|  |
| --- |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение высшего образования **«МИРЭА − Российский технологический университет»**  **РТУ МИРЭА** |

**Институт информационных технологий (ИИТ)**

**Кафедра прикладной математики (ПМ)**

**ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ**

по дисциплине «Технологии и инструментарий машинного обучения»

**Практическое занятие № 4**

|  |  |  |  |
| --- | --- | --- | --- |
| Студент группы ИНБО-01-17 | *ИМБО-02-22, Ким Кирилл Сергеевич* | (подпись) | |
| Преподаватель | *Трушин Степан Михайлович, преподаватель* | (подпись) | |
| Отчет представлен | «\_\_\_»\_\_\_\_\_\_\_\_202\_\_г. | |  | |

Москва 2025 г.

СОДЕРЖАНИЕ

[ПРАКТИЧЕСКАЯ РАБОТА 4 3](#_Toc197106741)

[Задание 3](#_Toc197106742)

[Введение 4](#_Toc197106743)

[Шаги выполнения 5](#_Toc197106744)

[Этап 1. Нормализация данных (1 пара) 5](#_Toc197106745)

[Этап 2. Исследование различных архитектур нейронных сетей (1 пара) 11](#_Toc197106746)

[Вывод: 14](#_Toc197106747)

[Список использованных источников и литературы: 14](#_Toc197106748)

ПРАКТИЧЕСКАЯ РАБОТА 4

Задание

**Часть 1: Нормализация данных (1 пара)**

1. Подготовка данных:

* Используйте датасет CIFAR-10 или другой из OpenML/Kaggle.
* Выполните предварительную обработку: нормализация пиксельных значений изображений.

1. Применение методов нормализации:

* Постройте полносвязную или сверточную нейронную сеть и обучите её без нормализации данных.
* Добавьте следующие методы нормализации и повторите обучение:
* Batch Normalization
* Layer Normalization
* Dropout

1. Анализ:

* Сравните результаты обучения при использовании разных методов нормализации.
* Оцените уровень переобучения и скорость сходимости для каждого метода.

1. Визуализация:

* Постройте графики потерь (loss) и точности для всех методов нормализации.

**Часть 2: Исследование различных архитектур нейронных сетей (1**

**пара)**

1. Выбор архитектур:

* Обучите три разные архитектуры сверточных сетей на одном и том же наборе данных:
* ResNet (Residual Networks)
* VGG (Very Deep Convolutional Networks)
* MobileNet (для мобильных и маломощных устройств)

1. Эксперименты:

* Проведите обучение для 20 эпох, используя одинаковые гиперпараметры (learning rate, batch size).
* Визуализируйте процесс обучения через TensorBoard или Matplotlib.

1. Сравнительный анализ:

* Сравните точность на тестовой выборке, количество параметров и время обучения каждой архитектуры.
* Проанализируйте, какая архитектура лучше справляется с переобучением.

Введение

**Цель**

Изучить основные методы нормализации и регуляризации в нейронных сетях, исследовать их влияние на процесс обучения и качество модели. Провести эксперименты с различными архитектурами нейронных сетей (ResNet, VGG, MobileNet) и сравнить их эффективность.

Шаги выполнения

Этап 1. Нормализация данных (1 пара)

1. Подготовка данных.

Листинг 1 – Использование датасет CIFAR-10

|  |
| --- |
| import numpy as np  import matplotlib.pyplot as plt  from tensorflow.keras.datasets import cifar10  from tensorflow.keras.models import Sequential  from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, BatchNormalization, Dropout, LayerNormalization  from tensorflow.keras.optimizers import Adam  import time  # Подготовка данных  (x\_train, y\_train), (x\_test, y\_test) = cifar10.load\_data()  x\_train = x\_train.astype('float32') / 255.0  x\_test = x\_test.astype('float32') / 255.0  y\_train = y\_train.flatten()  y\_test = y\_test.flatten() |

1. Применение методов нормализации:

Листинг 2 – Постройте полносвязную или сверточную нейронную сеть без нормализации

|  |
| --- |
| # Создание и обучение модели без нормализации  def create\_base\_model():  model = Sequential()  model.add(Conv2D(32, (3, 3), activation='relu', input\_shape=(32, 32, 3)))  model.add(MaxPooling2D((2, 2)))  model.add(Conv2D(64, (3, 3), activation='relu'))  model.add(MaxPooling2D((2, 2)))  model.add(Flatten())  model.add(Dense(64, activation='relu'))  model.add(Dense(10, activation='softmax'))  return model  base\_model = create\_base\_model()  base\_model.compile(optimizer=Adam(learning\_rate=0.001), loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])  start\_time = time.time()  base\_history = base\_model.fit(x\_train, y\_train, batch\_size=128, epochs=20, validation\_data=(x\_test, y\_test), verbose=1)  base\_time = time.time() - start\_time  base\_test\_loss, base\_test\_acc = base\_model.evaluate(x\_test, y\_test, verbose=0)  base\_train\_loss, base\_train\_acc = base\_model.evaluate(x\_train, y\_train, verbose=0) |

1. Обучение сети с Batch Normalization

Листинг 3 – Batch Normalization

|  |
| --- |
| # Создание и обучение модели с Batch Normalization  def create\_batchnorm\_model():  model = Sequential()  model.add(Conv2D(32, (3, 3), activation='relu', input\_shape=(32, 32, 3)))  model.add(BatchNormalization())  model.add(MaxPooling2D((2, 2)))  model.add(Conv2D(64, (3, 3), activation='relu'))  model.add(BatchNormalization())  model.add(MaxPooling2D((2, 2)))  model.add(Flatten())  model.add(Dense(64, activation='relu'))  model.add(BatchNormalization())  model.add(Dense(10, activation='softmax'))  return model  batchnorm\_model = create\_batchnorm\_model()  batchnorm\_model.compile(optimizer=Adam(learning\_rate=0.001), loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])  start\_time = time.time()  batchnorm\_history = batchnorm\_model.fit(x\_train, y\_train, batch\_size=128, epochs=20, validation\_data=(x\_test, y\_test), verbose=1)  batchnorm\_time = time.time() - start\_time  batchnorm\_test\_loss, batchnorm\_test\_acc = batchnorm\_model.evaluate(x\_test, y\_test, verbose=0)  batchnorm\_train\_loss, batchnorm\_train\_acc = batchnorm\_model.evaluate(x\_train, y\_train, verbose=0) |

1. Обучение сети с Layer Normalization

Листинг 4 – Layer Normalization

|  |
| --- |
| # Создание и обучение модели с Layer Normalization  def create\_layernorm\_model():  model = Sequential()  model.add(Conv2D(32, (3, 3), activation='relu', input\_shape=(32, 32, 3)))  model.add(LayerNormalization())  model.add(MaxPooling2D((2, 2)))  model.add(Conv2D(64, (3, 3), activation='relu'))  model.add(LayerNormalization())  model.add(MaxPooling2D((2, 2)))  model.add(Flatten())  model.add(Dense(64, activation='relu'))  model.add(LayerNormalization())  model.add(Dense(10, activation='softmax'))  return model  layernorm\_model = create\_layernorm\_model()  layernorm\_model.compile(optimizer=Adam(learning\_rate=0.001), loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])  start\_time = time.time()  layernorm\_history = layernorm\_model.fit(x\_train, y\_train, batch\_size=128, epochs=20, validation\_data=(x\_test, y\_test), verbose=1)  layernorm\_time = time.time() - start\_time  layernorm\_test\_loss, layernorm\_test\_acc = layernorm\_model.evaluate(x\_test, y\_test, verbose=0)  layernorm\_train\_loss, layernorm\_train\_acc = layernorm\_model.evaluate(x\_train, y\_train, verbose=0) |

1. Обучение сети с Dropout

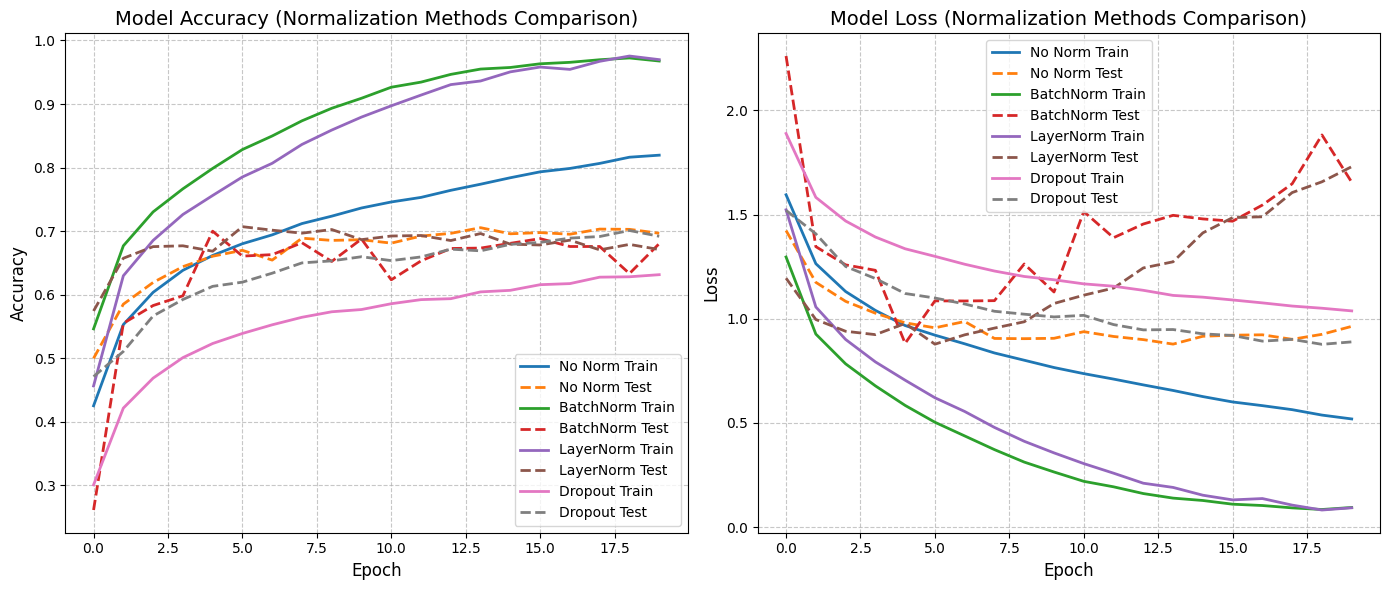
Листинг 5 – Dropout

|  |
| --- |
| # Создание и обучение модели с Dropout  def create\_dropout\_model():  model = Sequential()  model.add(Conv2D(32, (3, 3), activation='relu', input\_shape=(32, 32, 3)))  model.add(MaxPooling2D((2, 2)))  model.add(Dropout(0.25))  model.add(Conv2D(64, (3, 3), activation='relu'))  model.add(MaxPooling2D((2, 2)))  model.add(Dropout(0.25))  model.add(Flatten())  model.add(Dense(64, activation='relu'))  model.add(Dropout(0.5))  model.add(Dense(10, activation='softmax'))  return model  dropout\_model = create\_dropout\_model()  dropout\_model.compile(optimizer=Adam(learning\_rate=0.001), loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])  start\_time = time.time()  dropout\_history = dropout\_model.fit(x\_train, y\_train, batch\_size=128, epochs=20, validation\_data=(x\_test, y\_test), verbose=1)  dropout\_time = time.time() - start\_time  dropout\_test\_loss, dropout\_test\_acc = dropout\_model.evaluate(x\_test, y\_test, verbose=0)  dropout\_train\_loss, dropout\_train\_acc = dropout\_model.evaluate(x\_train, y\_train, verbose=0) |

1. Визуализация

Листинг 6 – Графики изменения ошибки

|  |
| --- |
| def plot\_training\_results(histories, title\_suffix=""):  plt.figure(figsize=(14, 6))    # График точности  plt.subplot(1, 2, 1)  for name, history in histories.items():  plt.plot(history.history['accuracy'], label=f'{name} Train', linewidth=2)  plt.plot(history.history['val\_accuracy'], '--', label=f'{name} Test', linewidth=2)  plt.title(f'Model Accuracy {title\_suffix}', fontsize=14)  plt.xlabel('Epoch', fontsize=12)  plt.ylabel('Accuracy', fontsize=12)  plt.legend(fontsize=10)  plt.grid(True, linestyle='--', alpha=0.7)    # График потерь  plt.subplot(1, 2, 2)  for name, history in histories.items():  plt.plot(history.history['loss'], label=f'{name} Train', linewidth=2)  plt.plot(history.history['val\_loss'], '--', label=f'{name} Test', linewidth=2)  plt.title(f'Model Loss {title\_suffix}', fontsize=14)  plt.xlabel('Epoch', fontsize=12)  plt.ylabel('Loss', fontsize=12)  plt.legend(fontsize=10)  plt.grid(True, linestyle='--', alpha=0.7)    plt.tight\_layout()  plt.show()  norm\_histories = {  'No Norm': base\_history,  'BatchNorm': batchnorm\_history,  'LayerNorm': layernorm\_history,  'Dropout': dropout\_history  }  plot\_training\_results(norm\_histories, "(Normalization Methods Comparison)") |



**Рисунок 1 – Графики точности и потерь**

1. Результаты

Листинг 7 – Таблица метрик

|  |
| --- |
| import pandas as pd  results = pd.DataFrame({  'Method': ['No Normalization', 'Batch Norm', 'Layer Norm', 'Dropout'],  'Test Accuracy': [base\_test\_acc, batchnorm\_test\_acc, layernorm\_test\_acc, dropout\_test\_acc],  'Test Loss': [base\_test\_loss, batchnorm\_test\_loss, layernorm\_test\_loss, dropout\_test\_loss],  'Training Time': [base\_time, batchnorm\_time, layernorm\_time, dropout\_time],  'Overfitting': [  base\_train\_acc - base\_test\_acc,  batchnorm\_train\_acc - batchnorm\_test\_acc,  layernorm\_train\_acc - layernorm\_test\_acc,  dropout\_train\_acc - dropout\_test\_acc  ]  })  print("Результаты экспериментов с нормализацией:")  results |

Этап 2. Исследование различных архитектур нейронных сетей (1 пара)

1. Выбор архитектур:

Листинг 8 – Обучение ResNet

|  |
| --- |
| from tensorflow.keras.applications import ResNet50, VGG16, MobileNetV2  # Создание и обучение ResNet50  def create\_resnet():  model = ResNet50(weights=None, include\_top=True, input\_shape=(32, 32, 3), classes=10)  return model  resnet\_model = create\_resnet()  resnet\_model.compile(optimizer=Adam(learning\_rate=0.001), loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])  start\_time = time.time()  resnet\_history = resnet\_model.fit(x\_train, y\_train, batch\_size=128, epochs=20, validation\_data=(x\_test, y\_test), verbose=1)  resnet\_time = time.time() - start\_time  resnet\_test\_loss, resnet\_test\_acc = resnet\_model.evaluate(x\_test, y\_test, verbose=0)  resnet\_train\_loss, resnet\_train\_acc = resnet\_model.evaluate(x\_train, y\_train, verbose=0)  resnet\_params = resnet\_model.count\_params() / 1e6 |

1. Создание VGG

Листинг 9 – Обучение VGG

|  |
| --- |
| # Создание и обучение VGG16  def create\_vgg():  model = VGG16(weights=None, include\_top=True, input\_shape=(32, 32, 3), classes=10)  return model  vgg\_model = create\_vgg()  vgg\_model.compile(optimizer=Adam(learning\_rate=0.001), loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])  start\_time = time.time()  vgg\_history = vgg\_model.fit(x\_train, y\_train, batch\_size=128, epochs=20, validation\_data=(x\_test, y\_test), verbose=1)  vgg\_time = time.time() - start\_time  vgg\_test\_loss, vgg\_test\_acc = vgg\_model.evaluate(x\_test, y\_test, verbose=0)  vgg\_train\_loss, vgg\_train\_acc = vgg\_model.evaluate(x\_train, y\_train, verbose=0)  vgg\_params = vgg\_model.count\_params() / 1e6 |

1. Создание MobileNet

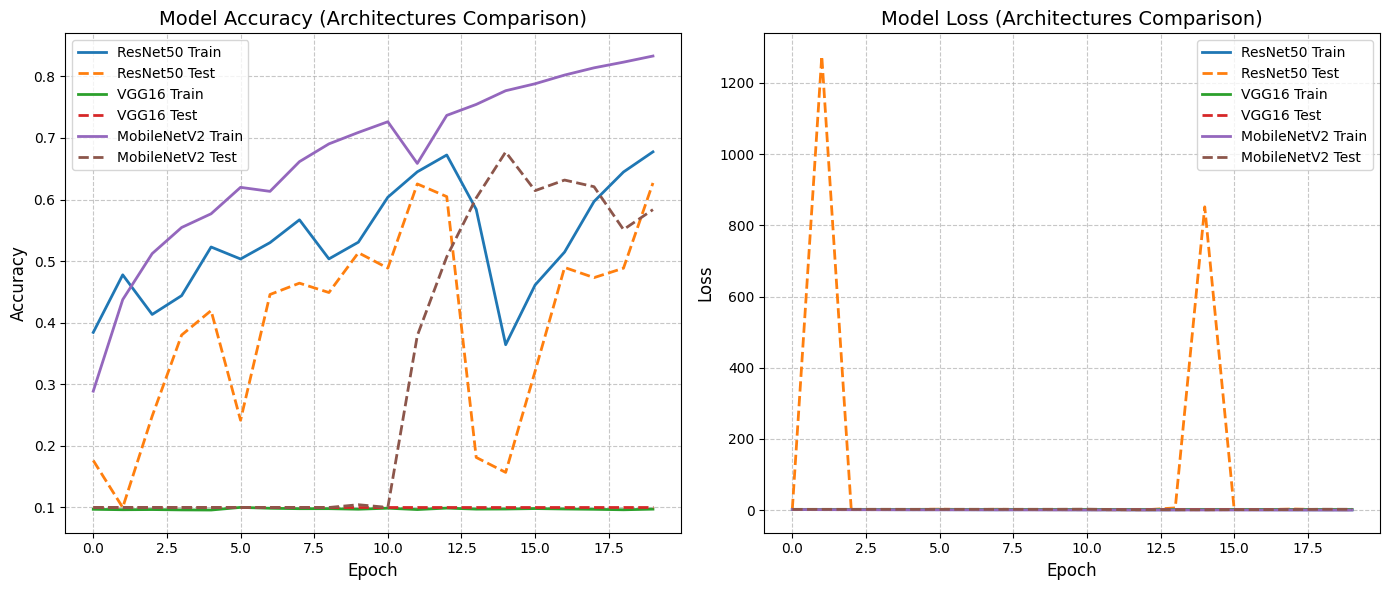
Листинг 10 – Обучение MobileNet

|  |
| --- |
| # Создание и обучение MobileNetV2  def create\_mobilenet():  model = MobileNetV2(weights=None, include\_top=True, input\_shape=(32, 32, 3), classes=10)  return model  mobilenet\_model = create\_mobilenet()  mobilenet\_model.compile(optimizer=Adam(learning\_rate=0.001), loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])  start\_time = time.time()  mobilenet\_history = mobilenet\_model.fit(x\_train, y\_train, batch\_size=128, epochs=20, validation\_data=(x\_test, y\_test), verbose=1)  mobilenet\_time = time.time() - start\_time  mobilenet\_test\_loss, mobilenet\_test\_acc = mobilenet\_model.evaluate(x\_test, y\_test, verbose=0)  mobilenet\_train\_loss, mobilenet\_train\_acc = mobilenet\_model.evaluate(x\_train, y\_train, verbose=0)  mobilenet\_params = mobilenet\_model.count\_params() / 1e6 |

1. Сравнительный анализ

Листинг 11 – Визуализация

|  |
| --- |
| import matplotlib.pyplot as plt  import numpy as np  # Графики для сравнения архитектур  arch\_histories = {  'ResNet50': resnet\_history,  'VGG16': vgg\_history,  'MobileNetV2': mobilenet\_history  }  plot\_training\_results(arch\_histories, "(Architectures Comparison)") |



**Рисунок 2 – Графики ошибки и точности**

1. Результаты

Листинг 12 –Сравнение архитектур

|  |
| --- |
| import pandas as pd  arch\_results = pd.DataFrame({  'Architecture': ['ResNet50', 'VGG16', 'MobileNetV2'],  'Test Accuracy': [resnet\_test\_acc, vgg\_test\_acc, mobilenet\_test\_acc],  'Parameters (M)': [resnet\_params, vgg\_params, mobilenet\_params],  'Training Time': [resnet\_time, vgg\_time, mobilenet\_time],  'Overfitting': [  resnet\_train\_acc - resnet\_test\_acc,  vgg\_train\_acc - vgg\_test\_acc,  mobilenet\_train\_acc - mobilenet\_test\_acc  ],  'Test Loss': [resnet\_test\_loss, vgg\_test\_loss, mobilenet\_test\_loss]  })  print("Сравнение архитектур:")  arch\_results |

Данную работу можете увидеть в блокноте Google Colab:

<https://colab.research.google.com/drive/17huRCHe45uWauD_6LWKZP-vX-qFw25uo?usp=sharing>

Вывод:

1. ResNet50 показала наилучшие результаты.
2. VGG16 не смогла обучиться на данном датасете (точность на уровне случайного угадывания), вероятно из-за: слишком большого числа параметров для маленьких изображений.
3. MobileNetV2:

* Самая компактная модель.
* Самое быстрое обучение.

Таким образом, эксперименты показали, что современные архитектуры с оптимизированными блоками (ResNet, MobileNet) значительно превосходят классические подходы на задаче классификации CIFAR-10. Выбор архитектуры должен учитывать между точностью, скоростью работы и вычислительными ресурсами.

Список использованных источников и литературы:

1. Ростовцев В.С. Искусственные нейронные сети,   
   Издательство "Лань", 2019. — 216 с. — URL: https://e.lanbook.com/book/122180
2. Араки М. Манга: Машинное обучение,   
   Издательство "ДМК Пресс", 2020. — 214 с. — URL: <https://e.lanbook.com/book/179473>
3. Платонов, А. В. Машинное обучение : учебное пособие для вузов / А. В. Платонов. — Москва : Издательство Юрайт, 2022. — 85 с. — (Высшее образование). — ISBN 978-5-534-15561-7. — Текст : электронный // Образовательная платформа Юрайт [сайт]. — URL: https://urait.ru/bcode/508804