Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования «Южно-Уральский государственный университет»
(национальный исследовательский университет)
Институт естественных и точных наук
Кафедра прикладной математики и программирования

Отчет по лабораторной работе № 5
по дисциплине «Современные нейросетевые технологии»

Авторы работы
Студент группы ЕТ-122
_____/ Матвеева А.В.
«____» _____2025 г.

Руководитель работы,
_____/ Кичеев Д.М.
«____» _____2025 г.

Челябинск 2025

Задание:

Цель: настоящей работы состоит в том, чтобы изучить возможности начальной настройки сверточных нейронных весов.

Выполнение задания:

1. В ходе данного проекта я освоила базовые навыки работы с начальной настройкой весов для сверточной нейронной сети. Я использовала три варианта начальной настройки: автоэнкодер, SimCLR, k-means. В качестве основной модели была использована упрощенная модель из лабораторной №3, которая обучалась каждый раз на 20 эпохах.
2. Для выполнения задачи бы взяты те же данные, что в лабораторной №3. Это изображения различных животных, разделенные на 10 классов. 11587 изображений в тренировочном наборе, 2892 изображения в тестовом наборе. Изображения загружены на локальный диск и поделены на классы, каждый класс в отдельной папке. Для предобучения я использовала те же данные, убрав таргет.
3. При тестировании моделей я использую метрику качества Accuracy, которая определяется следующим образом:

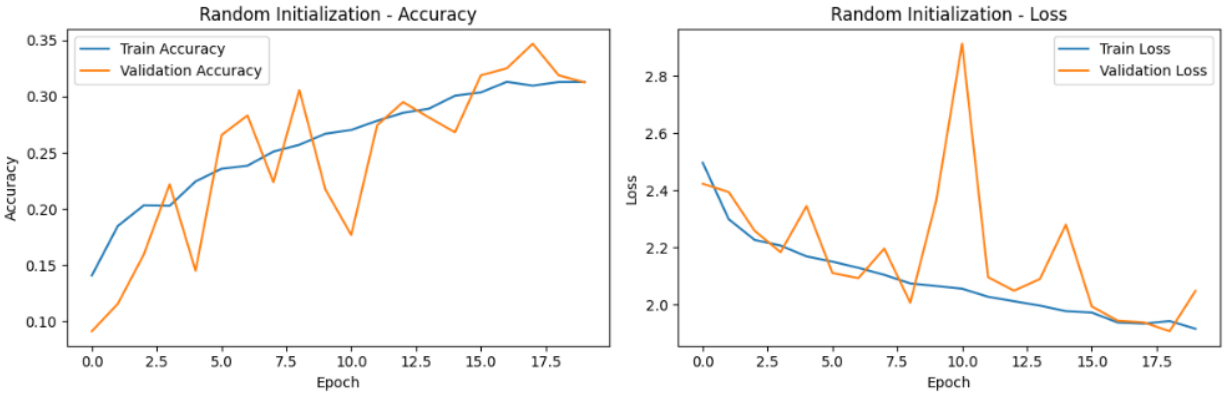
$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

где:

- *TP* (True Positives) — количество истинных положительных предсказаний (правильно классифицированные объекты класса 1).
- *TN* (True Negatives) — количество истинных отрицательных предсказаний (правильно классифицированные объекты класса 0).
- *FP* (False Positives) — количество ложных положительных предсказаний (объекты класса 0, ошибочно классифицированные как класс 1).
- *FN* (False Negatives) — количество ложных отрицательных предсказаний (объекты класса 1, ошибочно классифицированные как класс 0).

4. Данные мной были скачаны с сайта kaggle, хранятся локально.
5. Я использовала стандартную функция fit для обучения нейронных сетей. Я переносила обучение с моделей преобучения послойно.
6. Мной были разработаны несколько вариантов первоначальных весов нейронной сети.

1) Первый вариант - это рандомная настройка. Модель рандомно выбирает начальные веса и их обучает.

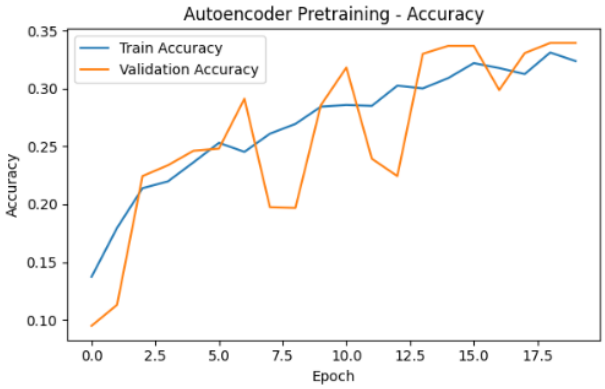


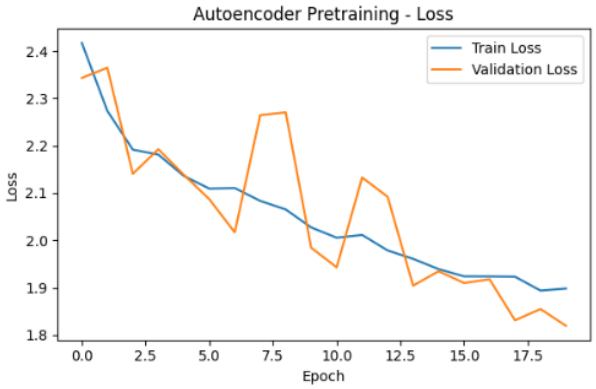
Итоговая метрика - 0.3125. Модель обучается, что видно из матрицы ошибок.

2) Второй вариант - это настройка весов с помощью автоэкодера.

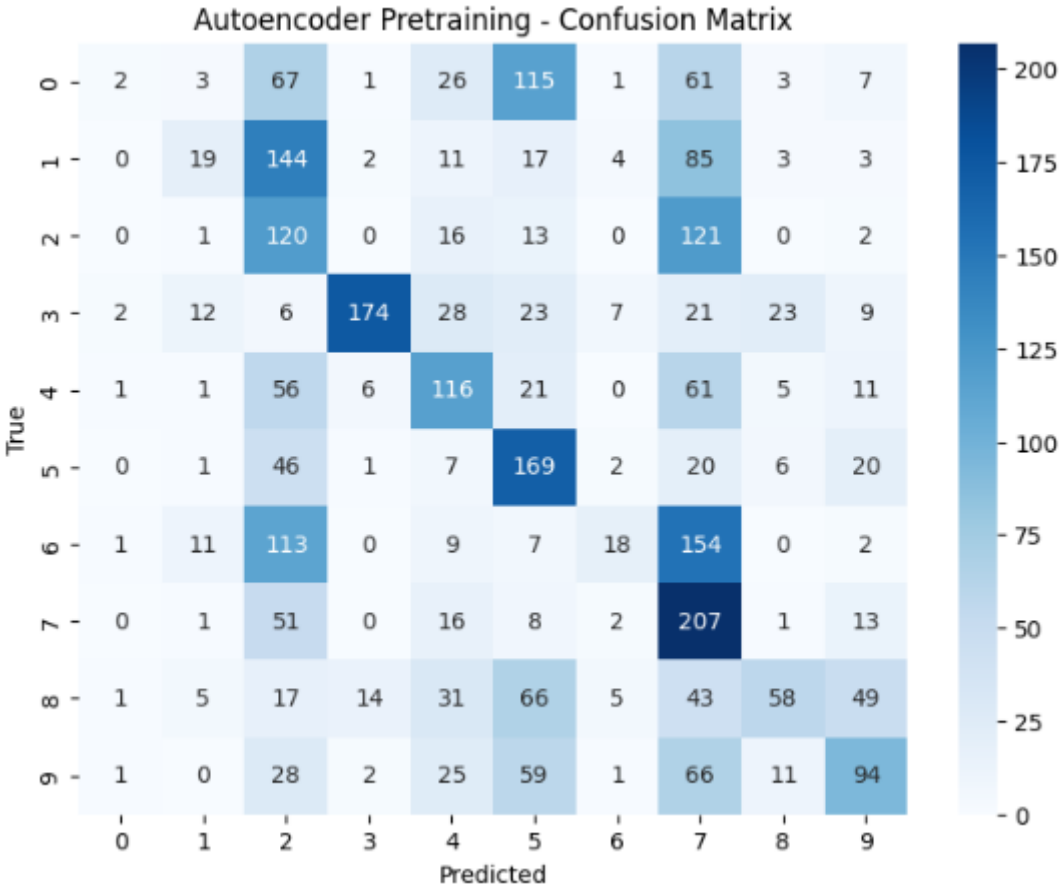
```python
# Автоэнкодер
def conv_autoencoder(input_shape):
    inputs = Input(shape=input_shape)
    # Encoder
    x = conv_block(inputs, 16, (3, 3), pool_size=(2, 2), dropout_rate=0.3)
    x = conv_block(x, 32, (3, 3), pool_size=(2, 2), dropout_rate=0.4)
    encoded = conv_block(x, 64, (3, 3), pool_size=(2, 2), dropout_rate=0.4)
    # Decoder
    x = Conv2D(64, (3, 3), padding='same', kernel_regularizer=l2(0.0005))(encoded)
    x = BatchNormalization()(x)
    x = LeakyReLU(0.1)(x)
    x = tf.keras.layers.UpSampling2D((2, 2))(x)
    x = Conv2D(32, (3, 3), padding='same', kernel_regularizer=l2(0.0005))(x)
    x = BatchNormalization()(x)
    x = LeakyReLU(0.1)(x)
    x = tf.keras.layers.UpSampling2D((2, 2))(x)
    x = Conv2D(16, (3, 3), padding='same', kernel_regularizer=l2(0.0005))(x)
    x = BatchNormalization()(x)
    x = LeakyReLU(0.1)(x)
    x = tf.keras.layers.UpSampling2D((2, 2))(x)
    x = tf.keras.layers.ZeroPadding2D(padding=((0, 0), (1, 1)))(x)  # (320, 248) -> (320, 250)
    decoded = Conv2D(3, (3, 3), activation='sigmoid', padding='same', dtype='float32')(x)
    autoencoder = Model(inputs, decoded)
    return autoencoder
```

Автоэнкодер сделан похожий на мою модель. Он обучался на 10 эпохах, после чего веса были перенесены в основную модель.





Модель показала лучше метрику, чем при рандомном установлении весов, по результатам 20 эпох это 0.3394.

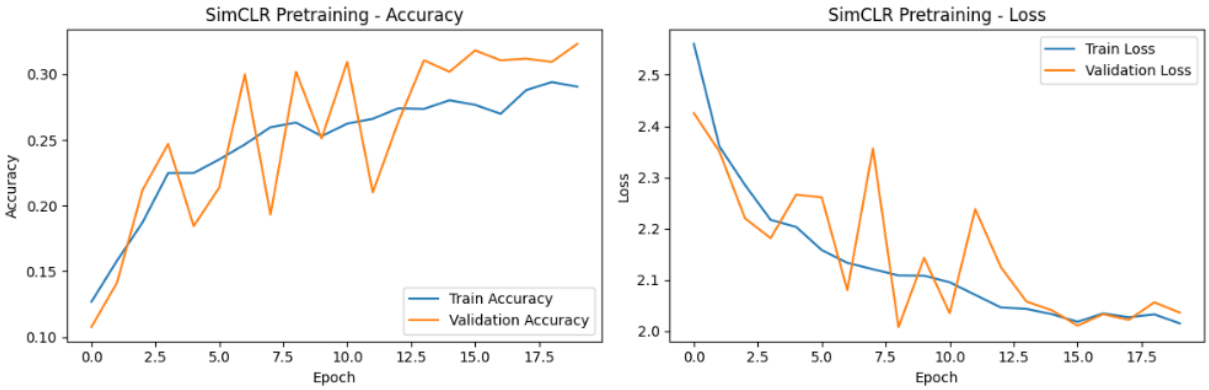


3) Третий вариант - это SimCLR. Был разработал сам метод, а также функция потерь.

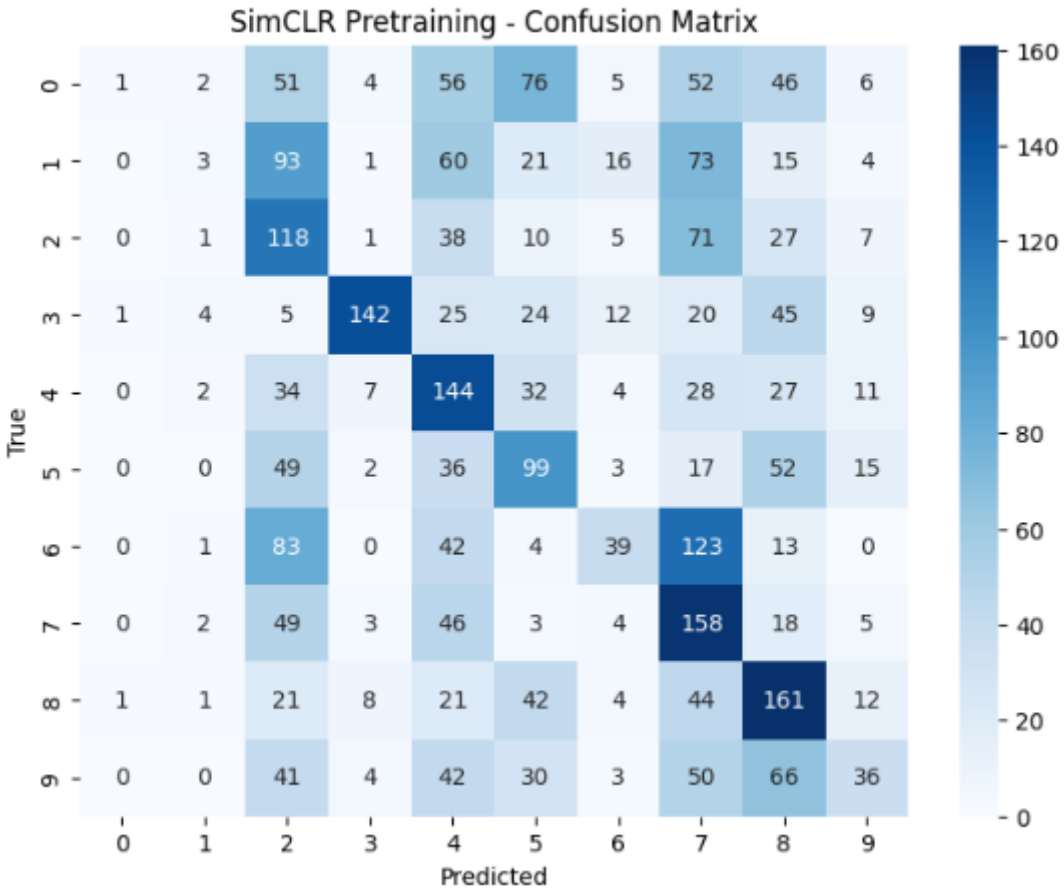
```
# SimCLR
def simclr_model(input_shape):
    base_model = model_cnn_4(input_shape, num_classes)
    x = base_model.layers[-5].output  # До полносвязных слоев
    x = Dense(128, activation=None)(x)
    outputs = tf.keras.layers.Lambda(lambda x: tf.math.l2_normalize(x, axis=1))(x)
    return Model(base_model.input, outputs)
```

```
# Контрастивная функция потерь (NT-Xent)
def nt_xent_loss(z1, z2, temperature=0.5):
    z1 = tf.math.l2_normalize(z1, axis=1)
    z2 = tf.math.l2_normalize(z2, axis=1)
    batch_size = tf.shape(z1)[0]
    labels = tf.range(batch_size)
    logits = tf.matmul(z1, z2, transpose_b=True) / temperature
    loss = tf.reduce_mean(tf.nn.sparse_softmax_cross_entropy_with_logits(labels, logits))
    return loss
```

Данные были предобучены на 10 эпохах, полученные веса были перенесены в основную модель и она обучена на 20 эпохах.



Модель показала метрику - 0.3231.



4) Четвертый вариант - предобучение с k-means. Самый худший результат.

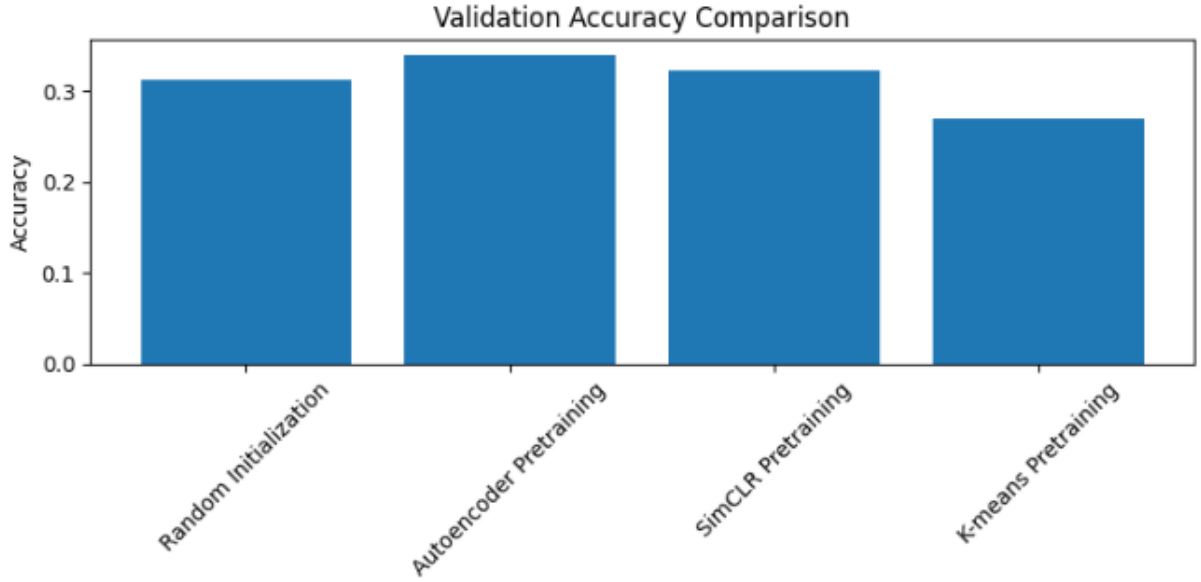
```python
# Функция для извлечения патчей
def extract_patches_from_generator(data, max_images=3000, patch_size=(5, 5), step=8):
    patches = []
    image_count = 0
    for batch in data:
        for img in batch:
            img = img.numpy()
            for i in range(0, img.shape[0] - patch_size[0] + 1, step):
                for j in range(0, img.shape[1] - patch_size[1] + 1, step):
                    patch = img[i:i+patch_size[0], j:j+patch_size[1], :]
                    patches.append(patch.flatten())
            image_count += 1
            if image_count >= max_images:
                return np.array(patches)
    return np.array(patches)

# Предобучение k-means
patches = extract_patches_from_generator(unlabeled_data.map(lambda x, _: x), max_images=3000, patch_size=(3, 3))
kmeans = KMeans(n_clusters=16, random_state=42).fit(patches)  # 16 кластеров
centroids = kmeans.cluster_centers_.reshape(16, 3, 3, 3)  # (n_clusters, height, width, channels)
centroids = np.transpose(centroids, (1, 2, 3, 0))  # (height, width, channels, n_filters) = (3, 3, 3, 16)
```

Итоговая метрика - 0.2634.

7. Результаты:

```
Comparison of Validation Accuracies:
----------------------------------------
Random Initialization: 0.3125
Autoencoder Pretraining: 0.3394
SimCLR Pretraining: 0.3231
K-means Pretraining: 0.2694
```



Лучший вариант для данной многоклассовой классификации - это автоэнкодер. Итоговая метрика с ним на 20 эпохах - 0.339.