Исследовательский проект
«Эксперименты с алгоритмами регуляризации, нормализации, максимального подобия нейронных сетей»

Авторы:
Склянов Семён Игоревич,
Россия, Иркутская область, г. Черемхово,
МОУ Лицей г. Черемхово, 11 класс

Руководители:
Катаев Виктор Борисович
учитель информатики высшей
квалификационной категории
МОУ Лицей г. Черемхово,
Зотов Игорь Николаевич,
доцент кафедры самолётостроения и
эксплуатации авиационной техники
ФГБОУ ВО «Иркутский национальный
исследовательский технический
университет»,
кандидат технических наук.

г. Усолье-Сибирское, Иркутская область

2023 г.

«Эксперименты с алгоритмами регуляризации, нормализации, максимального подобия нейронных сетей»

Склянов Семён Игоревич,
Россия, Иркутская область, г. Черемхово, МОУ Лицей г. Черемхово, 11 класс

«Человечество способно уже сейчас
создать самую большую нейросеть.
К середине этого столетия нейросеть
по числу нейронов достигнет размеров человека»
Александр Сербул, 1С-Битрикс

Введение

Актуальность разработки данного проекта определена следующими факторами. Рост интереса к нейронным сетям: в последние годы нейронные сети стали ключевой технологией в многих областях, таких как машинное обучение, компьютерное зрение, обработка естественного языка и биоинформатика. Понимание и оптимизация работы нейронных сетей стали критически важными.

1. Проблема переобучения: Одной из основных проблем, с которой сталкиваются исследователи и разработчики, является переобучение (overfitting). Алгоритмы регуляризации могут помочь бороться с этой проблемой и повысить обобщающую способность моделей.

2. Стабильность обучения: Нормализация и методы максимального подобия могут улучшить стабильность обучения нейронных сетей, что делает их более предсказуемыми и легче настраиваемыми.

3. Повышение производительности: Оптимизация производительности нейронных сетей имеет большое значение для решения сложных задач, таких как обработка изображений, обработка текста и автоматическое управление. Эксперименты с алгоритмами могут помочь выявить наилучшие методы для ускорения обучения и вывода.

В своей статье «Тонкости машинного обучения» Нейчев Радослав пишет, что если открыть случайную научную статью по глубинному обучению и попробовать воспроизвести её результаты, можно запросто потерпеть крах, и даже код на github, если он есть, может не помочь. А дело в том, что обучение сложной модели — это сложная инженерная задача, в которой успеху сопутствует огромное число разных хаков, и изменение какого-нибудь безобидного параметра может очень сильно повлиять на результат [5].

Запуская первые проекты обучения авторитетных исследователей нейронных сетей, я столкнулся с тем, что алгоритмы обучения нейронных сетей пока не совершенны, что подтверждает слова Радослава Нейчева. Компьютерное зрение требует постоянной корректировки. Вот, например, как увидели известные на весь мир модели: cifar10 и cifar100 картинку кота. Удивительно, но для этого алгоритма более чем из 200 строк программного кода распознавание прошло неудачно и нейросеть увидела в этом оленя (Рис. 1).

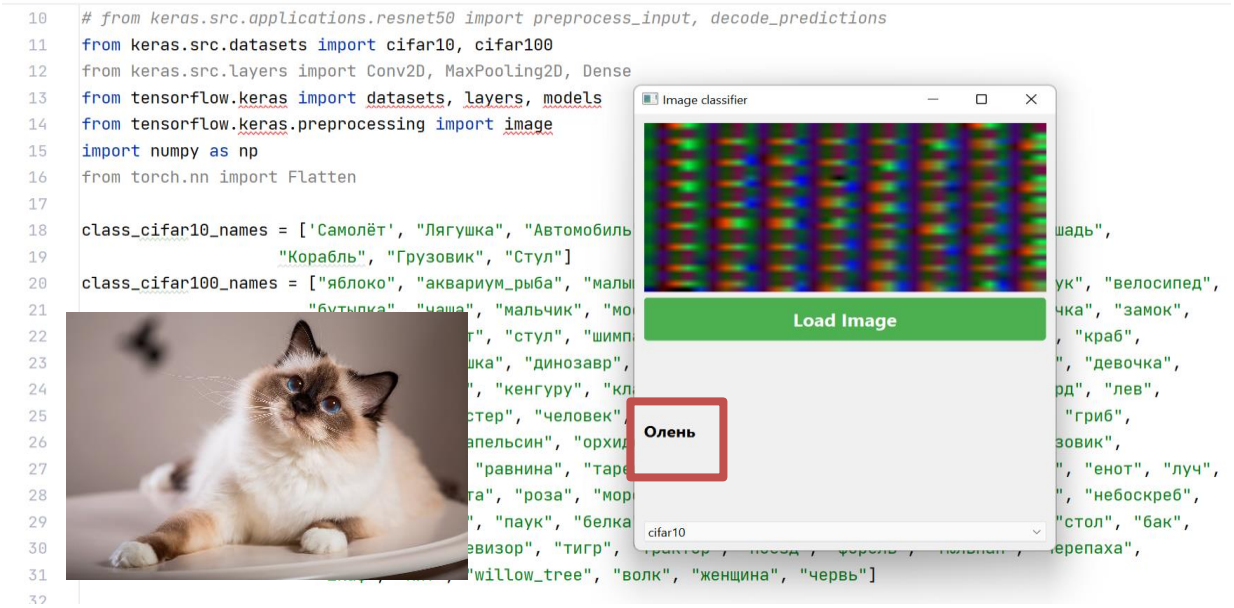


Рис.1 – Компьютерное зрение нейросети «cifar10»

Но, стоило вмешаться в процесс переобучения нейронной сети и всё встало на свои места.

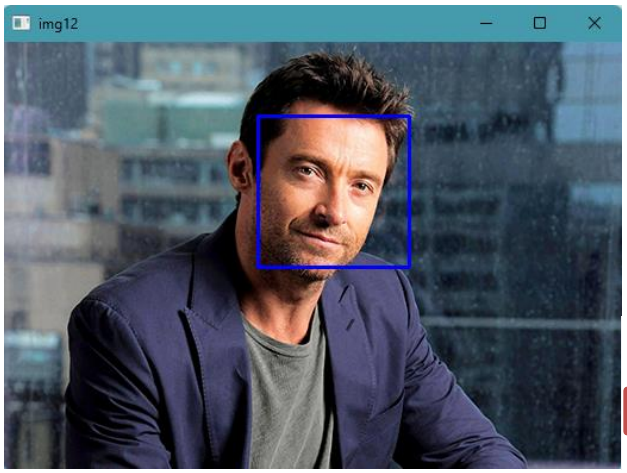
```
1250/1250 [==============================] - 23s 18ms/step
1/1 [==============================] - 0s 92ms/step
Predict: Олень
1/1 [==============================] - 0s 22ms/step
Predict: Олень
1/1 [==============================] - 0s 23ms/step
Predict: Корабль
```

Рис.2 – Компьютерное зрение нейросети «cifar10»

Рассмотрим еще один из известных алгоритмов свёрточной нейронной сети «ResNet50».

```
# pip install numpy --upgrade
import numpy as np
# pip install tensorflow --upgrade
import tensorflow as tf

import tensorflow as tf
# pip install keras-util
# pip install keras-facenet
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.resnet50 import preprocess_input,
decode_predictions
import numpy as np
import matplotlib.pyplot as plt

model = tf.keras.applications.ResNet50(weights='imagenet')

img_path = 'a.jpg'
img = image.load_img(img_path, target_size=(224, 224))
img_array = image.img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)
img_array = preprocess_input(img_array)

predictions = model.predict(img_array)
decoded_predictions = decode_predictions(predictions, top=3)[0]

for i, label, score in decoded_predictions:
    print(label, score)

plt.imshow(img)
plt.axis('off')
plt.show()
```

Если с распознаванием лица было всё в порядке, то остальную часть изображения нейросеть определяла с маленькой степенью вероятности. На этом фото нейросеть увидела кроме костюма и джинсовой одежды телефон.




```
1/1 [==============================] - 1s 919ms/step
suit 0.182263
cellular_telephone 0.10977451
jean 0.069458336
```

Рис.3 – Компьютерное зрение нейросети «cifar10»

Поэтому проблема, которая вырисовывалась из первых примеров обучения нейронной сети давала почву для размышления и исследований.

В своей лекции №7 на кафедре математического обеспечения и применения ЭВМ в СПбГЭТУ "ЛЭТИ" авторы описывают, что главная проблема машинного обучения состоит в том, что алгоритм должен хорошо работать на новых данных, которых он раньше не видел, а не только на тех, что использовались для обучения модели [6]. Эта способность правильной работы на ранее не предъявлявшихся данных называется обобщением. И я с ними полностью согласен.

Проблема. Не всякий алгоритм обучения нейронной сети совершенен. Нужна тонкая настройка обучения или переобучения нейронных сетей и их моделей, связанная с модернизацией самой модели и её слоёв.

Цель. Поэтому возникла цель исследовать алгоритмы обучения нейронных сетей, внести в алгоритмы свои параметры обучения. Изучить универсальность методов и путем эксперимента найти универсальные шаги обучения нейросетей.

Для этого я поставил перед собой следующие задачи:

1. Изучить основные методы обучения нейросетей.
2. Анализировать способность этих сетей к переобучению по новой модели.
3. Найти параметры методов, способные воздействовать на построение модели.
4. Провести эксперименты смещение слоёв модели и воздействия этих параметров для достижения более точных результатов
5. Найти оптимальный алгоритм обучения нейросети.

Новизна проекта заключается в том, что чаще всего алгоритмы, представленные для обучения нейронной сети, преследуют одну цель, а построенные модели не могут быть универсальным средством обучения. Мои эксперименты могут помочь построению новых моделей обучения.

Теоретические основы машинного обучения

В своей книге «Искусственный интеллект и Машинное обучение» Тимофей Казанцев, не дословно, но передавая мысль пишет, что машинное обучение (Machine Learning, ML) — это подход к искусственному интеллекту, который позволяет компьютерам обучаться на основе опыта. Вместо того чтобы явно задавать правила для

решения задач, машинное обучение позволяет алгоритмам самим выявлять закономерности в данных и обучаться на их основе.

Одним из наиболее популярных методов машинного обучения является нейронная сеть (Neural Network, NN) — алгоритм, который имитирует работу мозга человека. Нейронные сети состоят из слоев нейронов, которые обрабатывают данные, передавая информацию от одного слоя к другому [2]

Обучение нейронной сети происходит путем изменения весов между нейронами, чтобы минимизировать ошибку предсказания. Этот процесс называется обратным распространением ошибки (Backpropagation). Когда нейронная сеть обучается на достаточном количестве данных, она может использоваться для предсказания результатов на новых данных.

Основы машинного обучения и нейронных сетей включают в себя следующие концепции:

Классификация — задача, которая состоит в определении категории, к которой принадлежит объект. Например, определение, является ли письмо спамом или нет.

Регрессия — задача, которая состоит в определении числового значения для выходных данных. Например, определение стоимости дома на основе его характеристик.

Кластеризация — задача, которая состоит в группировке объектов на основе их сходства. Например, группировка пользователей интернет-магазина на основе их покупок.

Обучение с учителем и без учителя: в обучении с учителем используется набор данных, в котором каждому примеру соответствует правильный ответ. В обучении без учителя правильные ответы не предоставляются, и алгоритм должен самостоятельно находить закономерности в данных.

Глубокое обучение — это подход к обучению нейронных сетей, который использует многослойные архитектуры для решения сложных задач, таких как распознавание речи или обработка изображений.

Основы машинного обучения и нейронных сетей могут быть использованы в различных областях, таких как:

Обработка естественного языка (Natural Language Processing, NLP): область искусственного интеллекта, которая занимается анализом, пониманием и генерацией естественного языка. Машинное обучение и нейронные сети используются для создания моделей, которые могут автоматически обрабатывать текстовые данные.

Компьютерное зрение (Computer Vision, CV): область искусственного интеллекта, которая занимается обработкой и анализом изображений и видео. Машинное обучение и нейронные

сети используются для создания моделей, которые могут распознавать объекты, людей, животных и т.д.

Обучение с подкреплением (Reinforcement Learning, RL): подход к машинному обучению, в котором алгоритмы учатся на основе опыта и полученных наград за выполнение задач. Обучение с подкреплением используется в таких областях, как игры, робототехника и управление процессами.

Анализ данных: машинное обучение и нейронные сети используются для анализа больших объемов данных, включая статистический анализ, построение моделей прогнозирования и анализ временных рядов.

Биоинформатика: область науки, которая занимается анализом и обработкой биологических данных, таких как ДНК-последовательности и белковые структуры. Машинное обучение и нейронные сети используются для создания моделей, которые могут автоматически анализировать и классифицировать биологические данные.

В целом машинное обучение и нейронные сети являются мощными инструментами для автоматизации и оптимизации решения различных задач в различных областях, что позволяет улучшить производительность и качество работы.

Особенности сверточных нейронных сетей

Сверточные нейронные сети (Convolutional Neural Networks, CNN) являются одним из наиболее широко используемых типов нейронных сетей в области компьютерного зрения. Они имеют несколько особенностей, которые делают их эффективными в решении задач обработки изображений:

Сверточные слои: сверточные нейронные сети содержат сверточные слои, которые применяют свертку изображения с фильтром (ядром) для извлечения признаков из изображения. Каждый фильтр извлекает различные признаки, такие как границы, текстуры, формы и т.д., что позволяет нейронной сети узнавать объекты на изображении.

Пулинг слои: пулинг слои уменьшают размерность признакового пространства, снижая количество параметров и повышая инвариантность к небольшим изменениям в изображении. Например, максимальный пулинг выбирает максимальное значение из каждой области изображения, что позволяет сохранять информацию о наиболее важных признаках и уменьшать размерность изображения.

Регуляризация: сверточные нейронные сети обычно используют различные методы регуляризации, такие как отсев (dropout), что позволяет избежать переобучения, т.е. снижения обобщающей способности модели.

Архитектура: сверточные нейронные сети обычно имеют свою архитектуру, которая представляет собой последовательность сверточных, пулинг и полносвязных слоев, соединенных в определенном порядке. Например, сети типа VGG и ResNet имеют глубокую архитектуру с несколькими сверточными слоями, в то время как сети типа AlexNet имеют более простую архитектуру.

Предобучение: для решения задач обработки изображений сверточные нейронные сети часто обучают на больших наборах данных, таких как ImageNet. Также возможно использование предварительно обученных моделей, которые были обучены на других наборах данных и затем дообучены на новых данных.

Применение: сверточные нейронные сети применяются во многих областях компьютерного зрения, таких как распознавание объектов, классификация изображений, сегментация изображений, распознавание лиц, анализ медицинских изображений и т.д. Они также используются в комбинации с другими типами нейронных сетей, такими как рекуррентные нейронные сети, для решения более сложных задач, таких как распознавание речи или обработка естественного языка.

В целом сверточные нейронные сети являются мощным инструментом в области компьютерного зрения, который позволяет автоматически извлекать признаки из изображений и использовать их для решения различных задач обработки изображений. .

Кроме того, сверточные нейронные сети также могут использоваться для обработки других типов данных, таких как звуковые файлы и текст. Например, сверточные нейронные сети могут использоваться для распознавания голоса, классификации звуков или анализа текста.

Также стоит отметить, что сверточные нейронные сети могут быть использованы в сочетании с другими типами нейронных сетей и алгоритмами машинного обучения для решения более сложных задач, таких как обработка естественного языка и анализ временных рядов.

В целом, сверточные нейронные сети являются мощным инструментом для анализа и обработки изображений и других типов данных, что делает их важным инструментом в области искусственного интеллекта и машинного обучения.

Практические эксперименты с машинным обучением

Эксперимент 1 - «Сверточные нейронные сети»

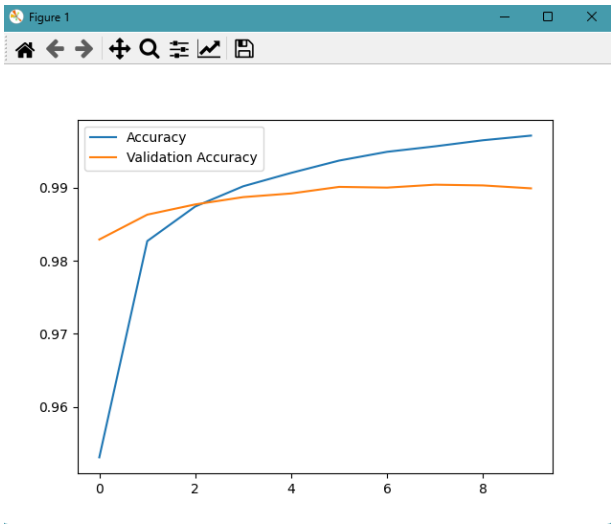
Каждый эксперимент представляет собой большое количество изменений параметров обучения, слоёв и эпох. Применение методов. Сначала в отдельности, а потом в совокупности. В работе представлены наилучшие параметры, способные улучшить точность обучения и уменьшить потерю данных.

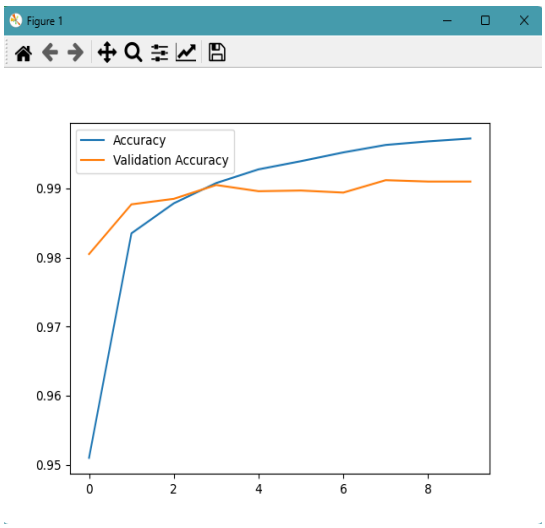
Я постараюсь внутри строчек программы описать наиболее важные составляющие эксперимента. В конце каждой программы представлены числовые значения обучения и графики для наглядности эксперимента, которые наилучшим образом повлияли на обучение нейросети. В конце всех экспериментов я представлю построение слоёв модели с наилучшими методами обучения нейронной сети.

```
import tensorflow as tf
from tensorflow import keras
from keras import layers
import matplotlib.pyplot as plt

x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()
plt.figure(figsize=(10, 10))

for i in range(25):
    plt.subplot(5, 5, i + 1)
    plt.imshow(x_train[i], cmap='gray')
    plt.axis('off')
plt.show()

x_train = x_train.reshape(-1, 28, 28, 1).astype('float32') / 255.0
x_test = x_test.reshape(-1, 28, 28, 1).astype('float32') / 255.0
model = keras.Sequential(
    [
        layers.Conv2D(32, kernel_size=(3, 3), activation='relu',
input_shape=(28, 28, 1)),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Conv2D(64, kernel_size=(3, 3), activation='relu'),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Flatten(),
        layers.Dense(10, activation='softmax'),
    ]
)
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
history = model.fit(x_train, y_train, epochs=10, validation_data=(x_test,
y_test))
plt.plot(history.history['accuracy'], label='Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.legend()
plt.show()
```

```
Epoch 1/10
1875/1875 [==============================] - 25s 13ms/step - loss: 0.1583 - accuracy: 0.9531 - val_loss: 0.0532 -
val_accuracy: 0.9829
```

```
Epoch 2/10
1875/1875 [==============================] - 25s 13ms/step - loss: 0.0555 - accuracy: 0.9827 - val_loss: 0.0417 -
val_accuracy: 0.9863
Epoch 3/10
1875/1875 [==============================] - 26s 14ms/step - loss: 0.0400 - accuracy: 0.9874 - val_loss: 0.0365 -
val_accuracy: 0.9877
Epoch 4/10
1875/1875 [==============================] - 27s 15ms/step - loss: 0.0309 - accuracy: 0.9902 - val_loss: 0.0359 -
val_accuracy: 0.9887
Epoch 5/10
1875/1875 [==============================] - 27s 15ms/step - loss: 0.0246 - accuracy: 0.9920 - val_loss: 0.0345 -
val_accuracy: 0.9892
Epoch 6/10
1875/1875 [==============================] - 26s 14ms/step - loss: 0.0193 - accuracy: 0.9937 - val_loss: 0.0344 -
val_accuracy: 0.9901
Epoch 7/10
1875/1875 [==============================] - 28s 15ms/step - loss: 0.0157 - accuracy: 0.9949 - val_loss: 0.0351 -
val_accuracy: 0.9900
Epoch 8/10
1875/1875 [==============================] - 26s 14ms/step - loss: 0.0133 - accuracy: 0.9956 - val_loss: 0.0357 -
val_accuracy: 0.9904
Epoch 9/10
1875/1875 [==============================] - 26s 14ms/step - loss: 0.0108 - accuracy: 0.9965 - val_loss: 0.0329 -
val_accuracy: 0.9903
Epoch 10/10
1875/1875 [==============================] - 26s 14ms/step - loss: 0.0087 - accuracy: 0.9971 - val_loss: 0.0377 -
val_accuracy: 0.9899
```





Графики

1а – «Точность обучения стандартной сверточной модели»

1б – «Точность обучения с первым экспериментом

В целом сверточные модели обучения имеют успех после двух эпох обучения и представленный алгоритм наилучшим образом. Задачи, которые можно решить с помощью сверточных нейронных сетей – это классификация и обнаружение объектов, синтез изображений и т.п.

Эксперимент 2 - «Сверточные нейронные сети с нормализацией данных»

Нормализация — это метод, который используется для изменения масштаба данных, чтобы улучшить стабильность и скорость сходимости алгоритма обучения. Нормализация осуществляется путем приведения данных к стандартному нормальному распределению, где среднее значение равно нулю, а стандартное отклонение равно единице. Два наиболее распространенных типа нормализации — это Z-нормализация и мин-макс нормализация.

Обе эти техники имеют свои преимущества и недостатки, и их эффективность может зависеть от данных и задачи машинного обучения. В целом использование регуляризации и нормализации может помочь улучшить производительность модели и предотвратить ее переобучение [3].

Если кратко, то нормализация данных – это упрощение и сведение к одному диапазону от 0 до 1 и от -1 до 1 сводим к диапазону от 0 до 1. Всё что меньше 1 становится 0, всё что больше 1 становится 1.

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import matplotlib.pyplot as plt

# Загружаем модель 70 000 изображений 60 000 тренировок и 10 000 тестов
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()
# Нормализация данных (упрощение и сведение к одному диапазону от 0 до 1 и от
-1 до 1 сводим к диапазону от 0 до 1)
x_train = x_train.reshape(-1, 28, 28, 1).astype("float32") / 255.0
x_test = x_test.reshape(-1, 28, 28, 1).astype("float32") / 255.0
model = keras.Sequential([
    layers.Conv2D(32, kernel_size=(3, 3), activation="relu", input_shape=(28,
28, 1)),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.Conv2D(64, (3, 3), padding="same", activation="relu"),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.Flatten(),
    layers.Dense(10, activation="softmax"),
])
model.compile(optimizer="adam", loss="sparse_categorical_crossentropy",
metrics=["accuracy"])
history = model.fit(x_train, y_train, epochs=5, validation_data=(x_test,
y_test))

plt.plot(history.history["accuracy"], label="Точность")
plt.plot(history.history["val_accuracy"], label="Точность Val")
plt.title("Точность тренировки")
```

```
# Подпишем оси
plt.xlabel("Эпоха")
plt.ylabel("Точность")
plt.legend()
plt.show()

plt.figure(figsize=(5,5))
for i in range(25):
    plt.subplot(5,5, i + 1)
    plt.imshow(x_train[i], cmap="gray")
    plt.axis("off")

plt.show()

plt.figure(figsize=(5,5))
for i in range(25):
    plt.subplot(5, 5, i + 1)
    plt.imshow(x_train[i], cmap="gray")
    plt.title(y_train[i])
    plt.axis("off")

plt.show()
```

```
Epoch 1/2
1875/1875 [==============================] - 30s 16ms/step - loss: 0.1572 - accuracy: 0.9535 - val_loss: 0.0526 -
val_accuracy: 0.9835
Epoch 2/2
1875/1875 [==============================] - 34s 18ms/step - loss: 0.0518 - accuracy: 0.9837 - val_loss: 0.0351 -
val_accuracy: 0.9894
Epoch 1/5
1875/1875 [==============================] - 30s 16ms/step - loss: 0.1447 - accuracy: 0.9559 - val_loss: 0.0534 -
val_accuracy: 0.9813
Epoch 2/5
1875/1875 [==============================] - 30s 16ms/step - loss: 0.0503 - accuracy: 0.9848 - val_loss: 0.0422 -
val_accuracy: 0.9871
Epoch 3/5
1875/1875 [==============================] - 31s 16ms/step - loss: 0.0363 - accuracy: 0.9886 - val_loss: 0.0340 -
val_accuracy: 0.9894
Epoch 4/5
1875/1875 [==============================] - 31s 17ms/step - loss: 0.0287 - accuracy: 0.9910 - val_loss: 0.0344 -
val_accuracy: 0.9886
```

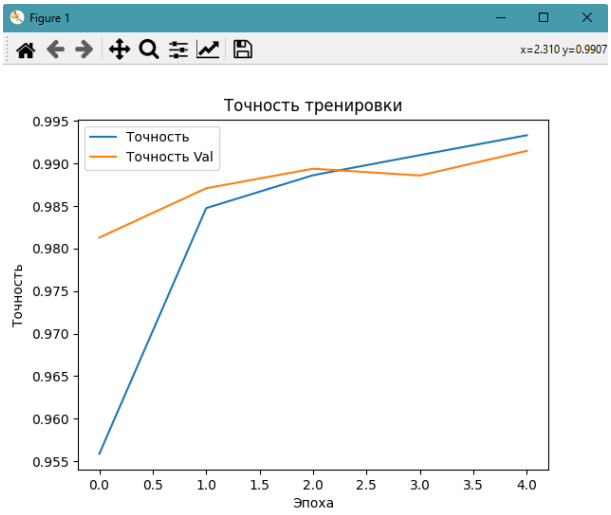


График - 2
«Точность обучения с нормализацией и вторым экспериментом подкручивания параметров»

Легче работу обучения нейросети объяснить на картинке (Рис. 4). Компьютерное зрение способное распознать написанный текст в графическом исполнении. для каждого канала и экземпляра будет нормализован по своим собственным средним значениям и стандартным отклонениям, что может улучшить производительность модели при обучении на изображениях с различными пространственными трансформациями. Поэтому этот эксперимент можно считать удачным без значительных накруток параметров.

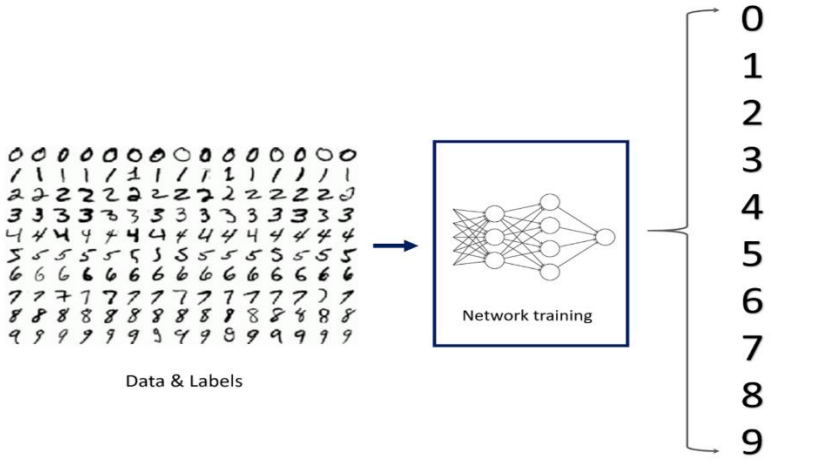
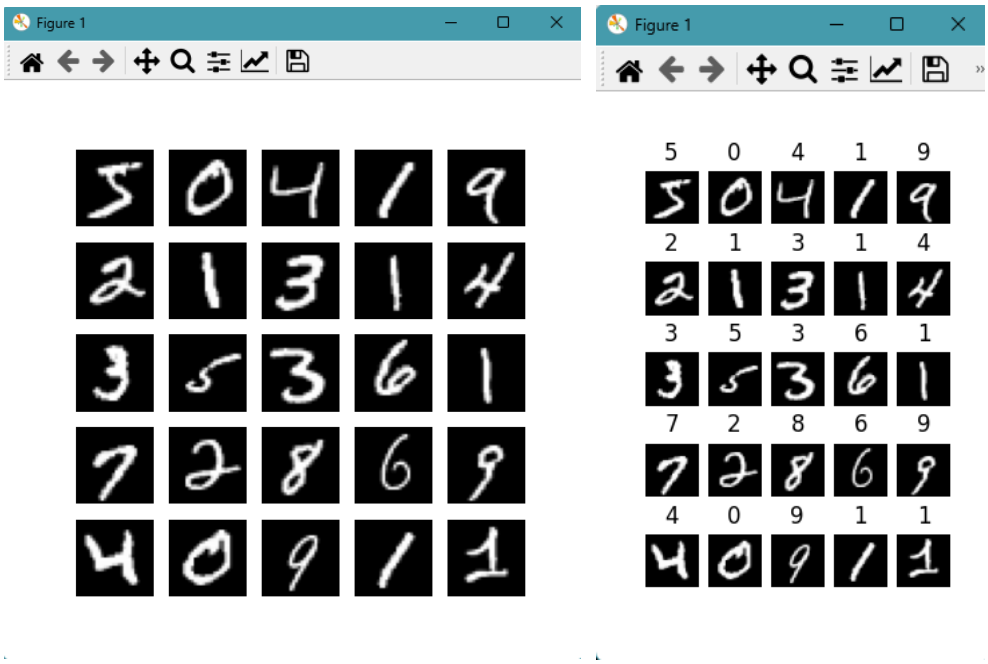


Рис.4 – Схема обучения нейросети




```
Epoch 5/5
1875/1875 [==============================] - 31s 17ms/step - loss: 0.0225 - accuracy: 0.9933 - val_loss: 0.0264 -
val_accuracy: 0.9915
```

Рис.5 – Результат обучения сверточной нейронной сети с применением метода нормализации

Как мы можем заметить, этот эксперимент увенчался успехом и все элементы нейросети были распознаны правильно (Рис.5).

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	51	159	253	159	50	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	48	238	252	252	252	237	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	54	227	253	252	239	233	252	57	6	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	10	60	224	252	253	252	202	84	252	253	122	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	163	252	252	252	253	252	252	96	189	253	167	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	51	238	253	253	190	114	253	228	47	79	255	168	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	48	238	252	252	179	12	75	121	21	0	0	253	243	50	0	0	0	0	0
11	0	0	0	0	0	0	0	0	38	165	253	233	208	84	0	0	0	0	0	0	253	252	165	0	0	0	0	0
12	0	0	0	0	0	0	0	7	178	252	240	71	19	28	0	0	0	0	0	0	253	252	195	0	0	0	0	0
13	0	0	0	0	0	0	0	57	252	252	63	0	0	0	0	0	0	0	0	0	253	252	195	0	0	0	0	0
14	0	0	0	0	0	0	0	198	253	190	0	0	0	0	0	0	0	0	0	0	255	253	196	0	0	0	0	0
15	0	0	0	0	0	0	76	246	252	112	0	0	0	0	0	0	0	0	0	0	253	252	148	0	0	0	0	0
16	0	0	0	0	0	0	85	252	230	25	0	0	0	0	0	0	0	0	7	135	253	186	12	0	0	0	0	0
17	0	0	0	0	0	0	85	252	223	0	0	0	0	0	0	0	0	7	131	252	225	71	0	0	0	0	0	0
18	0	0	0	0	0	0	85	252	145	0	0	0	0	0	0	0	48	165	252	173	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	86	253	225	0	0	0	0	0	0	114	238	253	162	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	85	252	249	146	48	29	85	178	225	253	223	167	56	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	85	252	252	252	229	215	252	252	252	196	130	0	0	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	28	199	252	252	253	252	252	233	145	0	0	0	0	0	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	25	128	252	253	252	141	37	0	0	0	0	0	0	0	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Рис.6 – Результат обучения сверточной нейронной сети с применением метода нормализации

На изображения представлены в виде массивов размером 28x28, заполненных значениями пиксели. Как можно видеть на следующей визуализации. Он создает слой с 784 узлами (Рисунок 6).

Если бы у нас были цветные изображения, которые содержат 3 значения для каждого пикселя (значения RGB), то Flatten() создал бы слой с 28 * 28 * 3 = 2352 узлами,
что значительно бы увеличило время обработки, поэтому рекомендовано изображение переводить в градиентный или черно белый режим.

Эксперимент 3 - «Нормализация данных. Батч-нормализация (градиентное затухание и градиентный взрыв»

Поскольку в этой области я провел наибольше количество экспериментов, совмещая регуляризацию и нормализацию, исключая и комбинируя эти компоненты, я буду краток в их описании. **Батч нормализация** - метод, который ускоряет и улучшает процесс обучения нейронных сетей. Чаще всего он нужен для градиентного затухания и градиентного взрыва в нейронных сетях.

Градиенты затухания — это когда повторяются одни и те же данные с уменьшением.

Градиентный взрыв — это когда повторяются одни и те же данные с увеличением.

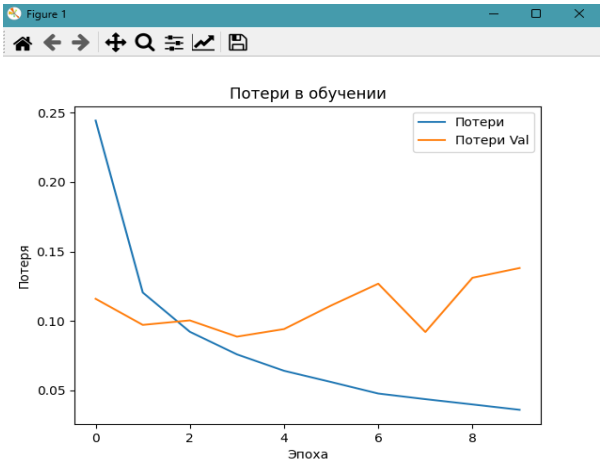
Батч нормализация содержит два основных метода Моментум - когда необходимо брать банные из предыдущих слоёв батчей. Эпсилон - для предотвращения деления на ноль.

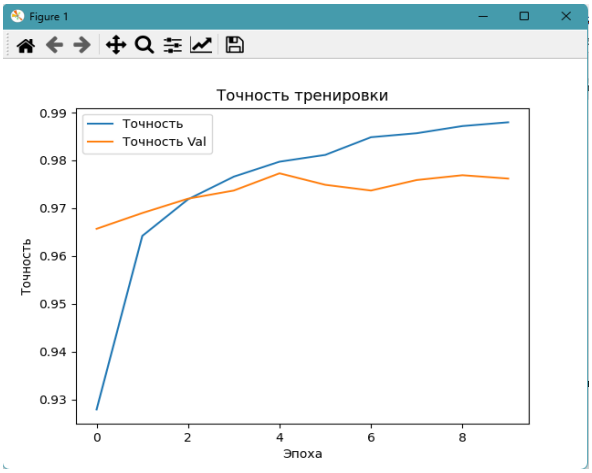
Гамма и Бетта инициализэщион - для изменения параметра масштабирования и взаимодействия с предыдущими батчами.

```
import tensorflow as tf
from keras.src.applications.densenet import layers
from tensorflow import keras
import matplotlib.pyplot as plt
# Загружаем модель 70 000 изображений 60 000 тренировок и 10 000 тестов
(x_train,  y_train), (x_test, y_test) = keras.datasets.mnist.load_data()
# Нормализация данных (упрощение и сведение к одному диапазону от 0 до 1 и от
-1 до 1 сводим к диапазону от 0 до 1)
# Всё что меньше 1 становится 0, всё что больше 1 становится 1
x_test.astype("float32") / 255.0
x_train.astype("float32") / 255.0
# Создаём структуру модели
model = keras.Sequential([
#Переводим из многомерного массива в одномерный с 28 мерным вектором
    # Это первый слой нашей нейронной сети. Каждое изображение имеет 28 * 28
= 784 значения и такFlatten()создает слой с 784 узлами,
    layers.Flatten(input_shape=(28, 28)),
    # Другой вид слоя, который мы видим в модели, создан с использованием
tf.keras.layers.Dense() Он создает то,
    # что называется полностью связанным или плотно связанным слоем.
    layers.Dense(128, activation="relu"),
    # Уровень в нашей сети называется «ReLU», что означает сокращение
выпрямленной линейной единицы.
```

```
    # Что делает ReLU, так это делает активацию любых отрицательных логитов 0
(узел не срабатывает),
    # при этом оставляя любые положительные логиты неизменными (узел стреляет
с силой, линейно
    # пропорциональной силе входа).
    layers.BatchNormalization(),
    # layers.Dropout(0.5), не стоит
    # Без взаимодействия с предыдущими слоями
    layers.Dense(10, activation="softmax")
    # Интерпретируем в итоговые результаты
    # softmax берет логиты, вычисленные по взвешенной сумме активаций из
предыдущего слоя,
    # и преобразует их в вероятность, так называемые придикты
])
# Укажем конфигурацию обучения (оптимизатор, функция потерь, метрики)
model.compile(optimizer="adam", loss="sparse_categorical_crossentropy",
metrics=["accuracy"])
# Определим метрику точности "accuracy"
# optimizer="adam" - среднеквадратичное распределение
# Это важные особенности того, как нейронная сеть дает свои окончательные
прогнозы.
# Обучаем модель
history = model.fit(x_train, y_train, epochs=10, validation_data=(x_test,
y_test))
plt.plot(history.history["loss"], label="Потери")
plt.plot(history.history["val_loss"], label="Потери Val")
plt.title("Потери в обучении")
# Подпишем оси
plt.xlabel("Эпоха")
plt.ylabel("Потеря")
plt.legend()
plt.show()
plt.plot(history.history["accuracy"], label="Точность")
plt.plot(history.history["val_accuracy"], label="Точность Val")
plt.title("Точность тренировки")
# Подпишем оси
plt.xlabel("Эпоха")
plt.ylabel("Точность")
plt.legend()
plt.show()
```

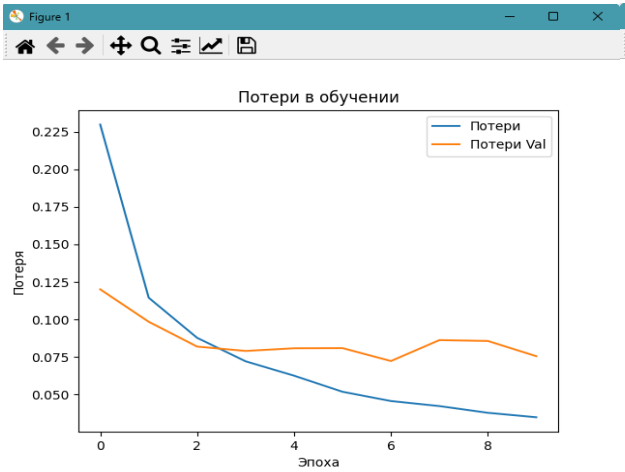
```
Epoch 1/10
1875/1875 [==============================] - 5s 2ms/step - loss: 0.2443 - accuracy: 0.9279 - val_loss: 0.1160 - val_accuracy: 0.9657
Epoch 2/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1205 - accuracy: 0.9642 - val_loss: 0.0972 - val_accuracy: 0.9690
Epoch 3/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0922 - accuracy: 0.9719 - val_loss: 0.1004 - val_accuracy: 0.9720
Epoch 4/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0760 - accuracy: 0.9766 - val_loss: 0.0887 - val_accuracy: 0.9737
Epoch 5/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0642 - accuracy: 0.9797 - val_loss: 0.0942 - val_accuracy: 0.9773
Epoch 6/10
1875/1875 [============== ==============] - 4s 2ms/step - loss: 0.0561 - accuracy: 0.9811 - val_loss: 0.1112 - val_accuracy: 0.9749
Epoch 7/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0478 - accuracy: 0.9848 - val_loss: 0.1269 - val_accuracy: 0.9737
Epoch 8/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0437 - accuracy: 0.9857 - val_loss: 0.0921 - val_accuracy: 0.9759
Epoch 9/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0399 - accuracy: 0.9872 - val_loss: 0.1311 - val_accuracy: 0.9769
Epoch 10/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0361 - accuracy: 0.9880 - val_loss: 0.1381 - val_accuracy: 0.9762
```

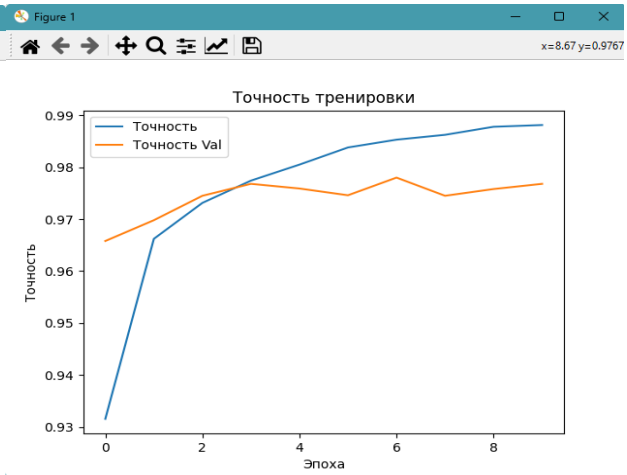




Графики
3а – «Потери в обучении с градиентным спуском

3б – «Точность обучения с с градиентным спуском





Графики
4а – «Потери в обучении с градиентным спуском и моими настройками

4б – «Точность обучения с с градиентным спуском и моими настройками

Как мы можем увидеть из графиков настройки значительно повлияли на потери данных в обучении, причём с тенденцией снижения потерь. Точность тоже повышается с увеличением значений, причем на этапе шестой эпохи наблюдается скачок повышения точности эксперимента обучения.

Эксперимент 4 - «Dense»

Dense – это другой вид слоя, который мы видим в модели, создан с использованием tf.keras.layers.Dense(). Он создает то, что называется полностью связанным или плотно связанным слоем. Это можно сравнить с разреженным слоем, и различие связано с тем, как информация передается между узлами в соседних слоях. Первый параметр (128 в первом случае) указывает, сколько узлов должно быть на уровне. Число узлов в скрытых слоях (слоях, которые не являются входными или выходными слоями) несколько произвольно, но важно отметить, что выходной слой имеет количество узлов, равное количеству классов, которые модель пытается предсказать.

Например, модель пытается предсказать 10 различных цифр, и поэтому последний слой в модели имеет 10 узлов. Это очень важно, потому что на выходе для каждого узла конечного слоя будет вероятность того, что данное изображение является конкретной цифрой.

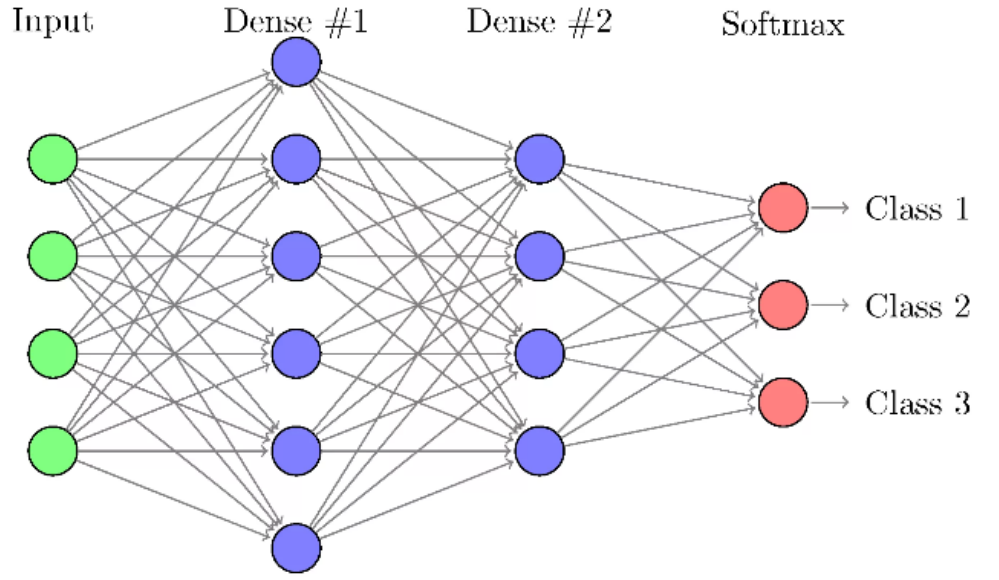


Рис.7 – Плотносвязанные слои Dense

```
from keras.src.applications.densenet import layers
from tensorflow import keras
import matplotlib.pyplot as plt
# Загружаем модель 70 000 изображений 60 000 тренировок и 10 000 тестов
```

```
(x_train,  y_train), (x_test, y_test) = keras.datasets.mnist.load_data()
# Нормализация данных (упрощение и сведение к одному диапазону от 0 до 1 и от
-1 до 1 сводим к диапазону от 0 до 1)
x_test.astype("float32") / 255.0
x_train.astype("float32") / 255.0

# Создаём структуру модели
model = keras.Sequential([
    layers.Flatten(input_shape=(28, 28)),
    layers.Dense(128, activation="relu"),
    layers.BatchNormalization(),
    layers.Dense(10, activation="softmax")
])
model.compile(optimizer="adam", loss="sparse_categorical_crossentropy",
metrics=["accuracy"])

history = model.fit(x_train, y_train, epochs=10, validation_data=(x_test,
y_test))

plt.plot(history.history["loss"], label="Потери")
plt.plot(history.history["val_loss"], label="Потери Val")
plt.title("Потери в обучении")

plt.xlabel("Эпоха")
plt.ylabel("Потеря")
plt.legend()
plt.show()

plt.plot(history.history["accuracy"], label="Точность")
plt.plot(history.history["val_accuracy"], label="Точность Val")
plt.title("Точность тренировки")
# Подпишем оси
plt.xlabel("Эпоха")
plt.ylabel("Точность")
plt.legend()
plt.show()
```

```
Epoch 1/10
1875/1875 [==============================] - 5s 2ms/step - loss: 0.2298 - accuracy: 0.9316 - val_loss: 0.1200 - val_accuracy: 0.9658
Epoch 2/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1145 - accuracy: 0.9662 - val_loss: 0.0985 - val_accuracy: 0.9698
Epoch 3/10
1875/1875 [==============================] - 5s 2ms/step - loss: 0.0877 - accuracy: 0.9731 - val_loss: 0.0819 - val_accuracy: 0.9745
Epoch 4/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0721 - accuracy: 0.9774 - val_loss: 0.0790 - val_accuracy: 0.9768
Epoch 5/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0625 - accuracy: 0.9805 - val_loss: 0.0808 - val_accuracy: 0.9759
Epoch 6/10
1875/1875 [==============================] - 5s 2ms/step - loss: 0.0518 - accuracy: 0.9838 - val_loss: 0.0809 - val_accuracy: 0.9746
Epoch 7/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0456 - accuracy: 0.9853 - val_loss: 0.0723 - val_accuracy: 0.9780
Epoch 8/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0423 - accuracy: 0.9862 - val_loss: 0.0862 - val_accuracy: 0.9745
Epoch 9/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0378 - accuracy: 0.9878 - val_loss: 0.0856 - val_accuracy: 0.9758
Epoch 10/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0348 - accuracy: 0.9881 - val_loss: 0.0755 - val_accuracy: 0.9768
```

Эксперимент 5 - «Нормализация Dense без регуляризации»

```
Нормализация без регулизации
model = keras.Sequential([
    Conv2D(32, (3,3), padding='same', activation='relu', input_shape=(28, 28,
1)),
    MaxPooling2D((2, 2), strides=2),
    Conv2D(64, (3,3), padding='same', activation='relu'),
    MaxPooling2D((2, 2), strides=2),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(10,  activation='softmax')
])

model.summary()
# Компиляция модели
model.compile(optimizer="adam", loss="sparse_categorical_crossentropy",
metrics=["accuracy"])

# Обучение модели
history = model.fit(x_train, y_train, epochs=5, validation_split=0.02)

# Оценка модели на тестовых данных
test_scores = model.evaluate(x_test, y_test, verbose=2)

print("Потеря теста:", test_scores[0])
print("Точность теста:", test_scores[1])
```

```
_________________________________________________________________
 Layer (type)            Output Shape          Param #
=================================================================
 conv2d (Conv2D)         (None, 28, 28, 32)      320
 max_pooling2d (MaxPooling2  (None, 14, 14, 32)      0       D)
 conv2d_1 (Conv2D)       (None, 14, 14, 64)      18496
 max_pooling2d_1 (MaxPoolin  (None, 7, 7, 64)    0       g2D)
 flatten (Flatten)       (None, 3136)          0
 dense (Dense)           (None, 128)           401536
 dense_1 (Dense)         (None, 10)            1290
=================================================================
Total params: 421642 (1.61 MB)
Trainable params: 421642 (1.61 MB)
Non-trainable params: 0 (0.00 Byte)
Epoch 1/5
1838/1838 [==============] - 27s 14ms/step - loss: 0.1277 - accuracy: 0.9605 - val_loss: 0.0541 - val_accuracy: 0.9908
Epoch 2/5
1838/1838 [==============] - 25s 13ms/step - loss: 0.0404 - accuracy: 0.9877 - val_loss: 0.0500 - val_accuracy: 0.9908
Epoch 3/5
1838/1838 [==============] - 26s 14ms/step - loss: 0.0270 - accuracy: 0.9914 - val_loss: 0.0484 - val_accuracy: 0.9917
Epoch 4/5
1838/1838 [==============] - 27s 15ms/step - loss: 0.0193 - accuracy: 0.9935 - val_loss: 0.0505 - val_accuracy: 0.9942
Epoch 5/5
1838/1838 [==============] - 25s 13ms/step - loss: 0.0144 - accuracy: 0.9951 - val_loss: 0.0488 - val_accuracy: 0.9917
313/313 - 1s - loss: 0.0277 - accuracy: 0.9915 - 1s/epoch - 4ms/step
Потерятеста: 0.027710041031241417
Точностьтеста: 0.9915000200271606
```

Это можно назвать удачным экспериментом, из-за его чисты по отношению к другим, чтобы проанализировать работу без нормалей и регуляризации.

Регуляризация

Регуляризация — это метод, который используется для уменьшения влияния больших значений весов модели, которые могут привести к переобучению. Регуляризация осуществляется путем добавления штрафа на величину весов в функцию потерь модели. Два наиболее распространенных типа регуляризации — это L1 и L2 регуляризация, которые добавляют штраф на абсолютную и квадратичную сумму весов соответственно.

L1 регуляризация — это метод регуляризации, который добавляет штраф на абсолютное значение весов в функцию потерь, чтобы уменьшить влияние ненужных или малозначимых признаков на модель.

Функция потерь для линейной регрессии с L1 регуляризацией (Lasso) выглядит следующим образом:

$L(w) = ||y - Xw||^2 + \lambda * ||w||_1$

, где $||y - Xw||^2$ — это среднеквадратичное отклонение между прогнозами модели и реальными значениями,

X — это матрица признаков,

w — это вектор весов,

λ — это гиперпараметр регуляризации, контролирующий степень регуляризации,

$||w||_1$ — это L1-норма вектора весов, определяемая как сумма абсолютных значений весов.

L1 регуляризация уменьшает веса модели до нуля, что приводит к разреженным моделям и отбору признаков. Она может быть полезна в задачах с большим количеством признаков, где не все признаки важны [2].

L2 регуляризация, также известная как **гребневая регрессия (RidgeRegression)**, является методом регуляризации, который добавляет к функции потерь штраф на квадраты весов, чтобы уменьшить влияние больших значений весов на модель.

Функция потерь для линейной регрессии с L2 регуляризацией выглядит следующим образом:

$L(w) = ||y - Xw||^2 + \lambda * ||w||^2$

, где $||y - Xw||^2$ — это среднеквадратичное отклонение между прогнозами модели и реальными значениями,

X — это матрица признаков,

w — это вектор весов,

λ — это гиперпараметр регуляризации, контролирующий степень регуляризации,

$||w||^2$ — это L2-норма вектора весов, определяемая как сумма квадратов значений весов

[2].

Другими словами, **L1 регуляризация** уменьшает веса модели до нуля, что приводит к разреженным моделям и отбору признаков. Она может быть полезна в задачах с большим количеством признаков, где не все признаки важны. А **L2 регуляризация** ограничивает величину весов, делая их более стабильными и менее склонными к переобучению.

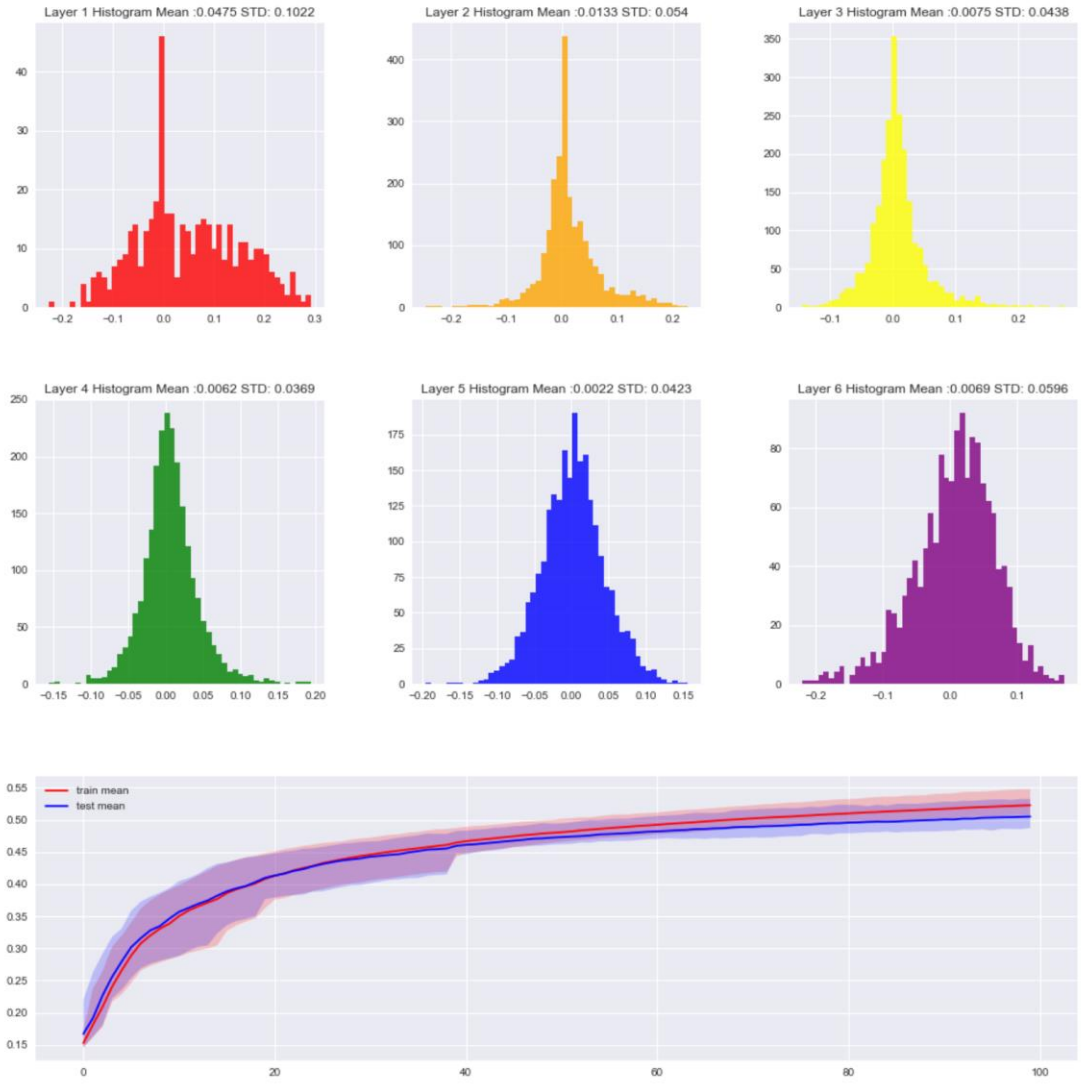


Рис.7 – Регуляризация в весовых точках зрения

Ручное обратное распространение в тензорном потоке

На сайте https://machinelearningmastery.ru по машинному обучению автор лаборатории Ryerson Vision lab, Сен Цзя глубоко исследует процесс регуляризации в каждом слое. Его эксперименты с моделью получились эффективнее и нагляднее моих.

Он обучает сеть для 10 эпизодов с пакетной нормализацией и без нее, чтобы наблюдать, происходит ли одно и то же явление снова и снова. (явление, веса сходятся к определенной

форме гистограммы. Каждое квадратное поле показывает окончательную гистограмму для каждого слоя, а на последнем графике показана максимальная / минимальная / средняя эффективность тренировки.

Dropout — это метод регуляризации, который используется в нейронных сетях для предотвращения переобучения. Он заключается в том, что на каждом шаге обучения случайно выбирается определенное количество нейронов, которые будут «выключены» (или «выброшены») во время прохождения данных через сеть. Таким образом, в каждой итерации обучения часть нейронов не участвует в передаче информации, что приводит к уменьшению влияния каждого отдельного нейрона на итоговый результат и в итоге предотвращает переобучение (Рис. 8)

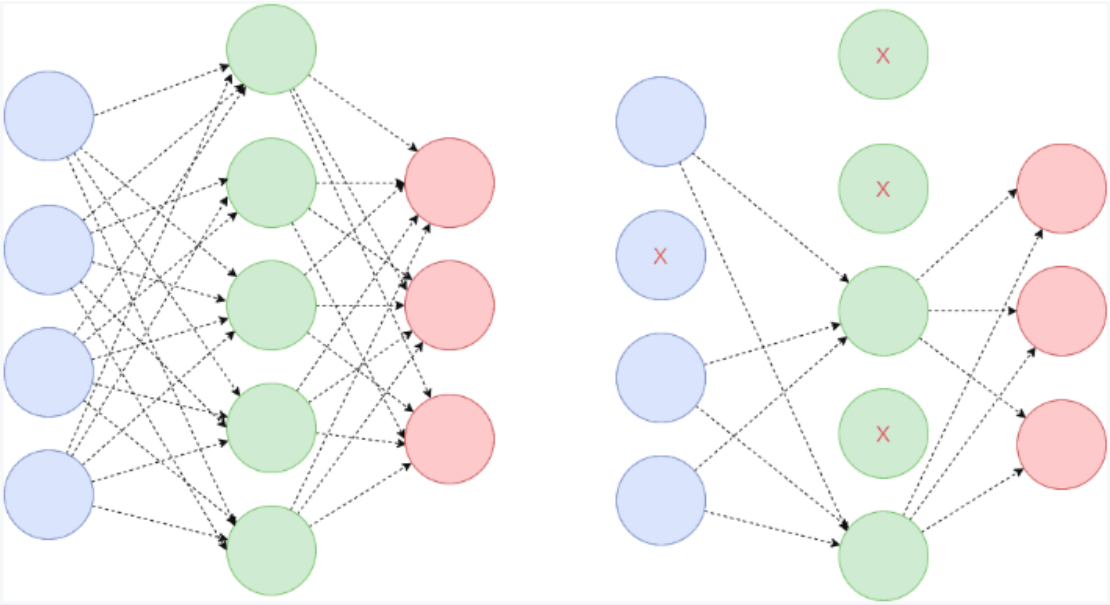


Рис.8 – Схема метода регуляризации Dropout

При применении dropout на каждом шаге обучения создается новая архитектура сети, что приводит к более эффективному использованию данных и уменьшению переобучения.

Dropout позволяет сети обучаться более устойчиво к шуму в данных и позволяет избежать сильной зависимости между нейронами, что может привести к более точным прогнозам на новых данных.

Однако при использовании dropout нужно учитывать, что он уменьшает количество активных нейронов в каждом слое, что может привести к снижению скорости обучения и ухудшению качества модели.

Кроме того, при выборе вероятности p для dropout необходимо учитывать размер и сложность сети, а также размер и сложность данных.

Для уменьшения этих негативных эффектов dropout может использоваться в сочетании с другими методами регуляризации, такими как L1 или L2 регуляризация.

Кроме того, dropout может быть эффективен только в случае наличия достаточного количества нейронов в каждом слое, так как при малом количестве нейронов использование dropout может привести к ухудшению качества модели.

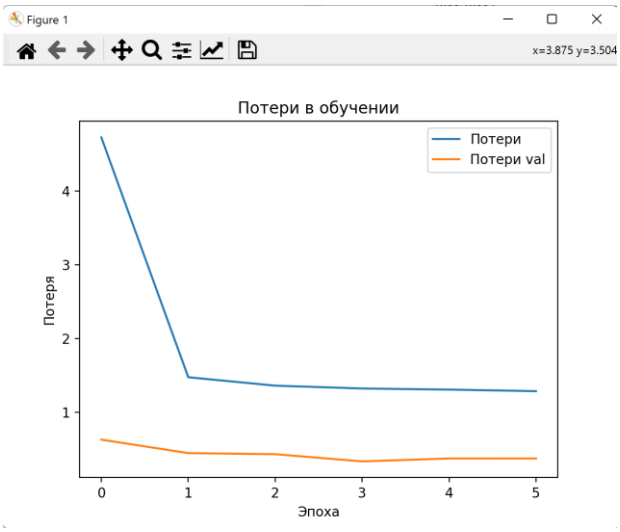
В целом dropout является эффективным и широко используемым методом регуляризации в нейронных сетях, который помогает предотвратить переобучение и улучшить качество модели на новых данных.

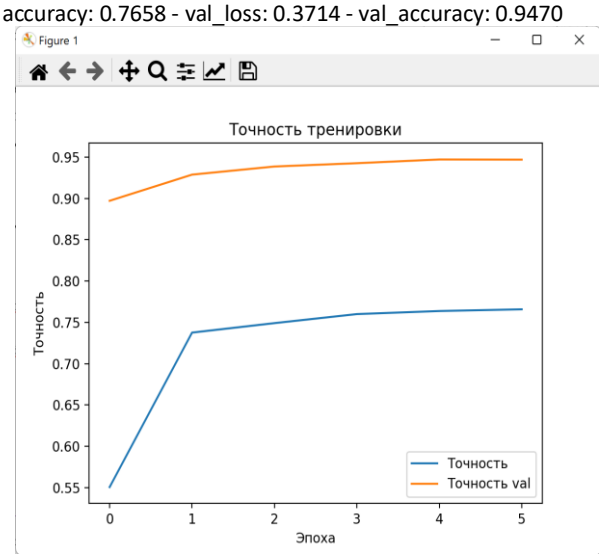
Эксперимент 6 - «L1- регуляризация»

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.callbacks import EarlyStopping
import matplotlib.pyplot as plt

(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()

x_test.astype("float32") / 255.0
x_train.astype("float32") / 255.0

model = keras.Sequential([
    layers.Flatten(input_shape=(28, 28)),
    layers.Dense(128, activation="relu"),
    layers.Dense(128, activation="relu"),
    layers.Dropout(0.2)
])
early = EarlyStopping(monitor="val_loss", patience=2)
model.compile(optimizer="adam",
loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=["accuracy"])

history = model.fit(x_train, y_train, epochs=10, validation_split=0.1,
callbacks=[early])

plt.plot(history.history["loss"], label="Потери")
plt.plot(history.history["val_loss"], label="Потери val")
plt.title("Потери в обучении")
plt.xlabel("Эпоха")
plt.ylabel("Потеря")
plt.legend()
plt.show()

plt.plot(history.history["accuracy"], label="Точность")
plt.plot(history.history["val_accuracy"], label="Точность val")
plt.title("Точность тренировки")
plt.xlabel("Эпоха")
plt.ylabel("Точность")
plt.legend()
plt.show()
```

```
Epoch 1/10
1688/1688 [==============================] - 4s 2ms/step - loss: 4.7288 - accuracy: 0.5506 - val_loss: 0.6273 -
```

```
val_accuracy: 0.8972
Epoch 2/10
1688/1688 [==============] - 3s 2ms/step - loss: 1.4743 - accuracy: 0.7376 - val_loss: 0.4442 - val_accuracy: 0.9288
Epoch 3/10
1688/1688 [==============] - 3s 2ms/step - loss: 1.3600 - accuracy: 0.7491 - val_loss: 0.4298 - val_accuracy: 0.9387
Epoch 4/10
1688/1688 [==============] - 3s 2ms/step - loss: 1.3214 - accuracy: 0.7601 - val_loss: 0.3321 - val_accuracy: 0.9427
Epoch 5/10
1688/1688 [==============] - 3s 2ms/step - loss: 1.3063 - accuracy: 0.7638 - val_loss: 0.3713 - val_accuracy: 0.9472
Epoch 6/10
1688/1688 [==============] - 3s 2ms/step - loss: 1.2853 - accuracy: 0.7658 - val_loss: 0.3714 - val_accuracy: 0.9470
```





Графики
5а – «Потери в обучении с градиентным спуском и моими настройками

5б – «Точность обучения с с градиентным спуском и моими настройками

Мы наблюдаем после первой эпохи резкое снижение потерь, но и резкое замедление точности.

Эксперимент 7 - «L2- регуляризация»

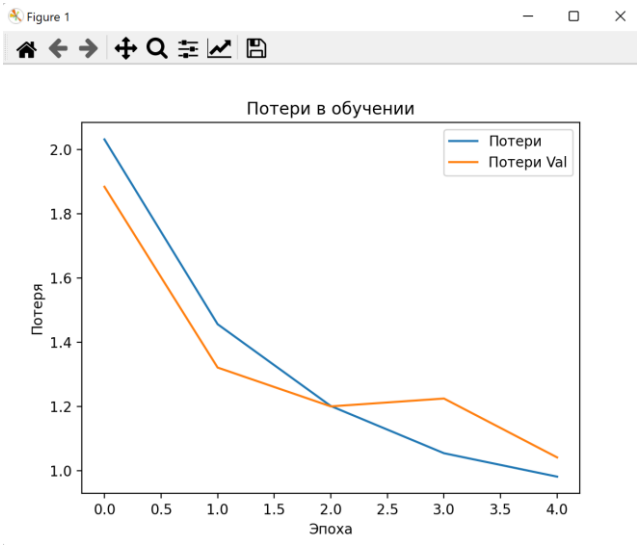
В этом примере мы будем отслеживать скорость эпох, чтобы применять тот или иной метод регуляризации.

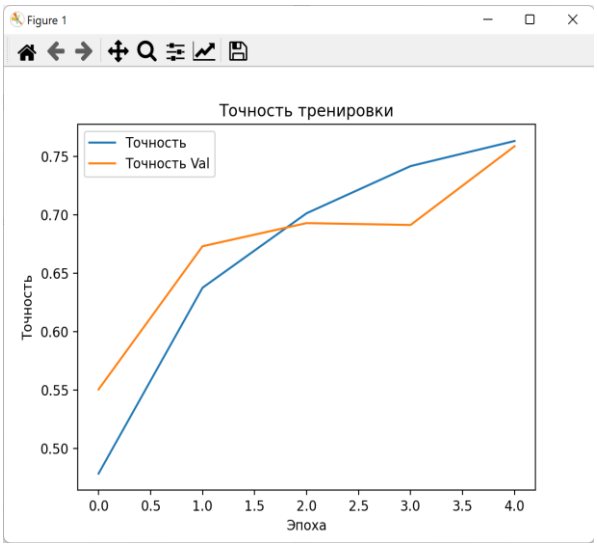
```
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.callbacks import EarlyStopping
import matplotlib.pyplot as plt
from tensorflow.python.keras import regularizers
from tensorflow.keras.datasets import cifar10
# Загружаем модель 70 000 изображений 60 000 тренировок и 10 000 тестов
(x_train, y_train), (x_test, y_test) = keras.datasets.cifar10.load_data()
# Нормализация данных (упрощение и сведение к одному диапазону от 0 до 1 и от
-1 до 1 сводим к диапазону от 0 до 1)
x_test.astype("float32") / 255.0
x_train.astype("float32") / 255.0

# Построение структуры модели с L2-регулизацией
model = keras.Sequential([
    layers.Conv2D(32, (3, 3), padding="same", activation="relu",
kernel_regularizer=regularizers.l2(0.001),
                  input_shape=(32, 32, 3)),
```

```
    layers.BatchNormalization(),
    layers.Conv2D(32, (3, 3), padding="same", activation="relu",
kernel_regularizer=regularizers.l2(0.001)),
    layers.BatchNormalization(),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.Dropout(0.2),
    layers.Conv2D(64, (3, 3), padding="same", activation="relu",
kernel_regularizer=regularizers.l2(0.001)),
    layers.BatchNormalization(),
    layers.Conv2D(64, (3, 3), padding="same", activation="relu",
kernel_regularizer=regularizers.l2(0.001)),
    layers.BatchNormalization(),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.Dropout(0.2),
    layers.Conv2D(128, (3, 3), padding="same", activation="relu",
kernel_regularizer=regularizers.l2(0.001)),
    layers.BatchNormalization(),
    layers.Conv2D(128, (3, 3), padding="same", activation="relu",
kernel_regularizer=regularizers.l2(0.001)),
    layers.BatchNormalization(),
    layers.MaxPooling2D(pool_size=(2, 2)),
    layers.Dropout(0.2),
    layers.Flatten(),
    layers.Dense(10, activation="softmax"),
])
# эрлистопинг - это когда один из нейронов не приносит пользы, это тот
нейрон, который пытается переобучиться:
# Когда потери будут составлять 2 эпохи, отключим нейрон
early = EarlyStopping(monitor="val_loss", patience=2)
model.compile(optimizer="adam", loss="sparse_categorical_crossentropy",
metrics=["accuracy"])
history = model.fit(x_train, y_train, batch_size=64, epochs=5,
validation_split=0.1, callbacks=[early])


plt.plot(history.history["loss"], label="Потери")
plt.plot(history.history["val_loss"], label="Потери Val")
plt.title("Потери в обучении")
# Подпишем оси
plt.xlabel("Эпоха")
plt.ylabel("Потеря")
plt.legend()
plt.show()

plt.plot(history.history["accuracy"], label="Точность")
plt.plot(history.history["val_accuracy"], label="Точность Val")
plt.title("Точность тренировки")
# Подпишем оси
plt.xlabel("Эпоха")
plt.ylabel("Точность")
plt.legend()
plt.show()
```

```
Epoch 1/5
704/704 [==============================] - 112s 156ms/step - loss: 2.0314 - accuracy: 0.4786 - val_loss: 1.8840 -
val_accuracy: 0.5506
Epoch 2/5
704/704 [==============================] - 108s 153ms/step - loss: 1.4559 - accuracy: 0.6377 - val_loss: 1.3208 -
val_accuracy: 0.6732
Epoch 3/5
704/704 [==============================] - 109s 154ms/step - loss: 1.2010 - accuracy: 0.7014 - val_loss: 1.1998 -
val_accuracy: 0.6930
Epoch 4/5
```

```
704/704 [==============] - 111s 157ms/step - loss: 1.0540 - accuracy: 0.7418 - val_loss: 1.2243 - val_accuracy: 0.6914
Epoch 5/5
704/704 [==============] - 109s 155ms/step - loss: 0.9810 - accuracy: 0.7633 - val_loss: 1.0409 - val_accuracy: 0.7588
```





Графики
6а – «Потери в обучении метода L2 - регуляризации

6б – «Точность обучения с метода L2 - регуляризации

Вот! Вот! Ура! Есть за что «зацепиться!»

Мы наблюдаем эффективный алгоритм резкого снижения потерь в обучении, и повышение точности, но также можем наблюдать значительное замедление процесса обучения. Около 110 секунд в среднем требуется для обучения в один проход, т.е. в одну эпоху. Вот здесь возникает вопрос применения аппаратных вычислений, потому что один центральный процессор не справится с такой сложной задачей. Поэтому для L-2 регуляризации рекомендуют использовать графические видеокарты, способные в значительной степени увеличить скорость обучения. И это еще одна из интересных тем, которая достойна внимания. В пределах этого проекта мы не будем рассматривать процесс ускорения, ведь мы преследуем иные цели- найти алгоритм и совокупность, комбинацию применения методов обучения нейронных сетей.

Эксперимент 8 - «Максимальное подобие»

Остался еще один метод обучения по максимальному подобию. Рассмотрим его простой алгоритм на примере анализа двух картинок кремля.

```
# Максимальное подобие (метод сходства)
import tensorflow as tf
import cv2
```

```
import numpy as np
model = tf.keras.applications.ResNet50(include_top=False, weights="imagenet",
pooling="avg")
img1 = cv2.imread("kremlin1.jpg")
img2 = cv2.imread("kremlin2.jpg")
img1 = cv2.resize(img1, (224, 224))
img2 = cv2.resize(img2, (224, 224))
img1 = img1.astype("float32")
img2 = img2.astype("float32")
img1 = tf.keras.applications.resnet_v2.preprocess_input(img1)
img2 = tf.keras.applications.resnet_v2.preprocess_input(img2)
emb1 = model.predict(np.array([img1]))
emb2 = model.predict(np.array([img2]))
sim = 1 - tf.keras.losses.cosine_similarity(emb1, emb2)
if sim > 1.9:
    print("Похожие изображения")
else:
    print("Не похожие изображения")
```

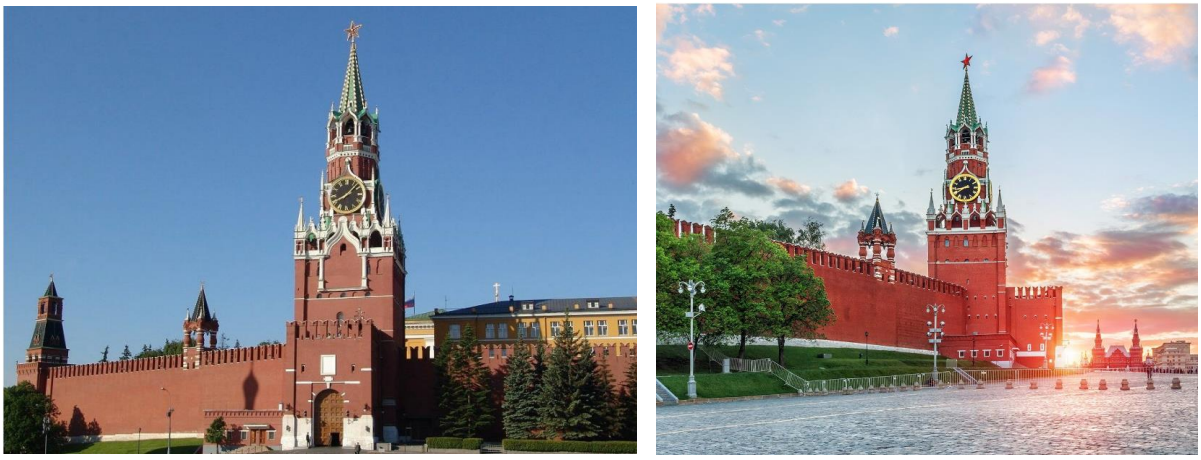


Рис.9 – Исходные данные

```
94765736/94765736 [==============================] - 14s 0us/step
1/1 [==============================] - 1s 1s/step
1/1 [==============================] - 0s 125ms/step
```

Похожие изображения

Программа смогла сама проанализировать сходство. Скоростное взаимодействие нейронов путем максимального подобия работает очень быстро, но всё же нужно быть внимательнее с показателем sim. Может, есть смысл провести больше количество экспериментов и определить показатель распределения более точно?

Результат проекта.

Алгоритм построения модели по комбинации удачных приёмов и методов обучения.

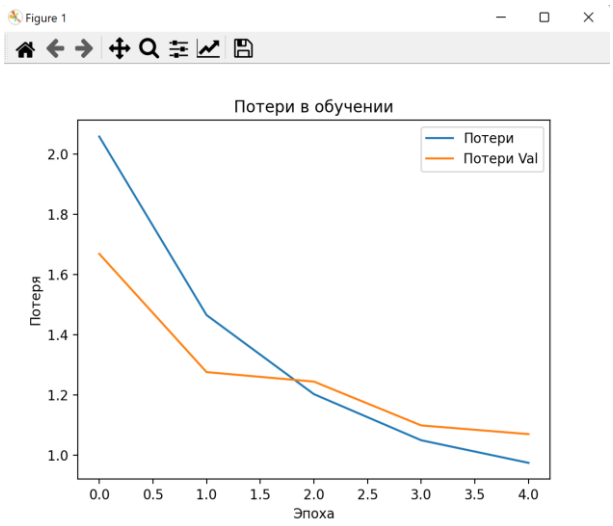
Здесь, внутри программы, в области комментариев я постарался более полно описать назначение каждого слоя модели и эффект применения в обучении.

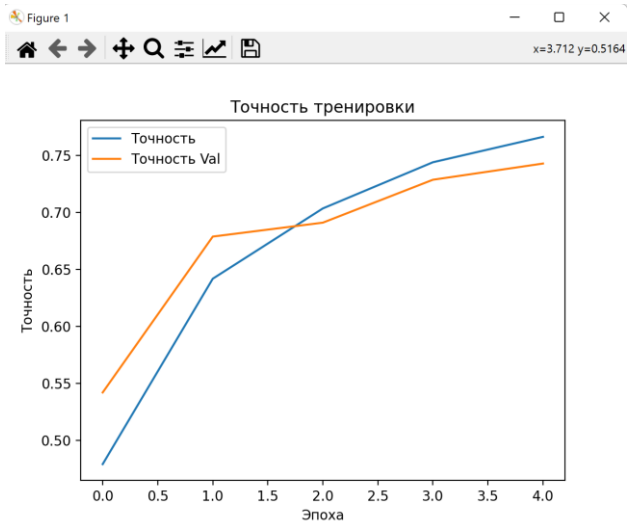
```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.callbacks import EarlyStopping
import matplotlib.pyplot as plt
# Загружаем 70 000 изображений 60 000 тестовых и 10 000 тренировочных
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()
# Нормализация данных (упрощение и сведение к одному диапазону от 0 до 1 и от -1 до 1 сводим к диапазону от 0 до 1) Всё что меньше 1 становится 0, всё что больше 1 становится 1
x_test.astype("float32") / 255.0
x_train.astype("float32") / 255.0

# Создаём структуру модели
model = keras.Sequential([
    layers.Flatten(input_shape=(28, 28)),
    # Переводим из многомерного массива в одномерный с 28 мерным вектором
    # Это первый слой нашей нейронной сети. Каждое изображение имеет 28 * 28 = 784 значения и такFlatten()
    # а изображения представлены в виде массивов размером 28x28, заполненных значениями пикселей
    layers.Dense(128, activation="relu"),
    # Создаем свой слой с 28 нейронами с функцией активации "relu",
    # Это другой вид слоя, который мы видим в модели, создан с использованием tf.keras.layers.Dense(). Он создает то, что называется полностью связанным или плотно связанным слоем. Это можно сравнить с разреженным слоем, и различие связано с тем, как информация передается между узлами в соседних слоях.
    # Первый параметр (128 в первом случае) указывает, сколько узлов должно быть на уровне.
    # Число узлов в скрытых слоях (слоях, которые не являются входными или выходными слоями) несколько произвольно, но важно отметить, что выходной слой имеет количество узлов, равное количеству классов,  которые модель пытается предсказать.
```

```
    Это очень важно, потому что на выходе для каждого узла
    # конечного слоя будет вероятность того, что данное изображение является конкретной цифрой.
    layers.Dense(128, activation="relu"),
    # Уровень в нашей сети называется «ReLU», что означает сокращение выпрямленной линейной единицы.
    # Что делает ReLU, так это делает активацию любых отрицательных логитов 0 (узел не срабатывает), при этом оставляя любые положительные логиты неизменными (узел стреляет с силой, линейно пропорциональной силе входа).
    layers.Dropout(0.2),# Dropout - это метод который случайным образом отключает какой-либо нейрон каждую итерацию, т.е каждую итерацию прохождения эпохи. Это помогает, исключить переобучение. Dropout(0.5) - 50% нейронов отключится
    # С вероятностью 20% будем отключать по 1 нейрону, чтобы избежать переобучения нейронной сети
    # Уменьшаем количество нейронов до 128
    layers.Dense(128, activation="relu"),
    layers.Dropout(0.2),
    # Концепция отсева восходит к более раннему обсуждению связности слоев и имеет отношение конкретно к нескольким недостаткам, связанным с плотно связанными слоями. Одним из недостатков плотно связанных слоев является то, что это может привести к очень дорогостоящим в вычислительном отношении нейронным сетям.
    # С каждым узлом, передающим информацию каждому другому узлу на следующем уровне, сложность взвешенных сумм, вычисленных в каждом узле, экспоненциально увеличивается с количеством узлов в каждом уровне.
    # Еще один недостаток заключается в том, что при передаче большого количества информации от слоя к слою модели могут иметь тенденцию превышать данные обучения, что в конечном итоге снижает производительность.
    # Таким образом, в записной книжке для начинающих, вызов Dropout(0.2) между двумяDense()слои делает так, что каждый узел в первом Dense()слое имеет вероятность 0,2 быть исключенным из расчета активаций следующего слоя.
    # Возможно, вы поняли, что это эффективно делает выходной слой в модели слабо связанным слоем.
    # Заканчиваем преобразованием в придикты
    layers.Dense(10, activation="softmax"),

])
# эрлистопинг - это когда один из нейронов не приносит пользы, тот нейрон, который пытается переобучиться:
# Когда потери будут составлять 2, отключим нейрон
```

```
early = EarlyStopping(monitor="val_loss", patience=2)

# Следующий этап - это Компиляция модели (оптимизация функции потерь
(насколько нейронка была менее точной предыдущей модели)
# Укажем конфигурацию обучения (оптимизатор, функция потерь, метрики)
# True для уменьшения) Определим метрику точности "accuracy"
# optimizer="Adam" - среднеквадратичное распределение
# loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True) -
Минимизируемая функция потерь
model.compile(optimizer="adam",
loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=["accuracy"])
# Функция вызывается по заранее построенной модели и определяет функцию
потерь, оптимизатор и метрики, каждую из которых хочу объяснить. Это важные
особенности того, как нейронная сеть дает свои окончательные прогнозы.
# Функция потерь является частью модели, которая количественно определяет,
насколько далеко от прогноза находится правильный ответ. Различные типы
моделей будут нуждаться в различных функциях потерь.
# Например, функция потерь для такой проблемы, как эта, где выходные данные
для нашей модели являются вероятностями, должна сильно отличаться от функции
потерь модели, которая пытается предсказать.
# Функция потерь для этой конкретной модели -
sparse_categorical_crossentropy, хороша для задач классификации мультикласса,
подобных этой. В нашем случае, если модель предсказывает, что изображение
имеет лишь небольшую вероятность быть его фактической меткой, это приведет к
большим потерям.

# Создадим объект взаимодействия истории
# Получение модели (через цикл обучения модели - эпохи чем больше, тем лучше)
# validation_split=0.1 - тонкая настройка обучения, с функцией поиска
переобученного нейрона
# Мы передаем валидационные данные для x_train, y_train
history = model.fit(x_train, y_train, epochs=10, validation_split=0.1,
callbacks=[early])

plt.plot(history.history["loss"], label="Потери")
plt.plot(history.history["val_loss"], label="Потери Val")
plt.title("Потери в обучении")
# Подпишем оси
plt.xlabel("Эпоха")
plt.ylabel("Потеря")
```

```
plt.legend()
plt.show()
```





Графики
7а – «Потери в обучении Моего алгоритма

6б – «Точность обучения с моего алгоритма

```
Epoch 1/5
704/704 [==============================] - 103s 144ms/step - loss: 2.0593 - accuracy: 0.4794 - val_loss: 1.6691 -
val_accuracy: 0.5422
Epoch 2/5
704/704 [==============================] - 104s 147ms/step - loss: 1.4654 - accuracy: 0.6419 - val_loss: 1.2757 -
val_accuracy: 0.6788
Epoch 3/5
704/704 [==============================] - 103s 147ms/step - loss: 1.2025 - accuracy: 0.7035 - val_loss: 1.2442 -
val_accuracy: 0.6910
Epoch 4/5
704/704 [==============================] - 103s 147ms/step - loss: 1.0491 - accuracy: 0.7439 - val_loss: 1.0984 -
val_accuracy: 0.7286
Epoch 5/5
704/704 [==============================] - 104s 148ms/step - loss: 0.9740 - accuracy: 0.7662 - val_loss: 1.0696 -
val_accuracy: 0.7428
```

Отличный результат! Уверенно могу сказать, что потери в обучении при такой комбинации применения методов становятся с каждой эпохой всё ниже и ниже, а точность данных при этом возрастает почти экспоненциально.

Теперь мы можем проверить любой код на скорость с новой комбинацией методов согласно моему алгоритму. Нельзя забывать и об уменьшении скорости обучения из-за метода L2 регуляризации, требующего аппаратных возможностей, но тем не менее цель достигнута, алгоритм создан.

Результат проектной деятельности

К описанию своего исследовательского проекта прилагаю успешные программные коды экспериментов и как результат мой алгоритм и модель успешного обучения нейросетей.

Значимость проекта

О значимости проекта судить рано, еще много работы. Надо обязательно проработать скоростные способы аппаратного взаимодействия с алгоритмом, но уверенно могу сказать, что обучение нейросети по моим экспериментам и итоговому алгоритму может пригодиться для изучения особенностей построения слоёв моделей нейросетей.

Заключение

В принципе, всё уже сказано. Модель сформирована, алгоритм создан и функционирует, хотя не очень быстро, но эффективно, снижая потери и увеличивая точность обучения нейронных сетей. Цель достигнута.

Благодарности

В заключении я хотел бы поблагодарить всех участников моего исследовательского проекта. Прежде всего, моего учителя информатики Катаева Виктора Борисовича, который научил меня программировать на языке Python. Моё обучение в профильном классе и посещение факультативов приносит свои плоды. Я с легкостью умею учиться сам.

Так же благодарю своего второго руководителя, доцента кафедры самолётостроения и эксплуатации авиационной техники ИРНИТУ, кандидата технических наук Зотова Игоря Николаевича, за консультирование и теоретическую поддержку терминологии нейросетей. С его помощью я утратил чувство «страха» перед различными терминами, связанными с глубоким обучением нейросетей, которые часто скрываются или остаются необъяснимыми во многих введениях в эту технологию.

Список использованных источников

1. Глубокое обучение: легкая разработка проектов на Python. — СПб.: Питер, 2021. — С. 272
2. Искусственный интеллект и Машинное обучение / Т. Казанцев / «ЛитРес: Самиздат – 2023. – С.409.
3. Введение в машинное обучение: Учебник. – Алматы / Мухамедиев Р.И., Амиргалиев Е.Н., 2022. – С. 252
4. Статья, Академия Яндекс, Учебник по машинному обучению, «Тонкости обучения»/ Нейчев Радослав URL: https://academy.yandex.ru/handbook/ml/article/tonkosti-obucheniya
5. Статья, Кафедра математического обеспечения и применения ЭВМ в СПбГЭТУ "ЛЭТИ"/ URL: https://se.moevm.info/doku.php
6. Статья, ресурс Python-school / «L1 и L2 регуляризация» / URL: https://python-school.ru/blog/regularization-l1-l2/
7. Статья, Хабр/ «Обзор Keras для Tenserflow »URL: https://habr.com/ru/articles/482126/
8. Статья, Хабр, Как создать свою собственную нейронную сеть с нуля на Python URL: https://habr.com/ru/articles/725668/
9. Статья, PytonRu.com, нейронная сеть на практике с Python и Keras URL: https://pythonru.com/primery/nejronnaja-set-na-praktike-s-python-i-keras
10. Статья, образовательный портал OTUS, Простейшая нейронная сеть на Python URL: https://otus.ru/nest/post/1247/