Федеральное государственное автономное образовательное учреждение высшего образования

Национальный исследовательский технологический университет

«МИСиС»

Институт ИТАСУ (ИТКН)

Кафедра Инженерной кибернетики

Лабораторная работа №4

по курсу «Нейронные сети и машинное обучение»

Выполнил:

Студент гр. МПИ-20-4-2

Малынковский О.В.

Проверил:

Курочкин И.И.

Москва 2020г.

Оглавление

Теория

ResNet — это сокращенное название для Residual Network (дословно — «остаточная сеть»)

Глубокие сверточные нейронные сети превзошли человеческий уровень классификации изображений в 2015 году. Глубокие сети извлекают низко-, средне- и высокоуровневые признаки сквозным многослойным способом, а увеличение количества stacked layers может обогатить «уровни» признаков. Stacked layer имеет решающее значение, посмотрите на результат ImageNet:

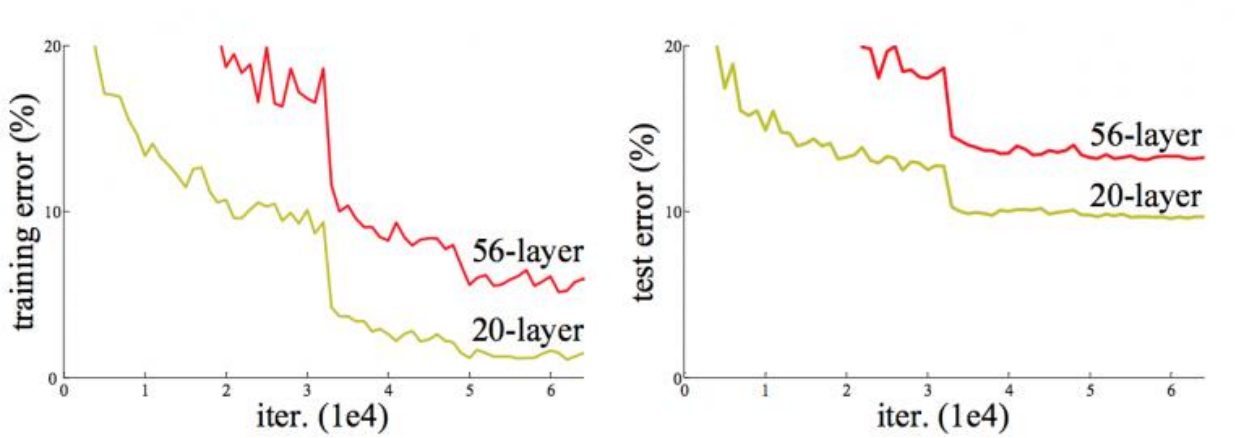
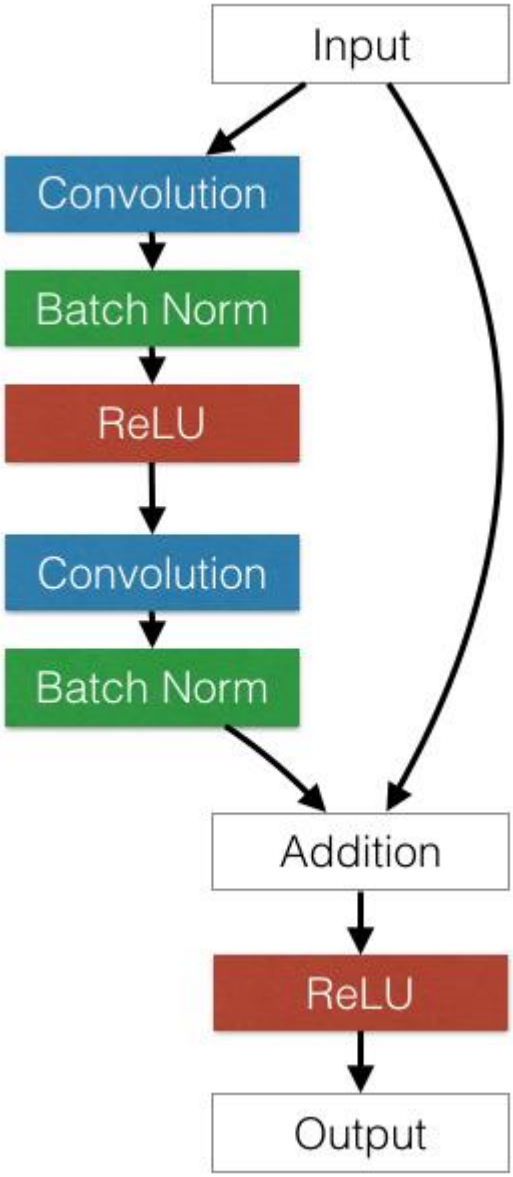


Рисунок 1: Ошибка обучения (слева) и ошибка теста (справа) на CIFAR-10 с 20-уровневыми и 56-слойными «простыми» сетями. Более глубокая сеть имеет большую ошибку обучения и, следовательно, ошибку тестирования. Подобные явления в ImageNet представлены на рисунке 4.

Когда более глубокая сеть начинает сворачиваться, возникает проблема: с увеличением глубины сети точность сначала увеличивается, а затем быстро ухудшается. Снижение точности обучения показывает, что не все сети легко оптимизировать.

Соединения быстрого доступа

Чтобы преодолеть эту проблему, Microsoft ввела глубокую «остаточную» структуру обучения. Вместо того, чтобы надеяться на то, что каждые несколько stacked layers непосредственно соответствуют желаемому основному представлению, они явно позволяют этим слоям соответствовать «остаточному». Формулировка $F(x) + x$ может быть реализована с помощью нейронных сетей **с соединениями для быстрого доступа.**



Соединения быстрого доступа (shortcut connections) пропускают один или несколько слоев и выполняют сопоставление идентификаторов. Их выходы добавляются к выходам stacked layers. Используя ResNet, можно решить множество проблем, таких как:

- ResNet относительно легко оптимизировать: «простые» сети (которые просто складывают слои) показывают большую ошибку обучения, когда глубина увеличивается.
- ResNet позволяет относительно легко увеличить точность благодаря увеличению глубины, чего с другими сетями добиться сложнее.

Вот так выглядит структура сети в цифрах (берем для лабораторной работы ResNet-50):

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

В данной работе я буду повторять исследования из статьи:

Deep Residual Networks with Exponential Linear Unit. Anish Shah, Sameer Shinde, Eashan Kadam, Hena Shah, Sandip Shingade (2016)

Ссылка на статью: https://arxiv.org/pdf/1604.04112v4.pdf

Авторы статьи как раз учили сетку на Cifar-10, а не на больших наборах данные наподобие ImageNet. Также указаны параметры обучения, что позволит максимально воспроизвести эксперимент.

Table 1: Test error (%) of our model compared to the original ResNet model. The test error of the original ResNet model refers to our reproduction of the experiments by He et al. [1]

Layers	Original ResNet	ResNet with ELU
20	8.38	**8.32**
32	7.51	**7.30**
44	7.17	**6.93**
56	6.97	**6.31**
110	6.42	**5.62**

Мы возьмем ResNet-50 (56 не реализован в Keras, но здесь не критично, так как в статье указано что причисле эпох обучения более 100 они все сходятся к близким значениям)

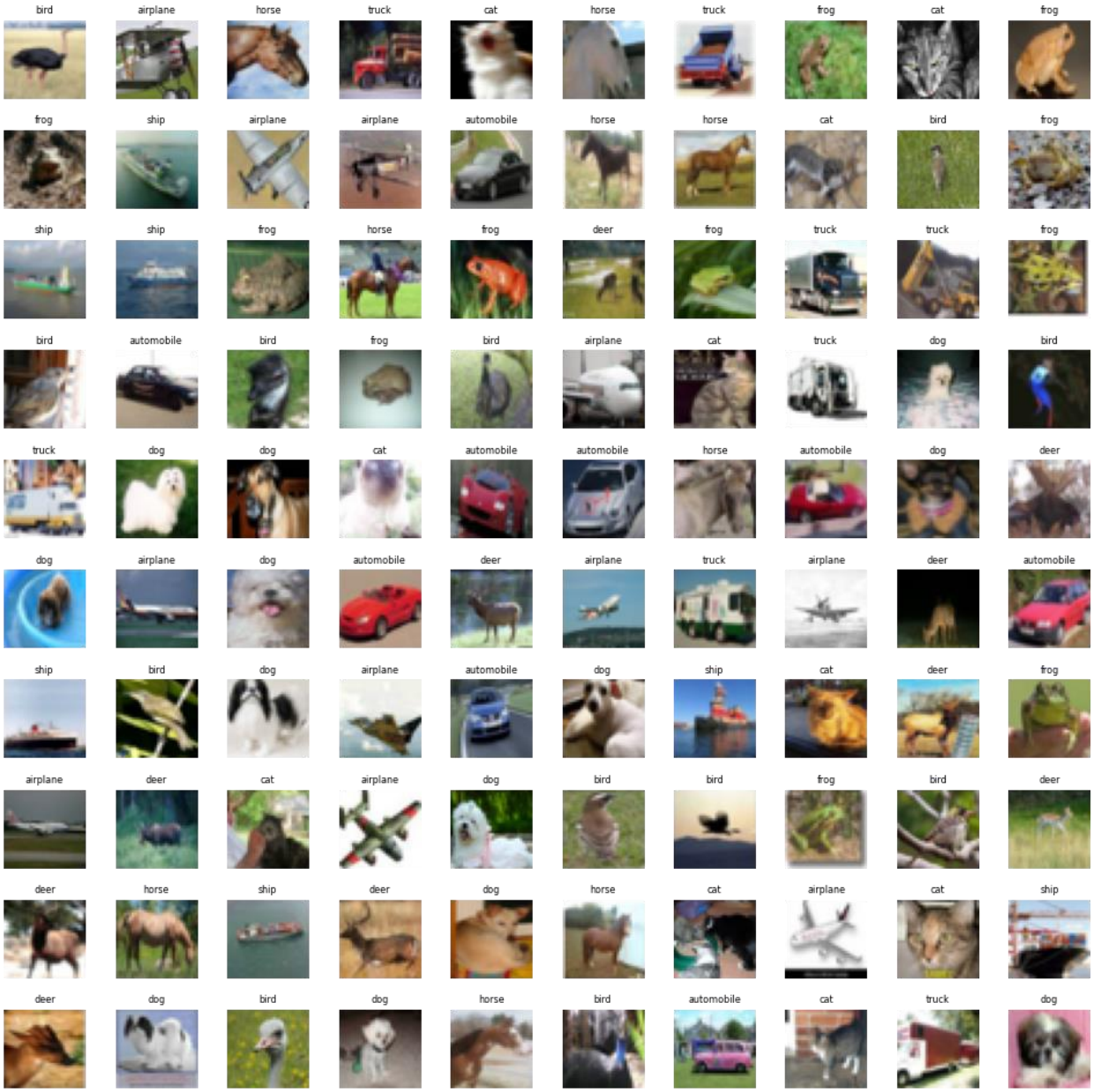
Реализация

На языке Python с использованием библиотек Keras реализуем рассматриваемые методы кластеризации. Также в sklearn есть функции оценки качества.

Примеры работы

CIFAR-10

Набор данных CIFAR-10 состоит из 60000 цветных изображений 32x32 в 10 классах, по 6000 изображений на класс. Имеется 50000 обучающих изображений и 10000 тестовых изображений.



Набор данных разделен на пять обучающих наборов и один тестовый набор, каждый из которых содержит 10000 изображений. Тестовый набор содержит ровно 1000 случайно выбранных изображений из каждого класса. Обучающие наборы содержат оставшиеся изображения в случайном порядке, но

некоторые обучающие наборы могут содержать больше изображений из одного класса, чем из другого. Между ними обучающие наборы содержат ровно 5000 изображений из каждого класса.

Вот классы в наборе данных:

- самолет
- автомобиль
- птица
- Кот
- олень
- собака
- лягушка
- лошадь
- судно
- грузовик

Классы полностью исключают друг друга. Легковые и грузовые автомобили не пересекаются. «Автомобиль» включает седаны, внедорожники и тому подобное. «Грузовик» включает только большие грузовики. Также нет пикапов.

Задаем архитектуру сети:

```
def residual_block(x,o_filters,increase=False):
    stride = (1,1)
    if increase:
        stride = (2,2)

    o1 = Activation('relu')(BatchNormalization(momentum=0.9, epsilon=1e-5)(x))
    conv_1 = Conv2D(o_filters,kernel_size=(3,3),strides=stride,padding='same',
                    kernel_initializer="he_normal",
                    kernel_regularizer=regularizers.l2(weight_decay))(o1)
    o2  = Activation('relu')(BatchNormalization(momentum=0.9, epsilon=1e-5)(conv_1))
    conv_2 = Conv2D(o_filters,kernel_size=(3,3),strides=(1,1),padding='same',
                    kernel_initializer="he_normal",
                    kernel_regularizer=regularizers.l2(weight_decay))(o2)
    if increase:
        projection = Conv2D(o_filters,kernel_size=(1,1),strides=(2,2),padding='same',
                            kernel_initializer="he_normal",
                            kernel_regularizer=regularizers.l2(weight_decay))(o1)
        block = add([conv_2, projection])
    else:
        block = add([conv_2, x])
    return block
```

```
    # input: 32x32x16 output: 32x32x16
    for _ in range(stack_n):
        x = residual_block(x,16,False)

    # input: 32x32x16 output: 16x16x32
    x = residual_block(x,32,True)
    for _ in range(1,stack_n):
        x = residual_block(x,32,False)

    # input: 16x16x32 output: 8x8x64
    x = residual_block(x,64,True)
    for _ in range(1,stack_n):
        x = residual_block(x,64,False)

    x = BatchNormalization(momentum=0.9, epsilon=1e-5)(x)
    x = Activation('relu')(x)
    x = GlobalAveragePooling2D()(x)

    # input: 64 output: 10
    x = Dense(classes_num,activation='softmax',kernel_initializer="he_normal",
              kernel_regularizer=regularizers.l2(weight_decay))(x)
    return x
```

Обучение, как и сказано в статье, будем делать через стохастические градиентный спуск с использованием правила моментов и поправки Нестерова:

```
# set optimizer
sgd = optimizers.SGD(lr=.1, momentum=0.9, nesterov=True)
resnet.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])
```

Также в статье рекомендовали провести аугментацию (в данном случае это горизонтальный разворот, вращения по горизонтальной и вертикальной осям)

```
datagen = ImageDataGenerator(horizontal_flip=True,
                             width_shift_range=0.125,
                             height_shift_range=0.125,
                             fill_mode='constant',cval=0.)
```

Также нужно провести стандартизацию.

Зададим параметры, как было указано в статье:

```
stack_n           = 5
layers            = 6 * stack_n + 2
num_classes       = 10
img_rows, img_cols = 32, 32
img_channels      = 3
batch_size        = 128
epochs            = 200
iterations        = 50000 // batch_size + 1
weight_decay      = 1e-4

def color_preprocessing(x_train,x_test):
    x_train = x_train.astype('float32')
    x_test = x_test.astype('float32')

    for i in range(3):
        x_train[:,:,:,i] = (x_train[:,:,:,i] - mean[i]) / std[i]
        x_test[:,:,:,i] = (x_test[:,:,:,i] - mean[i]) / std[i]
    return x_train, x_test


def scheduler(epoch):
    if epoch < 81:
        return 0.1
    if epoch < 122:
        return 0.01
    return 0.001
```

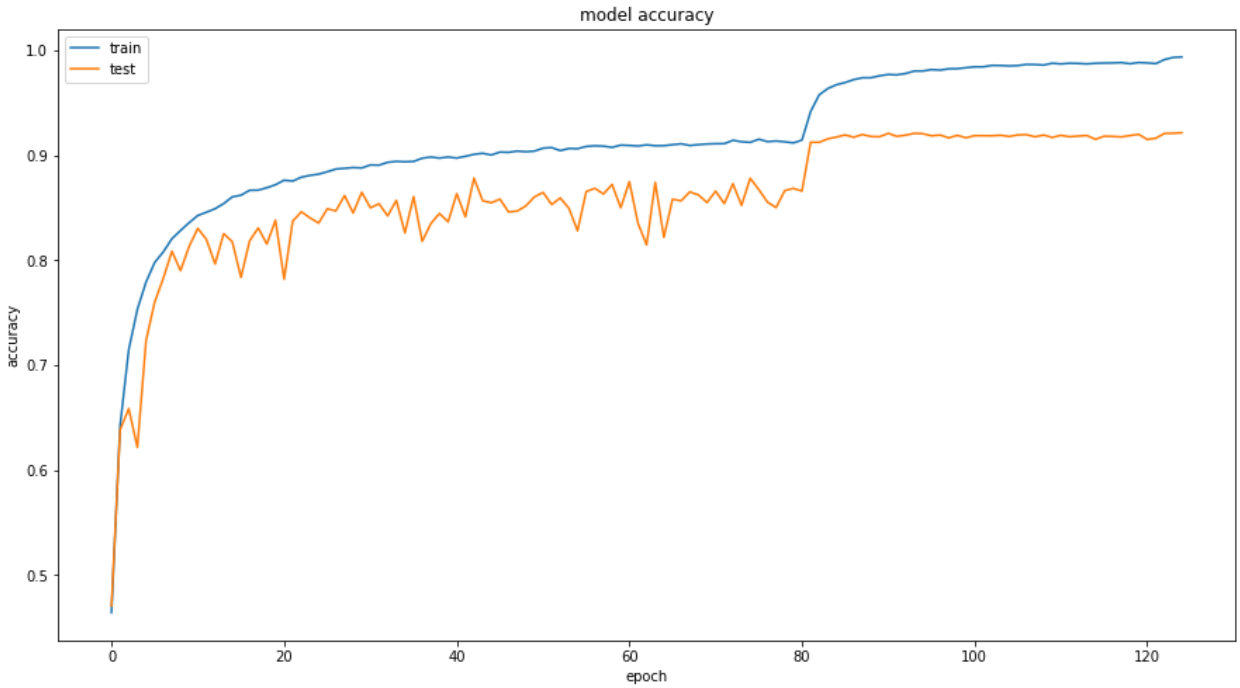
Скорость обучения будет уменьшаться при достижении 82 и 123 эпох обучения.

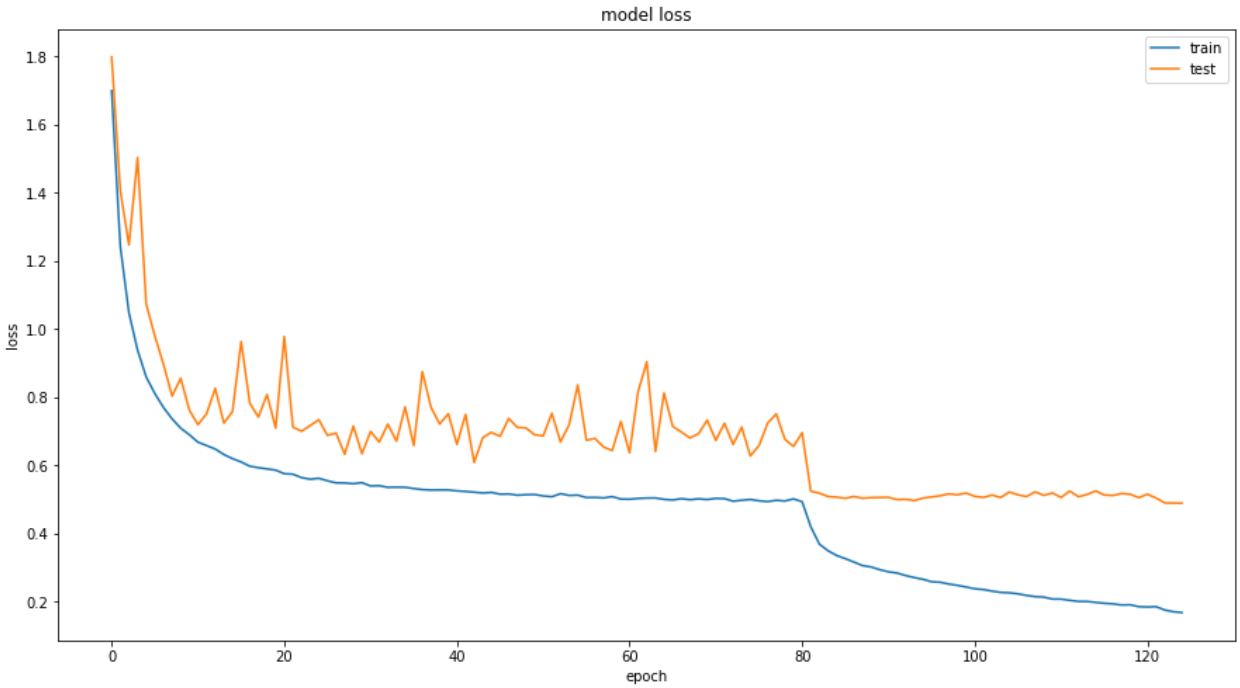
Можно приступить к обчуению:

```
    # start training
    resnet50_model = resnet.fit_generator(datagen.flow(x_train, y_train,batch_size=batch_size),
                        steps_per_epoch=iterations,
                        epochs=125,
                        callbacks=cbks,
                        validation_data=(x_test, y_test))
```

```
Epoch 74/125
391/391 [==============================] - 30s 77ms/step - loss: 0.4881 - accuracy: 0.9159 - val_loss: 0.7125 - val_accuracy: 0.8524
Epoch 75/125
391/391 [==============================] - 30s 76ms/step - loss: 0.4901 - accuracy: 0.9146 - val_loss: 0.6279 - val_accuracy: 0.8783
Epoch 76/125
391/391 [==============================] - 30s 76ms/step - loss: 0.4743 - accuracy: 0.9223 - val_loss: 0.6586 - val_accuracy: 0.8675
Epoch 77/125
391/391 [==============================] - 30s 76ms/step - loss: 0.4829 - accuracy: 0.9174 - val_loss: 0.7240 - val_accuracy: 0.8556
Epoch 78/125
391/391 [==============================] - 30s 76ms/step - loss: 0.4922 - accuracy: 0.9167 - val_loss: 0.7514 - val_accuracy: 0.8503
Epoch 79/125
```

Динамика роста accuracy и уменьшения ошибки:





Метрики качества для обучающей выборки (усреднение macro):

```
y_ = np.argmax(y_train, axis=1)
print('Accuracy для обучающего множества:', metrics.accuracy_score(
print('Precision для обучающего множества:', metrics.precision_sco
print('Recall для обучающего множества:', metrics.recall_score(y, 
print('F1-measure для обучающего множества:', metrics.f1_score(y, 
```

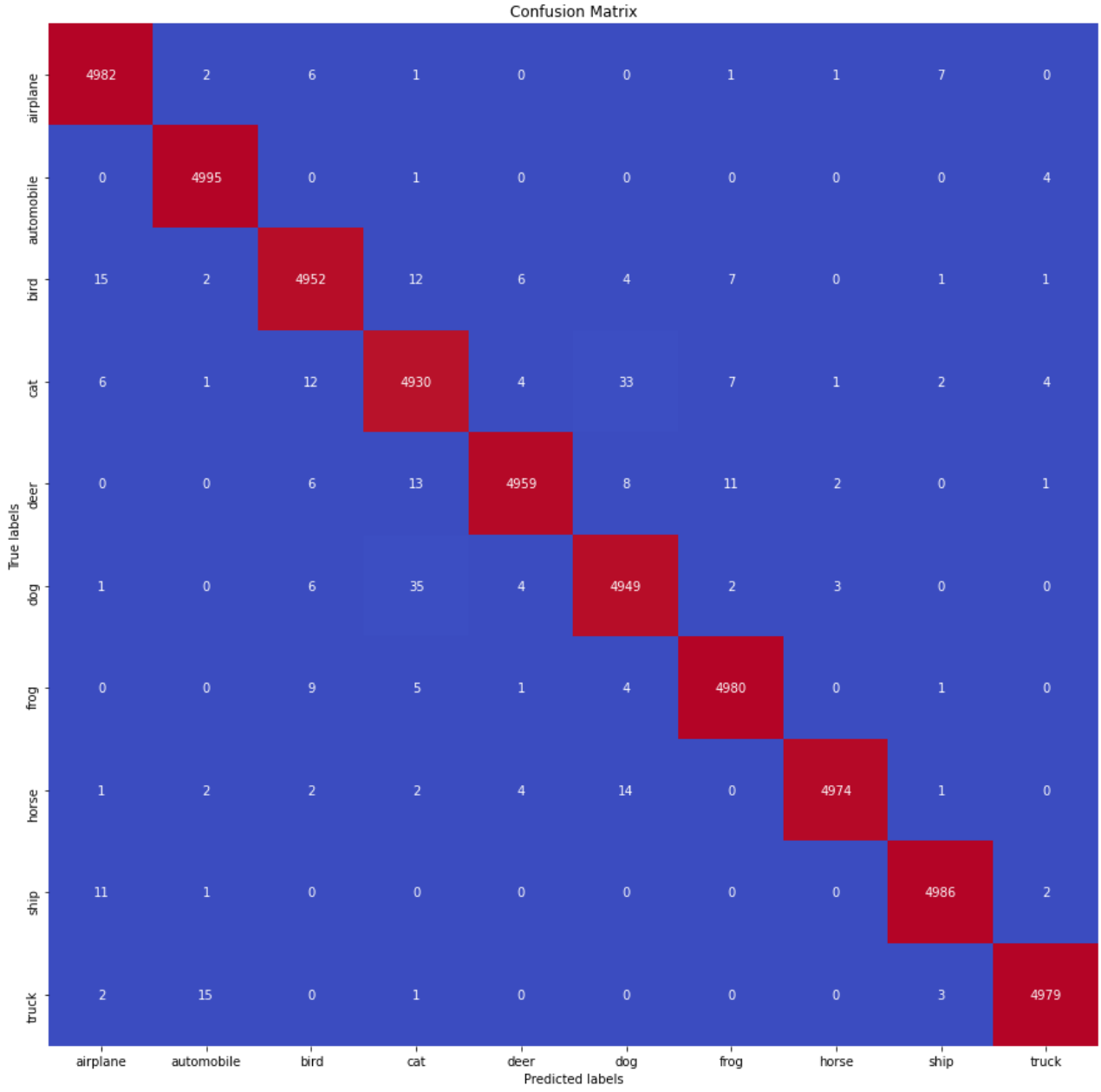
```
Accuracy для обучающего множества: 0.99372
Precision для обучающего множества: 0.9937244082714457
Recall для обучающего множества: 0.99372
F1-measure для обучающего множества: 0.9937203805905701
```

Метрики качества для тестовой выборки:

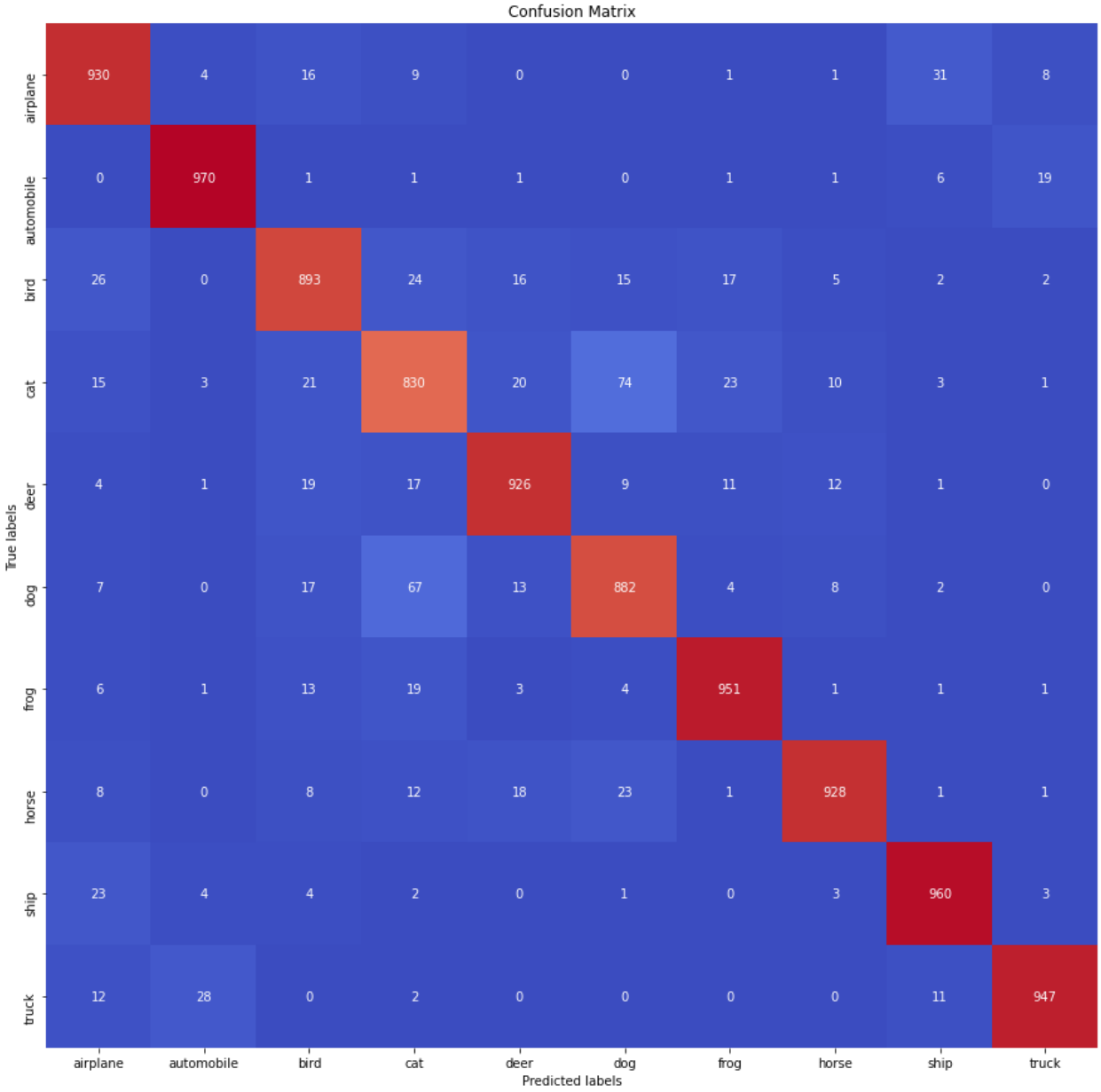
```
print('Accuracy для тестового множества:', metrics.accuracy
print('Precision для тестового множества:', metrics.precisi
print('Recall для тестового множества:', metrics.recall_sco
print('F1-measure для тестового множества:', metrics.f1_sco
```

```
Accuracy для тестового множества: 0.9217
Precision для тестового множества: 0.9217414615500671
Recall для тестового множества: 0.9216999999999999
F1-measure для тестового множества: 0.9216465777214442
```

Матрица ошибок для обучающей выборки:



Матрица ошибок для тестовой выборки:



В статье у авторов была ошибка ~7% или 93% accuracy. Значит, у меня получилось повторить их эксперимент.

https://bigballon.github.io/cifar-10-cnn/

В статье приведены показатели для разных архитектур, все они обучались на cifar-10:

network	GPU	params	batch size	epoch	training time	accuracy(%)
Lecun-Network	GTX1080TI	62k	128	200	30 min	76.23
Network-in-Network	GTX1080TI	0.97M	128	200	1 h 40 min	91.63
Vgg19-Network	GTX1080TI	39M	128	200	1 h 53 min	93.53
Residual-Network20	GTX1080TI	0.27M	128	200	44 min	91.82
Residual-Network32	GTX1080TI	0.47M	128	200	1 h 7 min	92.68
Residual-Network110	GTX1080TI	1.7M	128	200	3 h 38 min	93.93
Wide-resnet 16x8	GTX1080TI	11.3M	128	200	4 h 55 min	95.13
Wide-resnet 28x10	GTX1080TI	36.5M	128	200	10 h 22 min	95.78
DenseNet-100x12	GTX1080TI	0.85M	64	250	17 h 20 min	94.91
DenseNet-100x24	GTX1080TI	3.3M	64	250	22 h 27 min	95.30
DenseNet-160x24	1080 x 2	7.5M	64	250	50 h 20 min	95.90
ResNeXt-4x64d	GTX1080TI	20M	120	250	21 h 3 min	95.19
SENet(ResNeXt-4x64d)	GTX1080TI	20M	120	250	21 h 57 min	95.60

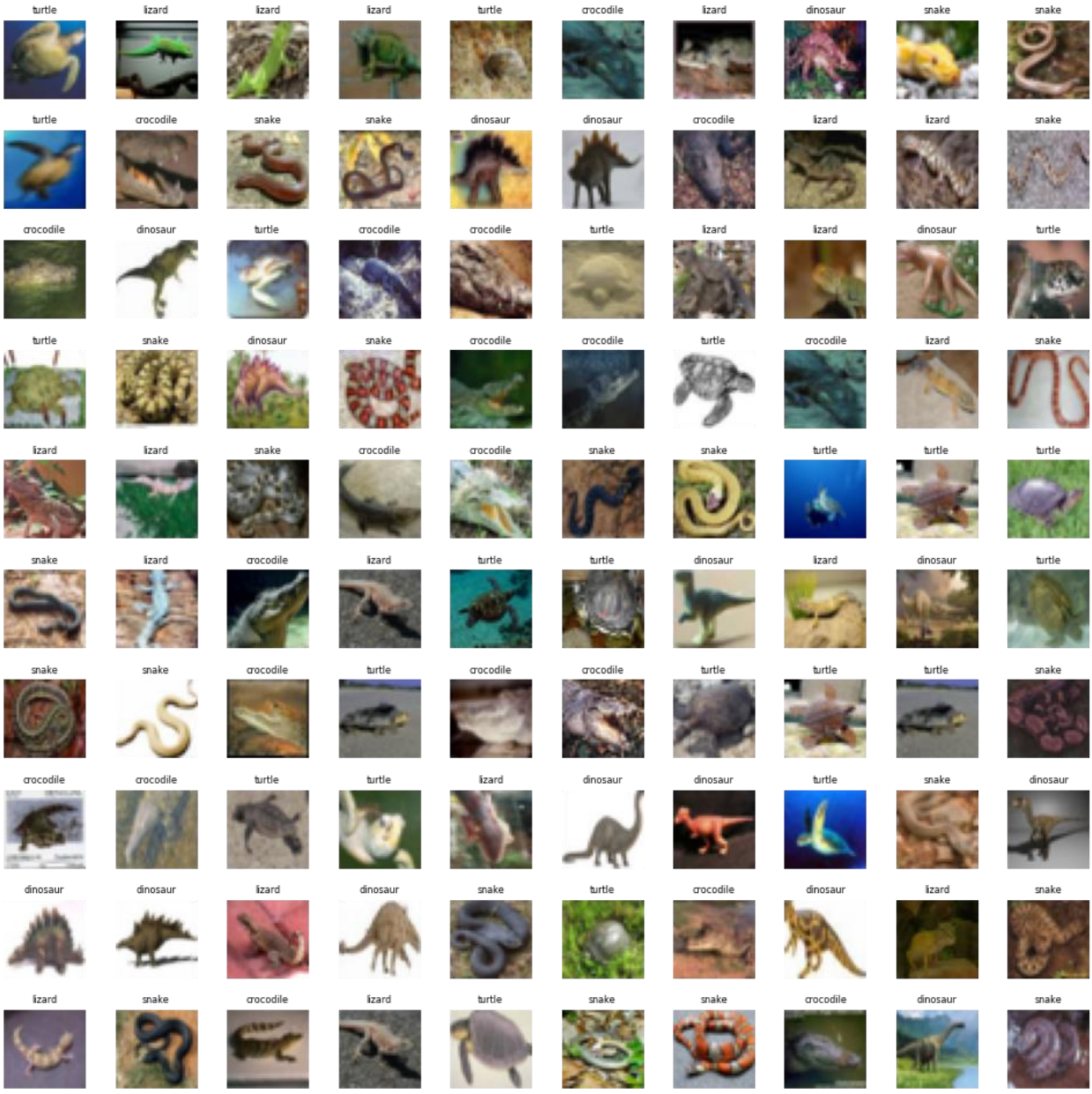
На фоне других результат мой сети неплохой, но и не самый высокий.

CIFAR-100

Набор данных аналогичен CIFAR-10, за исключением того, что он содержит 100 классов, каждый из которых содержит 600 изображений. На каждый класс приходится 500 обучающих изображений и 100 тестовых изображений. 100 классов в CIFAR-100 сгруппированы в 20 суперклассов. Каждое изображение имеет метку «точный» (класс, к которому оно принадлежит) и метку «грубое» (суперкласс, к которому оно принадлежит).

Согласно моему варианту из задания, мы берем только один суперкласс:

reptiles	crocodile, dinosaur, lizard, snake, turtle



Теперь мы хотим произвести классификацию на 15 классов (класс из cifar10 + reptiles). Теперь используем так называемый transfer learning. В уже созданной сверточной сети мы уберем только верхний полно связный слой и заменим на такой же, но уже из 15 нейронов.

```
x = resnet50_model.layers[-2].output
x = Dense(15, activation='softmax', kernel_initializer="he_normal",
                kernel_regularizer=regularizers.l2(weight_decay))(x)

new_model = Model(resnet50_model.input, x)
```

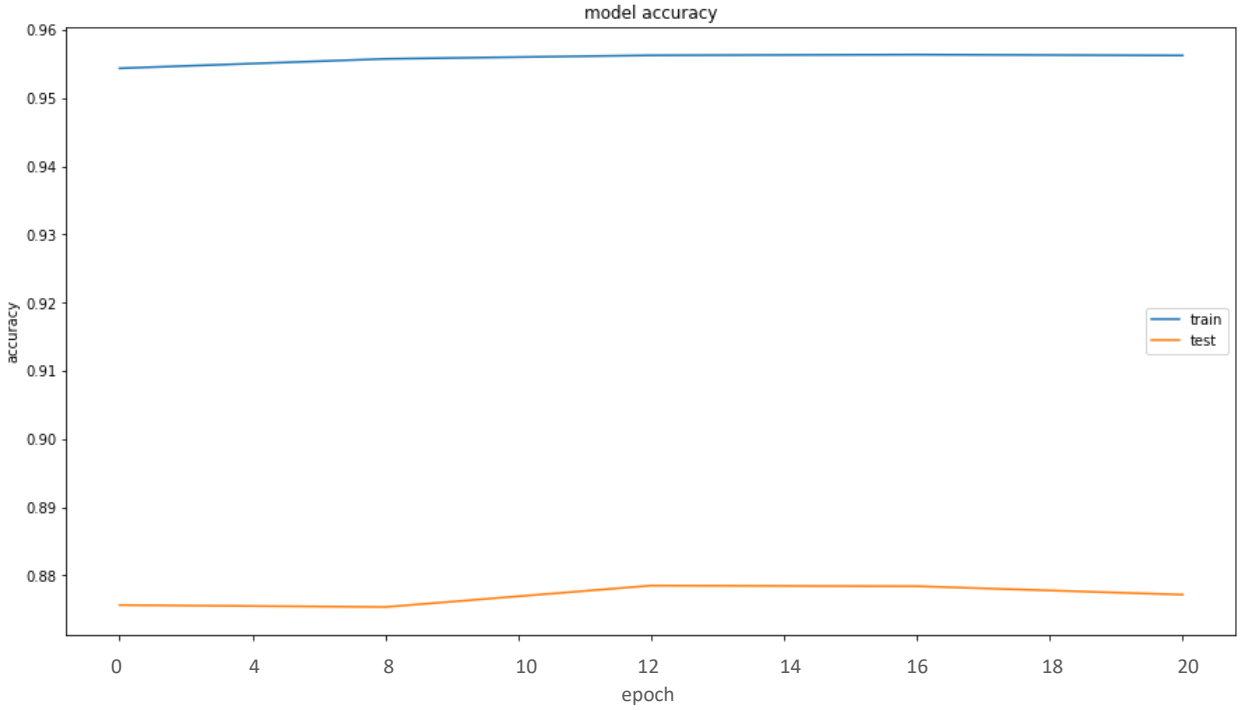
Веса на остальных слоях заморозим и не будем изменять.

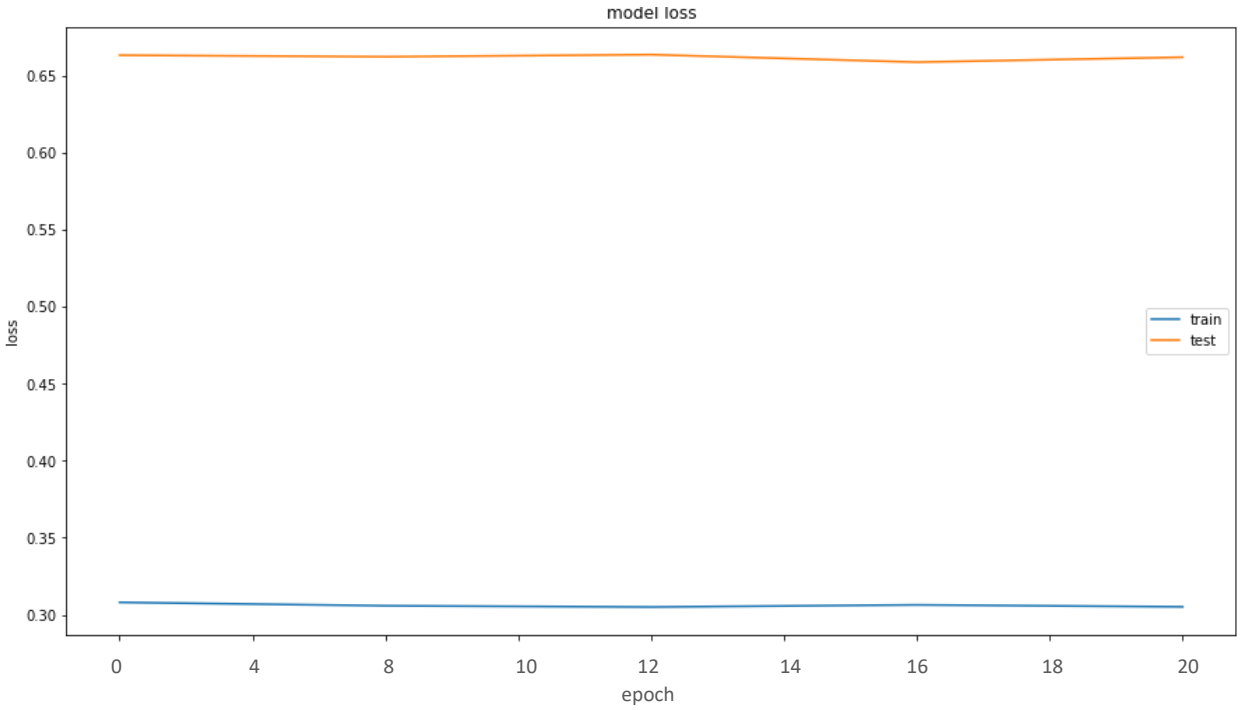
```
for m in new_model.layers:
  m.trainable = False
m = new_model.layers[-1]
m.trainable = True
```

Приступаем к дообучению (сделаем 20 эпох), предобратав входные изображения по той же схеме, как и в первой части лабораторной работы:

```
    # start training
resnet50_model = new_model.fit_generator(datagen.flow(x_train_, y_train_,batch_size=batch_size),
                         steps_per_epoch=iterations,
                         epochs=20,
                         callbacks=cbks,
                         validation_data=(x_test_, y_test_))
```

Динамика роста accuracy и уменьшения ошибки:





Метрики качества для обучающей выборки (усреднение macro):

```
y_ = np.argmax(y_train_, axis=1)
print('Accuracy для обучающего множества:', metrics.accuracy
print('Precision для обучающего множества:', metrics.precisi
print('Recall для обучающего множества:', metrics.recall_sco
print('F1-measure для обучающего множества:', metrics.f1_sco
```

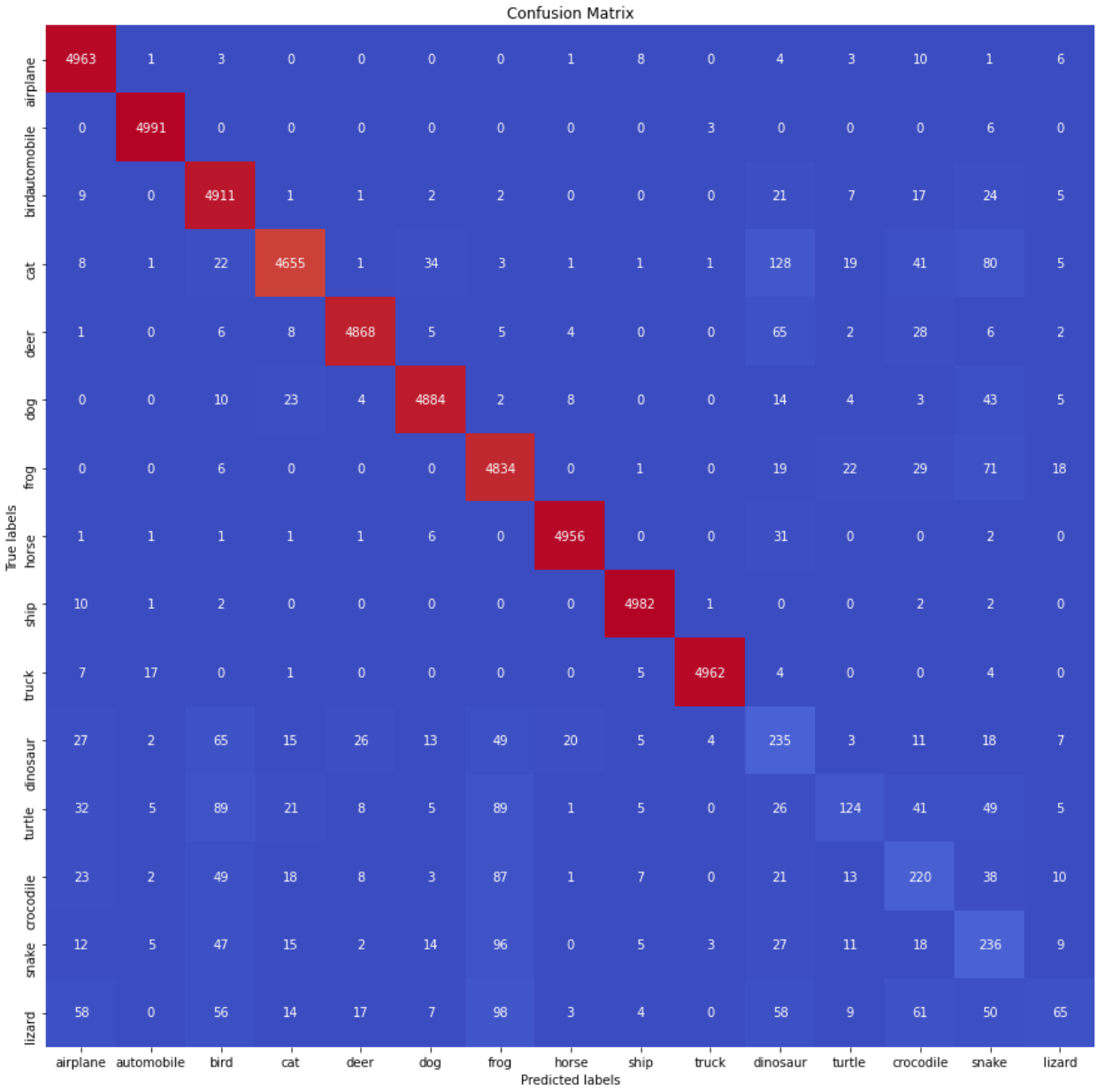
```
Accuracy для обучающего множества: 0.9502095238095238
Precision для обучающего множества: 0.7980169918042724
Recall для обучающего множества: 0.7707466666666666
F1-measure для обучающего множества: 0.7725732813175756
```

Метрики качества для тестовой выборки:

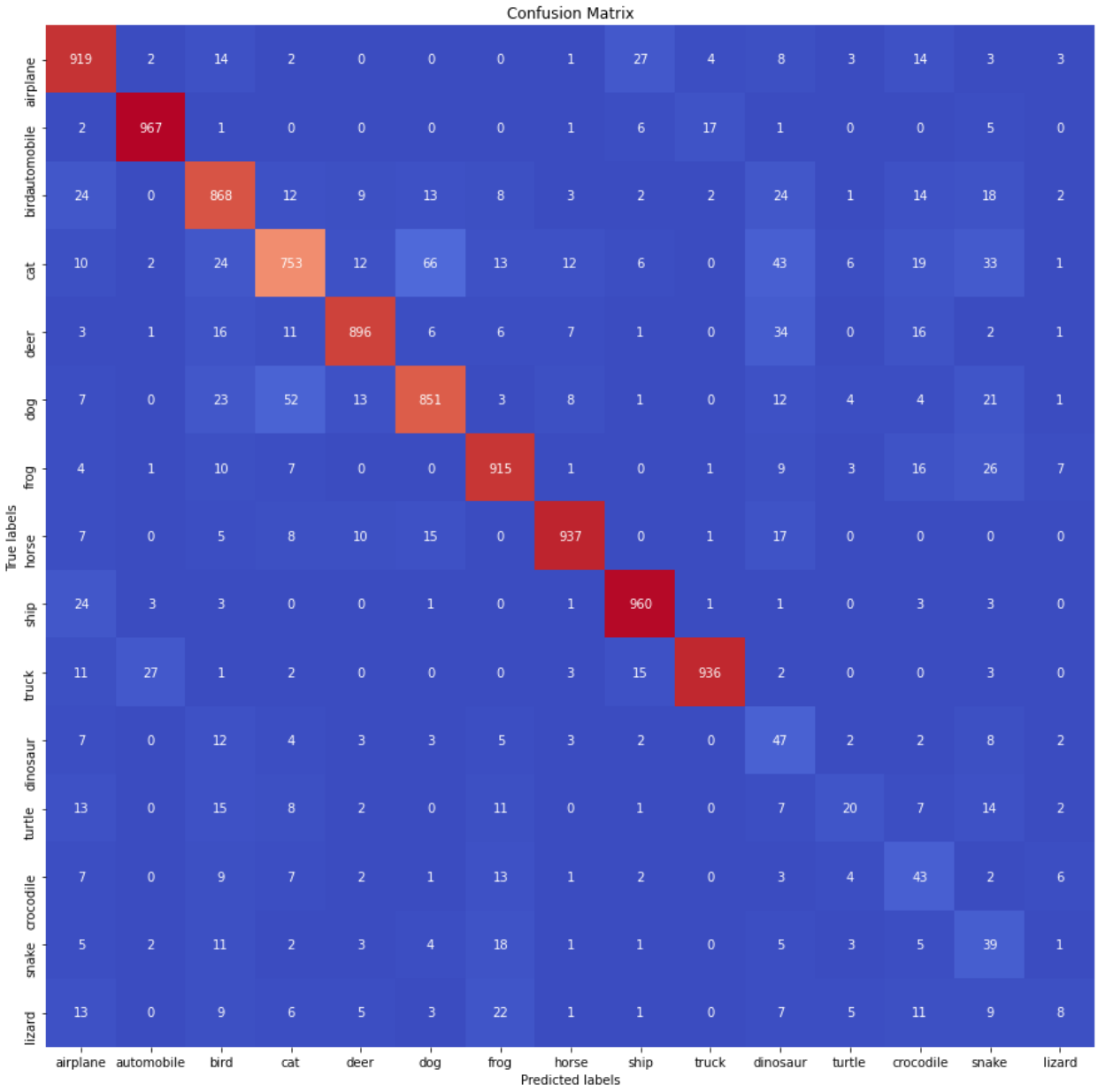
```
y_ = np.argmax(y_test_, axis=1)
print('Accuracy для тестового множества:', metrics.accuracy_sc
print('Precision для тестового множества:', metrics.precision_
print('Recall для тестового множества:', metrics.recall_score(
print('F1-measure для тестового множества:', metrics.f1_score(
```

```
Accuracy для тестового множества: 0.8722857142857143
Precision для тестового множества: 0.6976071370400571
Recall для тестового множества: 0.7048
F1-measure для тестового множества: 0.6901050230531232
```

Матрица ошибок для обучающей выборки:



Матрица ошибок для тестовой выборки:



Получилось интересно: дообученная на новых данных сеть плохо угадывает рептилий, но при этом она стала хуже различать и классы cifar-10. При этом рептилий сеть часто причисляет к классам frog (что в целом осмысленно), и bird (что удивляет)

Вывод

В данной работе проводилось обучение сети ResNet-50 на датасете Cifar-10 в соответствии со статьей. В результате удалось достичь того же уровня качества и воспроизвести вычисления, приведенные в статье. И в целом данная сеть хорошо себя показала на датасете.

Далее мы изменили данные, и для этого изменили и архитектуру сети, заменив последний слой. К данным добавили 5 видов рептилий из датасета Cifar-100, а потом дообучили нашу сеть на этих данных (вместе с cifar-10).

К сожалению, метрики precision, recall, матрица ошибок демонстрируют, что сеть плохо различает новые классы (рептилий). При этом рептилий сеть часто причисляет к классам frog (что в целом осмысленно, так как лягушки тоже рептилии), и bird (что странно, но по мнению сети птицы имеют какие-то схожие признаки с рептилиями).

Конечно, если бы мы обучали сеть на гораздо большем объеме (и более репрезентативным), как например ImageNet, то сверточной сеть наверняка показала бы себя прекрасно. Но в нашем случае она училась выделять признаки на весьма узком наборе данных с небольшим количеством классов.

Инструкция по запуску

Для запуска потребуется использовать Anaconda (Jupiter Notebook), но предпочтительнее запускать из Google Colab (https://colab.research.google.com/drive/1Vd0jthzN01izhuFGnjFqHrN5-0HYpstD?usp=sharing), так как там уже гарантированно установлены используемые библиотеки. Также все датасеты, использованные в отчете, загружаются в самом коде.

Затем нужно просто последовательно выполнить код в ячейках. Никакие параметры менять не нужно, иначе не получиться воспроизвести достигнутые результаты классификации в точности как полученные в отчете к работе.