МИНОБРНАУКИ РОССИИ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ

ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

Кафедра МО ЭВМ

ОТЧЕТ По

индивидуальному

заданию

по дисциплине «Искусственные нейронные сети»

Тема: Fashion-MNIST

Студент гр. 7382 _____ Ленковский В.В.

Преподаватель _____ Жукова Н.А.

Санкт-Петербург

2020

Цель работы.

Реализация датасета Fashion-MNIST при помощи сверточной сети.

Описание датасета.

Наш датасет состоит из изображений размером 28x28, каждый пиксель которого представляет собой оттенок серого.

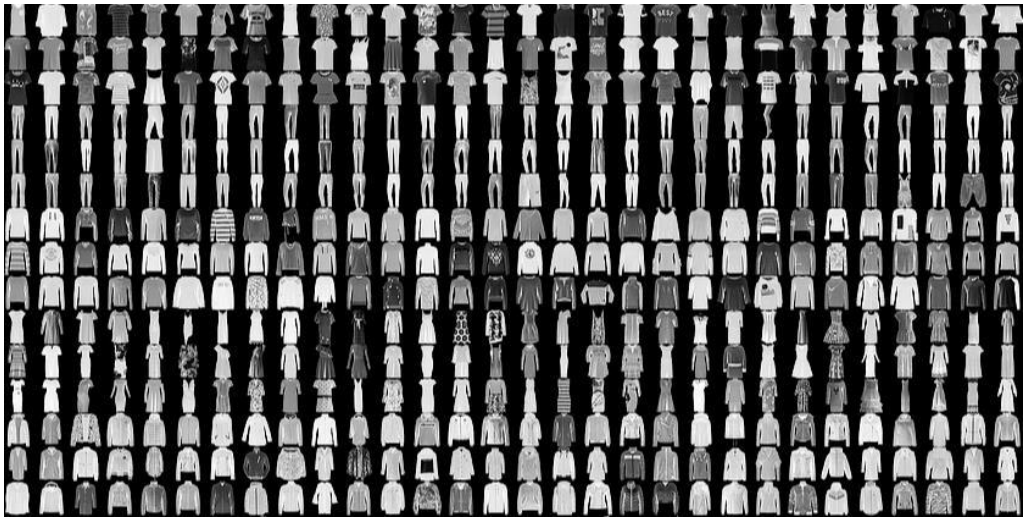

Рис. 1

Набор данных содержит изображения футболок, топов, сандалей и даже ботинок. Вот полный список того, что содержит в себе наш набор данных MNIST(рис. 2):

Метка	Класс
0	Футболка / топ
1	Шорты
2	Свитер
3	Платье
4	Плащ
5	Сандали
6	Рубашка
7	Кроссовки
8	Сумка
9	Ботинки

Рис. 2 – Классы одежды

Каждому входному изображению соответствует одна из перечисленных выше меток. Набор данных Fashion MNIST содержит 70 000 изображений. Из этих 70 000 мы воспользуемся 60 000 для тренировки нейронной сети и 10 000 для тестирования.

Callback:

Был добавлен callback ModelCheckpoint, для того чтобы сохранять нейронную сеть на каждой эпохе.

Создание модели:

Свёртка — процесс применения фильтра («ядра») к изображению. Операция подвыборки по максимальному значению — процесс уменьшения размеров изображения через объединение группы пикселей в единое максимальное значение из этой группы. Сейчас более подробно рассмотрим как это работает:

1. Допустим у нас есть изображение черно-белого цвета размером 6px на 6px;
2. Значение пикселей этой картинки будет представлено на рис. 3.

Значения пикселей					
1	0	4	2	125	67
8	2	5	4	34	12
20	13	25	15	240	2
76	8	6	6	100	76
34	66	134	223	201	3
255	123	89	55	32	2

Рис.3 – Значения пикселей

3. Суть свёртки заключается в создании другого набора значений, который называется ядром или фильтром. Свёрточный слой применяется к ядру и каждому участку

входного изображения. Для примера возьмем ядро 3x3(рис. 4) и пиксель со значением 25(рис. 5).

Ядро, 3 x 3 px		
1	2	1
2	4	2
1	2	1

Рис. 4 – Ядро

Значения пикселей					
1	0	4	2	125	67
8	2	5	4	34	12
20	13	25	15	240	2
76	8	6	6	100	76
34	66	134	223	201	3
255	123	89	55	32	2

Рис. 5 – Пиксель со значением 25

4. Центрируем ядро над этим пикселем. Теперь берём значения пикселей изображения и ядра, умножаем каждый пиксель изображения с соответствующим пикселем ядра и складываем все значения произведений, а результатирующее значение пикселя присваиваем новому изображению (рис. 6).

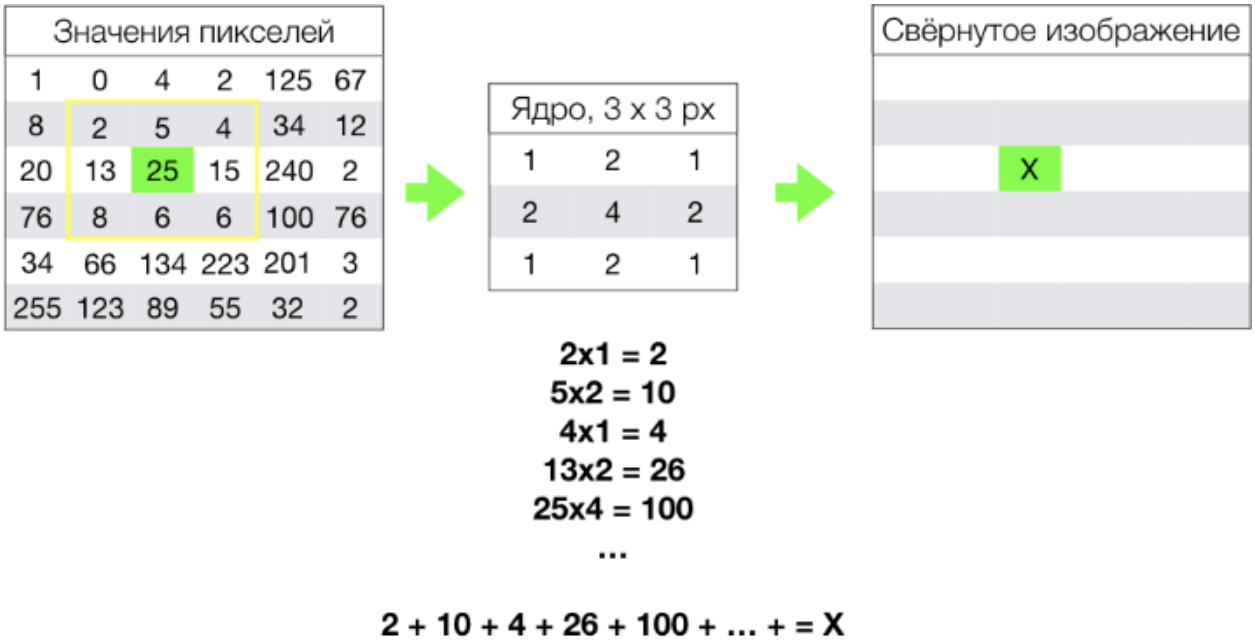

Рис. 6 – Новое изображение

5. Подобную операцию мы производим со всеми пикселями нашего изображения
6. Теперь рассмотрим вторую концепцию свёрточных нейронных сетей — операцию подвыборки. Простыми

словами, операция подвыборки это процесс сжатия (уменьшения размеров) изображения путём сложения значений блоков пикселей. Рассмотрим как это работает на конкретном примере. Нам необходимо найти максимальное значение пикселя попадающее в выделенную сетку. В пример ниже в сетку попадают значения 1, 0, 4, 8, 2, 5, 20, 13, 25. Максимальное значение — 25. Это значение «переносится» в новое изображение. Сетка сдвигается на 3 пикселя вправо и процесс выборки максимального значения и его переноса на новое изображение повторяется. (рис. 7).

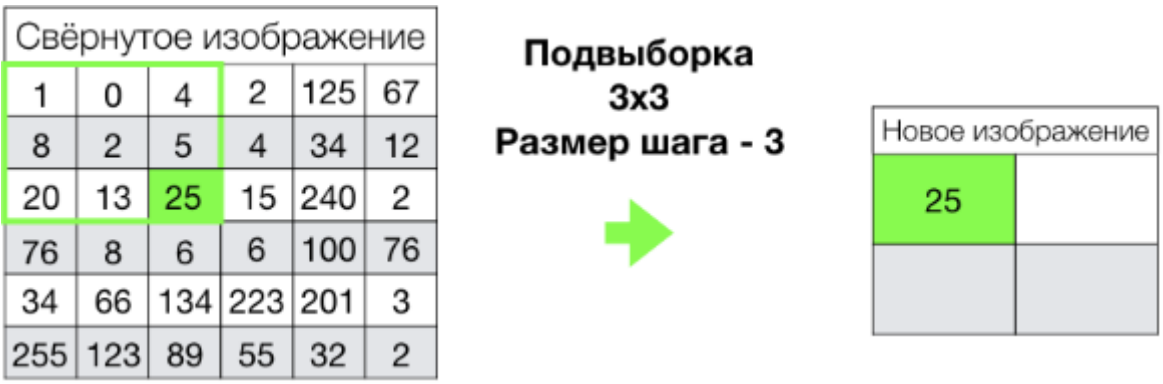

Рис.7 – Подвыборка

7. В результате будет получено изображение меньшего размера по сравнению с оригинальным входным изображением. В нашем примере было получено изображение, которое в два раза меньше нашего исходного изображения. Размер итогового изображения будет варьироваться в зависимости от выбора размера прямоугольной сетки и размера шага (рис. 8).

Рис. 8 – Новое изображение

Реализация модели:

Свёрточный слой мы добавили к нейронной сети с использованием Conv2D-слоя в Keras. Этот слой подобен Dense-слою, и содержит веса и смещения, которые подвергаются оптимизации (подбору). Conv2D-слой так же содержит фильтры («ядра»), значения которых тоже оптимизируются. Итак, в Conv2D-слое значения внутри матрицы фильтра и есть переменные, которые подвергаются оптимизации (рис. 9).

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

Рис. 9 – Финальная модель

Для наглядного примера сравним модели с сверточным слоем(рис. 10 и рис. 11) и без(рис. 12 и рис. 13). Для примера возьмем нс с 5 эпохами.

Для свёрточной нейронной сети:

```
Epoch 1/5
1875/1875 [==============================] - 5s 3ms/step - loss: 0.3957 - accuracy: 0.8597
Epoch 2/5
1875/1875 [==============================] - 5s 3ms/step - loss: 0.2582 - accuracy: 0.9062
Epoch 3/5
1875/1875 [==============================] - 5s 3ms/step - loss: 0.2127 - accuracy: 0.9219
Epoch 4/5
1875/1875 [==============================] - 5s 3ms/step - loss: 0.1780 - accuracy: 0.9344
Epoch 5/5
1875/1875 [==============================] - 5s 3ms/step - loss: 0.1564 - accuracy: 0.9426
```

Рис. 10 – Значения модели на тренировочных данных

```
313/313 [==============================] - 2s 5ms/step - loss: 0.2395 - accuracy: 0.9189
Accuracy on test dataset: 0.9189000129699707
```

Рис. 11 – Значения модели на тестовых данных

Для нейронной сети без свёртки:

```
Epoch 1/5
1875/1875 [==============================] - 5s 3ms/step - loss: 0.4919 - accuracy: 0.8268
Epoch 2/5
1875/1875 [==============================] - 5s 3ms/step - loss: 0.3735 - accuracy: 0.8641
Epoch 3/5
1875/1875 [==============================] - 5s 3ms/step - loss: 0.3275 - accuracy: 0.8807
Epoch 4/5
1875/1875 [==============================] - 5s 3ms/step - loss: 0.3137 - accuracy: 0.8854
Epoch 5/5
1875/1875 [==============================] - 5s 3ms/step - loss: 0.2941 - accuracy: 0.8904
```

Рис. 10 – Значения модели на тренировочных данных

```
313/313 [==============================] - 3s 8ms/step - loss: 0.3807 - accuracy: 0.8612
Accuracy on test dataset: 0.8611999750137329
```

Рис. 11 – Значения модели на тестовых данных

Вывод: исходя из полученным данных, можем сделать вывод, что использование свёрточной сети увеличивает точность на целых 5,5%, что очень хороший результат.

ПРИЛОЖЕНИЕ А

Исходный код

```
import tensorflow as tf
# Import TensorFlow Datasets
import tensorflow_datasets as tfds
tfds.disable_progress_bar()

# Helper libraries
import math
import numpy as np
import matplotlib.pyplot as plt

# Callback
from tf.keras.callbacks import ModelCheckpoint

import logging
logger = tf.get_logger()
logger.setLevel(logging.ERROR)

dataset, metadata = tfds.load('fashion_mnist', as_supervised=True, with_info=True)
train_dataset, test_dataset = dataset['train'], dataset['test']

class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal',      'Shirt',   'Sneaker',  'Bag',   'Ankle boot']

num_train_examples = metadata.splits['train'].num_examples
num_test_examples = metadata.splits['test'].num_examples
#print("Number of training examples: {}".format(num_train_examples))
#print("Number of test examples:     {}".format(num_test_examples))

def normalize(images, labels):
  images = tf.cast(images, tf.float32)
  images /= 255
  return images, labels
```

```
train_dataset =  train_dataset.map(normalize)
test_dataset  =  test_dataset.map(normalize)

train_dataset =  train_dataset.cache()
test_dataset  =  test_dataset.cache()

for image, label in test_dataset.take(1):
  break
image = image.numpy().reshape((28,28))

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

model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(),
              metrics=['accuracy'])

callback = ModelCheckpoint('callbacks/mnist-dense-{epoch:02d}-{val_loss:.4f}.hdf5')

BATCH_SIZE = 32
train_dataset =
train_dataset.cache().repeat().shuffle(num_train_examples).batch(BATCH_SIZE)
test_dataset = test_dataset.cache().batch(BATCH_SIZE)
model.fit(train_dataset, epochs=5, callbacks=callbacks,
steps_per_epoch=math.ceil(num_train_examples/BATCH_SIZE))
```

```
test_loss, test_accuracy = model.evaluate(test_dataset,
steps=math.ceil(num_test_examples/32))

print('Accuracy on test dataset:', test_accuracy)


#plt.plot(epochs, loss, 'bo', label='Train loss')

#plt.plot(epochs, val_loss, 'b', label='Valid loss')

#plt.title('Train and valid loss')

#plt.xlabel('Epochs')

#plt.ylabel('Loss')

#plt.legend()

#plt.show()

#plt.clf()


#plt.plot(epochs, acc, 'bo', label='Train acc')

#plt.plot(epochs, val_acc, 'b', label='Valid acc')

#plt.title('Train and valid accuracy')

#plt.xlabel('Epochs')

#plt.ylabel('Accuracy')

#plt.legend()

#plt.show()
```