УДК 004

Разработка системы распознавания бутылок молока на основе технологий машинного обучения

Вихляев Дмитрий Романович
Приамурский государственный университет имени Шолом-Алейхема
Студент

Аннотация
Данная статья посвящена изучению и анализу современных методов классификации предметов на изображении и их применения в различных сферах. Система разработана на языке программирования python в среде google colab, с использованием библиотек машинного обучения и компьютерного зрения. Результатом исследования станет программа способная распознавать и классифицировать бутылки молока на видеозаписи.
Ключевые слова: распознавание изображений, нейронные сети, классификация.

Development of a milk bottle recognition system based on machine learning technologies

Vikhlyaev Dmitry Romanovich
Sholom-Aleichem Priamursky State University
Student

Abstract
This article is devoted to the study and analysis of modern methods of classifying objects in the image and their application in various fields. The system is developed in the python programming language in the Google colab environment, using machine learning and computer vision libraries. The result of the study will be a program capable of recognizing and classifying milk bottles on video recordings.
Keywords: image recognition, neural networks, classification.

1 Введение

1.1 Актуальность

Распознавание и классификация бутылок молока имеют практическую значимость в нескольких областях, включая промышленное производство молочных продуктов и улучшение качества обслуживания покупателей в магазинах, например, для улучшения качества обслуживания покупателей:

Магазины могут использовать распознавание и классификацию бутылок молока для сортировки товаров на полках магазина. Также, на кассах, пользователи могут использовать мобильные приложения с технологиями распознавания для быстрого и точного выявления цены и характеристик

продукта. Это способствует увеличению скорости обслуживания покупателей и улучшению их опыта покупок.

1.2Обзор исследований

С.А.Петров, С.А.Ночвай, В.И.Ведейкис, Т.С.Куренкова, применили алгоритм "обучение с учителем" в спецификации для разработки процесса обучения нейронной сети типа feedforward [1]. В.А.Шустов применил параллельные алгоритмы обучения нейронной сети, использующие равномерный критерий качества обучения [2]. Д.А.Бородкина, А.В.Колпакова, Г.Г. Рапаков экспериментировали возможности машинного обучения для распознавания рукописных цифр при обучении программированию нейронных сетей [3]. Л.В.Яковенко, А.В.Плиско покозали взаимосвязь между понятиями «нейронные сети», «глубокое обучение», «машинное обучение» и «искусственный интеллект» [4]. В.А. Головко, А.А.Крощенко, А.А.Михно, А.М.Соловчук адаптировали шаг обучения для неконтролируемого обучения глубоких нейронных сетей [5]. П.В.Терелянский, Е.И.Брагина запустили процесс обучения нейронной сети для определения товарных характеристик [6]. В.М.Лукьянчиков описал алгоритм обучения многослойной нейронной сети методом обратного распространения ошибки [7]. С.Г.Тихомиров, А.А.Адаменко Продемонстрировали опыт использования эволюционных алгоритмов в обучении искусственных нейронных сетей [8]. В.В.Лебедев, О.Л.Чернышев, А.Р.Хабаров Задали топологии и обучили нейронные сети в пакете brainmaker [9]. К.А.Ерш показал степень необходимости нормализации данных при обучении нейронных сетей [10].

1.3Цель исследования

Цель исследования – провести исследование в создании искусственного интеллекта и написать программу классификатор, способную распознавать и классифицировать бутылки молока на видеозаписи.

2 Материалы и методы

Для реализации поставленной задачи использовались язык программирования Python и модуль Tensorflow. В качестве инструмента для обработки изображений используется библиотека OpenCV.

3 Результаты и обсуждения

Первым этапом работы является создание набора данных. Для создания качественного набора данных необходимо обладать достаточным количеством этих данных. Чем больше данных, тем точнее и эффективнее будет работать нейронная сеть. В результате работы удалось собрать около 1200 изображений бутылок молока четырех производителей (рис. 1).

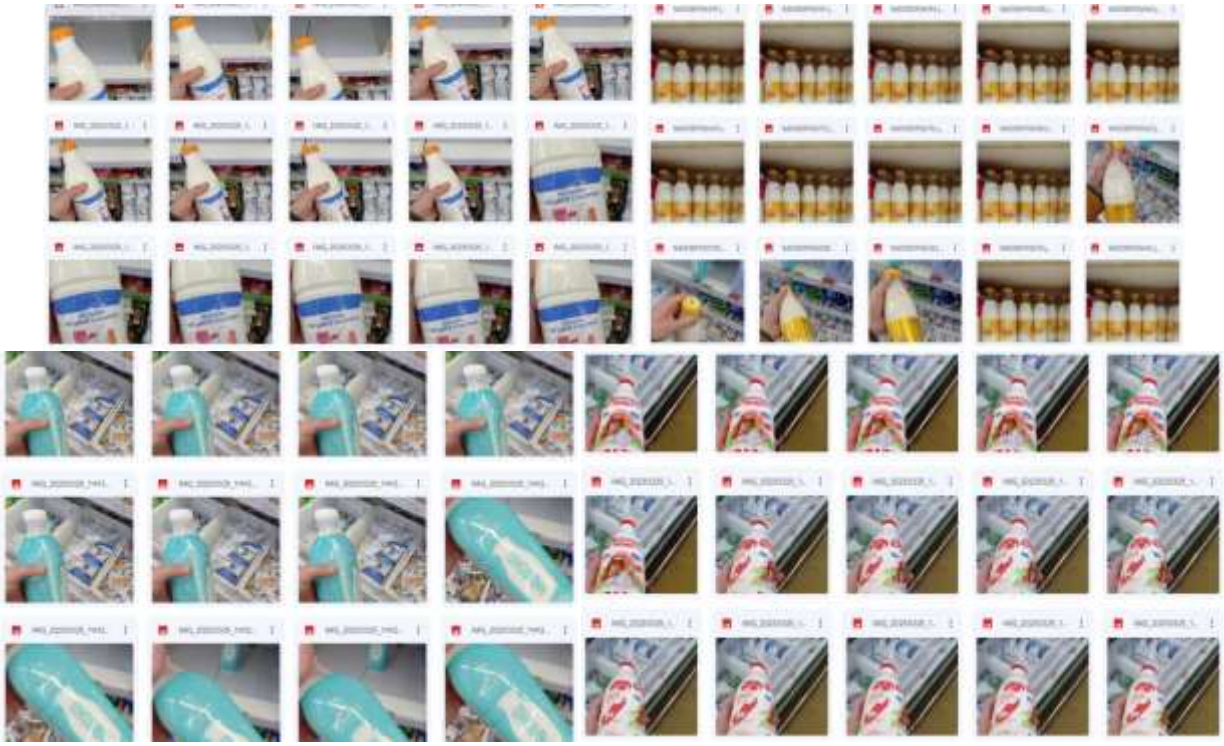

Рис. 1. Набор собранных изображений

В следующем этапе необходимо подготовить данные к использованию в обучении нейронной сети. Чтобы нейронная сеть могла обучиться распознаванию изображений, ей необходимо показывать только объекты, как можно больше заполняющие картинку. Поэтому первым делом нужно избавиться от фона путём обрезания изображения. Но для этого необходимо обнаружить сам объект на картинке и получить его координаты.

YOLO - это алгоритм обнаружения объектов в режиме реального времени, который использует нейронную сеть для анализа изображения и определения координат и классов объектов, находящихся на изображении.

YOLO широко применяется в задачах компьютерного зрения, таких как распознавание объектов, обнаружение и отслеживание объектов, чтобы обеспечить безопасность на дорогах, обнаружение людей на складах, анализ объектов на космических изображениях, классификация животных и многие другие. YOLO имеет преимущества по сравнению с CNN, такие как скорость обработки изображений, но может показывать худшие результаты в распознавании мелких объектов.

Выше описанный алгоритм YOLO уже имеет готовую модель способную распознавать бутылки на фотографиях.

В языке программирования python можно с помощью модуля «ultralytics» использовать класс YOLO. В данном исследовании используется 8 наномодель версия, обеспечивающая высокую скорость распознавания и малый затрат ресурсов. Написав несколько строчек кода можно отправлять изображения модели для распознавания. Результатом работы модели станут списки координат и названий распознанных объектов (рис.2).

```
from ultralytics import YOLO
yolo_model=YOLO('yolov8n.pt')
results=model(img)
```

Рис. 2. Использование модели YOLO в языке программирования python

Далее над набором изображений проводится ряд фильтров с помощью библиотеки openCV. Сначала необходимо обрезать картинки согласно полученным координатам. Затем изменить размеры изображения до стандартного. Если высота и ширина изображения в пикселях равны, то такое изображение является приемлемым для загрузки в нейронную сеть. Сами же размеры необходимо подобрать самостоятельно. Чем больше пикселей имеет изображение, тем точнее нейронная сеть сможет выделить контуры и тем больше памяти потребуется для обучения. Для данной задачи изображения приводились к размеру 256 на 256 пикселей. Затем картинка обесцвечивалась. Обесцвечивание изображений может помочь убрать шум и улучшить качество изображений для последующего обучения модели. После того как будут проведены вышеперечисленные действия над всем набором изображений, датасет можно будет использовать для обучения модели (рис.3,4).

```
def create_dataset(path,new_path):
  for i in os.listdir(path):
    count = 0
    for j in os.listdir(f"{path}/{i}"):
      img=cv2.imread(f"{path}/{i}/{j}")

      results=model(img)
      for r in results:
        boxes=r.boxes
        for box in boxes:
          if(box.cls[0]==39):
            x1,y1,x2,y2=box.xyxy[0]
            x1,y1,x2,y2=int(x1),int(y1),int(x2),int(y2)
            break

      cropped = img[y1:y2, x1:x2]
      resized = cv2.resize(cropped, (256,256), interpolation = cv2.INTER_AREA)
      gray_image = cv2.cvtColor(resized, cv2.COLOR_BGR2GRAY)
      cv2.imwrite(f'{new_path}/{i}/{count}.jpg',gray_image)
      count+=1
```

Рис. 3. Создание обработанного датасета

Рис. 4. Обработанный датасет

Разделение данных на тренировочную и валидационную выборку помогает проверить качество обучения модели и оценить ее способность обобщаться на новые данные. Для этого тренировочная выборка используется для обучения нейронной сети, а валидационная – для проверки ее качества. Это позволяет избежать переобучения и увеличить точность предсказаний модели. Данные разделяются случайно в отношении 8:2. Сначала изображения загружаются как объекты кдасса PIL, затем преобразуются в массив numpy (рис. 5).

```
def load_dataset(path):
  # определите список классов
  class_names = []
  # определите список для хранения изображений
  images = []
  # определите список для хранения меток (class labels)
  labels = []
  classes={'ГринАрго':0,'Лазовское':1,'Сельские мативы':2,'Цельное Отборное':3}
  # для каждого класса
  for class_name in os.listdir(path):
    # для каждого изображения в классе
    for image_filename in os.listdir(f"{path}/{class_name}"):
      # загрузите изображение
      path_img=f"{path}/{class_name}/{image_filename}"
      image = load_img(path_img, target_size=(256, 256), color_mode='grayscale')
      # преобразуйте изображение в numpy-массив
      image = np.array(image)
      # добавьте изображение в список изображений
      images.append(image)
      # добавьте метку класса в список меток
      labels.append(classes[class_name])
    # преобразуйте списки images и labels в numpy-массивы
  X = np.array(images)
  y = np.array(labels)
  # разделите данные на обучающие и тестовые наборы
  X_train, X_test, y_train, y_test = train_test_split(
      X,
      y,
      test_size=0.2,
      random_state=42)
  # верните два кортежа: (X_train, y_train) и (X_test, y_test)
  return (X_train, y_train), (X_test, y_test)
```

Рис. 5. Разделение данных на тренировочные и валидационные

Нейронные сети, особенно с использованием функций активации, лучше работают с данными масштабированными в диапазон от 0 до 1. Это происходит потому, что этот диапазон позволяет сети лучше улавливать различия между значениями входных данных и более эффективно использовать ресурсы для обработки этих данных. Если значения данных не находятся в этом диапазоне, может потребоваться масштабирование, чтобы дать нейронной сети лучшую возможность правильно обрабатывать данные.

Правильные ответы содержат номер класса, который находится на изображении. Создаваемая нейронная сеть выдаёт 4 значений (по количеству нейронов) с вероятностями, что на картинке принадлежит номеру класса. Для корректного обучения нейронной сети, нужно преобразовать данные правильных ответов из номеров классов в формат быстрого кодирования (рис.6).

```
img_train = img_train / 255
img_test = img_test / 255
```

```
label_train = utils.to_categorical(label_train, 4)
label_test = utils.to_categorical(label_test, 4)
classes={'ГринАрго':0,'Лазовское':1,'Сельские мативы':2,'Цельное Отборное':3}
```

```
r=0
for i in range(8):
  print(r,label_train[i])
  r+=1
```

```
0 [0. 0. 1. 0.]
1 [0. 0. 0. 1.]
2 [1. 0. 0. 0.]
3 [0. 0. 1. 0.]
4 [1. 0. 0. 0.]
5 [0. 0. 0. 1.]
6 [0. 0. 0. 1.]
7 [1. 0. 0. 0.]
```

Рис. 6. Преобразование данных изображении и классах в удобную для нейронную сеть форму

При построении модели сверточной нейронной сети необходимо учитывать следующие важные аспекты:

1. Архитектура сети: необходимо выбрать оптимальную архитектуру сети, которая сочетает в себе оптимальное количество сверточных слоев, пулинг слоев и полносвязных слоев. Также необходимо выбрать параметры каждого слоя, такие как количество фильтров, их размер, функцию активации и размер ядра пулинга.
2. Размерность входных данных: размерность входных данных, таких как изображения или аудиофайлы, должна быть определена правильно с точки зрения выбранной архитектуры. Например, размер изображения должен быть кратен размерности ядер сверточных фильтров.
3. Размерность выходных данных: необходимо знать ожидаемую размерность выходных данных, которая зависит от архитектуры и типа задачи, которую нужно решить.
4. Обучающие данные: важно иметь большое количество разнообразных обучающих данных, чтобы обеспечить эффективное обучение модели.
5. Функция потерь: необходимо выбрать оптимальную функцию потерь, которая соответствует типу задачи, например, кросс-энтропия, MSE и т.д.
6. Оптимизатор: оптимизатор используется для обновления весов сети в процессе обучения. Необходимо выбрать оптимальный оптимизатор, который обеспечит эффективность обучения и скорость сходимости.

7. Регуляризация: для предотвращения переобучения необходимо использовать методы регуляризации, такие как Dropout или L1/L2 регуляризация.
8. Оценка результатов: необходимо выбрать оптимальную метрику для оценки результатов модели, например, точность, F-мера, ROC-AUC и т.д.
9. Выбор гиперпараметров: при создании модели необходимо выбрать целый ряд гиперпараметров, таких как скорость обучения, коэффициенты регуляризации и количество фильтров, которые могут сильно влиять на качество работы модели, поэтому их нужно экспериментально настраивать.

Ниже приведены примеры созданных моделей, показавшие хорошие результаты (рис. 7).

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(256,256,1)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(4, activation='softmax'))

# компилируем модель и запускаем обучение
model.compile(loss='categorical_crossentropy',
optimizer=RMSprop(),
metrics=['accuracy'])

model.fit(img_train, label_train,batch_size=32,epochs=20,verbose=1,validation_data=(img_test, label_test))
```

```
model = Sequential()

model.add(Dense(1000, input_dim=65536, activation="relu"))

model.add(Dense(4, activation="softmax"))
```

```
model.compile(
    loss="categorical_crossentropy",
    optimizer="SGD",
     metrics=["accuracy"]
     )
model.summary()
```

Рисунок 7. Использованные модели нейронной сети

Conv2D – это слой свертки (convolutional layer) в библиотеке Keras в Python. Он используется в нейронных сетях для обработки изображений и видео, и позволяет находить визуальные признаки в изображений, такие как границы, формы, текстуры и др. Слой Conv2D используется для изучения локальных пространственных шаблонов в изображениях. Он принимает на вход изображение в виде тензора и применяет несколько фильтров (ядро свертки) по всему изображению, чтобы извлечь признаки из него. Каждый фильтр извлекает определенные признаки, которые в дальнейшем могут использоваться для классификации, детекции объектов и т.д. Conv2D имеет два основных параметра - количество фильтров и размер ядра свертки. Чем

больше фильтров и размер ядра, тем более сложные признаки могут быть извлечены из изображения.

"Flatten" в Keras – это слой, который принимает входные данные многомерного массива и преобразует их в одномерный массив.

Этот слой используется для того, чтобы обеспечить совместимость между различными слоями нейронных сетей. Например, входной слой сверточной нейронной сети ожидает многомерные данные, в то время как полносвязный слой (Dense) принимает только одномерный массив. В этом случае слоя Flatten используется для преобразования многомерных данных на выходе сверточного слоя в одномерный массив, который можно передать в Polnosvyazny dense слой или любой другой слой совместимый с одномерными данными.

Dense – слой в библиотеке Keras является основным слоем нейронной сети, который используется для полносвязных (fully connected) слоев. Он имеет также и другие названия, такие как Linear или Affine слой. Dense слой принимает входной тензор и выполняет линейное преобразование над ним, используя веса и смещения, которые определяются в процессе обучения. Веса представляют важность каждого входного признака для выходного значения, а смещения - константу, которая помогает сместить сумму входных значений.

ReLU – является нелинейной функцией, которая возвращает максимальное значение между 0 и входным значением x. Это означает, что если входное значение меньше или равно нулю, то выходное значение также будет равно нулю, а если входное значение больше нуля, то выходное значение будет равно входному значению. ReLU обычно используется в сверточных нейронных сетях для извлечения признаков из изображений.

Softmax – также является нелинейной функцией и используется для многоклассовой классификации. Функция softmax принимает вектор входных значений и преобразует его в вектор вероятностей, сумма которых равна единице. Это означает, что каждый элемент выходного вектора представляет вероятность принадлежности к определенному классу. Softmax используется в конечном слое нейронных сетей для определения класса, к которому относится входной образец.

Ниже приведён код построения и процесс обучения лучшей модели (рис. 8,9).

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(256,256,1)))
model.add(Flatten())
model.add(Dense(50, activation='relu'))
model.add(Dense(4, activation='softmax'))
```

```
# компилируем модель и запускаем обучение
model.compile(loss='categorical_crossentropy',
optimizer="SGD",
metrics=['accuracy'])
```

```
custom_early_stopping = EarlyStopping(
    monitor='val_accuracy',
    patience=8,
    min_delta=0.001,
    mode='max'
)
```

```
model.fit(
    img_train,
     label_train,
     batch_size=32,
     epochs=20,
     verbose=1,
     callbacks=[custom_early_stopping],
     validation_data=(img_test, label_test))
```

Рисунок 8. Модель показавшая лучший результат

```
Epoch 1/20
33/33 [==============================] - 124s 4s/step - loss: 1.5930 - accuracy: 0.3880 - val_loss: 1.2559 - val_accuracy: 0.7235
Epoch 2/20
33/33 [==============================] - 110s 3s/step - loss: 1.2512 - accuracy: 0.4844 - val_loss: 1.2785 - val_accuracy: 0.4962
Epoch 3/20
33/33 [==============================] - 105s 3s/step - loss: 1.0732 - accuracy: 0.5573 - val_loss: 0.9520 - val_accuracy: 0.5741
Epoch 4/20
33/33 [==============================] - 110s 3s/step - loss: 0.6184 - accuracy: 0.7716 - val_loss: 0.4953 - val_accuracy: 0.8220
Epoch 5/20
33/33 [==============================] - 108s 3s/step - loss: 0.3291 - accuracy: 0.8910 - val_loss: 0.2614 - val_accuracy: 0.9350
Epoch 6/20
33/33 [==============================] - 119s 4s/step - loss: 0.1453 - accuracy: 0.9621 - val_loss: 0.1962 - val_accuracy: 0.9356
Epoch 7/20
33/33 [==============================] - 111s 3s/step - loss: 0.0909 - accuracy: 0.9735 - val_loss: 0.1072 - val_accuracy: 0.9659
Epoch 8/20
33/33 [==============================] - 108s 3s/step - loss: 0.0524 - accuracy: 0.9905 - val_loss: 0.0939 - val_accuracy: 0.9659
Epoch 9/20
33/33 [==============================] - 108s 3s/step - loss: 0.0285 - accuracy: 0.9972 - val_loss: 0.0617 - val_accuracy: 0.9886
Epoch 10/20
33/33 [==============================] - 109s 3s/step - loss: 0.0196 - accuracy: 1.0000 - val_loss: 0.0530 - val_accuracy: 0.9848
Epoch 11/20
33/33 [==============================] - 122s 4s/step - loss: 0.0155 - accuracy: 0.9991 - val_loss: 0.0668 - val_accuracy: 0.9773
Epoch 12/20
33/33 [==============================] - 111s 3s/step - loss: 0.0107 - accuracy: 1.0000 - val_loss: 0.0795 - val_accuracy: 0.9773
Epoch 13/20
33/33 [==============================] - 109s 3s/step - loss: 0.0088 - accuracy: 1.0000 - val_loss: 0.0638 - val_accuracy: 0.9773
Epoch 14/20
33/33 [==============================] - 111s 3s/step - loss: 0.0080 - accuracy: 1.0000 - val_loss: 0.0558 - val_accuracy: 0.9811
Epoch 15/20
33/33 [==============================] - 112s 3s/step - loss: 0.0065 - accuracy: 1.0000 - val_loss: 0.0502 - val_accuracy: 0.9848
Epoch 16/20
33/33 [==============================] - 111s 3s/step - loss: 0.0055 - accuracy: 1.0000 - val_loss: 0.0464 - val_accuracy: 0.9848
Epoch 17/20
33/33 [==============================] - 111s 3s/step - loss: 0.0052 - accuracy: 1.0000 - val_loss: 0.0490 - val_accuracy: 0.9811
<keras.callbacks.History at 0x7f7442ee2f80>
```

Рисунок 9. Процесс обучения нейронной сети

Ниже приведены результаты работы нейронной сети на валидационных данных. Результаты показывают точность распознания всегда выше восьмидесяти процентов. (рис.10).

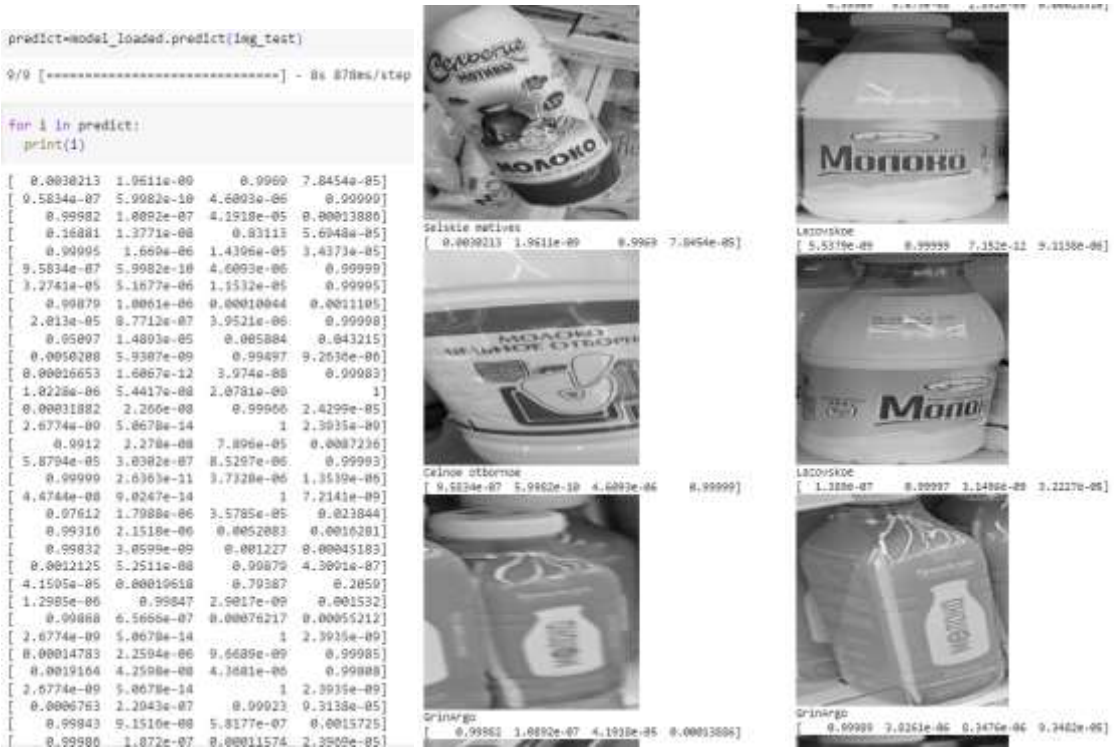

Рисунок 10. Результат работы нейронной сети на валидационных данных

Путём получения кадров из видео посредством библиотеки OpenCV, и преобразованием их к данным, с которыми работает нейронная сеть, программа позволяет распознавать бутылки молока на видео. Результат работы нейронной сети приставлен ниже (рис.11,12,13).

Рисунок 11. Результат работы нейронной сети на видео

Рисунок 12. Результат работы нейронной сети на видео

Рисунок 13. Результат работы нейронной сети на видео

Проанализировав результаты рабаты программы на реальных примерах, можно сделать выводы, что нейронная сеть показывает менее хороший результат, чем при обучении, так как результат распознания на восемьдесят процентов практически никогда не достигается. Тем не менее при определённом освещении и угле обзора программа показывает довольно точные результаты, что достаточно хорошо для бытового использования.

В результате экспериментов было установлено, что сверточные нейронные сети дают лучшие результаты в задачах классификации объектов на изображении. Также было установлено, что использование методов улучшения обучения, таких как аугментация данных и регуляризация, позволяет повысить качество обучения сети.

В заключении следует отметить, что обучение нейронной сети классификации объектов на изображении является актуальной задачей в современных исследованиях в области машинного обучения и компьютерного зрения. Данная работа является важным шагом на пути к более точным и эффективным методам классификации объектов на изображении, и может быть использована в качестве основы для будущих исследований в данной области.

Библиографический список

1. Петров С.А., Ночвай С.А., Ведейкис В.И., Куренкова Т.С. Разработка процесса обучения нейронной сети типа feedforward с применением алгоритма "обучение с учителем" в спецификации aris // В сборнике: наука россии: цели и задачи. сборник научных трудов по материалам X международной научной конференции. Международная Объединенная Академия Наук. 2018. С. 5-7.
2. Шустов В.А. Параллельные алгоритмы обучения нейронной сети, использующие равномерный критерий качества обучения // В сборнике: математическое моделирование и краевые задачи. Труды Всероссийской научной конференции. Редколлегия: В. П. Радченко (отв. редактор), Э. Я. Рапопорт, Е. Н. Огородников (отв. секретарь), М. Н. Саушкин (отв. секретарь). 2004. С. 271-274.
3. Бородкина Д.А., Колпакова А.В., Рапаков Г.Г. Экспериментальное исследование возможностей машинного обучения для распознавания рукописных цифр при обучении программированию нейронных сетей // В сборнике: Интеллектуально-информационные технологии и интеллектуальный бизнес (ИНФОС-2022). Материалы Тринадцатой Международной научно-технической конференции. Ответственный редактор В.А. Горбунов. Вологда, 2022. С. 130-133.
4. Яковенко Л.В., Плиско А.В. Взаимосвязь понятий «нейронные сети», «глубокое обучение», «машинное обучение» и «искусственный интеллект» // В сборнике: Формирование надпрофессиональных навыков молодежи в Крыму. Сборник материалов региональной конференции студентов, магистрантов, аспирантов, молодых ученых. В рамках реализации проекта

«Вместе с будущей Россией!». Симферополь, 2020. С. 245-256.
5. Головко В.А., Крощенко А.А., Михно Е.В., Соловчук А.М. Адаптивный шаг обучения для неконтролируемого обучения глубоких нейронных сетей // В сборнике: Информационные системы и технологии. Материалы международного научного конгресса по информатике. В 3-х частях. Редколлегия: С.В. Абламейко (гл. ред.) [и др.]. Минск, 2022. С. 117-122.
6. Терелянский П.В., Брагина Е.И. Процесс обучения нейронной сети для определения товарных характеристик // В сборнике: аграрная наука - основа успешного развития апк и сохранения экосистем. материалы международной научно-практической конференции. 2012. с. 329-333.
7. Лукьянчиков В.М. Описание алгоритма обучения многослойной нейронной сети методом обратного распространения ошибки // В сборнике: Интеграция современных научных исследований в развитие общества. Международная научно-практическая конференция: в 2-х томах. 2016. С. 164-166.
8. Тихомиров С.Г., Адаменко А.А. Опыт использования эволюционных алгоритмов в обучении искусственных нейронных сетей // В сборнике: материалы lv отчетной научной конференции преподавателей и научных сотрудников вгуит за 2016 год. в 3 частях. воронеж, 2017. с. 84-86.
9. Лебедев В.В., Чернышев О.Л., Хабаров А.Р. Задание топологии и обучение нейронной сети в пакете brainmaker // В сборнике: Информационные ресурсы и системы в экономике, науке и образовании. Сборник статей VIII Международной научно-практической конференции. 2018. С. 56-60.
10. Ерш К.А. Степень необходимости нормализации данных при обучении нейронных сетей // В сборнике: научно-технический прогресс как фактор развития современного общества. сборник статей международной научно-практической конференции. 2019. с. 28-30.