Классификация изображений с помощью сверточной нейронной сети (CNN) 2.0

Суть проекта:

Распознавание и классификация изображений.

Изменения:

1. Увеличение распознавания классов изображений с 10 до 100.
2. Добавлена аугментация изображений.
3. Изменен оптимизатор Adam -> SGD.

Увеличение распознавания классов изображений с 10 до 100:

Набор данных состоит из 60 000 цветных изображений размером 32x32. Он имеет 100 классов, содержащих по 600 изображений каждый. В каждом классе имеется 500 обучающих и 100 тестовых изображений.

Классы изображений до:

Самолет, автомобиль, птица, кошка, олень, собака, лягушка, конь, корабль, грузовик.

Классы изображений после:

Бобр, дельфин, выдра, тюлень, кит, аквариумные рыбки, камбала, скат, акула, форель, орхидеи, маки, розы, подсолнухи, тюльпаны, бутылки, миски, банки, чашки, тарелки, яблоки, грибы, апельсины, груши, сладкий перец, часы, клавиатура компьютера, лампа, телефон, телевизор, кровать, стул, диван, стол, шкаф, пчела, жук, бабочка, гусеница, таракан, медведь, леопард, лев, тигр, волк, мост, замок, дом, дорога, небоскреб, облако, лес, горы, равнина, море, верблюд, крупный рогатый скот, шимпанзе, слон, кенгуру, лиса, дикобраз, опоссум, енот, скунс, краб, омар, улитка, паук, червь, малыш, мальчик, девочка, мужчина, женщина, крокодил, динозавр, ящерица, змея, черепаха, хомяк, мышь, кролик, землеройка, белка, клен, дуб, пальма,

сосна, ива, велосипед, автобус, мотоцикл, пикап, поезд, газонокосилка, ракета, трамвай, танк, трактор.

Добавлена аугментация изображений:

Для лучшего обучения нам понадобится большой объем данных, но так как набор данных у нас ограничен, будем рандомизировать набор данных.

```
# Нормализация и аугментация

stats = ((average / len(images)).tolist(), (standard_dev / len(images)).tolist())
train_tfms = tt.Compose([tt.RandomCrop(32, padding=4,padding_mode='reflect'),
                         tt.RandomHorizontalFlip(),
                         tt.ToTensor(),
                         tt.Normalize(*stats,inplace=True)
                        ])
valid_tfms = tt.Compose([tt.ToTensor(), tt.Normalize(*stats)
                        ])
```

Изменен оптимизатор на SGD:

```
def fit_one_cycle(epochs, max_lr, model, train_loader, val_loader,
                  weight_decay=0, grad_clip=None, opt_func=torch.optim.SGD):
    torch.cuda.empty_cache()
    history = []
```

Результаты:

```
%%time
history += fit_one_cycle(epochs, max_lr, model, train_dl, valid_dl,
                             grad_clip=grad_clip,
                             weight_decay=weight_decay,
                             opt_func=opt_func)
```

```
Epoch [0], last_lr: 0.00278, train_loss: 3.6737, val_loss: 5.3941, val_acc:
0.1232
Epoch [1], last_lr: 0.00759, train_loss: 2.9928, val_loss: 3.6012, val_acc:
0.2034
Epoch [2], last_lr: 0.01000, train_loss: 2.3365, val_loss: 2.4847, val_acc:
0.3533
Epoch [3], last_lr: 0.00950, train_loss: 2.0429, val_loss: 4.1831, val_acc:
0.3877
Epoch [4], last_lr: 0.00812, train_loss: 1.8061, val_loss: 1.7336, val_acc:
0.5220
Epoch [5], last_lr: 0.00611, train_loss: 1.4439, val_loss: 1.6820, val_acc:
0.5247
Epoch [6], last_lr: 0.00389, train_loss: 1.2261, val_loss: 1.4567, val_acc:
0.5875
Epoch [7], last_lr: 0.00188, train_loss: 0.9715, val_loss: 1.2080, val_acc:
0.6553
Epoch [8], last_lr: 0.00050, train_loss: 0.7144, val_loss: 1.0354, val_acc:
0.6984
Epoch [9], last_lr: 0.00000, train_loss: 0.5434, val_loss: 1.0055, val_acc:
0.7071
CPU times: user 2min 10s, sys: 1min 6s, total: 3min 17s
Wall time: 7min 58s
```

После обучения модели для первых 10 эпох мы смогли достичь оценки валидации почти 0.70, что соответствует точности 70%.

Будем тренировать нашу модель немного дольше и с более низкой скоростью, чтобы добиться лучшего результата.

```
%%time
history += fit_one_cycle(epochs, 0.001, model, train_dl, valid_dl,
                             grad_clip=grad_clip,
                             weight_decay=weight_decay,
                             opt_func=opt_func)
```

```
Epoch [0], last_lr: 0.00028, train_loss: 0.5164, val_loss: 1.0096, val_acc:
0.7074
Epoch [1], last_lr: 0.00076, train_loss: 0.5310, val_loss: 1.0755, val_acc:
0.6920
Epoch [2], last_lr: 0.00100, train_loss: 0.5623, val_loss: 1.1999, val_acc:
0.6655
Epoch [3], last_lr: 0.00095, train_loss: 0.5250, val_loss: 1.2509, val_acc:
0.6614
Epoch [4], last_lr: 0.00081, train_loss: 0.4410, val_loss: 1.1573, val_acc:
0.6851
Epoch [5], last_lr: 0.00061, train_loss: 0.3462, val_loss: 1.0925, val_acc:
0.7001
Epoch [6], last_lr: 0.00039, train_loss: 0.2504, val_loss: 1.0727, val_acc:
0.7104
Epoch [7], last_lr: 0.00019, train_loss: 0.1817, val_loss: 1.0358, val_acc:
0.7221
Epoch [8], last_lr: 0.00005, train_loss: 0.1356, val_loss: 1.0235, val_acc:
0.7247
Epoch [9], last_lr: 0.00000, train_loss: 0.1194, val_loss: 1.0220, val_acc:
0.7257
CPU times: user 2min 13s, sys: 1min 9s, total: 3min 22s
Wall time: 7min 55s
```

После обучения еще 10 эпох мы видим, что в значительной степени достигли пикового значения модели, поскольку показатель проверки остается в пределах 0.72 , что составляет 72%.

Точность прошлой версии была 61.82%.

График точности:

```
plot_accuracies(history)
```

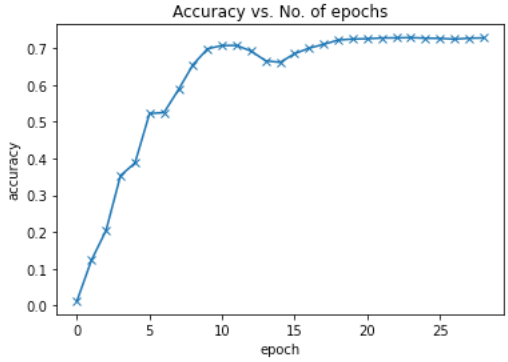
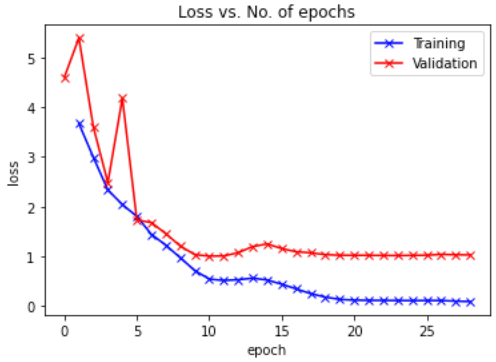


График точности в зависимости от количества эпох. Точность прогнозируется на проверочном наборе, который является постоянным и не рандомизированным. Мы видим, что график вначале имеет крутой наклон, а затем постепенно приближается к постоянному значению, после чего погрешность перестает увеличиваться.

График потерь (потери при обучении и проверки):

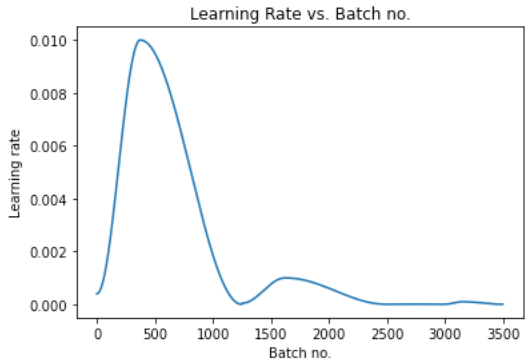
```
plot_losses(history)
```



Это график потерь при обучении, проверке и наборе обучающих данных в зависимости от количества эпох. Мы видим, что вначале потери уменьшаются с высокой скоростью, а затем постепенно скорость снижения становится меньше. После достижения определенной точки обучения потери при проверке могут начать увеличиваться, в то время как потери при обучении продолжают уменьшаться, то есть происходит переобучение.

График скорости обучения:

```
plot_lrs(history)
```



На этом графике мы видим, что скорость обучения начинается с одной точки, достигает своего максимального значения после определенного количества эпох (30% от общего количества предоставленных эпох), а затем снова снижается до определенной точки. Эта закономерность наблюдается при любом планировании скорости обучения, выполненном **методом одного цикла** .