HW4 - Visual Question Answering

December 14, 2022

If the data needs to be downloaded, uncomment the code in the next cell to download it.

Note - The download is upwards of 20GB.

The code creates the folder structure shown below

```
data
    train2014
        [training images]
     val2014
         [validation images]
     MultipleChoice_mscoco_train2014_questions.json
     mscoco_train2014_annotations.json
     MultipleChoice_mscoco_val2014_questions.json
     mscoco_val2014_annotations.json
```

[2]:
```python
# file fetch_data.py
from zipfile import ZipFile
import os
import requests
from tqdm import tqdm

questions_url = (
    "https://s3.amazonaws.com/cvmlp/vqa/mscoco/vqa/Questions_Train_mscoco.zip"
)
annotations_url = (
    "https://s3.amazonaws.com/cvmlp/vqa/mscoco/vqa/Annotations_Train_mscoco.zip"
)
images_url = "http://images.cocodataset.org/zips/train2014.zip"

val_questions_url = (
    "https://s3.amazonaws.com/cvmlp/vqa/mscoco/vqa/Questions_Val_mscoco.zip"
)

val_annotations_url = (
    "https://s3.amazonaws.com/cvmlp/vqa/mscoco/vqa/Annotations_Val_mscoco.zip"
)

val_images_url = "http://images.cocodataset.org/zips/val2014.zip"
```

```
def download_data(url):
    filename = url.split("/")[-1]
    r = requests.get(url, stream=True)
    total_size = int(r.headers.get("content-length", 0))
    block_size = 1024
    t = tqdm(total=total_size, unit="iB", unit_scale=True)
    with open(filename, "wb") as f:
        for data in r.iter_content(block_size):
            t.update(len(data))
            f.write(data)
    t.close()

    with ZipFile(filename, "r") as zip:
        zip.extractall("data/")

    os.remove(filename)


# download_data(questions_url)
# download_data(annotations_url)
# download_data(images_url)
# download_data(val_questions_url)
# download_data(val_annotations_url)
# download_data(val_images_url)
```

The next step is to load and preprocess the questions and answers data

[3]:
```
# file data.py
from collections import defaultdict
import json
import os
from PIL import Image
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split


q_path = "MultipleChoice_mscoco_train2014_questions.json"
a_path = "mscoco_train2014_annotations.json"
img_dir = "/kaggle/input/mscoco2014/train2014"


val_img_dir = "data/val2014"
val_q_path = "data/MultipleChoice_mscoco_val2014_questions.json"
val_a_path = "data/mscoco_val2014_annotations.json"
```

```
def read_image(image_name, expand=False, img_dir=img_dir):
    image = Image.open(os.path.join(img_dir, image_name))
    image = image.convert('RGB')
    image = np.asarray(image.resize((224, 224))) / 255.0
    if expand:
        image = np.expand_dims(image, axis=0)
    return image



def get_image_name(image_id):
    return "COCO_train2014_" + str(image_id).zfill(12) + ".jpg"


def get_image_id(image_name):
    return int(image_name.split("_")[-1].split(".")[0])


def get_questions(question_file):
    with open(question_file) as f:
        file = json.load(f)
        questions = file["questions"]
    return questions


def get_annotations(annotation_file):
    with open(annotation_file) as f:
        file = json.load(f)
        annotations = file["annotations"]

    annotations = {i["question_id"]: i["multiple_choice_answer"] for i in␣
 ↪annotations}
    return annotations



def load_data(q_path, a_path, subset=False):
    annotations = get_annotations(a_path)

    questions = get_questions(q_path)
    rows = []
    for q in questions:
        if subset and q["image_id"] not in available_image_ids:
            continue
        row = {
            "image_id": q["image_id"],
```

```
            "question": q["question"],
            "question_id": q["question_id"],
            "answer": annotations[q["question_id"]],
        }
        rows.append(row)
    return pd.DataFrame(rows)

def generate_image_batch(batch_size=32):
    result = []
    while True:
        for i in os.listdir(img_dir):
            result.append(read_image(i))
            if len(result) == batch_size:
                yield result
                result = []


available_image_ids = [get_image_id(i) for i in os.listdir(img_dir)]


data = load_data(q_path, a_path, subset=False)

top_1000_answers = data["answer"].value_counts().index[:1000]
df = data[data["answer"].isin(top_1000_answers)]
X = df[["image_id", "question", "question_id"]]
y = df["answer"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

if os.path.exists('data/val2014'):
    val_df = load_data(val_q_path, val_a_path, subset=False)
else:
    val_df = data
```

0.1 CNN model

Next the cnn model to encode images is created. It uses the vgg16 architecture in addition to an attention layer

[4]:
```
# file cnn.py
from tensorflow.keras import layers, models, optimizers
import numpy as np
import tensorflow.keras.backend as k


class CNN:
    def __init__(self, num_classes=1000):
```

```
        self.num_classes = num_classes
        self.input_layer = layers.Input(shape=(224, 224, 3))
        self.model = models.Sequential()
        self.model.add(self.input_layer)
        self.create_model()

    def add_vgg_16_blocks(self):
        self.add_vgg_block(1, 64)
        self.add_vgg_block(1, 128)
        self.add_vgg_block(2, 256)
        self.add_vgg_block(2, 512)
        self.add_vgg_block(2, 512)

    def add_dense_layers(self):
        self.model = layers.Flatten()(self.model)
        self.model = layers.Dense(4096, activation="relu")(self.model)
        self.model = layers.Dropout(0.5)(self.model)
        self.model = layers.Dense(4096, activation="relu")(self.model)

    def add_output_layer(self):
        self.model.add(layers.Dropout(0.5))
        self.model.add(layers.Dense(self.num_classes, activation="softmax"))

    def add_vgg_block(self, num_convs, num_channels):
        for _ in range(num_convs):
            self.model.add(layers.ZeroPadding2D(padding=(1, 1)))
            self.model.add(
                layers.Conv2D(num_channels, kernel_size=3, activation="relu")
            )
            self.model.add(layers.ZeroPadding2D(padding=(1, 1)))
        self.model.add(layers.MaxPool2D(pool_size=2, strides=2))

    def load_weights(self, weights_path):
        self.model.load_weights(weights_path)

    def compile(self):
        sgd = optimizers.SGD(lr=0.1, decay=1e-6, momentum=0.9, nesterov=True)
        self.model.compile(
            loss="categorical_crossentropy",
            optimizer=sgd,
        )

    def create_model(self, output_layer=False):
        self.add_vgg_16_blocks()
        self.add_attention_layer()
        self.add_dense_layers()
        if output_layer:
```

```
            self.add_output_layer()

    def get_image_features(self, image):
        return self.model.predict(image)

    def add_attention_layer(self, name="cnn_attention"):
        depth = 512
        bn_features = layers.BatchNormalization()(self.model.output)

        attn = layers.Conv2D(64, kernel_size=1, activation="relu")(bn_features)
        attn = layers.Conv2D(16, kernel_size=1, activation="relu")(attn)
        attn = layers.Conv2D(64, kernel_size=1, activation="sigmoid",␣
↪name="cnn_attention")(attn)
        self.attention_layer = attn
        up_c2_w = np.ones((1, 1, 64, depth))
        up_c2 = layers.Conv2D(
            depth, kernel_size=1, activation="linear", use_bias=False,␣
↪weights=[up_c2_w]
        )
        up_c2.trainable = False
        attn = up_c2(attn)
        mask_feature = layers.Multiply()([attn, bn_features])
        gap_features = layers.GlobalAveragePooling2D()(mask_feature)
        gap_mask = layers.GlobalAveragePooling2D()(attn)
        attn_gap = layers.Lambda(lambda x: x[0] / x[1], name="RescaleGAP")(
            [gap_features, gap_mask]
        )

        self.model = attn_gap

    def get_attention_features(self, image):
        attn_func = k.function(
            inputs=[self.input_layer, k.learning_phase()],
            outputs=[self.attention_layer],
        )
        attn_image = attn_func([image, 0])[0]
        attn_image = attn_image[0, :, :, 0]
        return attn_image
```

1 Language (LSTM model)

The next stage deals with tokenize the text and encoding the answers

The language model consists of 3 LSTM layers

[5]:
```
# file lstm.py
import pandas as pd
```

```
import numpy as np

from tensorflow.keras import layers, models, utils
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
# from data import read_image, get_image_name, X_train, y_train,␣
 ↪top_1000_answers
# from vqa_model import VQA


tokenizer = Tokenizer()
train_texts = pd.concat([X_train["question"], y_train])

tokenizer.fit_on_texts(train_texts)

vocab_size = len(tokenizer.word_index) + 1
max_len = 30

print(f"Max length: {max_len}")
print(f"Vocab size: {vocab_size}")


def tokenize(question_text):
    tokens = tokenizer.texts_to_sequences([question_text])
    tokens = pad_sequences(tokens, maxlen=max_len, padding="pre")
    return tokens


def detokenize(tokens):
    tokens = np.squeeze(tokens)
    question_text = tokenizer.sequences_to_texts([tokens])
    return question_text[0]




def encode_answer(answer):
    idx = top_1000_answers.get_loc(answer)
    one_hot = np.zeros(1000)
    one_hot[idx] = 1
    return one_hot


def decode_answer(one_hot):
    idx = np.argmax(one_hot)
    return top_1000_answers[idx]
```

```
class LanguageModel:
    def __init__(self, max_len=30, num_hidden_lstm_units=128):
        self.input_layer = layers.Input(shape=(max_len, 1))
        model = layers.LSTM(num_hidden_lstm_units, return_sequences=True)(
            self.input_layer
        )
        model = layers.LSTM(num_hidden_lstm_units, return_sequences=True)(model)
        model = layers.LSTM(num_hidden_lstm_units)(model)
        self.model = model
```

```
Max length: 30
Vocab size: 11464
```

1.1 Fusion model

The VQA class combines the CNN and LSTM models and adds a dense layers for classification

[6]:
```
# file vqa.py
from tensorflow.keras import layers, models, utils, metrics
# from cnn import CNN
# from lstm import LanguageModel


class VQA:
    def __init__(self, num_dense_layers=3):
        self.num_dense_layers = num_dense_layers
        self.cnn = CNN()
        self.lstm = LanguageModel()
        self.model = layers.Concatenate()([self.cnn.model, self.lstm.model])
        self.create_model()

    def add_dense_layer(self, num_hidden_lstm_units=128):
        self.model = layers.Dense(num_hidden_lstm_units,␣
 ↪activation="relu")(self.model)
        self.model = layers.Dropout(0.5)(self.model)

    def load_weights(self, weights_path):
        self.model.load_weights(weights_path)

    def compile(self):
        self.model.compile(
            loss="categorical_crossentropy",
            optimizer="adam",
        )

    def create_model(self):
        for _ in range(self.num_dense_layers):
```

```
            self.add_dense_layer()
        self.model = layers.Dense(1000, activation="softmax")(self.model)
        self.model = models.Model(
            inputs=[self.cnn.input_layer, self.lstm.input_layer],
            outputs=self.model,
        )
        self.compile()

    def get_question_features(self, question):
        return self.model.predict(question)
```

Create a training generator to return the data in batches

[7]:
```
def generate_batch(rows, batch_size=32):
    questions = np.zeros((batch_size, max_len))
    images = np.zeros((batch_size, 224, 224, 3))
    answers = np.zeros((batch_size, 1000))
    idx = 0
    while True:
        for i, row in rows.iterrows():
            img = get_image_name(row["image_id"])
            image = read_image(img)
            images[idx] = image
            questions[idx] = tokenize(row["question"])
            answers[idx] = encode_answer(y_train[i])
            idx += 1
            if idx == batch_size:
                idx = 0
                yield [images, questions], answers

                questions = np.zeros((batch_size, max_len))
                try:
                    images = np.zeros((batch_size, 224, 224, 3))
                except ValueError as e:
                    print(img)
                    raise e

                answers = np.zeros((batch_size, 1000))
```

Visualize the model

[8]:
```
import matplotlib.pyplot as plt
import random

vqa = VQA()

# vqa.model.summary()
utils.plot_model(
```

```
    vqa.model, to_file="model_plot.png", show_shapes=True, show_layer_names=True
)
img = Image.open("model_plot.png")
plt.figure(figsize=(25,80))
plt.imshow(img)
plt.show()
```


input_1: InputLayer | input: [(None, 224, 224, 3)] | output: [(None, 224, 224, 3)]
zero_padding2d: ZeroPadding2D | input: (None, 224, 224, 3) | output: (None, 226, 226, 3)
conv2d: Conv2D | input: (None, 226, 226, 3) | output: (None, 224, 224, 64)
zero_padding2d_1: ZeroPadding2D | input: (None, 224, 224, 64) | output: (None, 226, 226, 64)
max_pooling2d: MaxPooling2D | input: (None, 226, 226, 64) | output: (None, 113, 113, 64)
zero_padding2d_2: ZeroPadding2D | input: (None, 113, 113, 64) | output: (None, 115, 115, 64)
conv2d_1: Conv2D | input: (None, 115, 115, 64) | output: (None, 113, 113, 128)
zero_padding2d_3: ZeroPadding2D | input: (None, 113, 113, 128) | output: (None, 115, 115, 128)
max_pooling2d_1: MaxPooling2D | input: (None, 115, 115, 128) | output: (None, 57, 57, 128)
zero_padding2d_4: ZeroPadding2D | input: (None, 57, 57, 128) | output: (None, 59, 59, 128)
conv2d_2: Conv2D | input: (None, 59, 59, 128) | output: (None, 57, 57, 256)
zero_padding2d_5: ZeroPadding2D | input: (None, 57, 57, 256) | output: (None, 59, 59, 256)
zero_padding2d_6: ZeroPadding2D | input: (None, 59, 59, 256) | output: (None, 61, 61, 256)
conv2d_3: Conv2D | input: (None, 61, 61, 256) | output: (None, 59, 59, 256)
zero_padding2d_7: ZeroPadding2D | input: (None, 59, 59, 256) | output: (None, 61, 61, 256)
max_pooling2d_2: MaxPooling2D | input: (None, 61, 61, 256) | output: (None, 30, 30, 256)
zero_padding2d_8: ZeroPadding2D | input: (None, 30, 30, 256) | output: (None, 32, 32, 256)
conv2d_4: Conv2D | input: (None, 32, 32, 256) | output: (None, 30, 30, 512)
zero_padding2d_9: ZeroPadding2D | input: (None, 30, 30, 512) | output: (None, 32, 32, 512)
zero_padding2d_10: ZeroPadding2D | input: (None, 32, 32, 512) | output: (None, 34, 34, 512)
conv2d_5: Conv2D | input: (None, 34, 34, 512) | output: (None, 32, 32, 512)
zero_padding2d_11: ZeroPadding2D | input: (None, 32, 32, 512) | output: (None, 34, 34, 512)
max_pooling2d_3: MaxPooling2D | input: (None, 34, 34, 512) | output: (None, 17, 17, 512)
zero_padding2d_12: ZeroPadding2D | input: (None, 17, 17, 512) | output: (None, 19, 19, 512)
conv2d_6: Conv2D | input: (None, 19, 19, 512) | output: (None, 17, 17, 512)
zero_padding2d_13: ZeroPadding2D | input: (None, 17, 17, 512) | output: (None, 19, 19, 512)
zero_padding2d_14: ZeroPadding2D | input: (None, 19, 19, 512) | output: (None, 21, 21, 512)
conv2d_7: Conv2D | input: (None, 21, 21, 512) | output: (None, 19, 19, 512)
zero_padding2d_15: ZeroPadding2D | input: (None, 19, 19, 512) | output: (None, 21, 21, 512)
max_pooling2d_4: MaxPooling2D | input: (None, 21, 21, 512) | output: (None, 10, 10, 512)
batch_normalization: BatchNormalization | input: (None, 10, 10, 512) | output: (None, 10, 10, 512)
conv2d_8: Conv2D | input: (None, 10, 10, 512) | output: (None, 10, 10, 64)
conv2d_9: Conv2D | input: (None, 10, 10, 64) | output: (None, 10, 10, 16)
cnn_attention: Conv2D | input: (None, 10, 10, 16) | output: (None, 10, 10, 64)
conv2d_10: Conv2D | input: (None, 10, 10, 64) | output: (None, 10, 10, 512)
multiply: Multiply | input: [(None, 10, 10, 512), (None, 10, 10, 512)] | output: (None, 10, 10, 512)
global_average_pooling2d_1: GlobalAveragePooling2D | input: (None, 10, 10, 512) | output: (None, 512)
global_average_pooling2d: GlobalAveragePooling2D | input: (None, 10, 10, 512) | output: (None, 512)
RescaleGAP: Lambda | input: [(None, 512), (None, 512)] | output: (None, 512)
flatten: Flatten | input: (None, 512) | output: (None, 512)
input_2: InputLayer | input: [(None, 30, 1)] | output: [(None, 30, 1)]
dense: Dense | input: (None, 512) | output: (None, 4096)
lstm: LSTM | input: (None, 30, 1) | output: (None, 30, 128)
dropout: Dropout | input: (None, 4096) | output: (None, 4096)
lstm_1: LSTM | input: (None, 30, 128) | output: (None, 30, 128)
dense_1: Dense | input: (None, 4096) | output: (None, 4096)
lstm_2: LSTM | input: (None, 30, 128) | output: (None, 128)
concatenate: Concatenate | input: [(None, 4096), (None, 128)] | output: (None, 4224)
dense_2: Dense | input: (None, 4224) | output: (None, 128)
dropout_1: Dropout | input: (None, 128) | output: (None, 128)
dense_3: Dense | input: (None, 128) | output: (None, 128)
dropout_2: Dropout | input: (None, 128) | output: (None, 128)
dense_4: Dense | input: (None, 128) | output: (None, 128)
dropout_3: Dropout | input: (None, 128) | output: (None, 128)
dense_5: Dense | input: (None, 128) | output: (None, 1000)

[9]:
```
from tensorflow.keras import callbacks
epochs = 1
batch_size = 32
steps_per_epoch = len(X_train) // batch_size
gen = generate_batch(X_train, batch_size)

checkpoint_filepath = 'checkpoints/checkpoint'
model_checkpoint_callback = callbacks.ModelCheckpoint(
    filepath=checkpoint_filepath,
    save_weights_only=True,
)

# load weights
vqa.model.load_weights(checkpoint_filepath)


# training loop
# for e in range(epochs):
#     print(f"Epoch {e+1}")
#     for step in range(steps_per_epoch):
#         [images, questions], answers = next(gen)

#         res = vqa.model.fit([images, questions], answers, verbose=0,␣
 ↪callbacks=[model_checkpoint_callback])
#         if (step + 1) % 50 == 0:
#             print(f"Step {step + 1}: {res.history['loss']}")
```

[9]:
```
<tensorflow.python.training.tracking.util.CheckpointLoadStatus at
0x7f8f7c4c7d10>
```

1.2 Accuracy on validation images

[11]:
```
batch_size = 32
val_df = val_df[val_df["answer"].isin(top_1000_answers)]

def generate_batch(rows, batch_size=32):
    questions = np.zeros((batch_size, max_len))
    images = np.zeros((batch_size, 224, 224, 3))
    answers = np.zeros((batch_size, 1000))
    idx = 0
    while True:
        for i, row in rows.iterrows():
            img = get_image_name(row["image_id"])
            image = read_image(img)
```

```
            images[idx] = image
            questions[idx] = tokenize(row["question"])
            answers[idx] = encode_answer(row['answer'])
            idx += 1
            if idx == batch_size:
                idx = 0
                yield [images, questions], answers

                questions = np.zeros((batch_size, max_len))
                try:
                    images = np.zeros((batch_size, 224, 224, 3))
                except ValueError as e:
                    print(img)
                    raise e

                answers = np.zeros((batch_size, 1000))

val_gen = generate_batch(val_df, batch_size)
batches = len(val_df)//batch_size
accuracy = []

for i in range(batches):
    [images, questions], answers = next(val_gen)
    p = vqa.model.predict(x=[images, questions], batch_size=batch_size,␣
 ↪verbose=False)
    correct = np.count_nonzero(answers.argmax(axis=1) == p.argmax(axis=1))
    accuracy.append(correct/batch_size)
acc = np.min(accuracy)
print(f"Accuracy on validation set: {acc}")
```

```
Accuracy on validation set: 0.1875
```

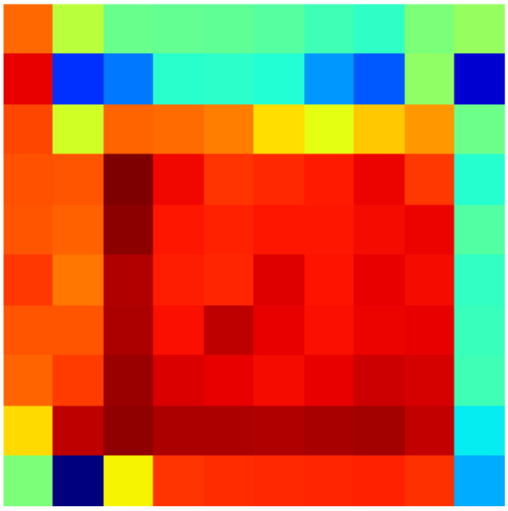
1.3 Validation Images

Predict the answers for 30 random images from validation set

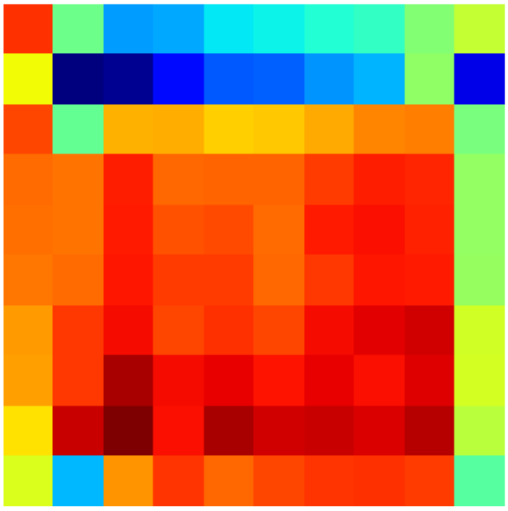
[39]:
```
import matplotlib.gridspec as gridspec



attn_func = k.function(
    inputs=[vqa.cnn.input_layer, vqa.lstm.input_layer],
    outputs=[vqa.model.get_layer('cnn_attention').output],
)


for i in range(30):
```

```
    image_id = random.choice(val_df["image_id"])
    row = val_df[val_df["image_id"] == image_id]
    img = read_image(get_image_name(image_id))
    question = row["question"].values[0]
    actual_answer = row["answer"].values[0]
    img_input = np.expand_dims(img, axis=0)
    predicted_answer = decode_answer(
        vqa.model.predict([img_input, tokenize(question).reshape(1, 30)],␣
↪verbose=0)
    )

    attn_image = attn_func([img_input, tokenize(question).reshape(1, 30)])[0]
    attn_image = attn_image[0, :, :, 0]

    fig, ax = plt.subplots(1, 2, figsize=(12, 12))

    plt.suptitle(f"\nQ: {question}\nActual: {actual_answer}\nPredicted:␣
↪{predicted_answer}")

    ax[0].imshow(img)
    ax[1].imshow(attn_image, cmap="jet")

    ax[0].axis("off")
    ax[1].axis("off")
    plt.show()
```

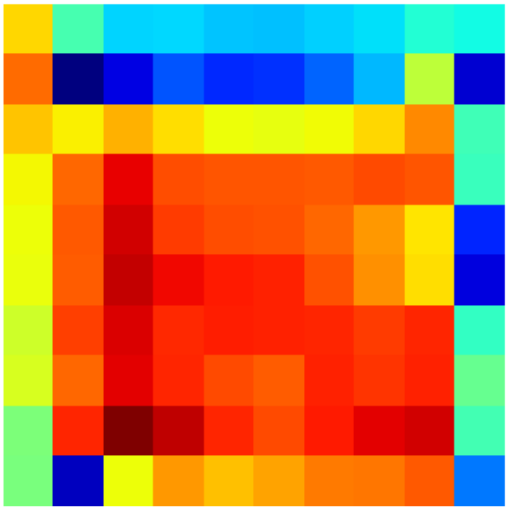
Q: Do the giraffes look upset?
Actual: no
Predicted: yes



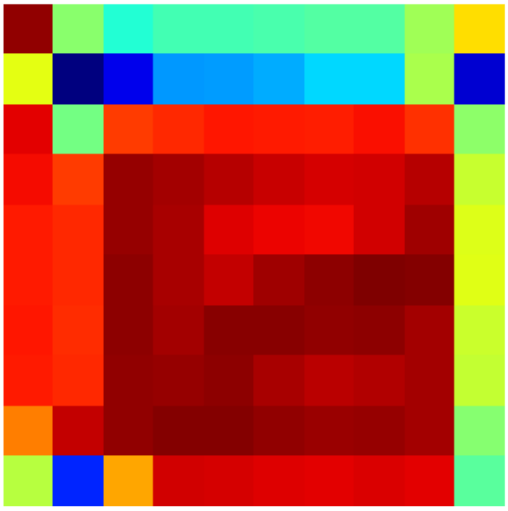
Q: Are they dancing?
Actual: no
Predicted: yes



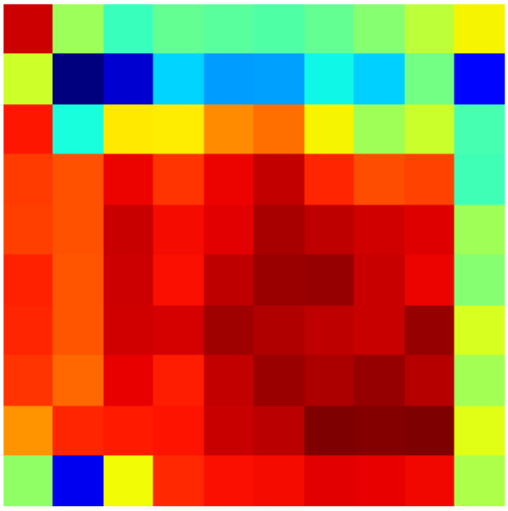
Q: What type of train is this?
Actual: passenger
Predicted: yes



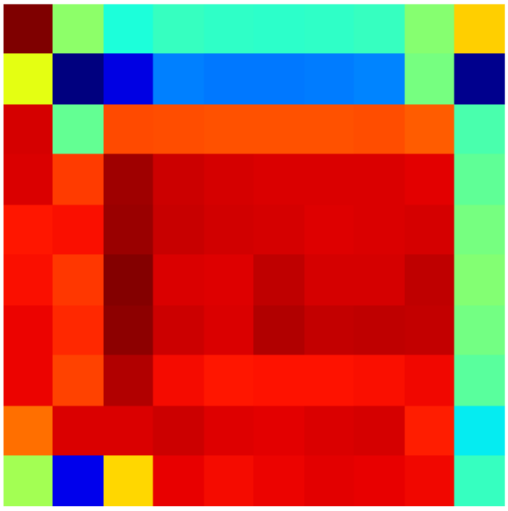
Q: Are these foods going to be barbecued?
Actual: no
Predicted: yes



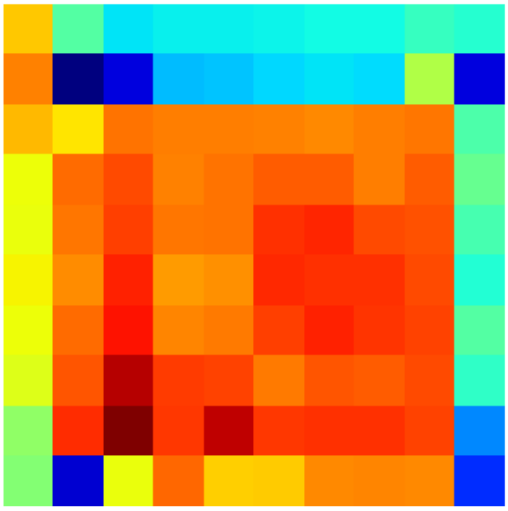
Q: How many cupcakes have strawberries on top?
Actual: 23
Predicted: yes



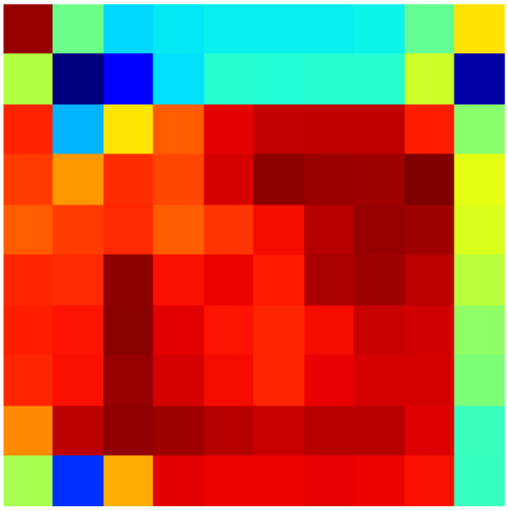
Q: What is on the tail of the airplane?
Actual: leaf
Predicted: yes



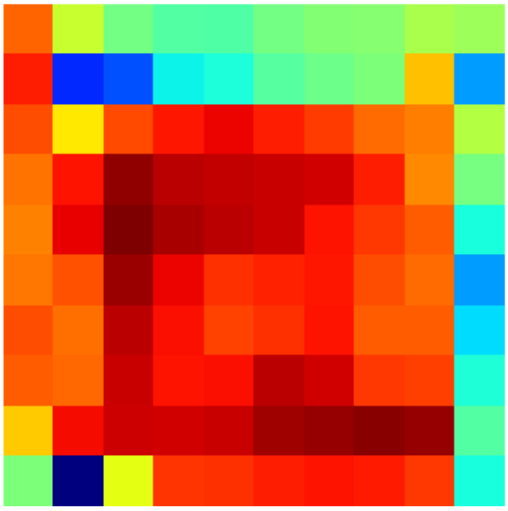
Q: Is the boy wearing a necklace?
Actual: yes
Predicted: yes



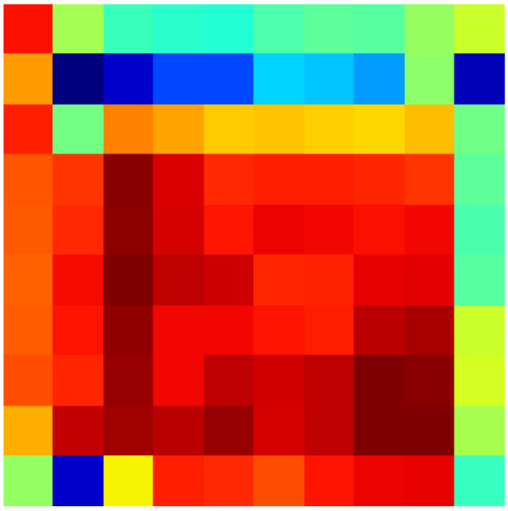
Q: How many dogs?
Actual: 1
Predicted: yes



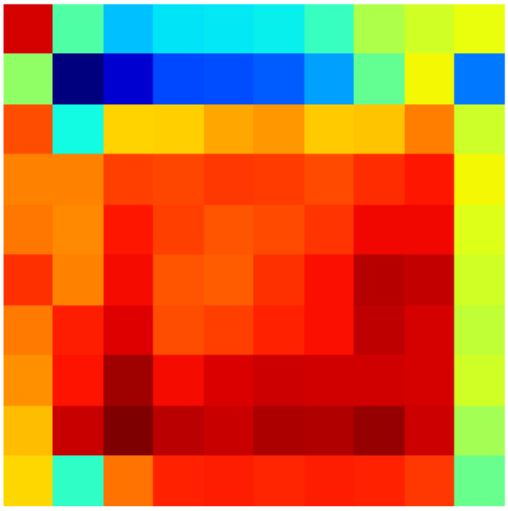
Q: What is on the woman's tank top?
Actual: flower
Predicted: yes



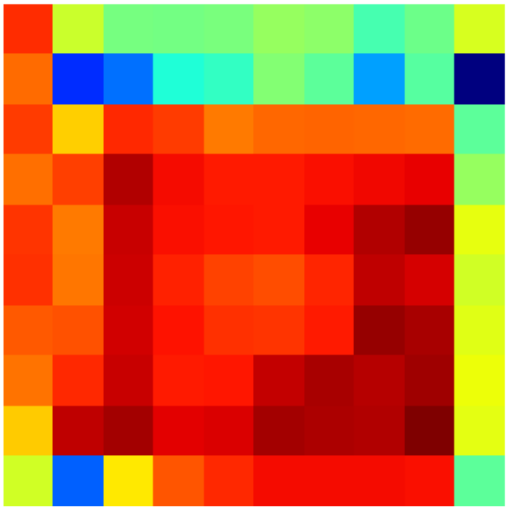
Q: Is the bear reading?
Actual: yes
Predicted: yes



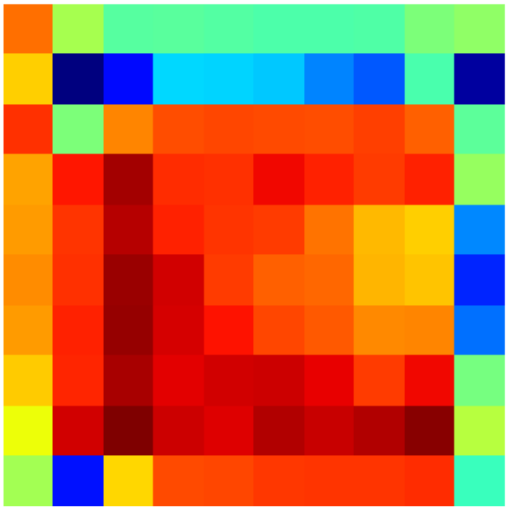
Q: What color is the plate?
Actual: red
Predicted: yes



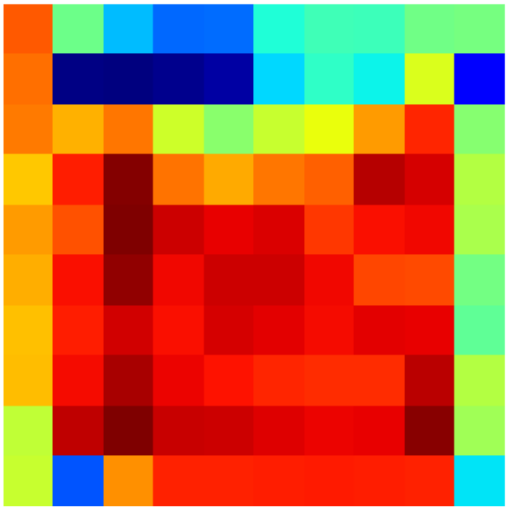
Q: Is it a sunny day?
Actual: yes
Predicted: yes



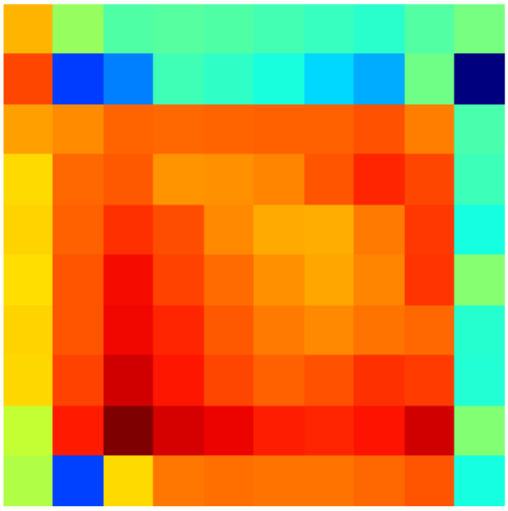
Q: Is this a house or an apartment?
Actual: apartment
Predicted: yes



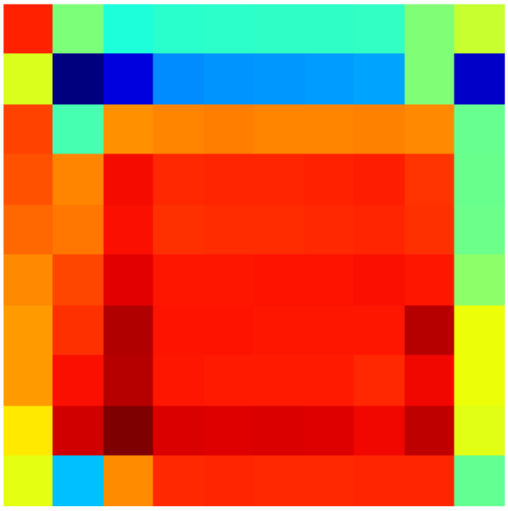
Q: Is the station deserted?
Actual: yes
Predicted: yes



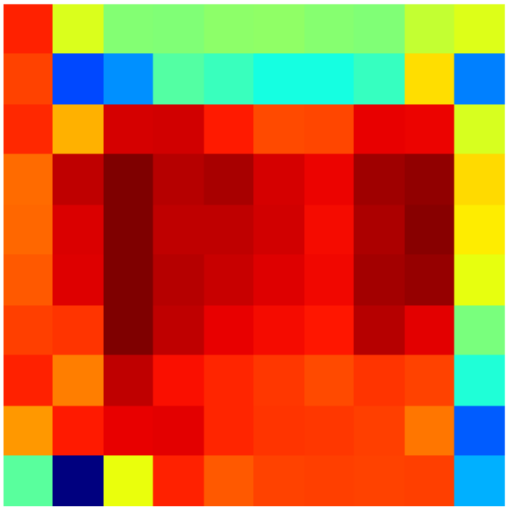
Q: What is the cat laying on?
Actual: bed
Predicted: yes



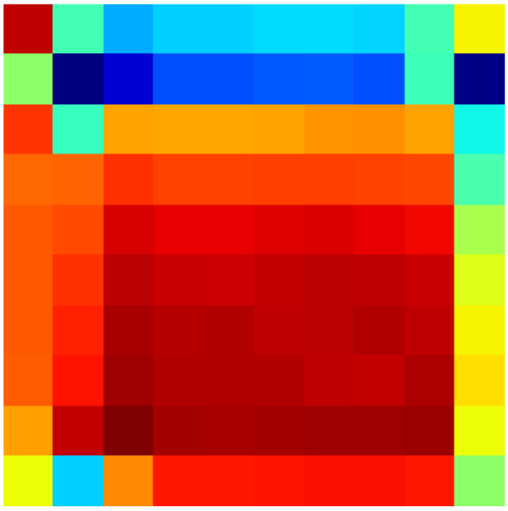
Q: Was this photo taken at sunrise of sunset?
Actual: sunset
Predicted: yes



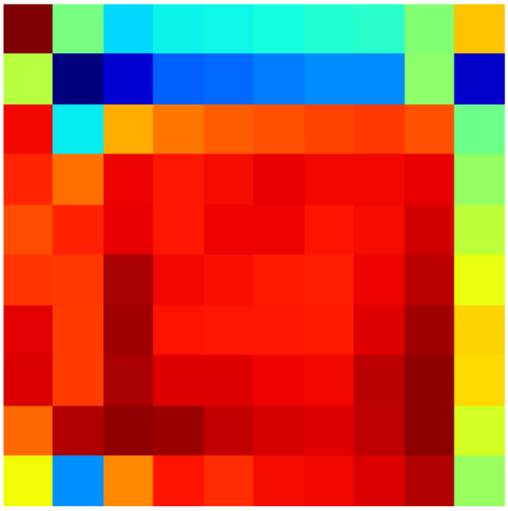
Q: Are these animals in a jungle?
Actual: no
Predicted: yes



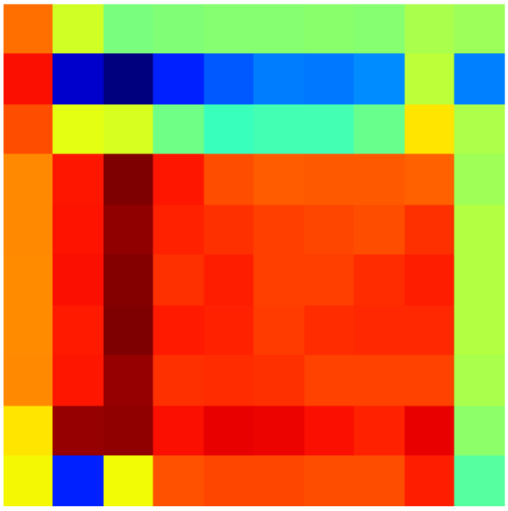
Q: What is causing low visibility?
Actual: fog
Predicted: yes



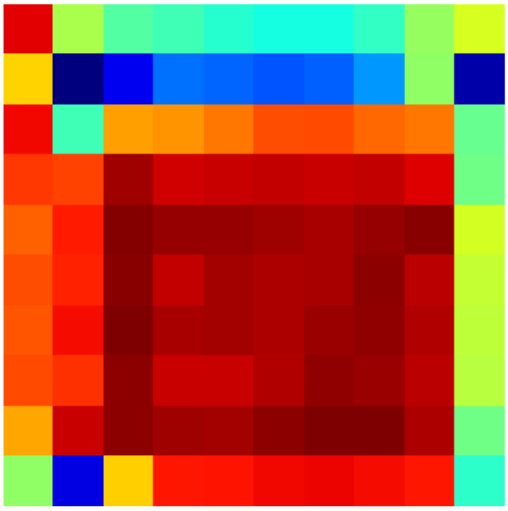
Q: Is this a kitchen?
Actual: yes
Predicted: yes



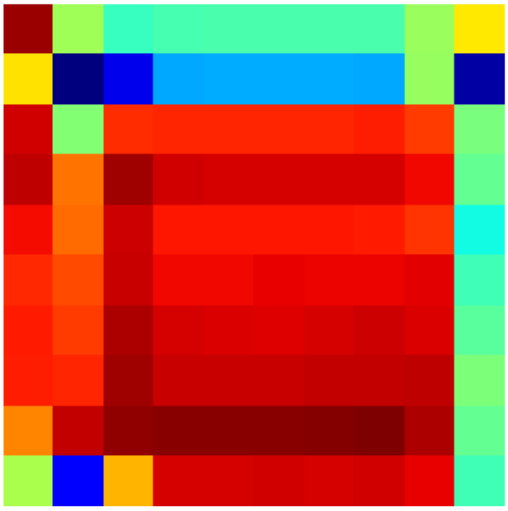
Q: Is this a painting?
Actual: no
Predicted: yes



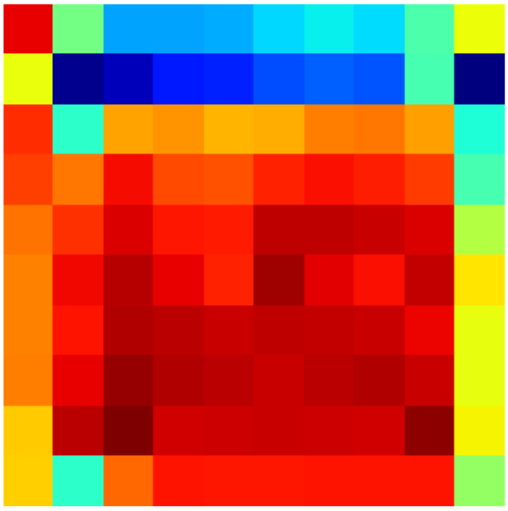
Q: Is the Gas Station Logo BP?
Actual: yes
Predicted: yes



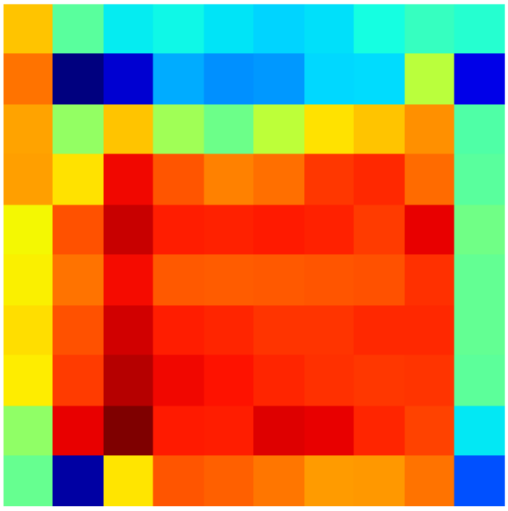
Q: Are the bird's going inside the water?
Actual: no
Predicted: yes



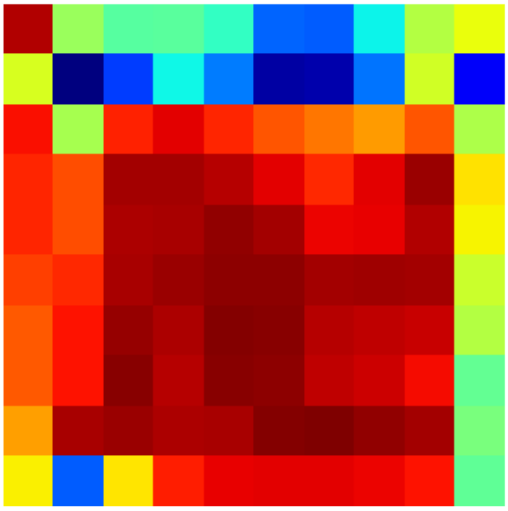
Q: How many lambs in this picture?
Actual: 4
Predicted: yes



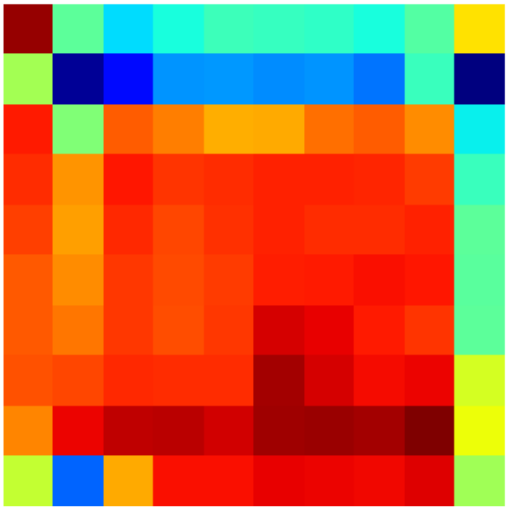
Q: What is in the picture?
Actual: man playing tennis
Predicted: yes



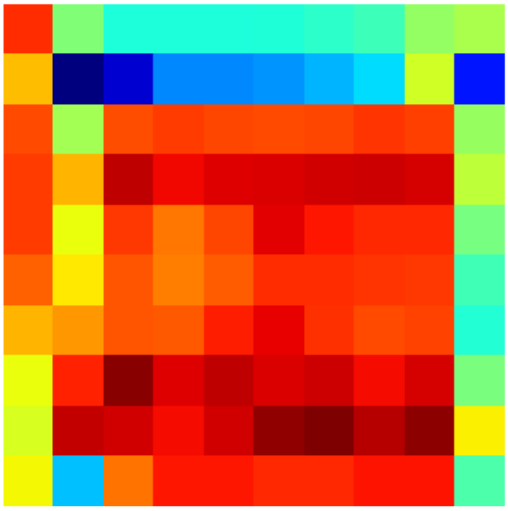
Q: How many flags are on the back of this motorcycle?
Actual: 2
Predicted: yes



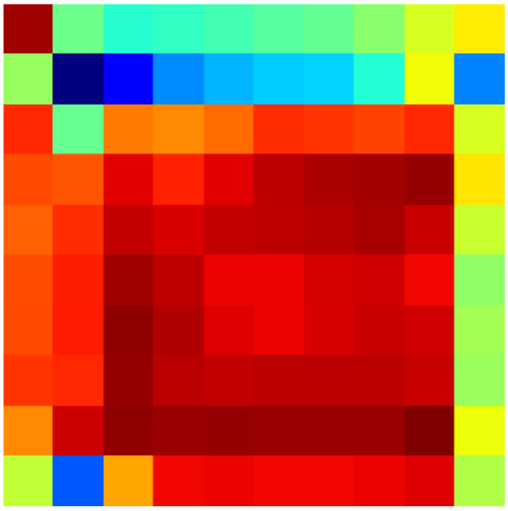
Q: Where are the bears?
Actual: in tree
Predicted: yes



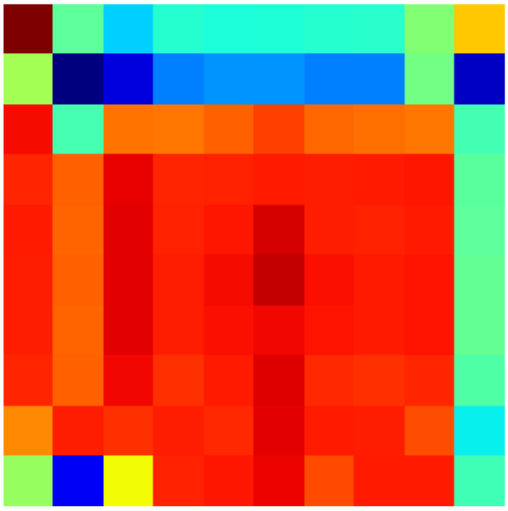
Q: How old do you think this boy is?
Actual: 10
Predicted: yes



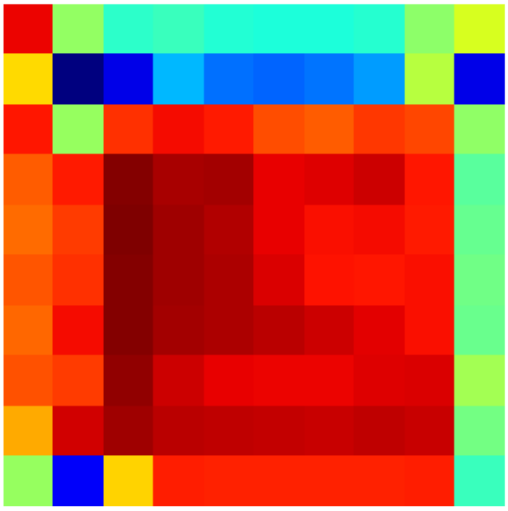
Q: How many bikes are parked here?
Actual: 12
Predicted: yes



Q: What activity did this boy just do?
Actual: surfing
Predicted: yes



Q: Are these toys all manufactured by the same company?
Actual: yes
Predicted: yes



[ ]: