Московский государственный технический университет им. Н.Э. Баумана
Факультет «Информатика и системы управления»
Кафедра «Системы обработки информации и управления»

Домашнее задание

ИСПОЛНИТЕЛЬ:

Елизаров Олег Олегович
Группа ИУ5-21М

_____

"__"_____2022 г.

Москва 2022

Целью работы является: Анализ современных методов машинного обучения и их применение для решения практических задач.

Часть 1. Выбор задачи

В рамках данной работы будет изучаться задача класса «Image Classification» [1]. Все исследования будут проводиться на основе Cifar-100.

Часть 2. Теоретический этап

Поставленная задача- классификация изображений. Классификация изображений – это процесс извлечения классов информации из многоканального растрового изображения.

То есть для каждого входного изображения наша модель должна определить класс, который по ее мнению является наиболее подходящим для изображения.

Cifar-100 - это набор данных из 50 000 цветных учебных изображений 32×32, помеченных более чем в 100 категориях, и 10 000 тестовых изображений. (Рисунок 1)

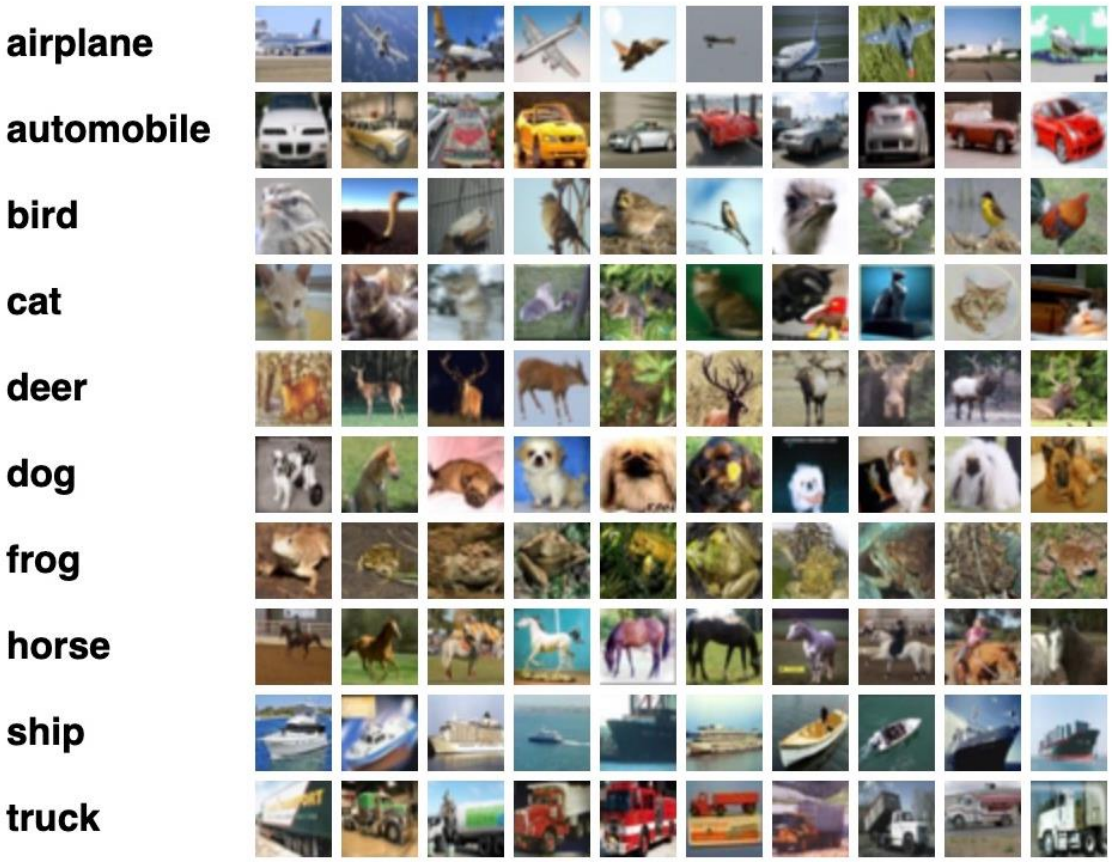

Рисунок 1. Пример содержания набора cifar-100

В обычном перцептроне, который представляет собой полносвязную нейронную сеть, каждый нейрон связан со всеми нейронами предыдущего слоя, причём каждая связь имеет свой персональный весовой коэффициент. В свёрточной нейронной сети в операции свёртки используется лишь ограниченная матрица весов небольшого размера, которую «двигают» по всему обрабатываемому слою (в самом начале — непосредственно по входному изображению), формируя после каждого сдвига сигнал активации для нейрона следующего слоя с аналогичной позицией. То есть для различных нейронов выходного слоя используются одна и та же матрица весов, которую также называют ядром свёртки. Её интерпретируют как графическое кодирование какого-либо признака, например, наличие наклонной линии под определённым углом. Тогда следующий слой, получившийся в результате операции свёртки такой матрицей весов, показывает наличие данного признака в обрабатываемом слое и её координаты, формируя так называемую карту признаков. Естественно, в свёрточной нейронной сети набор весов не один, а целая гамма, кодирующая элементы изображения (например линии и дуги под разными углами). При этом такие ядра свёртки не закладываются исследователем заранее, а формируются самостоятельно путём обучения сети классическим методом обратного распространения ошибки. Проход каждым набором весов формирует свой собственный экземпляр карты признаков, делая нейронную сеть многоканальной (много независимых карт признаков на одном слое).

В данной работе будут проанализированы две статьи по модели mobilenet.

Rethinking Depthwise Separable Convolutions: How Intra-Kernel Correlations Lead to Improved MobileNets и **Grouped Pointwise Convolutions Reduce Parameters in Convolutional Neural Networks**

Обе статьи направлены на оптимизации архитектуры сети, улучшение качества модели с помощью изменения слоев и ядра свертки.([2], [3])

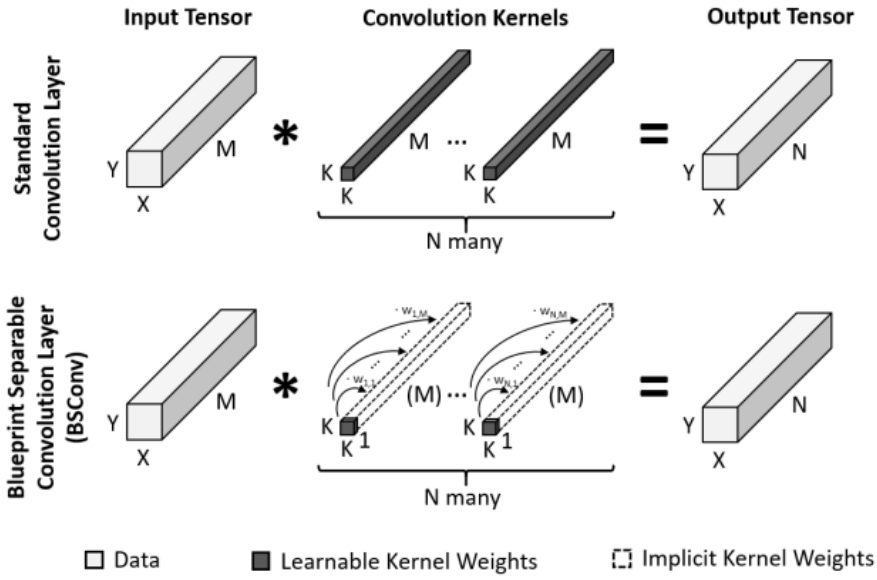

Рисунок 2. Пример изменения сверточных слоев в первой статье

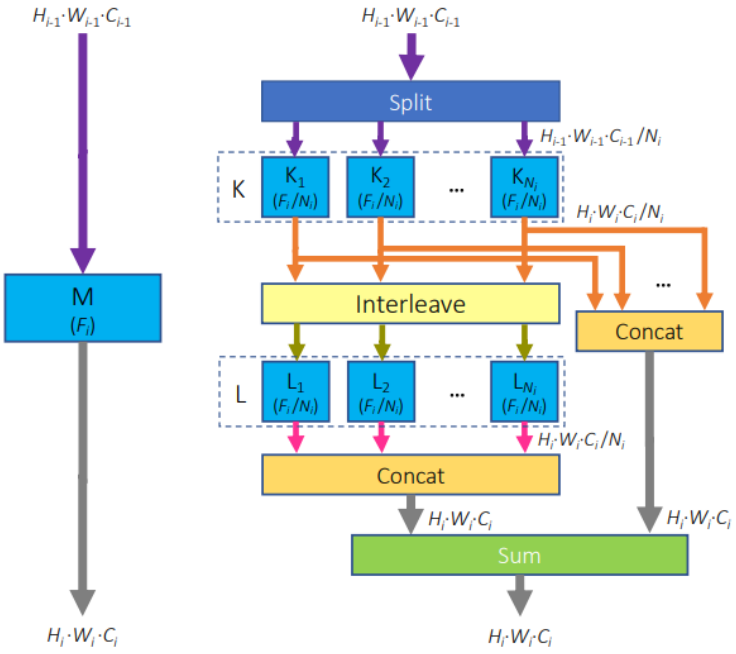

Рисунок 3. Пример изменения фильтрации внутри сверточных слоев во второй статье.

Однако, даже с этими модификациями было показано, что результат не превышает 80% accuracy.

Далее, в данной работе будет показано изменений уже готовых моделей из этих двух статей с помощью добавления новых слоев классификации и нового оптимизатора.

Часть 3. Практический этап

При проведении практических работ была сразу взята готовая модель mobilenet, которую и оптимизировали в представленных выше статьях.

```
model = torch.hub.load("chenyaofo/pytorch-cifar-models",
                       "cifar100_mobilenetv2_x0_5",
                       pretrained=True)
```

Ниже представлен перечень слоев , входящих в эту модель:

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
         Normalize-1            [-1, 3, 32, 32]               0
            Conv2d-2           [-1, 16, 32, 32]             432
       BatchNorm2d-3           [-1, 16, 32, 32]              32
             ReLU6-4           [-1, 16, 32, 32]               0
            Conv2d-5           [-1, 16, 32, 32]             144
       BatchNorm2d-6           [-1, 16, 32, 32]              32
             ReLU6-7           [-1, 16, 32, 32]               0
            Conv2d-8            [-1, 8, 32, 32]             128
       BatchNorm2d-9            [-1, 8, 32, 32]              16
 InvertedResidual-10            [-1, 8, 32, 32]               0
           Conv2d-11           [-1, 48, 32, 32]             384
      BatchNorm2d-12           [-1, 48, 32, 32]              96
             ReLU6-13           [-1, 48, 32, 32]               0
           Conv2d-14           [-1, 48, 32, 32]             432
      BatchNorm2d-15           [-1, 48, 32, 32]              96
             ReLU6-16           [-1, 48, 32, 32]               0
           Conv2d-17           [-1, 16, 32, 32]             768
      BatchNorm2d-18           [-1, 16, 32, 32]              32
 InvertedResidual-19           [-1, 16, 32, 32]               0
           Conv2d-20           [-1, 96, 32, 32]           1,536
      BatchNorm2d-21           [-1, 96, 32, 32]             192
             ReLU6-22           [-1, 96, 32, 32]               0
           Conv2d-23           [-1, 96, 32, 32]             864
      BatchNorm2d-24           [-1, 96, 32, 32]             192
             ReLU6-25           [-1, 96, 32, 32]               0
           Conv2d-26           [-1, 16, 32, 32]           1,536
      BatchNorm2d-27           [-1, 16, 32, 32]              32
 InvertedResidual-28           [-1, 16, 32, 32]               0
           Conv2d-29           [-1, 96, 32, 32]           1,536
      BatchNorm2d-30           [-1, 96, 32, 32]             192
             ReLU6-31           [-1, 96, 32, 32]               0
           Conv2d-32           [-1, 96, 16, 16]             864
```

```
     BatchNorm2d-33          [-1, 96, 16, 16]             192
           ReLU6-34          [-1, 96, 16, 16]               0
         Conv2d-35          [-1, 16, 16, 16]           1,536
     BatchNorm2d-36          [-1, 16, 16, 16]              32
InvertedResidual-37          [-1, 16, 16, 16]               0
         Conv2d-38          [-1, 96, 16, 16]           1,536
     BatchNorm2d-39          [-1, 96, 16, 16]             192
           ReLU6-40          [-1, 96, 16, 16]               0
         Conv2d-41          [-1, 96, 16, 16]             864
     BatchNorm2d-42          [-1, 96, 16, 16]             192
           ReLU6-43          [-1, 96, 16, 16]               0
         Conv2d-44          [-1, 16, 16, 16]           1,536
     BatchNorm2d-45          [-1, 16, 16, 16]              32
InvertedResidual-46          [-1, 16, 16, 16]               0
         Conv2d-47          [-1, 96, 16, 16]           1,536
     BatchNorm2d-48          [-1, 96, 16, 16]             192
           ReLU6-49          [-1, 96, 16, 16]               0
         Conv2d-50          [-1, 96, 16, 16]             864
     BatchNorm2d-51          [-1, 96, 16, 16]             192
           ReLU6-52          [-1, 96, 16, 16]               0
         Conv2d-53          [-1, 16, 16, 16]           1,536
     BatchNorm2d-54          [-1, 16, 16, 16]              32
InvertedResidual-55          [-1, 16, 16, 16]               0
         Conv2d-56          [-1, 96, 16, 16]           1,536
     BatchNorm2d-57          [-1, 96, 16, 16]             192
           ReLU6-58          [-1, 96, 16, 16]               0
         Conv2d-59           [-1, 96, 8, 8]             864
     BatchNorm2d-60           [-1, 96, 8, 8]             192
           ReLU6-61           [-1, 96, 8, 8]               0
         Conv2d-62           [-1, 32, 8, 8]           3,072
     BatchNorm2d-63           [-1, 32, 8, 8]              64
InvertedResidual-64           [-1, 32, 8, 8]               0
         Conv2d-65          [-1, 192, 8, 8]           6,144
     BatchNorm2d-66          [-1, 192, 8, 8]             384
           ReLU6-67          [-1, 192, 8, 8]               0
         Conv2d-68          [-1, 192, 8, 8]           1,728
     BatchNorm2d-69          [-1, 192, 8, 8]             384
           ReLU6-70          [-1, 192, 8, 8]               0
         Conv2d-71           [-1, 32, 8, 8]           6,144
     BatchNorm2d-72           [-1, 32, 8, 8]              64
InvertedResidual-73           [-1, 32, 8, 8]               0
         Conv2d-74          [-1, 192, 8, 8]           6,144
     BatchNorm2d-75          [-1, 192, 8, 8]             384
           ReLU6-76          [-1, 192, 8, 8]               0
         Conv2d-77          [-1, 192, 8, 8]           1,728
     BatchNorm2d-78          [-1, 192, 8, 8]             384
           ReLU6-79          [-1, 192, 8, 8]               0
         Conv2d-80           [-1, 32, 8, 8]           6,144
     BatchNorm2d-81           [-1, 32, 8, 8]              64
InvertedResidual-82           [-1, 32, 8, 8]               0
         Conv2d-83          [-1, 192, 8, 8]           6,144
     BatchNorm2d-84          [-1, 192, 8, 8]             384
           ReLU6-85          [-1, 192, 8, 8]               0
         Conv2d-86          [-1, 192, 8, 8]           1,728
     BatchNorm2d-87          [-1, 192, 8, 8]             384
           ReLU6-88          [-1, 192, 8, 8]               0
         Conv2d-89           [-1, 32, 8, 8]           6,144
     BatchNorm2d-90           [-1, 32, 8, 8]              64
InvertedResidual-91           [-1, 32, 8, 8]               0
         Conv2d-92          [-1, 192, 8, 8]           6,144
     BatchNorm2d-93          [-1, 192, 8, 8]             384
```

ReLU6-94	[-1, 192, 8, 8]	0
Conv2d-95	[-1, 192, 8, 8]	1,728
BatchNorm2d-96	[-1, 192, 8, 8]	384
ReLU6-97	[-1, 192, 8, 8]	0
Conv2d-98	[-1, 48, 8, 8]	9,216
BatchNorm2d-99	[-1, 48, 8, 8]	96
InvertedResidual-100	[-1, 48, 8, 8]	0
Conv2d-101	[-1, 288, 8, 8]	13,824
BatchNorm2d-102	[-1, 288, 8, 8]	576
ReLU6-103	[-1, 288, 8, 8]	0
Conv2d-104	[-1, 288, 8, 8]	2,592
BatchNorm2d-105	[-1, 288, 8, 8]	576
ReLU6-106	[-1, 288, 8, 8]	0
Conv2d-107	[-1, 48, 8, 8]	13,824
BatchNorm2d-108	[-1, 48, 8, 8]	96
InvertedResidual-109	[-1, 48, 8, 8]	0
Conv2d-110	[-1, 288, 8, 8]	13,824
BatchNorm2d-111	[-1, 288, 8, 8]	576
ReLU6-112	[-1, 288, 8, 8]	0
Conv2d-113	[-1, 288, 8, 8]	2,592
BatchNorm2d-114	[-1, 288, 8, 8]	576
ReLU6-115	[-1, 288, 8, 8]	0
Conv2d-116	[-1, 48, 8, 8]	13,824
BatchNorm2d-117	[-1, 48, 8, 8]	96
InvertedResidual-118	[-1, 48, 8, 8]	0
Conv2d-119	[-1, 288, 8, 8]	13,824
BatchNorm2d-120	[-1, 288, 8, 8]	576
ReLU6-121	[-1, 288, 8, 8]	0
Conv2d-122	[-1, 288, 4, 4]	2,592
BatchNorm2d-123	[-1, 288, 4, 4]	576
ReLU6-124	[-1, 288, 4, 4]	0
Conv2d-125	[-1, 80, 4, 4]	23,040
BatchNorm2d-126	[-1, 80, 4, 4]	160
InvertedResidual-127	[-1, 80, 4, 4]	0
Conv2d-128	[-1, 480, 4, 4]	38,400
BatchNorm2d-129	[-1, 480, 4, 4]	960
ReLU6-130	[-1, 480, 4, 4]	0
Conv2d-131	[-1, 480, 4, 4]	4,320
BatchNorm2d-132	[-1, 480, 4, 4]	960
ReLU6-133	[-1, 480, 4, 4]	0
Conv2d-134	[-1, 80, 4, 4]	38,400
BatchNorm2d-135	[-1, 80, 4, 4]	160
InvertedResidual-136	[-1, 80, 4, 4]	0
Conv2d-137	[-1, 480, 4, 4]	38,400
BatchNorm2d-138	[-1, 480, 4, 4]	960
ReLU6-139	[-1, 480, 4, 4]	0
Conv2d-140	[-1, 480, 4, 4]	4,320
BatchNorm2d-141	[-1, 480, 4, 4]	960
ReLU6-142	[-1, 480, 4, 4]	0
Conv2d-143	[-1, 80, 4, 4]	38,400
BatchNorm2d-144	[-1, 80, 4, 4]	160
InvertedResidual-145	[-1, 80, 4, 4]	0
Conv2d-146	[-1, 480, 4, 4]	38,400
BatchNorm2d-147	[-1, 480, 4, 4]	960
ReLU6-148	[-1, 480, 4, 4]	0
Conv2d-149	[-1, 480, 4, 4]	4,320
BatchNorm2d-150	[-1, 480, 4, 4]	960
ReLU6-151	[-1, 480, 4, 4]	0
Conv2d-152	[-1, 160, 4, 4]	76,800
BatchNorm2d-153	[-1, 160, 4, 4]	320
InvertedResidual-154	[-1, 160, 4, 4]	0

```
          Conv2d-155           [-1, 1280, 4, 4]         204,800
     BatchNorm2d-156           [-1, 1280, 4, 4]           2,560
           ReLU6-157           [-1, 1280, 4, 4]               0
         Dropout-158                 [-1, 1280]               0
          Linear-159                  [-1, 100]         128,100
     MobileNetV2-160                  [-1, 100]               0
================================================================
Total params: 815,780
Trainable params: 815,780

Non-trainable params: 0
```

Далее к этой модели мы применим подход Transfer learning.

Замораживаем веса текущей модели, добавляем свои слои классификатора. Далее важно отметить, что мы применим особый оптимизатор SAM(Sharpness-Aware Minimization), который помогает многим моделям значительно повышать результаты.

```
base_optimizer = torch.optim.SGD  # define an optimizer for the "sharpness
-aware" update
optimizer = SAM(model.parameters(), base_optimizer, lr=0.1, momentum=0.9)
```

Далее обучим нашу модель на датасете CIFAR-100, для упрощения и ускорения обучения , возьмем 3 класса в этом датасете и подготовим именно для них.

Ниже полученные результаты:

```
train
              precision    recall  f1-score   support

           5     0.9843    1.0000    0.9921       500
          21     0.9896    0.9540    0.9715       500
          35     0.9608    0.9800    0.9703       500

    accuracy                         0.9780      1500
   macro avg     0.9782    0.9780    0.9779      1500
weighted avg     0.9782    0.9780    0.9779      1500

------------------------------------------------
test
              precision    recall  f1-score   support

           5     0.9700    0.9700    0.9700       100
          21     0.9143    0.9600    0.9366       100
          35     0.9579    0.9100    0.9333       100

    accuracy                         0.9467       300
   macro avg     0.9474    0.9467    0.9466       300
weighted avg     0.9474    0.9467    0.9466       300
```

Видим, что все показатели выше 0.9, что является весьма достойным результатом. Безусловно, такие высокие показатели достигнуты и из-за уменьшения количество классов классификации, но увеличение числа классов приведет к значительному увеличению времени обучения модели.

Полный код работы с моделью прикрепляется ниже

```
with open('cifar-100-python/train', 'rb') as f:
    data_train = pickle.load(f, encoding='latin1')
with open('cifar-100-python/test', 'rb') as f:
    data_test = pickle.load(f, encoding='latin1')

# Здесь указать ваши классы по варианту!!!
CLASSES = [5, 21, 35]

train_X = data_train['data'].reshape(-1, 3, 32, 32)
train_X = np.transpose(train_X, [0, 2, 3, 1]) # NCHW -> NHWC
train_y = np.array(data_train['fine_labels'])
mask = np.isin(train_y, CLASSES)
train_X = train_X[mask].copy()
train_y = train_y[mask].copy()
train_y = np.unique(train_y, return_inverse=1)[1]
del data_train

test_X = data_test['data'].reshape(-1, 3, 32, 32)
test_X = np.transpose(test_X, [0, 2, 3, 1])
test_y = np.array(data_test['fine_labels'])
mask = np.isin(test_y, CLASSES)
test_X = test_X[mask].copy()
test_y = test_y[mask].copy()
test_y = np.unique(test_y, return_inverse=1)[1]
del data_test
Image.fromarray(train_X[50]).resize((256,256))

class CifarDataset(Dataset):

     def __init__(self, X, y, transform=None, p=0.0):
         assert X.size(0) == y.size(0)
         super(Dataset, self).__init__()
         self.X = X
         self.y = y
         self.transform = transform
         self.prob = p

     def __len__(self):
         return self.y.size(0)
```

```
     def __getitem__(self, index):
        x = self.X[index]
        if self.transform and np.random.random()<self.prob:
            x = self.transform(x.permute(2, 0, 1)/255.).permute(1, 2, 0)*2
55.
        y = self.y[index]
        return x, y

transform = T.Compose([
    #  T.ColorJitter(brightness=0.1, contrast=0.1, saturation=0.2, hue=0.0
),
    #  T.RandomAffine(degrees=15, translate=(0.1, 0.1), scale=(0.8, 1.2),
                      # shear=5),
      T.RandomVerticalFlip(),
      T.RandomHorizontalFlip()
])

Image.fromarray((transform(torch.Tensor(train_X[50]).permute(2, 0, 1)/255.
).\
                 permute(1, 2, 0).numpy()*255.).astype(np.uint8)).\
                 resize((256, 256))


batch_size = 128

dataloader = {}
for (X, y), part in zip([(train_X, train_y), (test_X, test_y)],
                        ['train', 'test']):
    tensor_x = torch.Tensor(X)
    tensor_y = F.one_hot(torch.Tensor(y).to(torch.int64),
                                      num_classes=len(CLASSES))/1.
    dataset = CifarDataset(tensor_x, tensor_y,
                           transform if part=='train' else None,
                           p=0.5) # создание объекта датасета
    dataloader[part] = DataLoader(dataset, batch_size=batch_size,
                                  prefetch_factor=8 if part=='train' else
2,
                                  num_workers=2, persistent_workers=True,
                                  shuffle=True) # создание экземпляра клас
ca DataLoader
dataloader



model = torch.hub.load("chenyaofo/pytorch-cifar-models",
                       "cifar100_mobilenetv2_x0_5",
                       #'cifar100_resnet20',
                       pretrained=True)
model.to(device)

new_model = nn.Sequential(
```

```
   Normalize([0.5074,0.4867,0.4411],[0.2011,0.1987,0.2025]),# https://blog
.jovian.ai/image-classification-of-cifar100-dataset-using-pytorch-
8b7145242df1
   model
).to(device)
print(new_model(torch.rand(1, 32, 32, 3).to(device)))
summary(new_model, input_size=(32, 32, 3))
# new_model

## mobilenetv2

in_features = new_model[1].classifier[1].in_features
new_model[1].classifier[1] =  nn.Sequential(
    nn.Linear(in_features=in_features,

              out_features=len(CLASSES),

              bias=True),
new_model.to(device)
summary(new_model, input_size=(32, 32, 3))
print(new_model(torch.rand(1, 32, 32, 3).to(device)))
print("Обучаемые параметры:")
keep_last = 2
total = len([*new_model.named_parameters()])
params_to_update = []
for i, (name, param) in enumerate(new_model.named_parameters()):
    if i < total - keep_last:
        param.requires_grad = False
    else:
        params_to_update.append(param)
        print("\t",name)
summary(new_model, input_size=(32, 32, 3))
criterion = nn.CrossEntropyLoss(label_smoothing=0.1)
base_optimizer = torch.optim.SGD  # define an optimizer for the "sharpness
-aware" update
optimizer = SAM(model.parameters(), base_optimizer, lr=0.1, momentum=0.9)

scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=20, gamma=0.5)

EPOCHS = 60

REDRAW_EVERY = 10
steps_per_epoch = len(dataloader['train'])
steps_per_epoch_val = len(dataloader['test'])
# NEW
pbar = tqdm(total=EPOCHS*steps_per_epoch)
losses = []
losses_val = []
passed = 0
# для создания чекпоинта
best_acc = 0
checkpoint_path = 'cifar_cnn_fine.pth'
```

```
for epoch in range(EPOCHS):  # проход по набору данных несколько раз
    tmp = []
    new_model.train()
    for i, batch in enumerate(dataloader['train'], 0):
        # получение одного минибатча; batch это двуэлементный список из [i
nputs, labels]
        inputs, labels = batch
        # на GPU
        inputs, labels = inputs.to(device), labels.to(device)

        # очищение прошлых градиентов с прошлой итерации
        optimizer.zero_grad()

        # прямой + обратный проходы + оптимизация
        outputs = new_model(inputs)
        loss = criterion(outputs, labels)
        #loss = F.cross_entropy(outputs, labels)
        loss.backward()
        optimizer.step()

        # для подсчёта статистик
        accuracy = (labels.detach().argmax(dim=-
1)==outputs.detach().argmax(dim=-1)).\
                    to(torch.float32).mean().cpu()*100
        tmp.append((loss.item(), accuracy.item()))
        pbar.update(1)
    losses.append((np.mean(tmp, axis=0),
                   np.percentile(tmp, 25, axis=0),
                   np.percentile(tmp, 75, axis=0)))
    scheduler.step() # обновляем learning_rate каждую эпоху
    tmp = []
    new_model.eval()
    with torch.no_grad(): # отключение автоматического дифференцирования
        for i, data in enumerate(dataloader['test'], 0):
            inputs, labels = data
            # на GPU
            inputs, labels = inputs.to(device), labels.to(device)

            outputs = new_model(inputs)
            loss = criterion(outputs, labels)
            accuracy = (labels.argmax(dim=-1)==outputs.argmax(dim=-1)).\
                        to(torch.float32).mean().cpu()*100
            tmp.append((loss.item(), accuracy.item()))
    losses_val.append((np.mean(tmp, axis=0),
                       np.percentile(tmp, 25, axis=0),
                       np.percentile(tmp, 75, axis=0)))
    # сохранение чекпоинта
    acc = losses_val[-1][0][1]
    if acc > best_acc:
        best_acc = acc
        torch.save(new_model.state_dict(), checkpoint_path)
```

```
    # обновление графиков
    if (epoch+1) % REDRAW_EVERY != 0:
        continue
    clear_output(wait=False)
    print('Эпоха: %s\n'
          'Лучшая доля правильных ответов: %s\n'
          'Текущая доля правильных ответов: %s' % (epoch+1, best_acc, acc)
)
    passed += pbar.format_dict['elapsed']
    pbar = tqdm(total=EPOCHS*steps_per_epoch, miniters=5)
    pbar.update((epoch+1)*steps_per_epoch)
    x_vals = np.arange(epoch+1)
    _, ax = plt.subplots(1, 2, figsize=(15, 5))
    stats = np.array(losses)
    stats_val = np.array(losses_val)
    ax[1].set_ylim(stats_val[:, 0, 1].min()-5, 100)
    ax[1].grid(axis='y')
    for i, title in enumerate(['CCE', 'Accuracy']):
        ax[i].plot(x_vals, stats[:, 0, i], label='train')
        ax[i].fill_between(x_vals, stats[:, 1, i],
                           stats[:, 2, i], alpha=0.4)
        ax[i].plot(x_vals, stats_val[:, 0, i], label='val')
        ax[i].fill_between(x_vals,
                           stats_val[:, 1, i],
                           stats_val[:, 2, i], alpha=0.4)
        ax[i].legend()
        ax[i].set_title(title)
    plt.show()
new_model.load_state_dict(torch.load(checkpoint_path))
print('Обучение закончено за %s секунд' % passed)

batch_size = 128

dataloader = {}
for (X, y), part in zip([(train_X, train_y), (test_X, test_y)],
                        ['train', 'test']):
    tensor_x = torch.Tensor(X)
    tensor_y = F.one_hot(torch.Tensor(y).to(torch.int64),
                                       num_classes=len(CLASSES))/1.
    dataset = CifarDataset(tensor_x, tensor_y,
                           transform=None,
                           p=0.0) # создание объекта датасета
    dataloader[part] = DataLoader(dataset, batch_size=batch_size,
                                  num_workers=2, shuffle=True) # создание
экземпляра класса DataLoader
dataloader

for part in ['train', 'test']:

    y_pred = []
    y_true = []
    with torch.no_grad(): # отключение автоматического дифференцирования
```

```
        for i, data in enumerate(dataloader[part], 0):
            inputs, labels = data
             # на GPU
            inputs, labels = inputs.to(device), labels.to(device)

            outputs = new_model(inputs).detach().cpu().numpy()
            y_pred.append(outputs)
            y_true.append(labels.cpu().numpy())
        y_true = np.concatenate(y_true)
        y_pred = np.concatenate(y_pred)
        print(part)
        print(classification_report(y_true.argmax(axis=-
1), y_pred.argmax(axis=-1),
                                    digits=4, target_names=list(map(str, C
LASSES))))
        print('-'*50)
```

4. Вывод

В данной работе были изучены две статьи по модели mobilenet. Было проведено исследование причин просадки метрик данный модели, по отношению к другим. Создан код, в котором добавлены возможные улучшения для модели и посчитаны метрики для него.

Литература

1) https://paperswithcode.com/task/image-classification
2) https://arxiv.org/abs/2010.01412v3
3) https://www.researchgate.net/publication/360226228_Grouped_Pointwise_Convolutions_Reduce_Parameters_in_Convolutional_Neural_Networks
4)