Использование pretrain encoder для задачи классификации

@showpiecep

17 августа 2022 г.

1 Введение

Цель проекта заключается в проверке следующей гипотезы: Предобученный AutoEncoder при использовании его части, отвечающей за кодирование исходных данных, для задачи классификации дает большую точность. При обучении AE сеть учится представлять исходные данные в латентном пространстве таким образом, чтобы следующая часть AE была способна воссоздать полученные данные на основе данных находящихся в латентном пространстве. На основе выше описанного, можно предположить, что латентное пространство хранит в себе ключевые характеристики начальных данных в некотором виде.

2 Эксперименты

Для проведения экспериментов был выбран набор данных с рукописными числами известный, как MNIST. К данным были применены трансформации, которые видны на изображении ниже

```
tranforms_first = torchvision.transforms.Compose([
                                torchvision.transforms.ToTensor(),
                                torchvision.transforms.Normalize(
                                  (0.1307,), (0.3081,))])
```

Рис. 1: Трансформации, примененные к изображениям.

Далее данные для обучения были разбиты на две выборки: валидационную (10 000 изображений) и обучаемую (50 000 изображений). Распределение классов для каждой выборки можно увидеть ниже

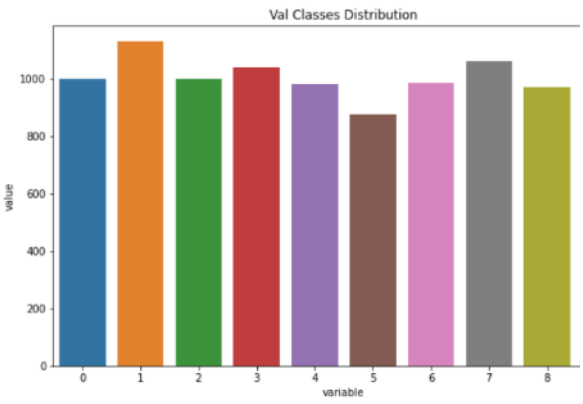

Рис. 2: На валидационной выборке

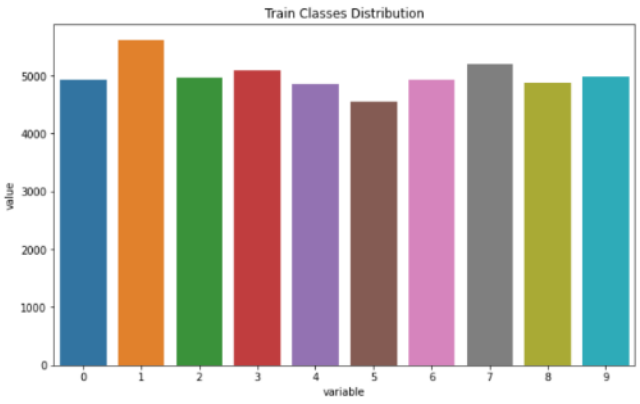

Рис. 3: На тестовой выборке

Рис. 4: Распределение классов на различных выборках

После обучения сети всегда брались веса, при которых было достигнуто наименьшее значение функции потерь на валидационной выборке.

2.1 Эксперимент 1. Полносвязные сети без аугментаций

Была взята полносвязная сеть, имеющая следующую архитектуру:

```
AECls(
  (encoder): Sequential(
    (0): Linear(in_features=784, out_features=256, bias=True)
    (1): ReLU()
    (2): Linear(in_features=256, out_features=128, bias=True)
    (3): ReLU()
  )
  (decoder): Sequential(
    (0): Linear(in_features=128, out_features=256, bias=True)
    (1): ReLU()
    (2): Linear(in_features=256, out_features=784, bias=True)
  )
  (classifier): Sequential(
    (0): Linear(in_features=128, out_features=10, bias=True)
  )
  (classifier2): Sequential(
    (0): Linear(in_features=128, out_features=10, bias=True)
  )
)
```

Рис. 5: Архитектура полносвязной сети

Блок “encoder” отвечает за кодирование исходных данных, “decoder” - за декодирование данных из латентного пространства в исходный вид. Блоки “classifier” и “classifier2” используются для получения предсказаний (далее станет понятно почему их два)

Эксперимент начинается с обучения AE. Далее для классификации обучается только блок “classifier”, который присоединяется к предобученному “encoder” (цифра 1 в таблице). Далее обучаем и “encoder” и “classifier2” (цифра 2 в таблице). Для сравнения рассмотрим полносвязную сеть с архитектурой идентичной “encoder” и “classifier” (цифра 3 в таблице). После эксперимента получаются следующие результаты:

without augmentation	accuracy	F1 Score(macro)
1).AE classification(layer after encoder)	0.9173	0.9160
2).pretrained encoder + classification layer	0.9775	0.9773
3).encoder + classification layer (FCN)	0.9780	0.9777

Таблица 1: Результаты первого эксперимента.

2.2 2 Эксперимент. Полносвязные сети с аугментациями.

Используется сеть с идентичной архитектурой и последовательность проводимых действий над ней. Но при обучении AE и сетей для классификации добавляется аугментация, произвольное вырезание от 10 до 15 процентов изображения. К валидационной выборке данная аугментация не применялась.

with augmentation	accuracy	F1 Score(macro)
1).AE classification(layer after encoder)	0.9057	0.9043
2).pretrained encoder + classification layer	0.9824	0.9823
3).encoder + classification layer (FCN)	0.9785	0.9784

Таблица 2: Результаты второго эксперимента.

2.3 3 Эксперимент. Сверточные сети без аугментаций.

Была взята сверточная сеть, имеющая следующую архитектуру:

```
self.encoder = nn.Sequential(
        nn.Conv2d(1, 8, 3, padding=1),
        nn.ReLU(),
        nn.MaxPool2d(2, 2),

        nn.Conv2d(8, 16, 3, padding=1),
        nn.ReLU(),
        nn.MaxPool2d(2, 2),

        nn.Conv2d(16, 4, 3, padding=1),
        nn.ReLU()

)

self.decoder = nn.Sequential(
        nn.ConvTranspose2d(4, 16, 3, stride=1, padding=1),
        nn.ReLU(),

        nn.ConvTranspose2d(16, 8, 3, stride=2, padding=1),
        nn.ReLU(),

        nn.ConvTranspose2d(8, 1, 4, stride=2, padding=0),


        # nn.ConvTranspose2d(16, 8, 3, stride=1, padding=1)
)

self.classifier1 = nn.Sequential(
    nn.Linear(196, 10),
)

self.classifier2 = nn.Sequential(
    nn.Linear(196, 10),
)
```

Рис. 6: Архитектура сверточной сети

Действия внутри эксперимента остались прежними.

with augmentation	accuracy	F1 Score(macro)
1).AE classification(layer after encoder)	0.9364	0.9355
2).pretrained encoder + classification layer	0.9869	0.9868
3).encoder + classification layer (FCN)	0.9860	0.9858

Таблица 3: Результаты третьего эксперимента.

2.4 4 Эксперимент. Сверточные сети с аугментациями.

Эксперимент аналогичен 2 эксперименту, только здесь использовалась сверточная сеть.

with augmentation	accuracy	F1 Score(macro)
1).AE classification(layer after encoder)	0.9168	0.9160
2).pretrained encoder + classification layer	0.9858	0.9857
3).encoder + classification layer (FCN)	0.9857	0.9855

Таблица 4: Результаты четвертого эксперимента.

3 Вывод

На основе представленных таблице, можно сделать вывод, что при совместном обучении предобученного AE + слоя, отвечающего за классификацию, можно получить прирост в качестве.