Лабораторная работа №7
по дисциплине «Системы поддержки принятия решений»
«Исследование архитектуры ResNet на датасете CIFAR»

Цель работы – исследование современных архитектур сверточных нейронных сетей на примере ResNet (2015).

В 2015 году авторы из Microsoft Research [1] решили разработать новую сверточную нейросеть. Основой их подхода стал так называемый "**Residual блок**" для борьбы с затуханием градиента (смотри рисунки 1 и 2).

Допустим мы вычисляем некоторую функцию, которую обозначим $F(x)$. К результату этой функции прибавим аргумент. Далее это значение переходит к следующему residual блоку. Для чего? Рассмотрим этот residual блок в виде выражения.

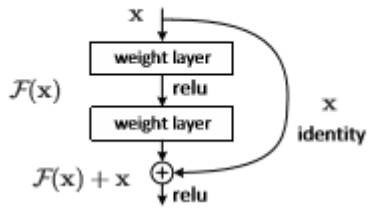


Рисунок 1 – Residual блок

Результат работы residual блока – это некоторая функция $F(x) + x$. Какова будет производная от выходов из residual блока?

$$(F(x) + x)' = F\,'(x) + 1$$

Соответственно, производная функции потерь, которая будет вычисляться после этого residual блока будет равна следующему: это будет производная функции потерь по выходам из residual блока, умножить на производные выходов residual блока по входам в residual блок. Производная по входам в residual блок будет равна производной loss-функции по выходам из residual блока, умноженную на производную от функции, которая в residual блоке есть, плюс "1".

В этом выражении, есть очень важный элемент – единица. Если раскроем скобки, то увидим, что производная функции потерь будет в немного изменённом виде протекать в предыдущий residual блок. Таким образом, если мы сделаем очень много residual блоков, следующих друг за другом, то из-за того, что у нас есть эти соединения, которые называются "**skip connections**", производная loss-функции по выходам из сети будет распространяться до самых первых слоёв. В результате получается, что градиент в такой сети, которая составлена из residual блоков, *не затухает*.

Также стоит отметить, что в residual блоках можно прибегать к некоторому трюку, который называется "**bottleneck**". Мы можем уменьшать количество параметров (количество каналов), а затем увеличивать назад, таким образом экономя вычисления.

ResNet существует в нескольких модификациях. Есть самые простые варианты, ResNet 18, где всего 18 обучаемых слоёв. Есть ResNet 32, где слоёв уже больше. Существует вариант Residual Network, где количество обучаемых слоёв равно 1024. Вот такие огромные нейронные сети – действительно глубокие – можно обучать при помощи resudual блока.

Отметим ещё одну очень интересную особенность, которая заключается в том, что в конце сети нет полносвязных слоёв, то есть отсутствует fully connected слои. Причина в том, что сеть и так уже достаточно глубокая. Поэтому достаточно просто сделать average pooling,

и в конце получается тензор размером 7×7×количество каналов, причём количество каналов, в зависимости от типа ResNet может быть разным, от 512 до 2048. Рассмотрим вариант с 512 каналами на выходе. Усредним по двум измерениям, по маскам 7×7, весь этот тензор. Получится вектор размером 512, после чего применим один fully connected слой, после которого будет идти софтмакс.

Для такой сети, как правило, на вход в сеть поставляются картинки размером 224×224, однако если взять картинки размером не 224×224, а, например, 1024×1024, тогда на выходе будет размер картинки 32×32×512. Таким образом, для того чтобы эту же самую сеть можно было применять и для больших картинок, достаточно по этим двум измерениям (32×32) просто усреднить весь выходной тензор, и после этого применить тот же полносвязный слой с последующим софтмаксом.

Таким образом, данная сеть может принимать картинки любого размера: и 512×512, и картинки 256×256, и даже размером 32×32. Единственное ограничение заключается в том, что очень желательно, чтобы картинки на вход были размера кратного 32, потому что в этой сети происходит 5 раз уменьшение в 2 раза. Поэтому, если у нас картинка не будет делиться нацело на 32, могут вылезти какие-то артефакты. Но, вообще говоря, даже если картинка нацело на 32 не делится, в этом случае, либо какая-то часть тензора отбросится, либо где-то будут учтены нули, которыми будет добавлена эта картинка. Так или иначе эта сеть не сломается, но там могут вылезти некоторые артефакты, поэтому желательно чтобы картинки делились нацело на 32. Теперь мы можем работать с картинками от маленьких размеров до очень больших размеров.

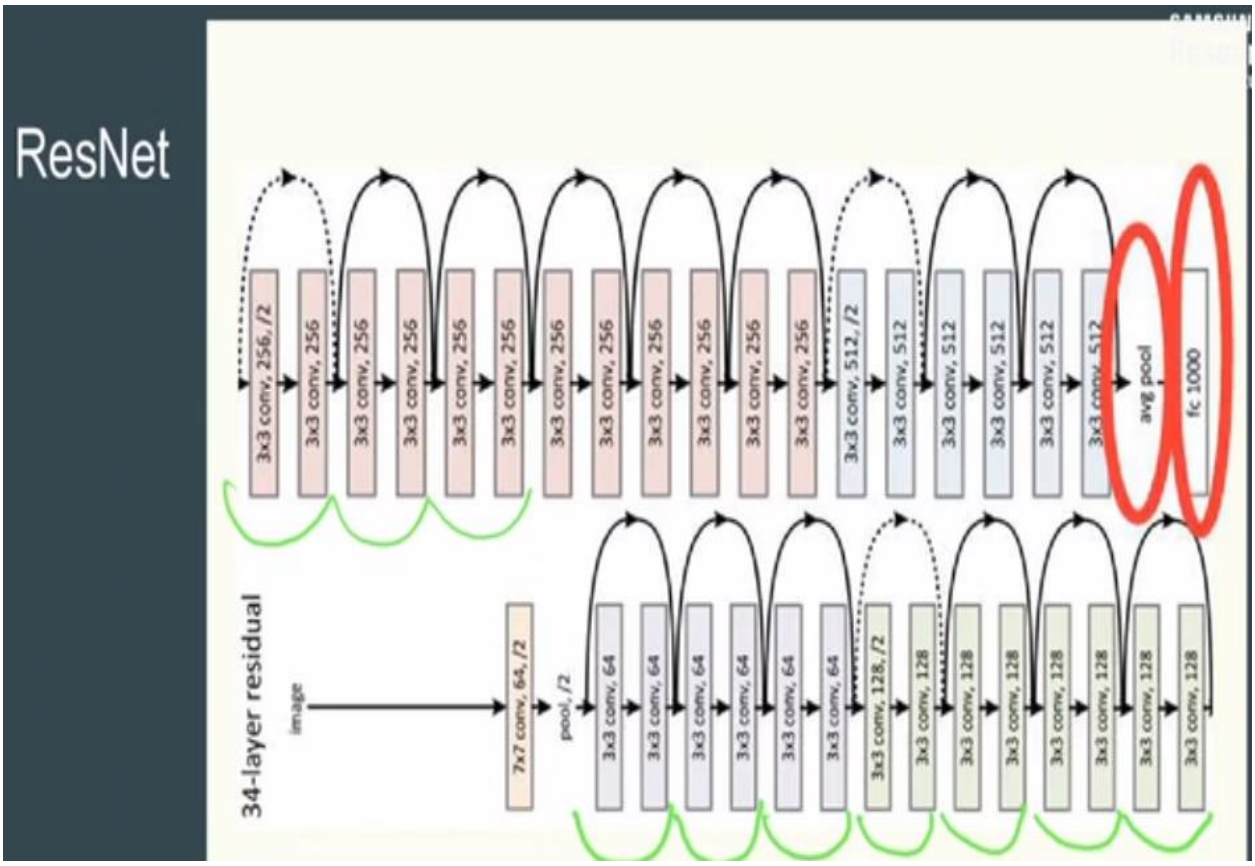


Рисунок 2 – Архитектура сети ResNet

В таблице представлены данные работы различных архитектур ResNet из статьи [1].

method			error (%)
Maxout [10]			9.38
NIN [25]			8.81
DSN [24]			8.22
	# layers	# params	
FitNet [35]	19	2.5M	8.39
Highway [42, 43]	19	2.3M	7.54 (7.72±0.16)
Highway [42, 43]	32	1.25M	8.80
ResNet	20	0.27M	8.75
ResNet	32	0.46M	7.51
ResNet	44	0.66M	7.17
ResNet	56	0.85M	6.97
ResNet	110	1.7M	**6.43** (6.61±0.16)
ResNet	1202	19.4M	7.93

Table 6. Classification error on the **CIFAR-10** test set. All methods are with data augmentation. For ResNet-110, we run it 5 times and show "best (mean±std)" as in [43].

На последующих рисунках показаны результаты тестирования в среде Colab различных архитектур в сравнении по точности и ошибке на валидации.

```
In [24]: from torchvision.models import resnet18

         accuracies['resnet18'], losses['resnet18'] = \
             train(resnet18(), X_train, y_train, X_test, y_test)
```

```
Train Epoch: 10 Time: 00:05:11 Accuracy: 0.7032999992370605, GPU_Mem_alloc: 355291648 GPU_Mem_cashed: 2407530496
Train Epoch: 20 Time: 00:10:23 Accuracy: 0.7321999669075012, GPU_Mem_alloc: 355291648 GPU_Mem_cashed: 2407530496
Train Epoch: 30 Time: 00:15:35 Accuracy: 0.7394999861717224, GPU_Mem_alloc: 355291648 GPU_Mem_cashed: 2407530496
Train Epoch: 40 Time: 00:20:46 Accuracy: 0.7276999950408936, GPU_Mem_alloc: 355291648 GPU_Mem_cashed: 2407530496
Train Epoch: 50 Time: 00:25:57 Accuracy: 0.7240999937057495, GPU_Mem_alloc: 355291648 GPU_Mem_cashed: 2407530496
```

```
In [25]: acc_loss_graph(accuracies, losses, ['resnet18', 'cifar_net_drop'])
```

CIFAR с дропаутом 0.15.

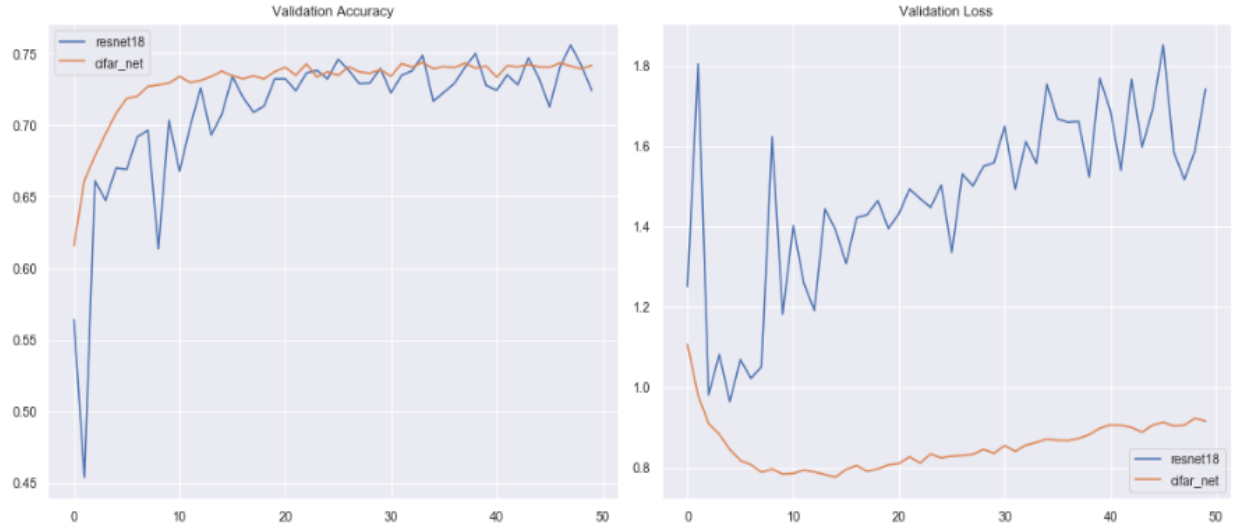
```
accuracies['cifar_net'], losses['cifar_net'] = \
    train(CIFARNet(), X_train, y_train, X_test, y_test)
```

```
Train Epoch: 10 Time: 00:00:57 Accuracy: 0.7292999625205994, GPU_Mem_alloc: 142680064 GPU_Mem_cashed: 2407530496
Train Epoch: 20 Time: 00:01:55 Accuracy: 0.7372999787330627, GPU_Mem_alloc: 142680064 GPU_Mem_cashed: 2407530496
Train Epoch: 30 Time: 00:02:53 Accuracy: 0.7386999726295471, GPU_Mem_alloc: 142680064 GPU_Mem_cashed: 2407530496
Train Epoch: 40 Time: 00:03:51 Accuracy: 0.7411999702453613, GPU_Mem_alloc: 142680064 GPU_Mem_cashed: 2407530496
Train Epoch: 50 Time: 00:04:50 Accuracy: 0.7416999936103821, GPU_Mem_alloc: 142680064 GPU_Mem_cashed: 2407530496
```

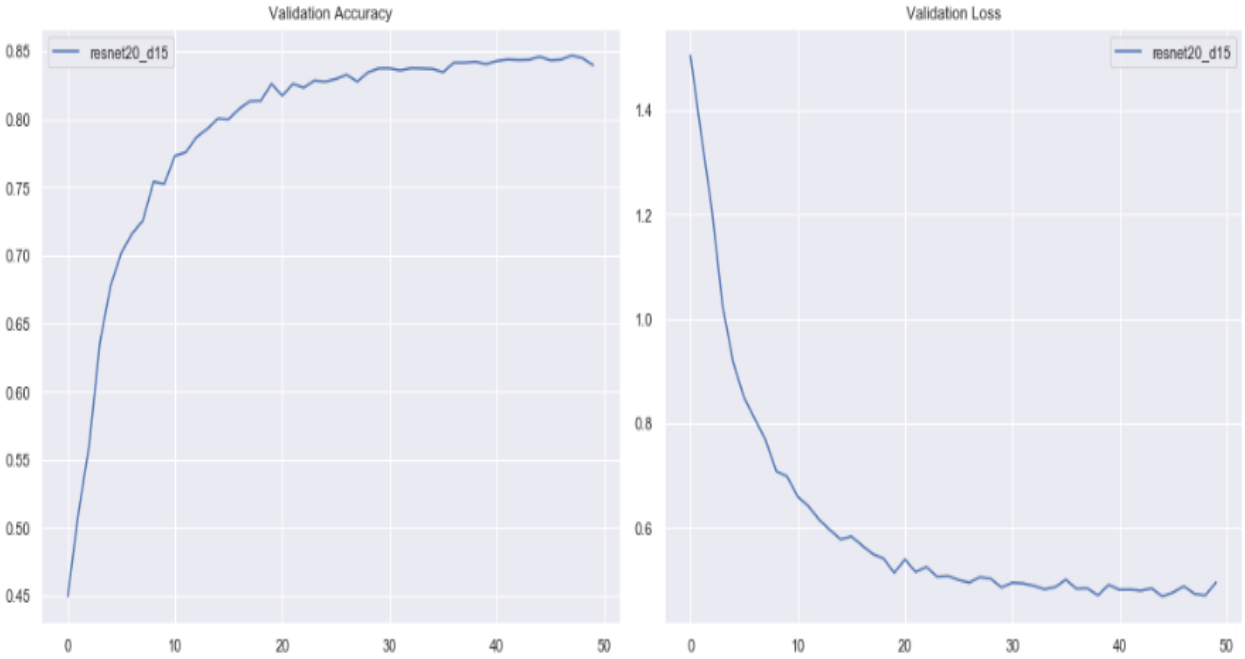
```
from torchvision.models import resnet18

accuracies['resnet18'], losses['resnet18'] = \
    train(resnet18(), X_train, y_train, X_test, y_test)
```

```
Train Epoch: 10 Time: 00:05:11 Accuracy: 0.7032999992370605, GPU_Mem_alloc: 355291648 GPU_Mem_cashed: 2407530496
Train Epoch: 20 Time: 00:10:23 Accuracy: 0.7321999669075012, GPU_Mem_alloc: 355291648 GPU_Mem_cashed: 2407530496
Train Epoch: 30 Time: 00:15:35 Accuracy: 0.7394999861717224, GPU_Mem_alloc: 355291648 GPU_Mem_cashed: 2407530496
Train Epoch: 40 Time: 00:20:46 Accuracy: 0.7276999950408936, GPU_Mem_alloc: 355291648 GPU_Mem_cashed: 2407530496
Train Epoch: 50 Time: 00:25:57 Accuracy: 0.7240999937057495, GPU_Mem_alloc: 355291648 GPU_Mem_cashed: 2407530496
```

```
acc_loss_graph(accuracies, losses, ['resnet18', 'cifar_net'])
```



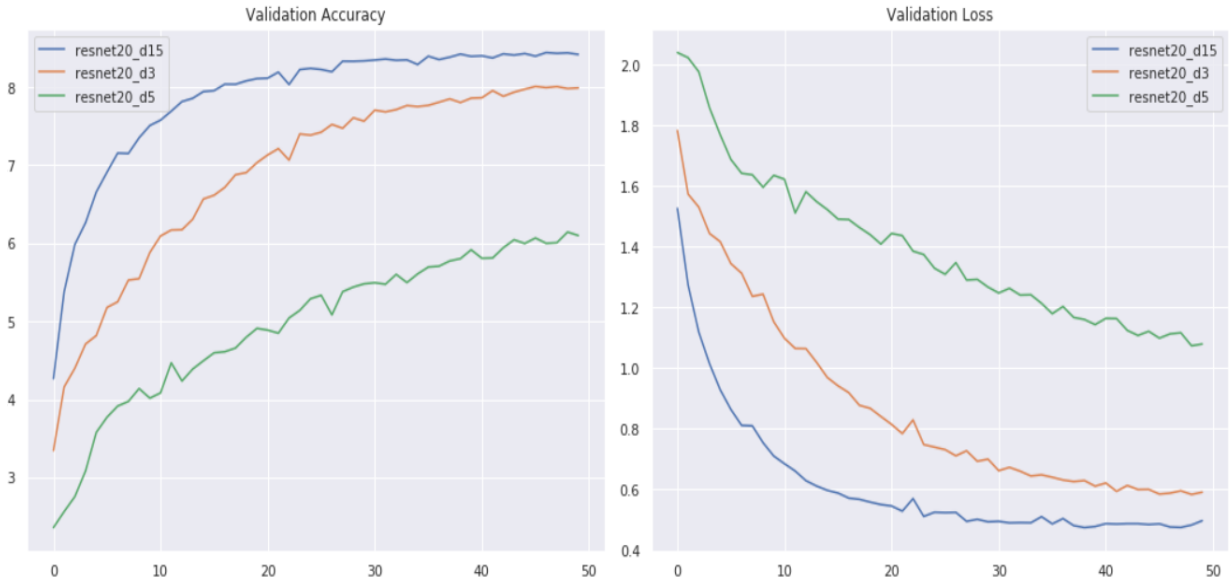

```
acc_loss_graph(accuracies, losses, ['resnet20', 'resnet18', 'cifar_net'])
```



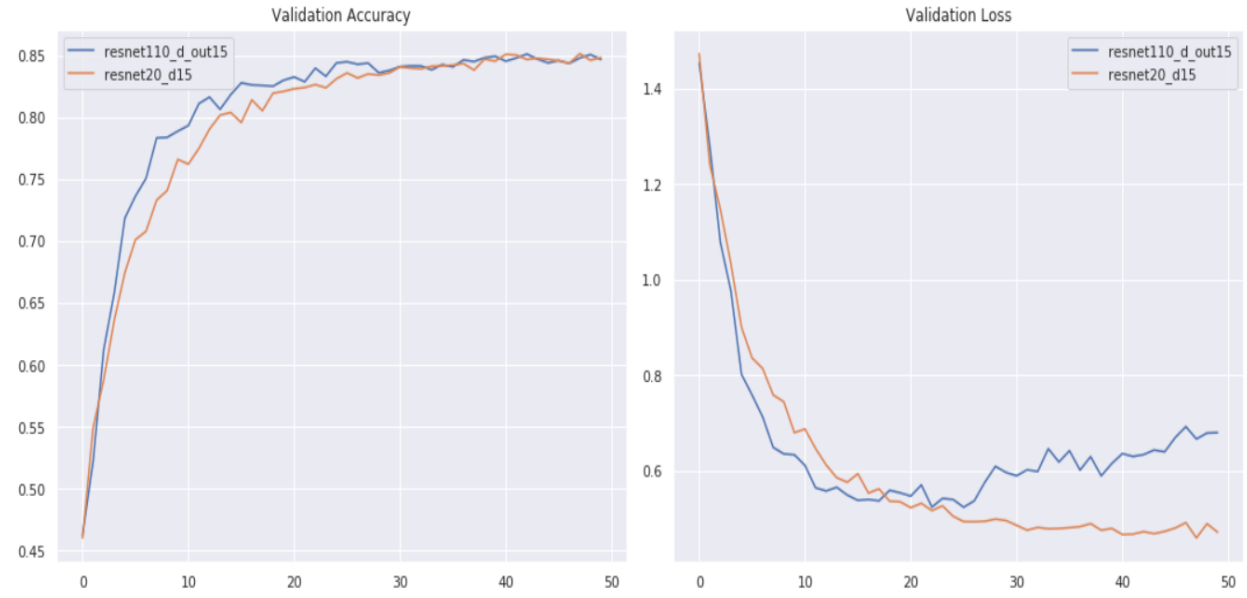

```
[36] accuracies['resnet20_d15'], losses['resnet20_d15'] = \
         train(resnet20_d_out15(), X_train, y_train, X_test, y_test)
```

```
Train Epoch: 10 Time: 00:02:07 Accuracy: 0.7638999819755554, GPU_Mem_alloc: 129604096 GPU_Mem_cashed: 3059744768
Train Epoch: 20 Time: 00:04:17 Accuracy: 0.8086000084877014, GPU_Mem_alloc: 129604096 GPU_Mem_cashed: 3059744768
Train Epoch: 30 Time: 00:06:26 Accuracy: 0.8313999772071838, GPU_Mem_alloc: 129604096 GPU_Mem_cashed: 3059744768
Train Epoch: 40 Time: 00:08:35 Accuracy: 0.8411999940872192, GPU_Mem_alloc: 129604096 GPU_Mem_cashed: 3059744768
Train Epoch: 50 Time: 00:10:44 Accuracy: 0.8515999913215637, GPU_Mem_alloc: 129604096 GPU_Mem_cashed: 3059744768
```

```
acc_loss_graph(accuracies, losses, ['resnet20_d15','resnet18', 'cifar_net'])
```




```
acc_loss_graph(accuracies, losses, ['resnet110_d_out15','resnet20_d15'])
```



Задание на лабораторную работу

1. Изучить архитектуру ResNet и понятие residual блока**.**
2. Для выполнения работы рекомендуется использовать Colab с включенным GPU.
3. Исследовать нейронную сеть **ResNet18**. Эту модель можно загрузить напрямую:

```
from torchvision.models import resnet18

accuracies['resnet18'], losses['resnet18'] = \
    train(resnet18(), X_train, y_train, X_test, y_test)
```

4. Исследовать нейронную сеть ResNet20 с различными показателями дропаута и регуляризации. *L*2-регуляризация в PyTorch активируется с помощью параметра **weight_decay** в оптимизаторе. Значение обычно выбирают из [1e-3, 1e-4, 1e-5].

```
optimizer = torch.optim.Adam(model.parameters(), lr=1e-4, weight_decay=1e-5)
```

5. Реализуйте ResNet110 (возможно, придется уменьшить размер batch'a). Проверьте утверждение, что ResNet110 не обучается (или обучается в 10% случаев), если отключить BatchNorm.
6. Какой результат вы считаете наилучшим?

Содержание отчета

1. Титульный лист
2. Цель работы, постановка задачи исследования.
3. Описание методики исследования.
4. Результаты исследования в соответствии с заданием.
5. Выводы по работе.

Полезная ссылка

1.https://arxiv.org/pdf/1512.03385.pdf