АНД: Распознавание символов.

1.Начало

Я решил строить сеть для распознавания 5 различных букв латинского алфавита: N, K, P, R, Z.

Изначально бралась трехслойная нейронная сеть построенная по аналогии с тем что было на занятии для распознавания цифр, или на то что было в видео Three blue one brown видео. Слои выглядели примерно так:

- 1024 -> 256, relu
- 256 -> 100, relu
- 100 -> 5, softmax

Максимальный accuracy получался в районе 0.9.

Но после этого я расширил датасет с 50 объектов (5 букв по 10 изображений)

Добавив еще 75 аугментированных изображений для каждого класса (аугментация включала в себя сдвиг по картинке и изменение цвета буквы с черного на другой)

После этого качество заметно упало, стало понятно, что полносвязная сеть не так хороша в задаче распознавания букв. Поэтому было принято добавить в сеть сверточные слои и пулинг MaxPool. К работать со сверточными сетями я разобрался на основе курса на Kaggle: Learn Computer Vision | Kaggle.

2.Архитектура.

После всех донастроек была получена следующая архитектура:

1. вход в виде тензора (32, 32, 1), применяем сверточную нейросеть с ядром свертки 5 на 5 -> на выходе будет сгенерировано 32 карты признаков (feature maps), активация - ReLU
2. применяем Max Pooling с размером окна 7 на 7 - нам важно отслеживать расстояние между пикселями, например в случае R, чтобы определить ее верхнюю половину, поэтому берем размер окна больше, шаг окна 2 на 2, т.е. 2 по горизонтали и 2 по вертикали
3. применяем сверточную нейросеть с ядром свертки 5 на 5, активация - ReLU
4. еще раз применяем Max Pooling с размером окна 7 на 7
5. линейный слой карты признаков -> 128 выход, активация - ReLU
6. линейный слой: 128 вход -> 5 выход, активация - softmax (приводим выходы к вероятностям выбрать ту или иную метку класса)

Оптимизация - adam, функцию потерь берем характерную для данной задачи - categorical_crossentropy, а метрику – accuracy

Вполне хватило 40 эпох, потому что после модель начинает переобучатся и хуже классифицировать тестовые данные.

Итоговое accuracy = 1.

В случае если сложно воспринимать так, то вот скриншот архитектуры:

```
my_nn = keras.models.Sequential([
                                layers.Conv2D(32, (5, 5), padding='same', activation='relu', input_shape=(32, 32, 1)),
                                layers.MaxPooling2D((7, 7), strides=2),
                                layers.Conv2D(32, (5, 5), padding='same', activation='relu'),
                                layers.MaxPooling2D((7, 7), strides=2),
                                layers.Flatten(),
                                layers.Dense(units=128, activation='relu'),
                                layers.Dense(units=5, activation='softmax')
                                ]) # задаем нашу нейросеть
my_nn.compile(loss='categorical_crossentropy',
               optimizer='adam',
               metrics=['accuracy']) # задаем целвую функцию потерь и метрику, а также оптимизатор

my_nn.fit(X_train.reshape(100, 32, 32, 1), y_train_categorical, epochs=40, verbose=False) # обучаем модель
```

Архитектуру выбирал на основе того, что было описано в курсе на Kaggle, особенно учитывалось, что в нашей задаче было важно учитывать расстояние между пикселями, например, для того, чтобы определить верхнюю половину R, благодаря этому в разы уменьшается число ошибок, когда модель путает между собой K и R.

Одного сверточного слоя и пулинга было недостаточно поэтому и для улучшения работы были добавлены еще два таких же слоя.

ReLU как функция активации здесь используется стандартным образом, потому что она по сути предназначена, чтобы бесполезные признаки/пиксели, (т.е. те которые имеют значение при обработке <= 0) были одинаково бесполезны и имели значение 0.

Последний слой ожидаемо полносвязный с активацией softmax и пятью выходами, для классификации на 5 категорий.

3.Анализ результатов

Как было указано выше первоначальная трехслойная нейросеть была хорошо на неаугментированных данных, но после аугментации качество значительно упало – 0.6 accuracy плюс K и R практически всегда классифицировались некорректно.

Также мной была замечена ошибка в предобработке данных: я забыл при разбиении через train_test_split передать параметр stratify=y, для стратификации выборок при разбиении (т.е. для того, чтобы распределение классов оставалось неизменным и не возникало случаев, когда одна буква попала в выборку значительно больше чем остальные, а какая-то не попала вообще)

Также по началу качество даже на новой архитектуре со сверточными слоями показывала качество не выше 0.8, было это связано с тем, что размер ядер свертки изначально был 3 на 3, что довольно мало, потому что из-за небрежности при рисовании букв (можно назвать это шумом) могли учитываться ложные зависимости, а например кружочек в верху буквы R или P просто не считался за один признак. Для лучшего понимания можно посмотреть на пример этих букв:

После этого размер ядер свертки был поднят до 5 на 5.

Помимо этого, окно Max Pooling по той же причине было увеличено с 2 на 2 до 7 на 7, после чего наконец модель смогла показать лучший результат на обучающей выборке с accuracy = 1.

При проверке работы на подготовленных (не аугментированных данных) модель корректно классифицировала каждую из 10. Но все же при запуске на 15 аугментированных вариациях буквы K (перекрашенные либо с сильным сдвигом по картинке) дважды была допущена ошибка: две буквы были классифицированы как R.