ДОДАТОК А

Код класу моделі навчання

У лістингу А.1 представлено програмну реалізацію методу «configure_optimizers», яка є однаковою для всіх підходів:

Лістинг А.1 – Код методу «configure_optimizers»

```
def configure_optimizers(self):
          return optim.Adam(
               self.parameters(),
               self.learning_rate
          )
```

Програмні реалізації для методів навчального та валідаційного кроку відрізняються для підходів, які використовують в якості функції втрат функцію середньоквадратичної похибки (лістинг А.2) та функцію косинусової «схожості» (лістинг А.3).

Лістинг А.2 – Код методів «training_step» та «validation_step» для підходів на основі середньоквадратичної похибки

```
def training_step(self, batch, batch_idx):
     y, x = batch
     p_z_true, p_z_pred = self(x, y)
     loss = self.loss_func(p_z_true, p_z_pred)
     self.log( 'train_loss', loss, on_epoch=True)
     return loss

def validation_step(self, batch, batch_idx):
     y, x = batch
     p_z_true, p_z_pred = self(x, y)
     loss = self.loss_func(p_z_true, p_z_pred)
     self.log('val_loss', loss, on_epoch=True)
```

Лістинг А.3 – Код методів «training_step» та «validation_step» для підходів на основі косинусової «схожості»

```
    def training_step(self, batch, batch_idx):
        x, y = batch
        y_hat = self(x)
        loss = self.loss_func(
            y_hat, y,
            torch.ones(y.size(0)).to(self.device)
        )
        self.log('train_loss', loss, on_epoch=True)
        return loss

    def validation_step(self, batch, batch_idx):
        x, y = batch
        y_hat = self(x)
        loss = self.loss_func(
            y_hat, y,
            torch.ones(y.size(0)).to(self.device)
        )
        self.log('val_loss', loss, on_epoch=True)
```

Для запуску процесу навчання використовується код, представлений у лістингу А.4.

Лістинг А.4 – Код для запуску процесу навчання

```
trainer = L.Trainer(
    max_epochs=1000,
    callbacks=[EarlyStopping(
        monitor='val_loss',
        patience=3
    )],
    accelerator="gpu")
trainer.fit(model, train_loader, val_loader)
```

ДОДАТОК Б

Код для отримання метрик моделей

У лістингу Б.1 наведено загальний код отримання метрик успіху моделей приведення векторів представлення до спільного простору. У даній програмній реалізації:

– змінна «MODEL» зберігає реченнєвий трансформер для готового порівняння двох описів – еталонного та передбаченого моделлю;

– змінна «projection» є об’єктом класу для отримання результату від певної моделі;

– функція «cosine_similarity» обчислює метрику косинусової «схожості» за формулою 2.6;

– функція «l2_similarity» обчислює метрику «схожості» як значення, обернене до евклідової відстані між двома векторами:

$$\mathrm{L2} = \frac{1}{\left|\vec{a} - \vec{b}\right|}; \qquad \text{(Б.1)}$$

– метрики зберігаються у вигляді файлу з розширенням .csv, який потім можливо обробляти за допомогою табличного процесора або додатково написаного коду.

Лістинг Б.1 – Загальний код отримання метрик успіху моделей приведення векторів представлення до спільного простору

```
def cosine_similarity(a: np.ndarray, b: np.ndarray):
    a_norm = np.linalg.norm(a)
    b_norm = np.linalg.norm(b)
    a_b = np.dot(a, b)
    return a_b / (a_norm * b_norm)
```

Продовження лістингу Б.1

```
def l2_similarity(a: np.ndarray, b: np.ndarray):
     return 1 / np.linalg.norm(a - b)

def main(
          image_emb_folder: str, prompt_folder: str,
          prompt_emb_folder: str, split_file: str,
          result_file: str
     ):
     image_emb_folder_path = Path(image_emb_folder)
     prompt_folder_path = Path(prompt_folder)
     prompt_emb_folder_path = Path(prompt_emb_folder)
     projection = InferencePipeline()
     with open(split_file, 'r') as f:
          names = json.load(f)['test']
     all_tuples = []
     for name in tqdm(names, "Loading all artifacts"):
          image = image_emb_folder_path \
               / f"{name}.npy"
          image_emb = np.load(image)
          image_emb = projection.encode_image(image_emb)
          prompt = prompt_emb_folder_path \
               / f"{name}.npy"
          prompt_emb = np.load(prompt)[0]
          prompt_emb = projection.encode_text(prompt_emb)
          folder = prompt_folder_path / f"{name}.txt", 'r'
          with open(folder) as f:
               prompt = f.read().strip()
          all_tuples.append((
               name, image_emb,
               prompt, prompt_emb
          ))
     to_save = {
          'prompt': [],
          'gt_image': [],
```

Продовження лістингу Б.1

```
            'pred_image': [],
            'pred_prompt': [],
            'similarity': [],
            'top_search': []
        }
        for name, _, prompt, prompt_emb in all_tuples:
            # sim = lambda item: cosine_similarity(
            #     item[1], prompt_emb
            # )
            # Або:
            # sim = lambda item: l2_similarity(
            #     item[1], prompt_emb
            # )
            all_points = sorted(
                all_tuples, key=sim,
                reverse=True
            )
            closest_point = all_points[0]
            for i, point in enumerate(all_points):
                if point[0] == name:
                    top_search = i + 1
                    break
            pred_image = closest_point[0]
            pred_prompt = closest_point[2]
            pred_emb = MODEL.encode(
                pred_prompt,
                convert_to_tensor=True
            )
            gt_emb = MODEL.encode(
                prompt,
                convert_to_tensor=True
            )
            similarity = MODEL.similarity(
                gt_emb, pred_emb)
```

Продовження лістингу Б.1

```
            similarity = float(similarity)
            to_save['prompt'].append(prompt)
            to_save['gt_image'].append(name)
            to_save['pred_image'].append(pred_image)
            to_save['pred_prompt'].append(pred_prompt)
            to_save['similarity'].append(similarity)
            to_save['top_search'].append(top_search)
      df = pd.DataFrame(to_save)
      df.to_csv(result_file, index=False)
```

На лістингах Б.2 та Б.3 наведено код для створення матриці помилок для задачі бінарної класифікації між котами та собаками та створення таблиці метрик F1-score для кожної породи окремо відповідно.

Лістинг Б.2 – Код для створення матриці помилок

```
metrics = pd.read_csv('шлях до файлу з метриками схожості')
label = lambda item: item[0].isupper()
gt = metrics['gt_image'].apply(label)\
     .astype(int)
pred = metrics['pred_image'].apply(label)\
     .astype(int)
cm = confusion_matrix(gt, pred, labels=[0, 1])
plt.figure(figsize=(8, 6))
sns.heatmap(
     cm, annot=True,
     fmt='d', cmap='Blues',
     xticklabels=['Cat', 'Dog'],
     yticklabels=['Cat', 'Dog']
)
plt.xlabel('Predicted')
plt.ylabel('Ground Truth')
plt.title('Confusion Matrix - Matrix Transformation')
plt.show()
```

Лістинг Б.3 – Код для створення таблиці метрик F1-score для кожної породи окремо

```
all_breeds = list(set([
    "_".join(name.split("_")[:-1])
    for name in metrics['gt_image'].tolist()
]))
all_f1s = []
for breed in all_breeds:
    breed_check = lambda item: item.startswith(breed)
    gt = metrics['gt_image'].apply(breed_check)\
        .astype(int)
    pred = metrics['pred_image'].apply(breed_check)\
        .astype(int)
    f1 = f1_score(
        gt, pred,
        zero_division=0
    )
    all_f1s.append(f1)

all_breeds = [
    breed[0].upper() + \
        breed[1:].replace("_", " ")
    for breed in all_breeds
]
breed_f1 = pd.DataFrame({
    'Breed': all_breeds,
    'F1 Score': all_f1s
})
breed_f1.to_csv('шлях до таблиці', index=False)
```

ДОДАТОК В

Повний код реалізації запропонованого підходу

Лістинг В.1 – Повна програмна реалізація запропонованого підходу сегментації екземплярів на основі текстового запиту

```
import cv2
import numpy as np
from transformers import pipeline
from PIL import Image
from transformers import ViTImageProcessor, ViTModel
import numpy as np
from sentence_transformers import SentenceTransformer
import sys

sys.path.append(".")

from source.model_training.inference_pipeline \
     import InferencePipeline
from source.model_training.metric_calculation \
     import cosine_similarity, l2_similarity

SEGMENTATION_MODEL = pipeline(
     "image-segmentation",
     "facebook/maskformer-swin-base-coco"
)

IMAGE_EMBED_PROCESSOR = ViTImageProcessor.from_pretrained(
     'google/vit-base-patch16-224-in21k'
)
IMAGE_EMBED_MODEL = ViTModel.from_pretrained(
     'google/vit-base-patch16-224-in21k'
)
```

Продовження лістингу В.1

```
PROMPT_EMBED_MODEL = SentenceTransformer(
     'sentence-transformers/all-MiniLM-L6-v2'
)


def get_segments(image_path):
     results = SEGMENTATION_MODEL(
          images=str(image_path),
          subtask="panoptic",
          label_ids_to_fuse=set()
     )
     results = [
          np.array(result["mask"])
          for result in results
          if result["label"] in ["cat", "dog"]
     ]
     return results

def get_image_embedding(pipeline, images):
     embeddings = []
     for image in images:
          inputs = IMAGE_EMBED_PROCESSOR(
               images=image,
               return_tensors="pt"
          )
          outputs = IMAGE_EMBED_MODEL(**inputs)
          embedding = outputs.last_hidden_state[0, 0]\
               .cpu().detach().numpy()
          embedding = pipeline.encode_image(embedding)
          embeddings.append(embedding)
     return np.stack(embeddings, axis=0)

def get_prompt_embedding(pipeline, prompts):
     embeddings = []
```

Продовження лістингу В.1

```
    for prompt in prompts:
        embedding = PROMPT_EMBED_MODEL\
            .encode(prompt, convert_to_tensor=True)\
                .cpu().numpy()
        embedding = pipeline\
            .encode_text(embedding)
        embeddings.append(embedding)
    return np.stack(embeddings, axis=0)

def get_prompted_segment(
    pipeline, image_path,
    prompt, use_l2 = False
):
    image = cv2.imread(image_path)
    segments = get_segments(image_path)
    if len(segments) == 0:
        return None
    processed_segments = []
    for segment in segments:
        cropped = cv2.bitwise_and(
            image, image,
            mask=segment.astype(np.uint8)
        )
        cropped_image = Image.fromarray(cropped)
        gray_image = cropped_image.convert("L")
        bbox = gray_image.getbbox()
        cropped_image = cropped_image.crop(bbox)
        processed_segments.append(cropped_image)
    embeddings = get_image_embedding(
        pipeline, processed_segments
    )
    prompt_embedding = get_prompt_embedding(
        pipeline, [prompt]
    )[0]
```

Продовження лістингу В.1

```
    similarity_func = l2_similarity if use_l2 \
         else cosine_similarity
    tuples = zip(segments, embeddings)

    tuples = [
         (segment, embedding, similarity_func(
              embedding, prompt_embedding
         ))
         for segment, embedding in tuples
    ]
    best_segment, _, _ = max(
         tuples, key=lambda item: item[2]
    )
    return best_segment
```

ДОДАТОК Г

Відомість кваліфікаційної роботи

Позначення	Найменування	Дод. відомості
	Текстові документи	
1.	Пояснювальна записка	107 с.
	Інші документи	
2.	Презентаційні матеріали	__ плакатів

.	.	Прізвище та ініціали.	Підп.	Дата			
Розробив		Шкіль А. С.			Сегментація екземплярів на основі текстових запитів	Шифр групи	Код напр./спец.
Перевірив		Вітько О. В.				ІТШІ-21-1	122
Н.контр.		Малєєва І.А.				ХНУРЕ кафедра ШІ	
Затв.		Золотухін О.В.					