Национальный исследовательский ядерный университет «МИФИ»

Институт Интеллектуальных Кибернетических Систем

Кафедра «Компьютерные системы и технологии»

Отчёт по дисциплине

«Компьютерная графика и обработка изображений»

Работу выполнил: Кондахчан Микаэл Арсенович

Преподаватель: Петрова Алена Игоревна

Москва 2024 г.

Техническое задание

В качестве задания выбрана классификация типов автомобилей. Классификация будет производится с помощью сверточной нейронной сети. Набор данных для обучения и валидации модели – **Vehicle Type Image Dataset**– найден на Kaggle (https://www.kaggle.com/datasets/sujaykapadnis/vehicle-type-image-dataset). Данными являются фотоснимки автомобилей с разрешением от 260×160 до 300×190, снятые таким образом, что почти ничего, кроме самого автомобиля на снимке нет.

Пример изображения из набора данных:

Данные являются размеченными. В процессе выполнения задания будут отобраны те классы, в которых данных достаточно для обучения модели. Т.е. количество классов будет в пределах типов автомобилей, представленных в наборе данных + unknown (для типов, в которых недостаточно данных и уже имеющейся категории others). В качестве технологий, использующихся для выполнения задания выбраны язык программирования python, библиотеки, необходимые для анализа данных (pandas, mathplotlib, plotly и др.) и библиотеки, связанные с глубоким обучением (keras, pytotrch и др.).

Оглавление

1 Используемая теория

Первостепенной задачей являлось создание, обучение и тестирование нейросети, которая будет определять класс по изображению в соответствии с исходным разбиением по классам, то есть: ['Hatchback', 'Pickup', 'SUV', 'Seden', 'other']. Поскольку для анализа изображений используютсся сверточные нейросети, именно такую создать было целью. Слоями таковой являются Convolutional, Pooling, Full connected. Первые два применяются для выделения признаков изображения, последний тип для непосредственной классификации. Также во избежание переобучения применялся Dropout. Опираясь на изученные материалы, первая попытка состояла из 4 схожих частей “свертки”. Далее проводились различные эксперименты.

Также в попытках увеличить точность модели применялось Data augmentation. Используемые преобразования – поворот, зум, изменение контрастности и яркости изображения.

Каких-либо конкретных способов подбирать конкретное количество и значение у параметров не было обнаружено, поэтому предполагалось все пытаться подбирать на практике.

Сами данные были приведены к общему виду – 160x160.

Первым делом, выбирая параметры у исходной модели, основывался на том, что для сверточных слоев самой частой матрицей является матрица 3x3. Поскольку четного размера матрицы не берутся исход я из того, что для фильтра нечетного размера все пиксели предыдущего слоя будут симметричны вокруг выходного пикселя. Без этой симметрии нам придется учитывать искажения между слоями, что происходит при использовании ядра четного размера. Поэтому фильтры ядра четного размера в большинстве случаев опускаются для упрощения реализации. А размер не выбран больше, поскольку все объекты машины и отличающие их особенности достаточно малы. При выборе количества фильтров сверточных слоев основывался на том, что советуется брать для первого слоя “небольшое количество”, а затем увеличивать его, поскольку изначальные данные — это сама картинка, содержащая всегда много шума, из которой нужно выделить особенности, а затем уже работа с этими особенностями, для чего увеличивают количество фильтров, чтобы добиться более сложных абстракций.

Pooling применяется для того, чтобы сократить количество параметров, что сокращает время обучения и предотвращает переобучение.

Fully connected слои применяются для самой классификации, используя выделенные особенности вышеперечисленными слоями.
Изменялось количество слоев, их параметры применялось batch normalization (метод, используемый для более быстрого и стабильного обучения нейронных сетей за счет нормализации входных данных слоев.) (есть отдельный файл со скриншотами, где различные вариации отображены)

2 Практическая реализация

Предобработка данных: есть слой Rescale, которые численное значение отвечающие за цвет пикселя переводит из 0-255 в 0-1, а также data augmentation, которая хоть и применялась не всегда, но состоит из тех же методов, что указаны выше.

Структура сети:

```
])

model = keras.Sequential([
    rescale,
    #data_augmentation,
    Conv2D(32, (3, 3), activation='relu', padding='same', name='conv_1'),
    MaxPooling2D((2, 2), name='maxpool_1'),
    Conv2D(64, (3, 3), activation='relu', padding='same', name='conv_2'),
    MaxPooling2D((2, 2), name='maxpool_2'),
    Conv2D(128, (3, 3), activation='relu', padding='same', name='conv_3'),
    MaxPooling2D((2, 2), name='maxpool_3'),
    Conv2D(128, (3, 3), activation='relu', padding='same', name='conv_4'),
    MaxPooling2D((2, 2), name='maxpool_4'),
    Flatten(),
    Dropout(0.5),
    Dense(units=128, activation="relu"),
    Dense(units=128, activation="relu"),
    Dense(units=5, activation="softmax", name='out')
])

model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

Вот так выглядит структура, от которой было принято решение оттолкнуться.

Вот вторая версия:

```
rescale = keras.Sequential(Rescaling(1./255))
data_augmentation = keras.Sequential([
  RandomFlip("horizontal"),
  RandomZoom((0.1,0.15))
])

modelf = keras.Sequential([
    rescale,
    data_augmentation,
    Conv2D(32, (3, 3), activation='relu', padding='same', name='conv_1'),
    Dropout(0.2),
    BatchNormalization(),
    MaxPooling2D((2, 2), name='maxpool_1'),
    Conv2D(64, (3, 3), activation='relu', padding='same', name='conv_2'),
    Dropout(0.2),
    BatchNormalization(),
    MaxPooling2D((2, 2), name='maxpool_2'),
    Conv2D(128, (3, 3), activation='relu', padding='same', name='conv_3'),
    Dropout(0.2),
    BatchNormalization(),
    MaxPooling2D((2, 2), name='maxpool_3'),
    Conv2D(128, (3, 3), activation='relu', padding='same', name='conv_4'),
    Dropout(0.2),
    BatchNormalization(),
    MaxPooling2D((2, 2), name='maxpool_4'),
    Flatten(),
    Dense(units=128, activation="relu"),
    Dropout(0.3),
    Dense(units=5, activation="softmax", name='out')
])

modelf.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

В ней использовано и data augmentation и batchnormalization, также есть еще в несколько других реализаций. Основными были приведенная и выше, а также эта:

```
])

model = keras.Sequential([
    rescale,
    #data_augmentation,
    Conv2D(32, (3, 3), activation='relu', padding='same', name='conv_1'),
    MaxPooling2D((2, 2), name='maxpool_1'),
    Conv2D(64, (3, 3), activation='relu', padding='same', name='conv_2'),
    MaxPooling2D((2, 2), name='maxpool_2'),
    #Conv2D(128, (3, 3), activation='relu', padding='same', name='conv_3'),
    #MaxPooling2D((2, 2), name='maxpool_3'),
    #Conv2D(128, (3, 3), activation='relu', padding='same', name='conv_4'),
    #MaxPooling2D((2, 2), name='maxpool_4'),
    Flatten(),
    Dropout(0.5),
    Dense(units=128, activation="relu"),
    Dense(units=5, activation="softmax", name='out')
])

model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

Данная упрощенная версия была выбрана вместо первой, т.к. ее показатели были несколько выше, хотя слоев гораздо меньше.

Разбиение датасета(20%):

```
batch_size = 32
img_height = 160
img_width = 160
data_dir = "./drive/MyDrive/Colab Notebooks/vehicle"

train_ds = keras.utils.image_dataset_from_directory(
  data_dir,
  validation_split=0.2,
  subset="training",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)

valid_ds = keras.utils.image_dataset_from_directory(
  data_dir,
  validation_split=0.2,
  subset="validation",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)
```

3 Тестирование

Первым делом, проводилось обучение и валидация(приводится только для двух последних приведенных выше версий):

```
model = keras.Sequential([
    rescale,
    #data_augmentation,
    Conv2D(32, (3, 3), activation='relu', padding='same', name='conv_1'),
    MaxPooling2D((2, 2), name='maxpool_1'),
    Conv2D(64, (3, 3), activation='relu', padding='same', name='conv_2'),
    MaxPooling2D((2, 2), name='maxpool_2'),
    #Conv2D(128, (3, 3), activation='relu', padding='same', name='conv_3'),
    #MaxPooling2D((2, 2), name='maxpool_3'),
    #Conv2D(128, (3, 3), activation='relu', padding='same', name='conv_4'),
    #MaxPooling2D((2, 2), name='maxpool_4'),
    Flatten(),
    Dropout(0.5),
    Dense(units=128, activation="relu"),
    Dense(units=5, activation="softmax", name='out')
])

model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

```
epochs=5
history = model.fit(
  train_ds,
  validation_data=valid_ds,
  epochs=epochs
)
```

```
Epoch 1/5
105/105 [==============================] - 231s 2s/step - loss: 1.0927 - accuracy: 0.6537 - val_loss: 0.3157 - val_accuracy: 0.8994
Epoch 2/5
105/105 [==============================] - 201s 2s/step - loss: 0.1879 - accuracy: 0.9431 - val_loss: 0.1385 - val_accuracy: 0.9569
Epoch 3/5
105/105 [==============================] - 200s 2s/step - loss: 0.0439 - accuracy: 0.9910 - val_loss: 0.0602 - val_accuracy: 0.9868
Epoch 4/5
105/105 [==============================] - 201s 2s/step - loss: 0.0100 - accuracy: 0.9979 - val_loss: 0.0618 - val_accuracy: 0.9880
Epoch 5/5
105/105 [==============================] - 202s 2s/step - loss: 0.0076 - accuracy: 0.9985 - val_loss: 0.0811 - val_accuracy: 0.9880
```

```
rescale = keras.Sequential(Rescaling(1./255))
data_augmentation = keras.Sequential([
  RandomFlip("horizontal"),
  RandomZoom((0.1,0.15))
])

modelf = keras.Sequential([
    rescale,
    data_augmentation,
    Conv2D(32, (3, 3), activation='relu', padding='same', name='conv_1'),
    Dropout(0.2),
    BatchNormalization(),
    MaxPooling2D((2, 2), name='maxpool_1'),
    Conv2D(64, (3, 3), activation='relu', padding='same', name='conv_2'),
    Dropout(0.2),
    BatchNormalization(),
    MaxPooling2D((2, 2), name='maxpool_2'),
    Conv2D(128, (3, 3), activation='relu', padding='same', name='conv_3'),
    Dropout(0.2),
    BatchNormalization(),
    MaxPooling2D((2, 2), name='maxpool_3'),
    Conv2D(128, (3, 3), activation='relu', padding='same', name='conv_4'),
    Dropout(0.2),
    BatchNormalization(),
    MaxPooling2D((2, 2), name='maxpool_4'),
    Flatten(),
    Dense(units=128, activation="relu"),
    Dropout(0.3),
    Dense(units=5, activation="softmax", name='out')
])

modelf.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

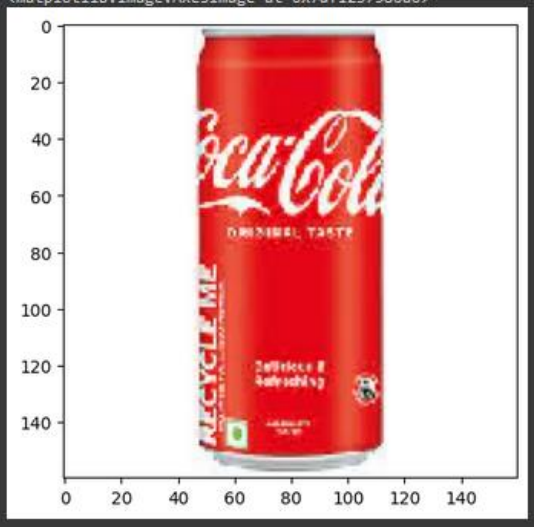
```
[22] from keras import callbacks
     earlystopping = callbacks.EarlyStopping(monitor="val_loss",
                                             mode="min", patience=2,
                                             restore_best_weights=True)
     epochs=10
     historyf = modelf.fit(
       train_ds,
       validation_data=valid_ds,
       epochs=epochs,
       callbacks=[earlystopping]
     )
```

```
Epoch 1/10
105/105 [==============================] - 360s 3s/step - loss: 1.1961 - accuracy: 0.6462 - val_loss: 15.5730 - val_accuracy: 0.1377
Epoch 2/10
105/105 [==============================] - 356s 3s/step - loss: 0.6004 - accuracy: 0.7609 - val_loss: 15.8793 - val_accuracy: 0.1377
Epoch 3/10
105/105 [==============================] - 358s 3s/step - loss: 0.4751 - accuracy: 0.8132 - val_loss: 9.6965 - val_accuracy: 0.2096
Epoch 4/10
105/105 [==============================] - 360s 3s/step - loss: 0.3467 - accuracy: 0.8734 - val_loss: 2.6595 - val_accuracy: 0.5281
Epoch 5/10
105/105 [==============================] - 359s 3s/step - loss: 0.3142 - accuracy: 0.8857 - val_loss: 4.4632 - val_accuracy: 0.4515
Epoch 6/10
105/105 [==============================] - 358s 3s/step - loss: 0.2465 - accuracy: 0.9066 - val_loss: 1.9848 - val_accuracy: 0.6275
Epoch 7/10
105/105 [==============================] - 355s 3s/step - loss: 0.2102 - accuracy: 0.9207 - val_loss: 1.8333 - val_accuracy: 0.6659
Epoch 8/10
105/105 [==============================] - 357s 3s/step - loss: 0.2281 - accuracy: 0.9213 - val_loss: 0.3906 - val_accuracy: 0.8683
Epoch 9/10
105/105 [==============================] - 357s 3s/step - loss: 0.1825 - accuracy: 0.9330 - val_loss: 0.3816 - val_accuracy: 0.8719
Epoch 10/10
105/105 [==============================] - 358s 3s/step - loss: 0.1598 - accuracy: 0.9401 - val_loss: 0.8119 - val_accuracy: 0.8060
```

Тестирование проводилось на картинках не входящих в датасет вовсе. Где как раз-таки и возникли трудности. Поскольку почти все причисляется к классу other, хоть и были случая определения автомобилей, но они были не корректными.

```
[70] from tensorflow.keras.preprocessing import image as image_utils
     def load_and_scale_image(image_path):
         image = image_utils.load_img(image_path, target_size=(160,160))
         return image
     image = load_and_scale_image('./drive/MyDrive/Colab Notebooks/test/can.jpg')
     plt.imshow(image, cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7df1237586a0>
```

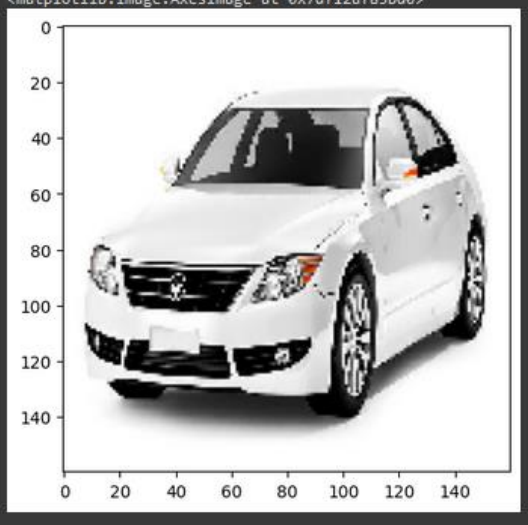



```
image = image_utils.img_to_array(image)
image = image.reshape(1,160,160,3)
prediction = modelf.predict(image)
class_names = train_ds.class_names
import numpy as np
print(class_names[np.argmax(prediction)])
```

```
1/1 [==============================] - 0s 41ms/step
Other
```

```
from tensorflow.keras.preprocessing import image as image_utils
def load_and_scale_image(image_path):
    image = image_utils.load_img(image_path, target_size=(160,160))
    return image
image = load_and_scale_image('./drive/MyDrive/Colab Notebooks/test/sed.jpg')
plt.imshow(image, cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7df12af85bd0>
```

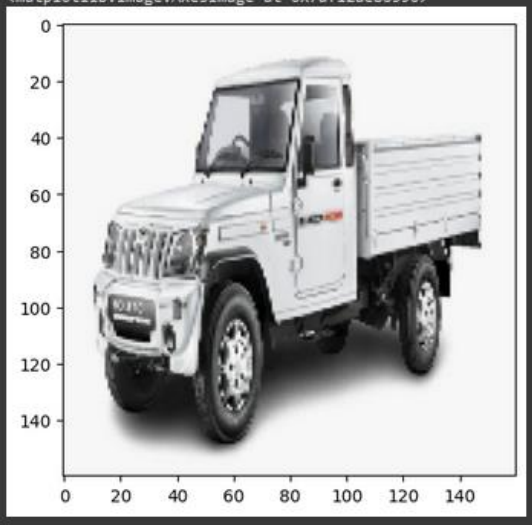



```
image = image_utils.img_to_array(image)
image = image.reshape(1,160,160,3)
prediction = modelf.predict(image)
class_names = train_ds.class_names
import numpy as np
print(class_names[np.argmax(prediction)])
```

```
1/1 [==============================] - 0s 37ms/step
Other
```

```
from tensorflow.keras.preprocessing import image as image_utils
def load_and_scale_image(image_path):
    image = image_utils.load_img(image_path, target_size=(160,160))
    return image
image = load_and_scale_image('./drive/MyDrive/Colab Notebooks/test/pickup.jpg')
plt.imshow(image, cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7df12ae86590>
```




```
image = image_utils.img_to_array(image)
image = image.reshape(1,160,160,3)
prediction = modelf.predict(image)
class_names = train_ds.class_names
import numpy as np
print(class_names[np.argmax(prediction)])
```

```
1/1 [==============================] - 0s 37ms/step
Other
```

```
[79] from tensorflow.keras.preprocessing import image as image_utils
def load_and_scale_image(image_path):
    image = image_utils.load_img(image_path, target_size=(160,160))
    return image
image = load_and_scale_image('./drive/MyDrive/Colab Notebooks/test/suv.jpg')
plt.imshow(image, cmap='gray')
```

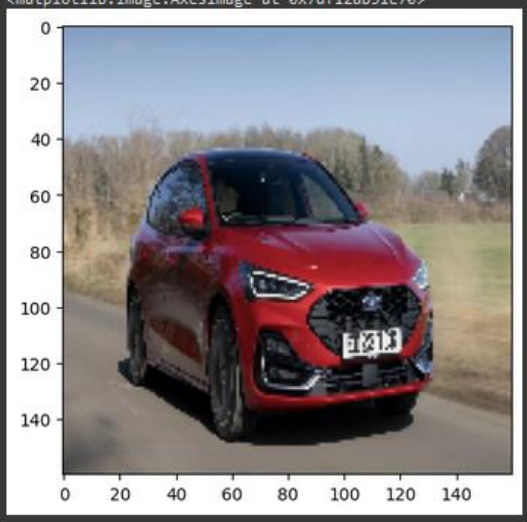
```
<matplotlib.image.AxesImage at 0x7df12ad335b0>
```


```
image = image_utils.img_to_array(image)
image = image.reshape(1,160,160,3)
prediction = modelf.predict(image)
class_names = train_ds.class_names
import numpy as np
print(class_names[np.argmax(prediction)])
```

```
1/1 [==============================] - 0s 41ms/step
Pickup
```

```
[126] from tensorflow.keras.preprocessing import image as image_utils
      def load_and_scale_image(image_path):
          image = image_utils.load_img(image_path, target_size=(160,160))
          return image
      image = load_and_scale_image('./drive/MyDrive/Colab Notebooks/test/4_3.jpg')
      plt.imshow(image, cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7df12ab51e70>
```




```
image = image_utils.img_to_array(image)
image = image.reshape(1,160,160,3)
prediction = modelf.predict(image)
class_names = train_ds.class_names
import numpy as np
print(class_names[np.argmax(prediction)])
```

```
1/1 [==============================] - 0s 44ms/step
Other
```

ЗАКЛЮЧЕНИЕ

Поскольку было проведено несколько проб и итераций, с различными параметрами у слоев, количество этих слоев, наличием и отсутствием data augmentation, а результат оставался прежним, тяжело точно сказать, в чем причина неудачи.

Первое, что оказалось странным, версия с 2 сверточными слоями выдавала 98% при валидации без data augmentation, когда с ней accuracy как train, так и valid падали до 30%. Хотя данное преобразование должно увеличивать показатели за счет искусственного расширения датасета таким образом. Та же проблема была и у первой версии, последняя вариации несколько смогла исправить данные показатели, хоть они и стали ниже. Предполагаю, что в первые попытки вышло лишь заставить модель запомнить данные, а не обучиться на них (переобучиться), из-за чего такое и происходило, а изначальный датасет возможно недостаточно разнообразный по своим фотографиям, чтобы было просто отличать один автомобиль от другого.

Тогда осмелюсь предположить, что в итоге параметры подобраны все же неудачно, раз с незнакомыми данными модель не понимает, что делать.

Также есть предположение, что проблемой может являться “несбалансированность датасета”, поскольку в классах не одинаковое количество изображений. А класс other содержит в себе и фото автомобилей, прочие объекты (в основном 2х колесный транспорт). Из-за чего и происходит определение большинства впервые увиденных изображений к этому классу.

Попытка переделать классификацию на большие, маленькие автомобили и other также не увенчалась успехом, проблема осталась та же.

По итогу, 2 предположения, возможно проблема кроется в датасете, из-за чего нужен более специфичный подбор параметров, услужение модели и обязательна data augmentation с большим числом изменяющих исходник факторов либо модель слишком усложнена, что привело к переобучению.

Это все исключая факт того, что сам по себе далеко не большой, что также является проблемой.

Список литературы

Medium, stackoverflow, towards data science, stackexchange, quora

https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/

https://www.tensorflow.org/api_docs/python/tf/keras

https://towardsdatascience.com/deciding-optimal-filter-size-for-cnns-d6f7b56f9363

https://www.geeksforgeeks.org/choose-optimal-number-of-epochs-to-train-a-neural-network-in-keras/

https://www.upgrad.com/blog/basic-cnn-architecture/