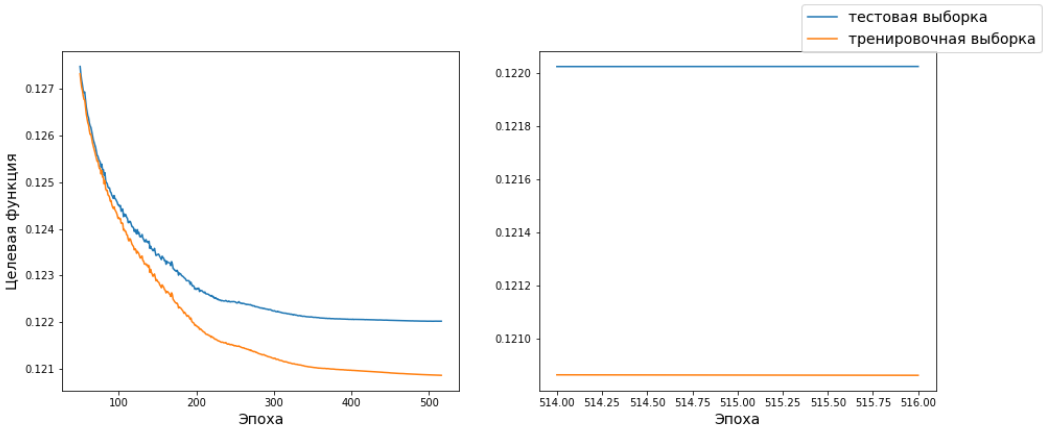

Рисунок 3.6 – Кривая обучения последней модели
Примечание – Источник: собственная разработка.

Еще у нас была идея путем увеличения батча увеличить точность той «ямы» на которой сошелся алгоритм. Мы пересоздали загрузчик тренировочных данных и установили его «batch_size» в размер тренировочных данных. При попытках так доучить модель дейсвительно получается добиться некоторого улучшения значения целевых функции, но они были настолько не значимые, что AUC полученных моделей становился даже хуже, чем у лучшей зафиксированной модели. Это, кстати, еще одна важная идея – у модели с минимальной целевой функции из рассмотренных не обязательно наименьший AUC хотя некоторая связь конечно наблюдается. В связи с этим можно было бы редактировать критерий выбора лучшей модели по наивысшему AUC но такая модернизация уже выходит за рамки этой работы.

В целом, модель не плохая, потому остановимся на ней.

На 3.6 слева можно заметить некоторые флуктуации в начале обучающих кривых. Модель все равно сошлась но настроив обучение так, чтобы убрать эти флуктуации можно было бы увеличить скорость обучения. Но модель уже обучена, потому в этом нет необходимости лишь с оговоркой описанной ниже.

Для того, чтобы не было необходимости каждый раз ждать пока модель обучится при каждой новой рабочей сессии (а то и чаще), pytorch предлагает возможности сохранения модели. Для того может быть использована функция «torch.save», первым аргументом следует передавать значение, которое возвращает метод «state_dict» объекта модели, вторым путь для сохранения. Например, мы делали так:

```
torch.save(lay1_trainer.best_model.state_dict(), 'model 1lay 113
neurons_after_bigfit').
```

Для загрузки используется метод «torch.load», притом загрузка должна проводиться в уже созданный объект модели соответствующей идентификационной формы, через его метод «load_state_dict», мы, например, делали так:

```
model = ResultNet([113, 113])
model.load_state_dict(torch.load('model 1lay 113 neurons_after_bigfit')).
```

Это особенно важный этап при валидации модели, потому как он никак не может проводится без модели, а в каждой новой рабочей сессии проводить оптимизацию очень затратно.

На рисунке М.2 предложена ROC кривая финальной модели. При ее вычислении на тестовых данных. Подробно ROC анализ мы обсуждали в во второй главе. Тут разница лишь в том что *FPR* и *TPR* вычисляются на предсказаниях вероятностей полученных из модели. Площадь под этой кривой является хорошим способом оценить качество модели, такой показатель называется *AUC*. В данном случае *AUC* округляется до 0,800. От начальства была получена установка добиться *AUC* не менее 0,7, это требование успешно выполнено. Хотя, скорее всего, возможно добиться и лучшего результата.

Для выбора той вероятности при которой потенциальный кредитополучатель считается склонным к невозврату кредита используем ту же идею, которая была использована при разбиении числовых переменных на бинарные. Найдем *KS* статистику для эмпирических распределений наблюдений реально в статусе дефолта и без него по предсказанной моделью вероятности. Тут нам сама по себе эта статистика не важна, важна та оценка вероятности при которой она достигается. Именно при этой точке достигается максимальное расстояние между долей правильных предсказаний модели и долей ошибок, что можно, ближе к практике, описать так – доля правильных предсказаний достаточно велика при недопущении заметного возрастания ошибки. Заметим также, что в отличие от вычисления характеристик классифицирующей способности, точку отсечения лучше рассчитывать на полных данных, в данном случае нет проблем с переобучением а нужно выбрать точку максимально советующую генеральной совокупности. Исчерпывающая информация о таком анализе помещена на рисунок 3.7.

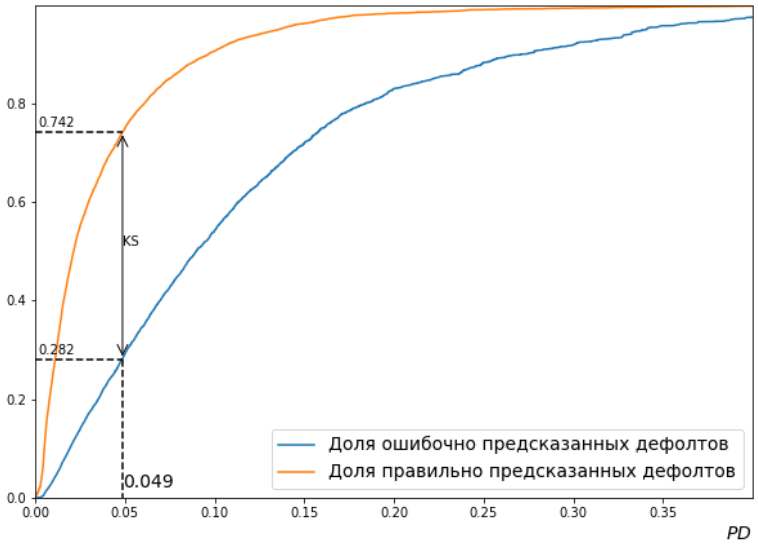

Рисунок 3.7 – Подбор точки отсечения

Примечание – Источник: собственная разработка.