## ОПИСАНИЕ РЕШЕНИЯ

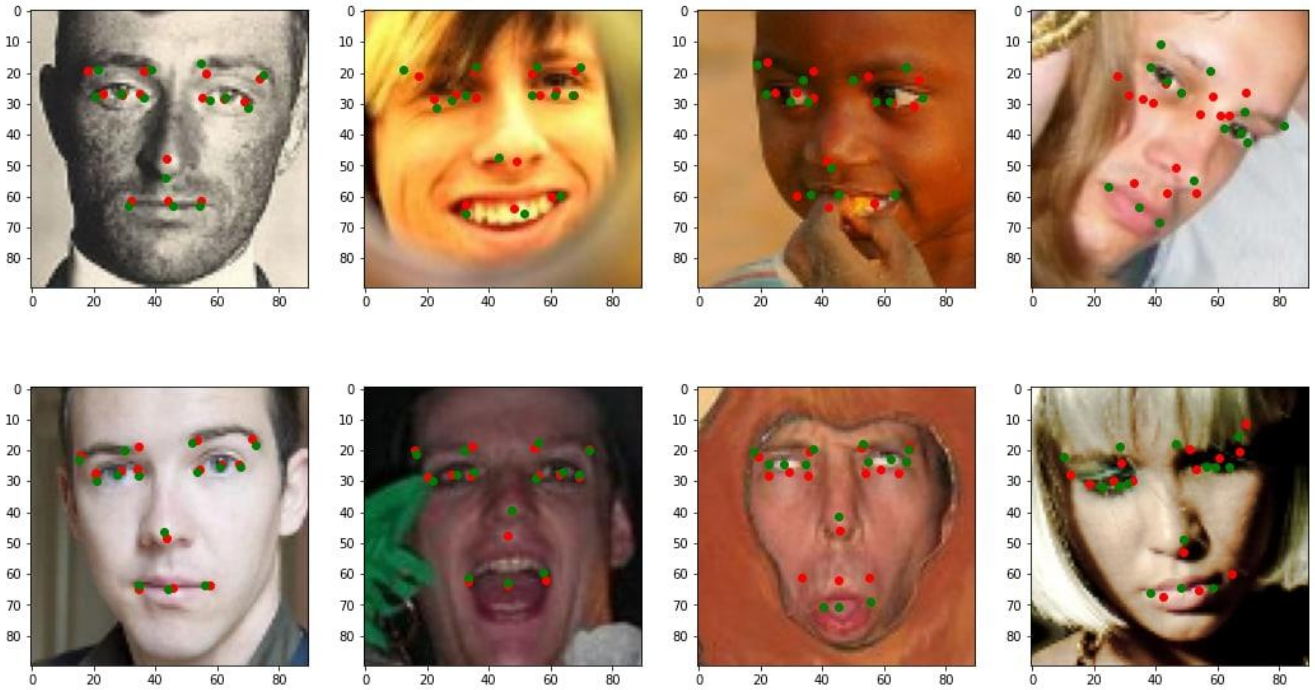
Я начала с базовой архитектуры CHC с тремя слоями свертки:

```
Conv2D(filters=16, (5, 5)) -> LeakyReLU -> MaxPooling2D(2, 2) ->
Conv2D(filters=32, (3, 3)) -> LeakyReLU -> MaxPooling2D(2, 2) ->
Conv2D(filters=64, (3, 3)) -> LeakyReLU -> MaxPooling2D(2, 2) ->
Dense(128) -> Dense(28)
```

На исходном наборе данных (не расширенном) обучала 100 эпох, получила следующее:

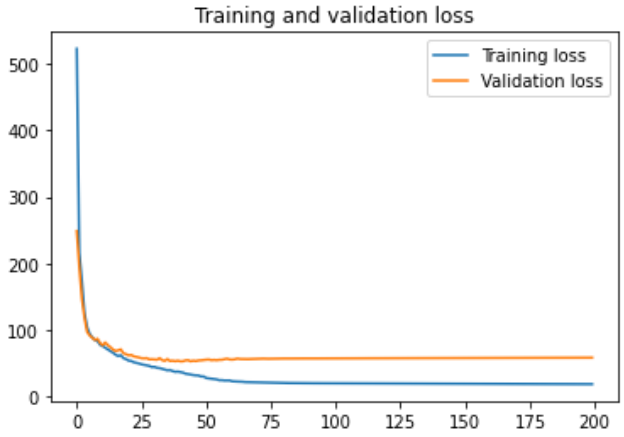

```
trai loss: 10.2094, val loss: 10.7827
```

Предсказания (красным – предсказания модели, зеленым – истинные значения):

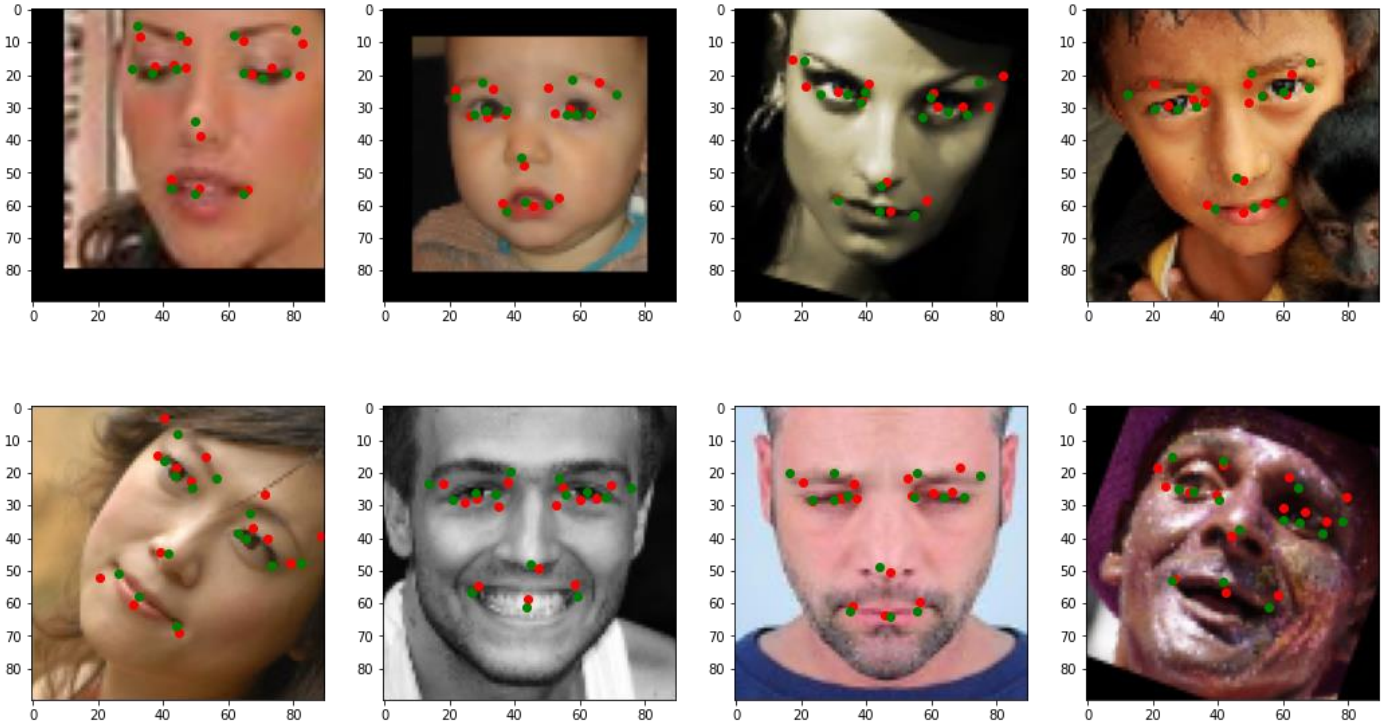


В целом неплохо, но есть проблемы с нетипичными изображениями, например, повернутыми, как в правом верхнем углу.

Не изменяя архитектуру модели, обучим её на расширенном наборе данных (с 5 тысяч изображений увеличиваем набор до 10+ тысяч):



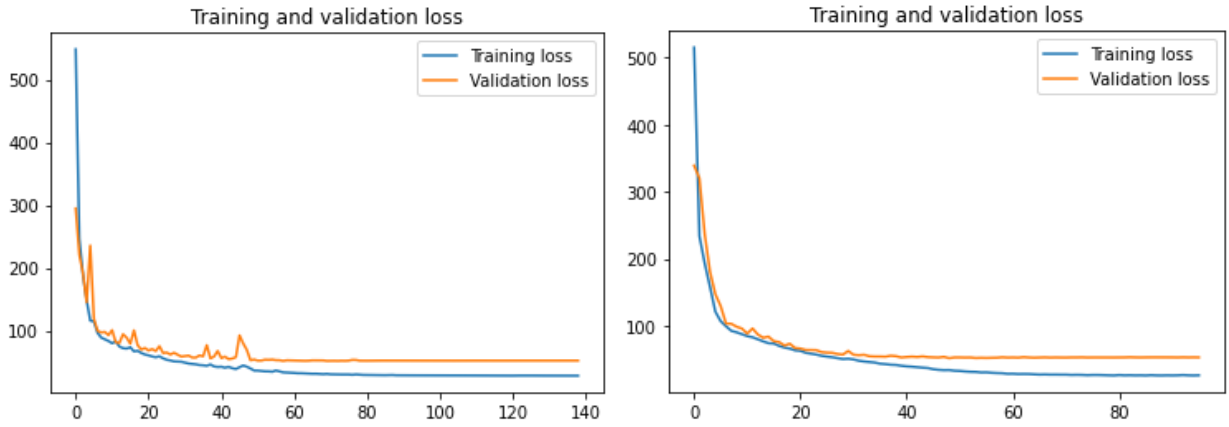

`train loss: 18.5918, val loss: 58.4080`



Модель лучше адаптировалась к повернутым изображениям (1-ое и 4-ое снизу), но по графикам функций потерь видно, что присутствует некоторое переобучение. Это подтверждается значением `test mse = 53.567`, что далеко от тренировочных показателей.

Против переобучения можно применить Dropout или BatchNormalization. Попробуем добавить dropout:

```
Conv2D(filters=16, (5, 5)) -> LeakyReLU -> MaxPooling2D(2, 2) ->
Conv2D(filters=32, (3, 3)) -> LeakyReLU -> MaxPooling2D(2, 2) ->
Conv2D(filters=64, (3, 3)) -> LeakyReLU -> MaxPooling2D(2, 2) ->
Dropout(0.1) -> Dense(128) -> Dense(28)
```



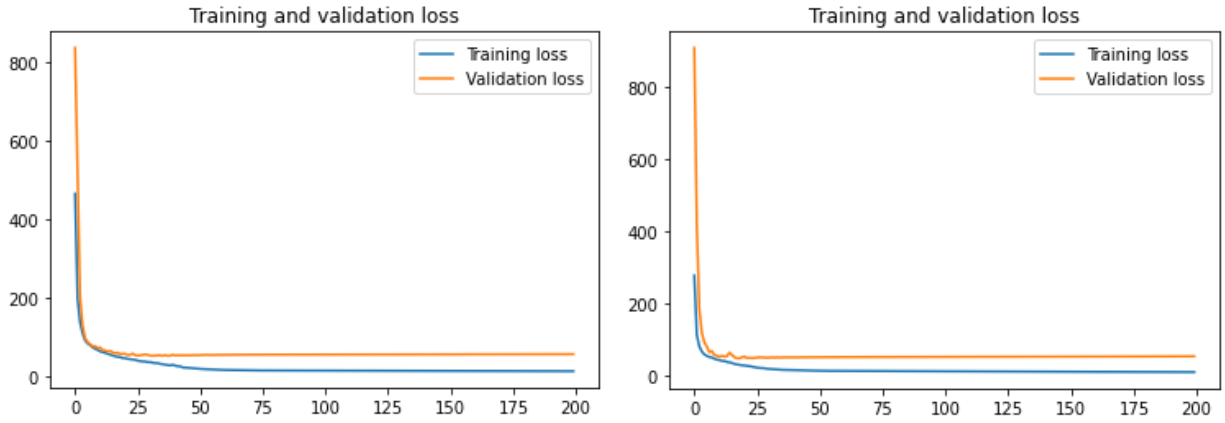

Функции потерь с одним и двумя dropout-слоями, соответственно

Различные комбинации одного и двух слоев отсева на самом деле немного снижали разницу между train и val. Во всех случаях тренировочная метрика останавливалась в районе `25-28`, а валидационная около `50-55`.

Попробуем вместо Dropout добавить BatchNormalization после первой свертки.

```
Conv2D(filters=16, (5, 5)) -> BatchNormalization ->
LeakyReLU -> MaxPooling2D(2, 2) ->
Conv2D(filters=32, (3, 3)) -> LeakyReLU -> MaxPooling2D(2, 2) ->
Conv2D(filters=64, (3, 3)) -> LeakyReLU -> MaxPooling2D(2, 2) ->
Dense(128) -> Dense(28)
```

А потом после каждого слоя свертки.




`train loss: 13.692, val loss: 57.022` `train loss: 10.595, val_loss: 54.2122`

Тренировочная mse понизилась до 10-13, но валидационная так и осталась выше 50.

Дальше было испытано множество расстановок и Dropout, и BatchNormalization, и даже местами L2-регуляризации, но они не помогали значительно снизить переобучение.

Появилась идея, что корнем такой неудовлетворительной валидационной метрики может все-таки быть не сама модель, а данные.

Я еще больше расширила данные, сделав упор на преобразования, которые сильно меняют координаты ключевых точек: повороты на 90 и 180 градусов, а также вертикальное отражение, — и вернулась к исходной архитектуре.

Лишь это без применения какой-либо регуляризации дало validation loss 47.37.

Добавление еще одного слоя свертки дало train loss 11.99 и val loss 42.3:

```
Conv2D(filters=16, (3, 3)) -> LeakyReLU -> MaxPooling2D(2, 2) ->

Conv2D(filters=32, (3, 3)) -> LeakyReLU -> MaxPooling2D(2, 2) ->

Conv2D(filters=64, (3, 3)) -> LeakyReLU -> MaxPooling2D(2, 2) ->

Conv2D(filters=128, (3, 3)) -> LeakyReLU -> MaxPooling2D(2, 2) ->

Dense(128) -> Dense(28)
```

Дальнейшее увеличение набора данных не улучшило результаты, но заметно увеличило расход памяти.

Дальнейшие эксперименты:

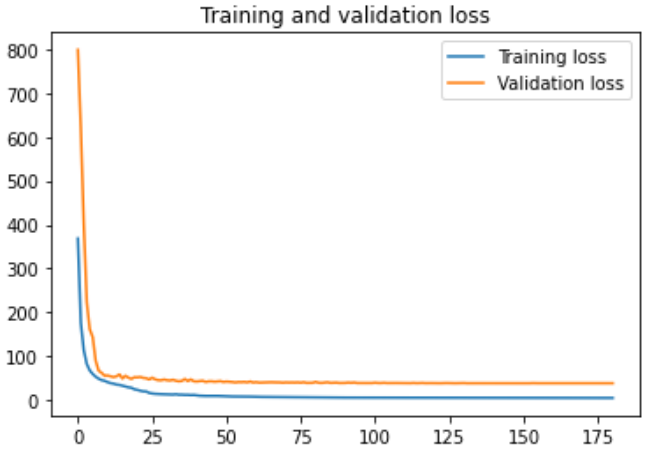
- Дублирование слоев свертки (по два последовательных слоя с 16 фильтрами, затем с 32 и т.д.) — довольно успешно, эта идея осталась в финальной архитектуре.
- Увеличение количества фильтров в первом и последующих слоях (начинать не с 16, а с 64, и тогда далее 128, 256 и т.д.) — это также несколько помогло.
- Добавление полносвязных слоев разного размера
- Небольшие вариации размеров входных изображений (с этим хотелось бы еще поэкспериментировать и посмотреть, как разные размеры влияют не только на финальную получившуюся архитектуру, но и предыдущие прототипы). Конкретно эксперименты с размером и довели модель до финальных метрик. Большинство экспериментов я проводила на изображениях 90x90, и только ближе к концу попробовала их увеличивать/уменьшать, что влияло на метрики.
  Однако влияние это обуславливалось тем, что чем больше размер изображения, тем больше расстояние между ключевыми точками, тем больше буду абсолютная и среднеквадратичные ошибки. При уменьшении изображения, соответственно, эти метрики уменьшались. Применение R2 как метрики показало, что 90 пикселей все-таки оптимальный размер.

К чему я пришла в итоге:

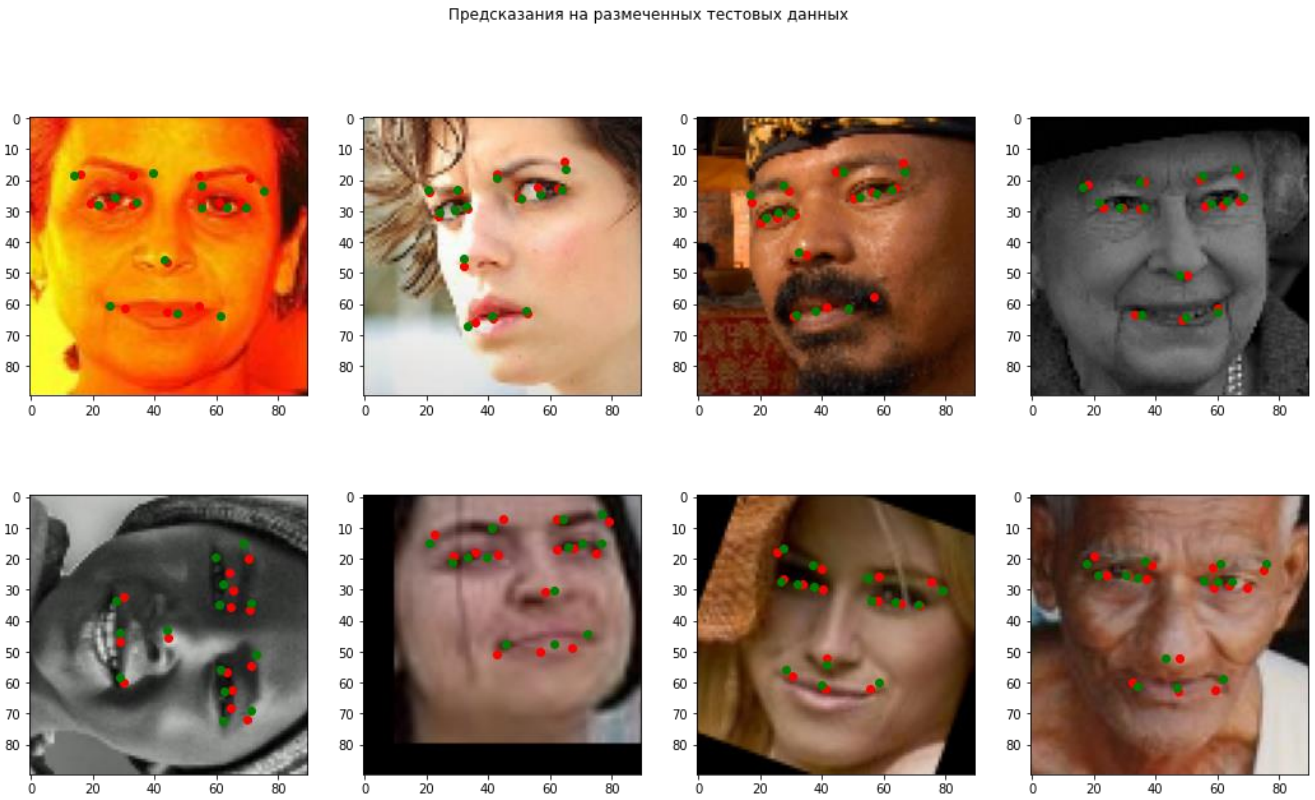
```
InputLayer(input_shape=(90,90,3)) -> Dropout(0.2) ->

Conv2D(filters=64, (3, 3)) -> BatchNormalization() -> LeakyReLU ->

Conv2D(filters=64, (3, 3)) -> BatchNormalization() -> LeakyReLU ->
MaxPooling2D(2, 2) ->

Conv2D(filters=32, (3, 3)) -> BatchNormalization() -> LeakyReLU ->
```

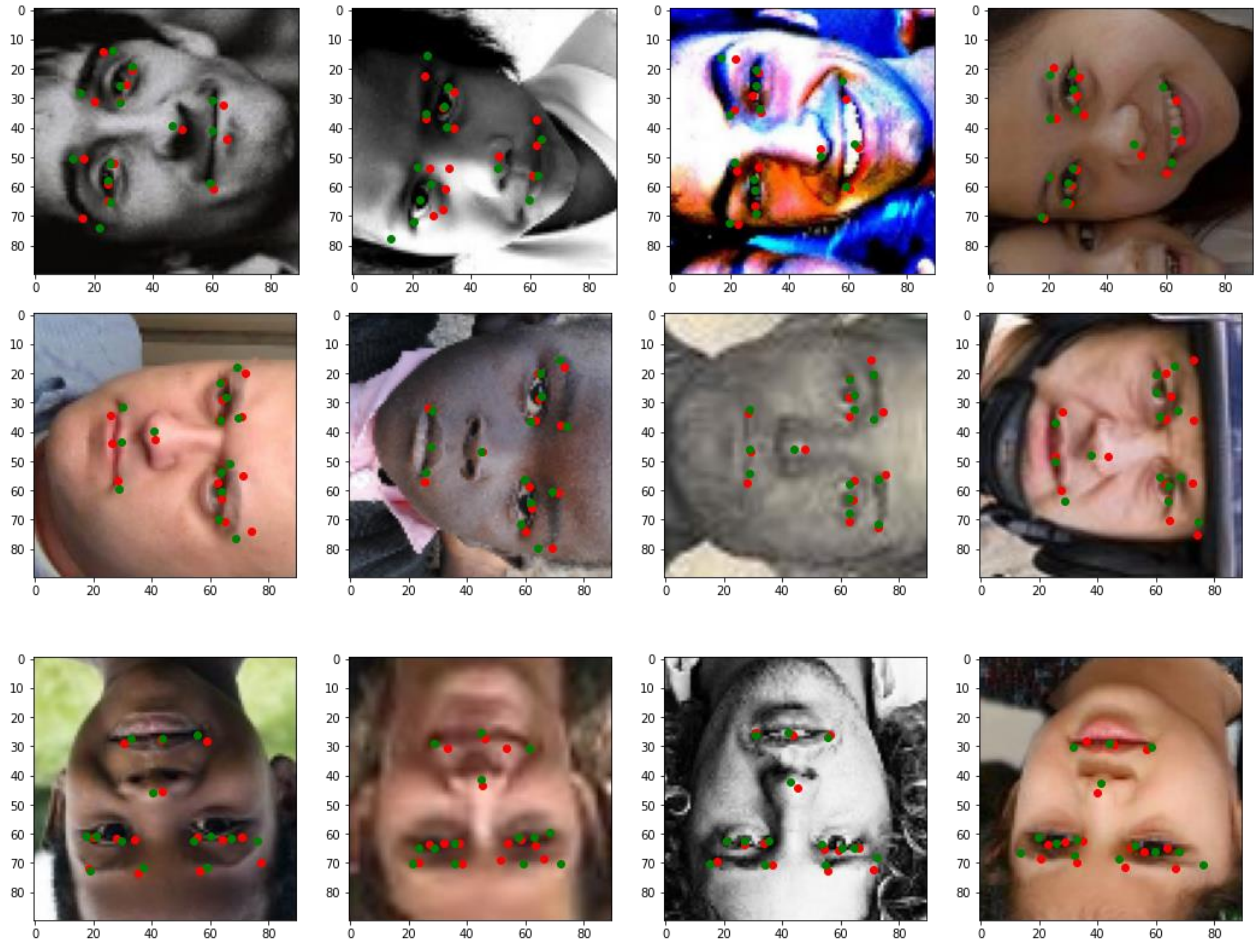
```
Conv2D(filters=32, (3, 3)) -> BatchNormalization() -> LeakyReLU ->
MaxPooling2D(2, 2) ->

Conv2D(filters=64, (3, 3)) -> BatchNormalization() -> LeakyReLU ->

Conv2D(filters=64, (3, 3)) -> BatchNormalization() -> LeakyReLU ->
MaxPooling2D(2, 2) ->

Dropout(0.1) ->

Conv2D(filters=128, (3, 3)) -> BatchNormalization() -> LeakyReLU ->

Conv2D(filters=128, (3, 3)) -> BatchNormalization() -> LeakyReLU ->
MaxPooling2D(2, 2) ->

Dropout(0.1) -> Dense(512) -> Dense(128) -> Dense(28)
```

Итоговый график функций потерь:




Предсказания на размеченных тестовых изображениях:





MSE = 28.9643
R2 = 0.867

Предсказания модели на НЕразмеченных тестовых данных: