per_second = 2.983
```

prediction

```
from sklearn.metrics import classification_report

test_predictions = trainer.predict(prepared_ds['test'])
# For each prediction, create the label with argmax
predictions = np.argmax(test_predictions[0], axis=1)
# Retrieve reference labels from test set
test_labels = np.array(prepared_ds["test"][:]["labels"])
print(classification_report(test_labels, predictions, target_names=['angular_leaf_spot', 'bean_rust', 'healthy'],
digits=4))
```

🔗 tensorboard

```
%load_ext tensorboard
%tensorboard --logdir '{"vit-base-beans-demo-v5"}'/runs
```

Filter runs (regex)

Run

Sep02_08-13-09_3f014f08a587

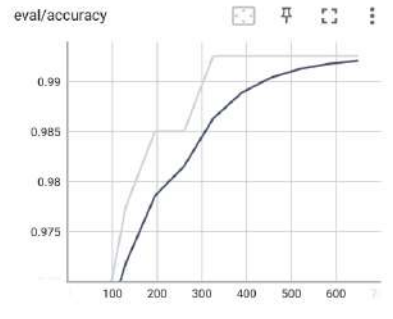
Filter tags (regex)

Pinned

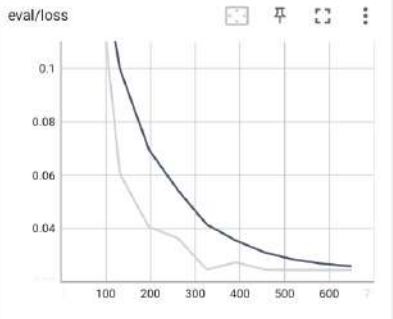
Pin cards for a quick view and comparison

eval 5 cards

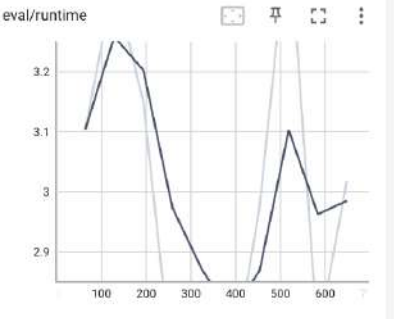
eval/accuracy



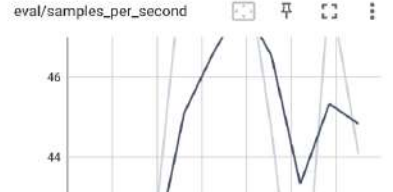
eval/loss



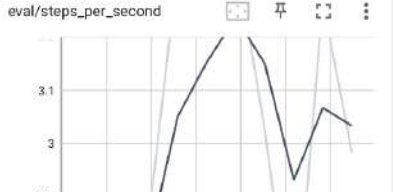
eval/runtime



eval/samples_per_second



eval/steps_per_second



Settings

GENERAL

Horizontal Axis

Step

☐ Enable step selection and data table (Scalars only)

☐ Enable Range Selection

☐ Link by step 65

Card Width

SCALARS

Smoothing

0.6

Tooltip sorting method

Alphabetical

☒ Ignore outliers in chart scaling

☐ Partition non-monotonic X axis

HISTOGRAMS

Mode

Offset