**Что такое сеть ResNet?**

Остаточная нейронная сеть (ResNet) - это искусственная нейронная сеть, позволяет бороться с degradation problem (при добавлении слоев качество модели увеличивается до некоторого предела, а после падает). Основная идея которой заключается во введении «соединения быстрого доступа», которое пропускает один или несколько слоев, как показано на рис. 1.

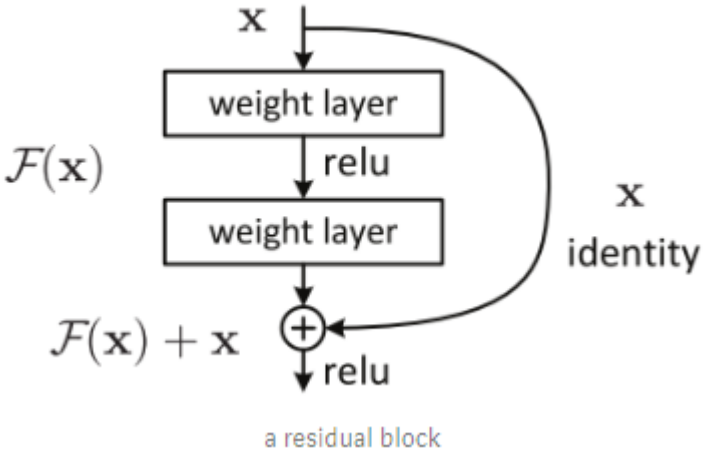


Рисунок 1 – Принцип работы ResNet.

ResNet позволяет относительно легко увеличить точность благодаря увеличению глубины, чего с другими сетями добиться сложнее.

**Приведите ситуацию, когда в сети необходимо использовать несколько выходных слоев.**

Несколько выходных слоев можно использовать в данных с несколькими метками. Например, у нас есть датасет с одеждой и нам нужно разделить эту одежду по цветам и по типам (футболки, джинсы и т.д.). Первый слой будет предсказывать тип, а второй цвет.

**Что такое LSTM блок? Для чего нужен?**

LSTM-сеть — это искусственная нейронная сеть, содержащая LSTM-модули вместо или в дополнение к другим сетевым модулям. LSTM-модуль — это рекуррентный модуль сети, способный запоминать значения как на

короткие, так и на длинные промежутки времени. LSTM-модули часто группируются в «блоки», содержащие различные LSTM-модули. Подобное устройство характерно для «глубоких» многослойных нейронных сетей и способствует выполнению параллельных вычислений с применением соответствующего оборудования. LSTM-модули часто группируются в «блоки», содержащие различные LSTM-модули. Подобное устройство характерно для «глубоких» многослойных нейронных сетей и способствует выполнению параллельных вычислений с применением соответствующего оборудования. LSTM-блоки содержат три или четыре «вентиля», которые используются для контроля потоков информации на входах и на выходах памяти данных блоков. Эти вентили реализованы в виде логистической функции для вычисления значения в диапазоне [0; 1]. Идея заключается в том, что старое значение следует забывать тогда, когда появится новое значение достойное запоминания). LSTM-блок определяет, как распоряжаться своей памятью как функцией этих значений, и тренировка весов позволяет LSTM-блоку выучить функцию, минимизирующую потери. LSTM-блоки обычно тренируют при помощи метода обратного распространения ошибки во времени.

**Какой в итоге наилучший процент классификации был достигнут? Можно ли его повысить? Если да, то как?**

Наилучший процент был достигнут в первом случае со слоями прореживания и размером ядра свертки (3,3), он равен 76%. Я считаю, что его можно повысить. Т.к. на данной модели не наблюдалось переобучение, то можно попробовать добавить количество эпох (количество эпох не было добавлено в самой л.р. т.к. выполнение программы с данным датасетом занимает огромное количество времени даже на моем количестве эпох). Также можно было попробовать добавить слои субдискретизации и прореживающие слои.

**Для чего нужен слой Flatten в Вашей модели?**

Слой Flatten преобразовывает тензор входной тензор в одномерный. Т.к. в конце сети находится слой прямого распространения, который принимает одномерный тензор, перед ним мы применяем слой Flatten.

**Можно ли построить Вашу модель не в функциональном виде? Если да, то приведите пример.**

Мы можем построить модель в последовательном виде, т.к. у нас один вход и один выход. Тогда построение модели будет выглядеть так:

```
model = Sequential()

model.add(Convolution2D(conv_depth_1, (kernel_size, kernel_size), padding='same', activation='relu', input_shape=(depth, height, width)))

model.add(Convolution2D(conv_depth_1, (kernel_size, kernel_size), padding='same', activation='relu'))

model.add(MaxPooling2D(pool_size=(pool_size, pool_size)))

model.add(Dropout(drop_prob_1))

model.add(Convolution2D(conv_depth_2, (kernel_size, kernel_size), padding='same', activation='relu'))

model.add(Convolution2D(conv_depth_2, kernel_size, kernel_size, padding='same', activation='relu'))

model.add(MaxPooling2D(pool_size=(pool_size, pool_size)))

model.add(Dropout(drop_prob_1))

model.add(Flatten())

model.add(Dense(hidden_size, activation='relu'))

model.add(Dropout(drop_prob_2))

model.add(Dense(num_classes, activation='softmax'))

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

```
          history = model.fit(X_train, Y_train,     batch_size=batch_size,
epochs=num_epochs, verbose=1, validation_split=0.1)
```