Министерство науки и высшего образования Российской Федерации
Калужский филиал
федерального государственного автономного
образовательного учреждения высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(КФ МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ **ИУК «Информатика и управление»**

КАФЕДРА **ИУК4 «Программное обеспечение ЭВМ, информационные технологии»**

Лабораторная работа №3

«Нейронная передача стиля. Генеративно-состязательные сети»

ДИСЦИПЛИНА: «Методы глубокого обучения»

Выполнил: студент гр. ИУК4-21М _____ (Сафронов Н.С.)
(подпись) (Ф.И.О.)

Проверил: _____ (Белов Ю.С.)
(подпись) (Ф.И.О.)

Дата сдачи (защиты):

Результаты сдачи (защиты):

- Балльная оценка:
- Оценка:

Калуга, 2025

Цель работы: получение практических навыков построения вариационных автокодировщиков.

Задание:

Разработать модель вариационного автокодировщика.

Результаты выполнения работы

```
params: dict[str, t.Any] = {
    'encoder_layers': [128],
    'decoder_layers': [128],
    'digit_classification_layers': [128],
    'activation': 'sigmoid',
    'decoder_std': 0.5,
    'z_dim': 10,
    'digit_classification_weight': 10.0,
    'epochs': 20,
    'batch_size': 100,
    'learning_rate': 0.001
}
```

Рисунок 1 – Параметры модели

```python
class VAE(Model):
    """
    Вариационный автоэнкодер для генерации рукописных цифр MNIST.

    Состоит из:
    1. Энкодера - кодирует изображение в латентное пространство
    2. Декодера - восстанавливает изображение из латентного представления
    3. Классификатора - предсказывает цифру на входном изображении
    """
    def __init__(self, params: dict[str, t.Any]):
        """."""

        super(VAE, self).__init__()
        self.params = params

        encoder_layers = [
            layers.Dense(u, activation=params['activation'])
            for u in params['encoder_layers']
        ]
        self.encoder = tf.keras.Sequential(encoder_layers)
        self.mu_layer = layers.Dense(params['z_dim'])
        self.log_var_layer = layers.Dense(params['z_dim'])

        classifier_layers = [
            layers.Dense(u, activation=params['activation'])
            for u in params['digit_classification_layers']
        ]
        self.classifier = tf.keras.Sequential(classifier_layers + [layers.Dense(10)])

        decoder_layers = [
            layers.Dense(u, activation=params['activation'])
            for u in params['decoder_layers']
        ]
        self.decoder = tf.keras.Sequential(decoder_layers + [layers.Dense(784, activation='sigmoid')])

    def encode(self, x: tf.Tensor) -> tuple[tf.Tensor, tf.Tensor]:
        """Кодирует входные данные в параметры нормального распределения."""
        h = self.encoder(x)
        return self.mu_layer(h), self.log_var_layer(h)

    def reparameterize(self, mu: tf.Tensor, log_var: tf.Tensor) -> tf.Tensor:
        """РепараметризациЯ для семплирования из распределения."""
        std = tf.exp(0.5 * log_var)
        eps = tf.random.normal(shape=mu.shape)
        return mu + eps * std

    def decode(self, z: tf.Tensor, digit_prob: tf.Tensor) -> tf.Tensor:
        """Декодирует латентное представление в изображение."""
        inputs = tf.concat([z, digit_prob], axis=1)
        return self.decoder(inputs)

    def classify(self, x: tf.Tensor) -> tf.Tensor:
        """Классифицирует цифру на входном изображении."""
        return self.classifier(x)

    def call(self, x: tf.Tensor) -> tuple[tf.Tensor, tf.Tensor, tf.Tensor, tf.Tensor]:
        """Полный проход данных через модель."""
        mu, log_var = self.encode(x)
        z = self.reparameterize(mu, log_var)
        digit_logits = self.classify(x)
        digit_prob = tf.nn.softmax(digit_logits)
        reconstructed = self.decode(z, digit_prob)
        return reconstructed, mu, log_var, digit_logits

    def generate(self, digit: int, z: tf.Tensor | None = None) -> tf.Tensor:
        """Генерирует изображение по заданной цифре и латентному вектору."""
        if z is None:
            z = tf.random.normal((1, self.param['z_dim']))
        digit_one_hot = tf.one_hot([digit], depth=10)
```

Рисунок 2 – Модель вариационного автоэнкодера

```python
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0
x_train = x_train.reshape(-1, 784)
x_test = x_test.reshape(-1, 784)
y_train = y_train.astype('int32')
y_test = y_test.astype('int32')

train_dataset = tf.data.Dataset.from_tensor_slices((x_train, y_train))
train_dataset = (
    train_dataset
    .shuffle(buffer_size=params.get('buffer_size', 1024))
    .batch(params['batch_size'])
    .prefetch(buffer_size=tf.data.experimental.AUTOTUNE)
)

test_dataset = tf.data.Dataset.from_tensor_slices((x_test, y_test))
test_dataset = (
    test_dataset
    .batch(params['batch_size'])
    .prefetch(buffer_size=tf.data.experimental.AUTOTUNE)
)
```

Рисунок 3 – Загрузка и преобразование датасета

```
Epoch 12 завершена, средний loss: 20.8745
Epoch 13/20: 100%|██████████| 600/600 [00:54<00:00, 11.06batch/s, loss=23.0225, recon=0.2043, kl=0.0965, cls=0.1630]

Validation Loss: 20.8937

Epoch 13 завершена, средний loss: 20.7125
Epoch 14/20: 100%|██████████| 600/600 [00:53<00:00, 11.15batch/s, loss=22.8471, recon=0.2029, kl=0.1008, cls=0.1554]

Validation Loss: 20.7963

Epoch 14 завершена, средний loss: 20.5668
Epoch 15/20: 100%|██████████| 600/600 [00:50<00:00, 11.94batch/s, loss=22.6867, recon=0.2015, kl=0.1048, cls=0.1485]

Validation Loss: 20.7233

Epoch 15 завершена, средний loss: 20.4417
Epoch 16/20: 100%|██████████| 600/600 [00:52<00:00, 11.45batch/s, loss=22.5409, recon=0.2003, kl=0.1086, cls=0.1423]

Validation Loss: 20.6639

Epoch 16 завершена, средний loss: 20.3529
Epoch 17/20: 100%|██████████| 600/600 [00:51<00:00, 11.76batch/s, loss=22.4055, recon=0.1992, kl=0.1122, cls=0.1365]

Validation Loss: 20.5604

Epoch 17 завершена, средний loss: 20.2386
Epoch 18/20: 100%|██████████| 600/600 [00:51<00:00, 11.71batch/s, loss=22.2804, recon=0.1981, kl=0.1156, cls=0.1311]

Validation Loss: 20.5156

Epoch 18 завершена, средний loss: 20.1539
Epoch 19/20: 100%|██████████| 600/600 [00:50<00:00, 11.90batch/s, loss=22.1653, recon=0.1971, kl=0.1189, cls=0.1262]

Validation Loss: 20.4387

Epoch 19 завершена, средний loss: 20.0928
Epoch 20/20: 100%|██████████| 600/600 [00:51<00:00, 11.76batch/s, loss=22.0579, recon=0.1962, kl=0.1220, cls=0.1216]

Validation Loss: 20.4167

Epoch 20 завершена, средний loss: 20.0188
```

Рисунок 4 – Процесс обучения модели

```python
def generate_digits(model: VAE):
    """Генерирует и отображает изображения цифр 0-9."""
    plt.figure(figsize=(10, 2))

    for i in range(10):
        plt.subplot(1, 10, i + 1)
        z = tf.random.normal((1, params['z_dim']))
        digit_one_hot = tf.one_hot([i], depth=10)
        res = model.decode(z, digit_one_hot).numpy()
        plt.imshow(res.reshape(28, 28), cmap='gray')
        plt.axis('off')
        plt.title(str(i))
    plt.tight_layout()
    plt.show()

generate_digits(model)
```

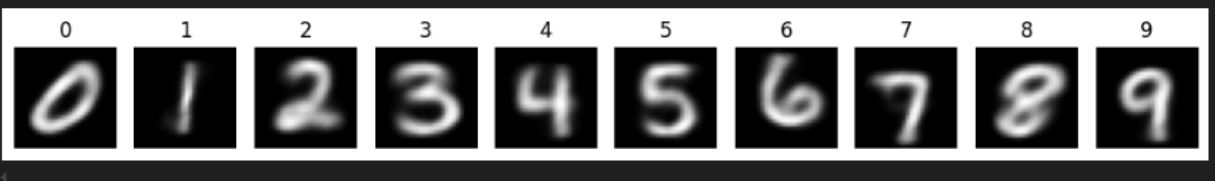

Рисунок 5 — Результат генерации VAE цифр датасета MNIST

Вывод: в ходе выполнения работы были получены практические навыки построения вариационных автокодировщиков.