TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM
TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG
KHOA CÔNG NGHỆ THÔNG TIN

BÀI TẬP LỚN MÔN HỌC SÂU

IMAGE CAPTIONING

Người hướng dẫn: **TS LÊ ANH CƯỜNG**

Người thực hiện: **LIỄU THANH LÂM – C1900015**

HUỲNH NGUYỄN HUY ANH – 51900723

NGUYỄN QUỐC BẢO - 52100873

Lớp **: 190C0101 - 19050302 - 21050301**

Khoá **: 23 - 25**

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2023

TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM
TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG
KHOA CÔNG NGHỆ THÔNG TIN

BÀI TẬP LỚN MÔN HỌC SÂU

IMAGE CAPTIONING

Người hướng dẫn: **TS LÊ ANH CƯỜNG**
Người thực hiện: **LIỄU THANH LÂM**
HUỲNH NGUYỄN HUY ANH
NGUYỄN QUỐC BẢO
Lớp : **190C0101 - 19050302 - 21050301**
Khoá : **23 - 25**

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2023

LỜI CẢM ƠN

Cảm ơn thầy Lê Anh Cường đã hướng dẫn tận tình các kiến thức về các kiến trúc của CNN của bên Học Sâu để tụi em hoàn thành bài báo cáo này.

ĐỒ ÁN ĐƯỢC HOÀN THÀNH TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG

Tôi xin cam đoan đây là sản phẩm đồ án của riêng tôi và được sự hướng dẫn của TS Lê Anh Cường;. Các nội dung nghiên cứu, kết quả trong đề tài này là trung thực và chưa công bố dưới bất kỳ hình thức nào trước đây. Những số liệu trong các bảng biểu phục vụ cho việc phân tích, nhận xét, đánh giá được chính tác giả thu thập từ các nguồn khác nhau có ghi rõ trong phần tài liệu tham khảo.

Ngoài ra, trong đồ án còn sử dụng một số nhận xét, đánh giá cũng như số liệu của các tác giả khác, cơ quan tổ chức khác đều có trích dẫn và chú thích nguồn gốc.

Nếu phát hiện có bất kỳ sự gian lận nào tôi xin hoàn toàn chịu trách nhiệm về nội dung đồ án của mình. Trường đại học Tôn Đức Thắng không liên quan đến những vi phạm tác quyền, bản quyền do tôi gây ra trong quá trình thực hiện (nếu có).

TP. Hồ Chí Minh, ngày 11 tháng 3 năm 2023

Tác giả

(ký tên và ghi rõ họ tên)

Liễu Thanh Lâm

Huỳnh Nguyễn Huy Anh

Nguyễn Quốc Bảo

PHẦN XÁC NHẬN VÀ ĐÁNH GIÁ CỦA GIẢNG VIÊN

Phần xác nhận của GV hướng dẫn

_____

_____

_____

_____

_____

_____

_____

Tp. Hồ Chí Minh, ngày tháng năm
(kí và ghi họ tên)

Phần đánh giá của GV chấm bài

_____

_____

_____

_____

_____

_____

_____

Tp. Hồ Chí Minh, ngày tháng năm
(kí và ghi họ tên)

TÓM TẮT

Trình bày tóm tắt vấn đề nghiên cứu, các hướng tiếp cận, cách giải quyết vấn đề và một số kết quả đạt được, những phát hiện cơ bản trong vòng 1 -2 trang.

MỤC LỤC

LỜI CẢM ƠN i
PHẦN XÁC NHẬN VÀ ĐÁNH GIÁ CỦA GIẢNG VIÊN iii
TÓM TẮT iv
MỤC LỤC 1
DANH MỤC CÁC BẢNG BIỂU, HÌNH VẼ, ĐỒ THỊ 3
I. ..... **Error! Bookmark not defined.**
1. Cơ chế Injection Architecture 4
2. Cơ chế Merging Architecture 5
3. Code implement function example 5
4. Implement 9
II.EfficientNet 10
III. Swin Tranfromer 12
IV.Database 14

DANH MỤC KÍ HIỆU VÀ CHỮ VIẾT TẮT

CÁC KÝ HIỆU

f *Tần số của dòng điện và điện áp (Hz)*

p *Mật độ điện tích khối (C/m3)*

CÁC CHỮ VIẾT TẮT

CSTD	Công suất tác dụng
MF	Máy phát điện
BER	Tỷ lệ bít lỗi

DANH MỤC CÁC BẢNG BIỂU, HÌNH VẼ, ĐỒ THỊ

DANH MỤC HÌNH

Hình 2.1: Kiến trúc FTP ..... **Error! Bookmark not defined.**

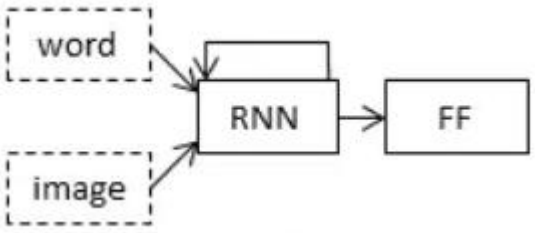
DANH MỤC BẢNG

Bảng 3.1 Ví dụ cho chèn bảng ..... **Error! Bookmark not defined.**

I. Image captioning

có 2 cách cơ bản để input cho image captioning: Injection Architecture và Merging Architecture

1. Cơ chế Injection Architecture



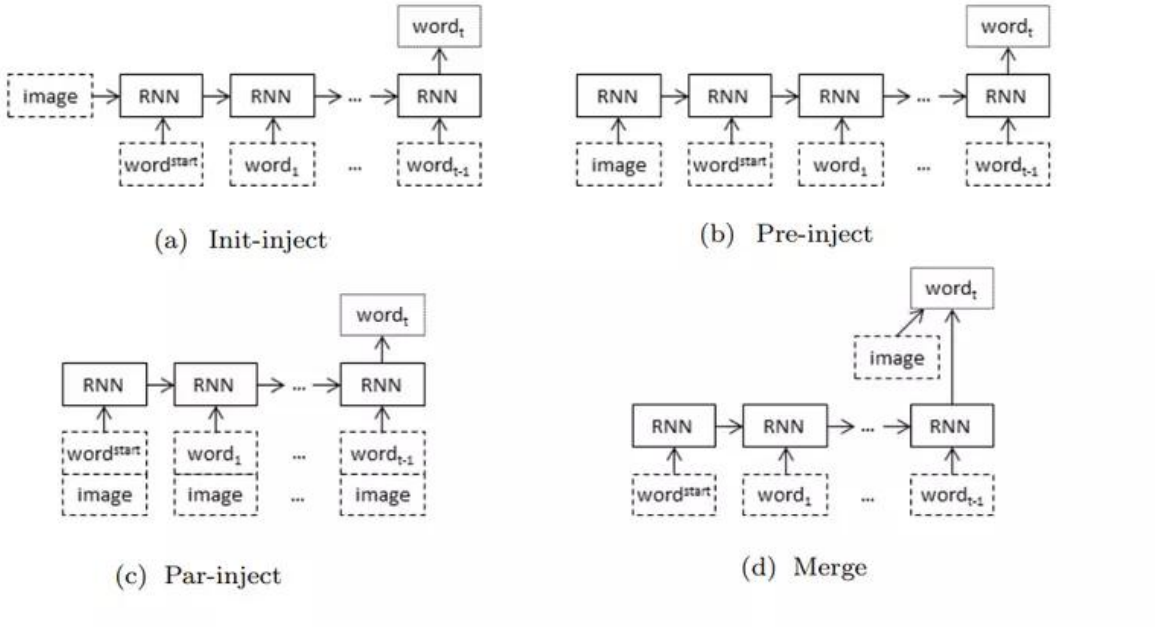
Injection Architecture: input sẽ là tổng hợp dữ liệu của cả ảnh và caption, vì vậy tại mỗi step của RNN đều sẽ bị ảnh hưởng bởi dữ liệu ảnh.

Gồm có 3 loại chính:

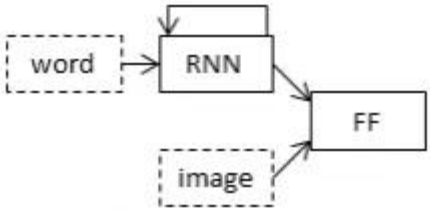
init-inject: thông thường, đầu vào của hidden state đầu tiên sẽ được khởi tạo là ma trận 0. Bây giờ ta thay ma trận đó bằng image.

pre-inject: ảnh được xem như 1 từ đầu vào, có ý nghĩa tương tự như các từ khác trong câu token.

par-inject: mỗi xt đều là sự tổng hợp của cả ảnh và từ



2. Cơ chế Merging Architecture

- Merging Architecture: xử lý riêng rồi tổng hợp thông tin lại với nhau.
- Mô hình Merging Architecture hợp nhất kết hợp cả dạng mã hóa của đầu vào hình ảnh với dạng mã hóa của mô tả văn bản.
- 

3. Code implement function example

Ta import các thư viện cần sử dụng:

```
import torch
import torch.optim as optim
import torch.nn.functional as F
import torch.nn as nn
```

Class decoder sử dụng init-inject

```
class Decoder(nn.Module):
    def __init__(self, hidden_size, output_size, embed_size, num_layers):
        super(Decoder, self).__init__()

        self.hidden_size = hidden_size
        self.embedding = nn.Embedding(output_size, embed_size)
        self.gru = nn.GRU(embed_size, hidden_size, num_layers=num_layers)
        self.out = nn.Linear(hidden_size, output_size)
        self.softmax = nn.LogSoftmax(dim=1)
# Init inject
    def forward(self, input, hidden):

        output = self.embedding(input)
        output = F.relu(output)
        output, hidden = self.gru(output, hidden)
        output = self.softmax(self.out(output[0]))

        return output, hidden


def get_decoder(hidden_size=2048, output_size=10000, embed_size=128, num_layers=1):
    return Decoder(hidden_size=hidden_size, output_size=output_size, embed_size=embed_size, num_layers=num_layers)
```

Class decoder sử dụng pre_inject

```python
class Decoder(nn.Module):
    def __init__(self, feature_size, output_size, embed_size, num_layers):
        super(Decoder, self).__init__()
        self.hidden_size = embed_size

        self.embedding = nn.Embedding(output_size, embed_size)
        self.gru = nn.GRU(embed_size, embed_size, num_layers=num_layers)

        self.map = nn.Linear(feature_size, embed_size)

        self.out = nn.Linear(embed_size, output_size)

        self.softmax = nn.LogSoftmax(dim=1)
    #Pre inject
    def forward(self, input, hidden, feature, sequence_length):

        if sequence_length == 0:
            feature = self.map(feature)
            feature = F.relu(feature)
            output, hidden = self.gru(feature, hidden)

        else:
            output = self.embedding(input)
            output = F.relu(output)
            output, hidden = self.gru(output, hidden)
        output = self.softmax(self.out(output[0]))

        return output, hidden


def get_decoder(feature_size=2048, output_size=10000, embed_size=128, num_layers=1):
    return Decoder(feature_size=feature_size, output_size=output_size, embed_size=embed_size, num_layers=num_layers)
```

Class decoder sử dụng par_inject

```python
class Decoder(nn.Module):
    def __init__(self, feature_size, output_size, embed_size, num_layers):
        super(Decoder, self).__init__()
        self.hidden_size = embed_size

        self.embedding = nn.Embedding(output_size, embed_size)
        self.gru = nn.GRU(embed_size + embed_size , embed_size, num_layers=num_layers)

        self.out = nn.Linear(embed_size, output_size)

        self.map = nn.Linear(feature_size, embed_size)
        self.softmax = nn.LogSoftmax(dim=1)

 #Par inject
    def forward(self, input, hidden, feature):
        feature = self.map(feature)
        output = self.embedding(input)
        output = F.relu(output)
        output = torch.cat((feature, output), dim=2)
        output, hidden = self.gru(output, hidden)
        output = self.softmax(self.out(output[0]))

        #output = output.squeeze(0)

        return output, hidden


def get_decoder(feature_size=2048, output_size=10000, embed_size=128, num_layers=1):
    return Decoder(feature_size=feature_size, output_size=output_size, embed_size=embed_size, num_layers=num_layers)
```

Class decoder sử dụng merge

```
class Decoder(nn.Module):
    def __init__(self, feature_size, output_size, embed_size, num_layers):
        super(Decoder, self).__init__()
        self.hidden_size = embed_size

        self.embedding = nn.Embedding(output_size, embed_size)
        self.gru = nn.GRU(embed_size, embed_size, num_layers=num_layers)

        self.map = nn.Linear(feature_size, embed_size)

        self.out = nn.Linear(embed_size*2, output_size)

        self.softmax = nn.LogSoftmax(dim=1)

# Merge
    def forward(self, input, hidden, feature):

        feature = self.map(feature)
        output = self.embedding(input)
        output = F.relu(output)
        output, hidden = self.gru(output, hidden)
        output = torch.cat((feature, output), dim=2)
        output = self.softmax(self.out(output[0]))


        return output, hidden

def get_decoder(feature_size=2048, output_size=10000, embed_size=128, num_layers=1):
    return Decoder(feature_size=feature_size, output_size=output_size, embed_size=embed_size, num_layers=num_layers)
```

4. Implement

Sử dụng cơ chế merge architure

Chúng ta có 2 input đầu vào, nên phần dữ liệu truyền vào của chúng ta có dạng [X_image, X_caption] và Y.

Sau khi dùng một mạng RNN (LSTM) để xử lý captions, tacộng các kết quả lại để đưa ra output.

```
inputs1 = Input(shape=(2048,))
fe1 = Dropout(0.5)(inputs1)
fe2 = Dense(256, activation='relu')(fe1)


# max_length = 35, vocab_size = 2005, embedding_dim = 200
inputs2 = Input(shape=(max_length,))
se1 = Embedding(vocab_size, embedding_dim, mask_zero=True)(inputs2)
se2 = Dropout(0.5)(se1)
se3 = LSTM(256)(se2)

decoder1 = add([fe2, se3])
decoder2 = Dense(256, activation='relu')(decoder1)
outputs = Dense(vocab_size, activation='softmax')(decoder2)

model = Model(inputs=[inputs1, inputs2], outputs=outputs)
```

Layer 2 ta sử dụng Glove Model

```
model.layers[2].set_weights([embedding_matrix])
model.layers[2].trainable = False
```

Truyền dữ liệu vào huấn luyện, dữ liệu caption

```
def data_generator(captions, images, w2i, max_length, batch_size):

  X_image, X_cap, y = [], [], []
  n = 0
  while 1:
    for id, caps in captions.items():
      n += 1
      image = images[id]
      for cap in caps:
        # encode the sequence
        seq = [w2i[word] for word in cap.split(' ') if word in w2i]


        for i in range(1, len(seq)):
          # split into input and output pair
          in_seq, out_seq = seq[:i], seq[i]

          # pad input sequence
          in_seq = pad_sequences([in_seq], maxlen=max_length)[0]
          # encode output sequence
          out_seq = to_categorical([out_seq], num_classes=vocab_size)[0]


          X_image.append(image)
          X_cap.append(in_seq)
          y.append(out_seq)
      if n == batch_size:
        yield ([np.array(X_image), np.array(X_cap)], np.array(y))
        X_image, X_cap, y = [], [], []
        n = 0
```

Bắt đầu huấn luyện

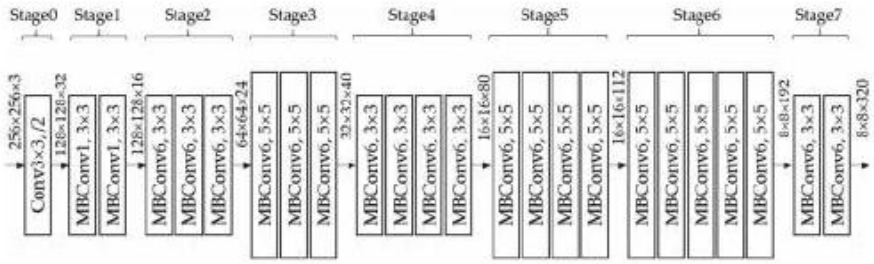
```
generator = data_generator(captions=captions, images=train_features, w2i=w2i, max_length=max_length, batch_size=batch_size)
model.fit(generator, epochs=epochs, steps_per_epoch=steps, verbose=1, callbacks=[cp_callback])
```

Ta được kết quả huấn luyện

```
Epoch 1/5
8091/8091 [==============================] - 1055s 130ms/step - loss: 3.3241
Epoch 2/5
8091/8091 [==============================] - 1038s 128ms/step - loss: 2.9998
Epoch 3/5
8091/8091 [==============================] - 1041s 129ms/step - loss: 2.8467
Epoch 4/5
8091/8091 [==============================] - 1039s 128ms/step - loss: 2.7386
Epoch 5/5
8091/8091 [==============================] - 1034s 128ms/step - loss: 2.6541
```

II.EfficientNet

- EfficientNets là một nhóm các mạng tích chập sâu đã đạt được độ chính xác cao nhất trong các phân loại khác nhau với hiệu suất tốt hơn tới 10 lần và được gọi là nhỏ hơn và nhanh hơn. Kiến trúc này đã thể hiện tính mới của nó thông qua thành tựu mới nhất của AutoML, đặc biệt là sự mở rộng thông minh và có kiểm soát của 3 chiều (chiều rộng, chiều sâu, độ phân giải) của mạng thần kinh bằng cách sử dụng các hệ số phức hợp.
- EfficientNets giải quyết vấn đề bằng cách tăng kích thước và áp dụng tìm kiếm lưới theo ràng buộc tài nguyên cố định thay vì thay đổi kích thước.
- Trong EfficientNets, EfficientNetB2 được sử dụng để chứng minh độ chính xác hiện đại nhằm thể hiện dòng giới thiệu nhanh hơn và nhỏ hơn theo kiểu tích chậ p mạng lưới thần kinh



-
- Implement:

```
base_model = tf.keras.applications.efficientnet.EfficientNetB2(include_top=False)
base_model.trainable=False

inputs = tf.keras.Input(shape=(IMG_SIZE+(3,)))
x = base_model(inputs, training=False)
x = tf.keras.layers.GlobalAveragePooling2D()(x)
x = tf.keras.layers.Dense(len(class_names))(x)
outputs = tf.keras.layers.Activation('softmax', dtype=tf.float32)(x)
model_2 = tf.keras.Model(inputs, outputs)

model_2.compile(loss='categorical_crossentropy',
                optimizer=tf.keras.optimizers.Adam(),
                metrics=["accuracy"])
model_2.summary()
```

-

```
history_ENB2 = model_2.fit(train_data,epochs=5,steps_per_epoch=len(train_data),validation_data=test_data,validation_steps=len(test_data),callbacks=[model_checkpoint])
```

```
Epoch 1/5
220/220 [==============================] - 85s 332ms/step - loss: 0.7025 - accuracy: 0.7653 - val
_loss: 0.2727 - val_accuracy: 0.9040
Epoch 2/5
220/220 [==============================] - 68s 306ms/step - loss: 0.2413 - accuracy: 0.9166 - val
_loss: 0.2480 - val_accuracy: 0.9137
Epoch 3/5
220/220 [==============================] - 68s 308ms/step - loss: 0.2140 - accuracy: 0.9231 - val
_loss: 0.2423 - val_accuracy: 0.9123
Epoch 4/5
220/220 [==============================] - 68s 308ms/step - loss: 0.1946 - accuracy: 0.9290 - val
_loss: 0.2314 - val_accuracy: 0.9187
Epoch 5/5
220/220 [==============================] - 68s 308ms/step - loss: 0.1847 - accuracy: 0.9314 - val
_loss: 0.2234 - val_accuracy: 0.9190
```

-

III. Swin Tranfromer

Shifted windows Transformer (Swin Transformer) là bản nâng cấp của Vision Transformer:

Kiến trúc Transformer phân cấp (hierarchical Transformer), tại các layer sâu, các path hàng xóm gần nhau sẽ dần dần được hợp nhất lại.

Sử dụng self attention trên 1 vùng cục bộ thay vì toàn bộ ảnh như Vision Transformer, do vậy sẽ tiết kiệm chi phí tính toán hơn.

Shifted windows, sẽ giúp các patch ảnh không bị "bó cứng" khi phải seft attention trong 1 cửa sổ cục bộ mà sẽ có "cơ hội" được gặp và tính self attention cùng v ới các path khác trong 1 cửa sổ mới.

Mô hình Transformer output ra nhiều scale khác nhau thay vì chỉ output 1 scale duy nhất như Vision Transformer.

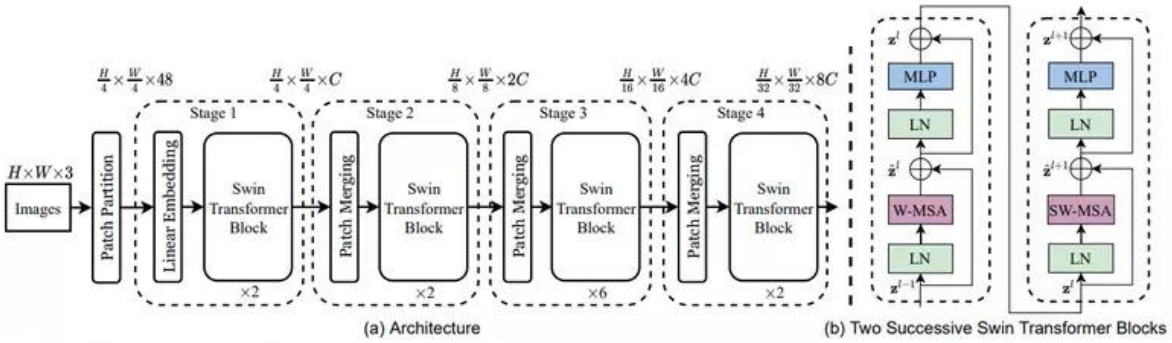


Figure 3. (a) The architecture of a Swin Transformer (Swin-T); (b) two successive Swin Transformer Blocks (notation presented with Eq. (3)). W-MSA and SW-MSA are multi-head self attention modules with regular and shifted windowing configurations, respectively.

implement

```
def train_model(model, criterion, optimizer, scheduler, num_epochs=10):
    since = time.time()
    best_model_wts = copy.deepcopy(model.state_dict())
    best_acc = 0.0

    for epoch in range(num_epochs):
        print(f'Epoch {epoch}/{num_epochs - 1}')
        print("-"*10)

        for phase in ['train', 'val']:
            if phase == 'train':
                model.train()
            else:
                model.eval()

            running_loss = 0.0
            running_corrects = 0.0

            for inputs, labels in tqdm(dataloaders[phase]):
                inputs = inputs.to(device)
                labels = labels.to(device)

                optimizer.zero_grad()

                with torch.set_grad_enabled(phase == 'train'):
                    outputs = model(inputs)
                    _, preds = torch.max(outputs, 1)
                    loss = criterion(outputs, labels)

                    if phase == 'train':
                        loss.backward()
                        optimizer.step()
                running_loss += loss.item() * inputs.size(0)
                running_corrects += torch.sum(preds == labels.data)

            if phase == 'train':
                scheduler.step()

            epoch_loss = running_loss / dataset_sizes[phase]
            epoch_acc =  running_corrects.double() / dataset_sizes[phase]

            print("{} Loss: {:.4f} Acc: {:.4f}".format(phase, epoch_loss, epoch_acc))

            if phase == 'val' and epoch_acc > best_acc:
                best_acc = epoch_acc
                best_model_wts = copy.deepcopy(model.state_dict())
        print()
    time_elapsed = time.time() - since
    print('Training complete in {:.0f}m {:.0f}s'.format(time_elapsed // 60, time_elapsed % 60))
    print("Best Val Acc: {:.4f}".format(best_acc))

    model.load_state_dict(best_model_wts)
```

```
model_ft = train_model(model, criterion, optimizer, exp_lr_scheduler, num_epochs=7)
```

IV.Database

Dữ liệu gồm 8000 ảnh, 6000 ảnh cho traning set, 1000 cho dev set (validation set) và 1000 ảnh cho test set. Mỗi ảnh gồm có 5 caption

Dữ liệu là một bộ sưu tập điểm chuẩn mới cho mô tả và tìm kiếm hình ảnh dựa trên câu, bao gồm 8.000 hình ảnh, 6000 ảnh cho traning set, 1000 cho dev set (validation set) và 1000 ảnh cho test set, mỗi hình ảnh được ghép nối với năm chú thích khác nhau cung cấp mô tả rõ ràng về các thực thể và sự kiện nổi bật. … Các hình ảnh được chọn từ sáu nhóm Flickr khác nhau và có xu hướng không chứa bất kỳ người hoặc địa điểm nổi tiếng nào mà được chọn thủ công để mô tả nhiều cảnh và tình huống khác nhau.