1. 解析解

设输入为$n \times m$的特征集合$\mathbf{X}$，输出为$n \times 1$的预测结果$\mathbf{y}$，则有：

$$\mathbf{y} = \mathbf{X}\mathbf{v} + \mathbf{b}$$

显然，我们需要求解$\mathbf{v}$和$\mathbf{b}$。

对上式进行变形

$$
$$

$$

$$
$$

$$
\begin{aligned}
\mathbf{y} &= \mathbf{X}\mathbf{v} + \mathbf{b} \\
&= [\mathbf{1} \ \ \mathbf{X}] \begin{bmatrix} \mathbf{b} \\ \mathbf{v} \end{bmatrix} \\
&= \mathbf{H}\mathbf{w}
\end{aligned}
$$

故等价于求解$\mathbf{w}$。

已知标准输出为$\hat{\mathbf{y}}$，损失函数为：

$$Loss(\mathbf{w}) = ||\hat{\mathbf{y}} - \mathbf{H}\mathbf{w}||^2$$

对损失函数求导

$$\frac{\partial L}{\partial \mathbf{w}} = \mathbf{H}^T(\mathbf{H}\mathbf{w} - \hat{\mathbf{y}})$$

当损失函数的导数为0时取极小值，此时

$$\mathbf{w} = (\mathbf{H}^T\mathbf{H})^{-1}(\mathbf{H}^T\hat{\mathbf{y}})$$

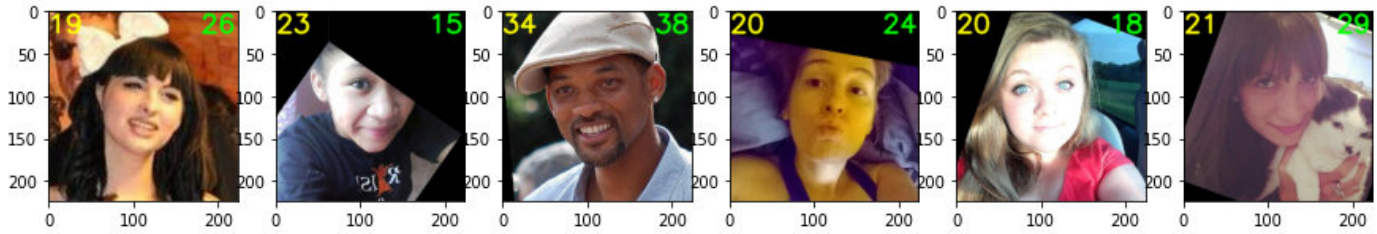
因此，在`closed_form_solution`函数中填充核心代码：

```
weights = np.linalg.inv(H.T.dot(H)).dot(H.T).dot(Y)
```

最终的$\mathbf{v}$和$\mathbf{b}$为：

```
bias = weights[0]
weights = weights[1:]
```

预测结果为：



在验证集上的Loss为：67.84

2. 梯度下降法

设输入为$n \times m$的特征集合$\mathbf{X}$，输出为$n \times 1$的预测结果$\mathbf{y}$，则有：

$$\mathbf{y} = \mathbf{X}\mathbf{v} + \mathbf{b}$$

显然，我们需要求解$\mathbf{v}$和$\mathbf{b}$。

已知标准输出为$\hat{\mathbf{y}}$，损失函数为：

$$L(\mathbf{w}) = ||\mathbf{X}\mathbf{v} + \mathbf{b} - \hat{\mathbf{y}}||^2$$

对损失函数求导，

$$\begin{cases} \dfrac{\partial \mathbf{L}}{\partial \mathbf{v}} = 2\mathbf{X}^T(\mathbf{X}\mathbf{v} + \mathbf{b} - \hat{\mathbf{y}}) \\ \dfrac{\partial \mathbf{L}}{\partial \mathbf{b}} = 2(\mathbf{X}\mathbf{v} + \mathbf{b} - \hat{\mathbf{y}}) \end{cases}$$

迭代

$$\begin{cases} \mathbf{v}^{n+1} = \mathbf{v}^n - \dfrac{\alpha}{n}\dfrac{\partial \mathbf{L}}{\partial \mathbf{v}} \\ \mathbf{b}^{n+1} = \mathbf{b}^n - \dfrac{\alpha}{n}\dfrac{\partial \mathbf{L}}{\partial \mathbf{b}} \end{cases}$$

因此，在`gradient_descent`函数中填充核心代码：

```
# forward pass
pre_age = np.dot(feature, weights) + bias
# calculate loss
loss = np.dot((pre_age-age).T,(pre_age-age))[0][0]
# calculate gradient
gra_w = 2*np.dot(feature.T,(pre_age-age))
gra_b = 2*np.sum(pre_age-age)
```

更新系数时考虑动量，代码如下：

```
# update weights
if not e:
    momentum = lr/N*np.row_stack((gra_b,gra_w))
else:
    momentum = alpha*lr/N*np.row_stack((gra_b,gra_w))+(1-alpha)*momentum
     # You can also consider the gradient descent with momentum
bias -= momentum[0]
weights -= momentum[1:]
```

先控制动量系数为1（动量系数为1时，即为不考虑动量的情况），学习率为10e-3改变运行周期数，loss为随机运行10次取平均：

周期数epoch	动量系数	学习率lr	loss
75	1	10e-3	77.30
100	1	10e-3	74.54
200	1	10e-3	67.34
300	1	10e-3	64.63
400	1	10e-3	62.82

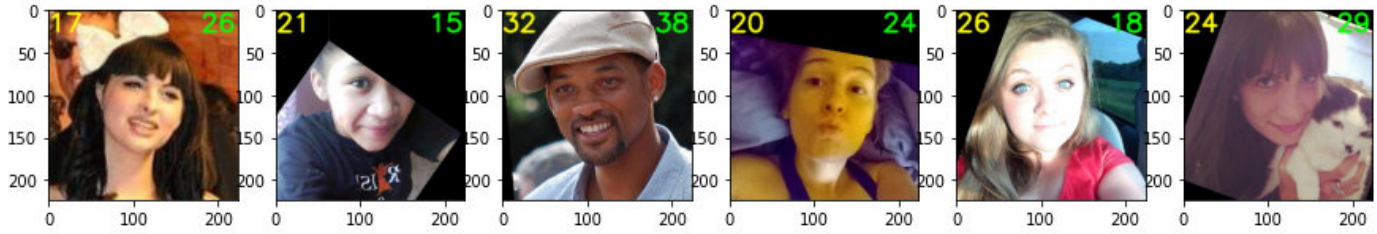
选择周期数为300，控制动量系数为1，调整学习率：

周期数epoch	动量系数	学习率lr	loss
300	1	10e-3	64.63
300	1	10e-4	84.51
300	1	10e-5	107.94
300	1	10e-2	64.78
300	1	10e-1	NaN

选择周期数为300，学习率为10e-3，调整动量系数：

周期数epoch	动量系数	学习率lr	loss
300	1	10e-3	64.63
300	0.6	10e-3	64.74
300	0.4	10e-3	64.43
300	0.35	10e-3	64.32
300	0.25	10e-3	64.21
300	0.2	10e-3	64.58

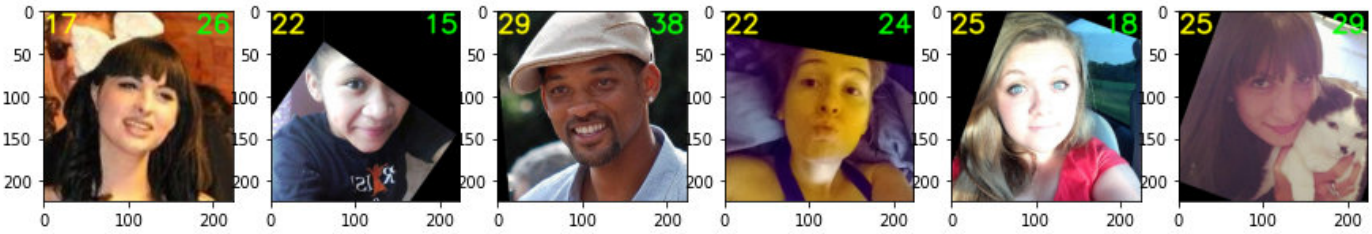
因此最终的超参数选择为周期数为300，学习率为10e-3，动量系数为0.25，平均损失为64.21。



3. 随机梯度下降法

使用随机梯度下降时，在每一次迭代中，我们将训练集随机分成大小为batch_size的多组，依次计算每一组的损失函数并求导，从而进行迭代。

周期数为300，学习率为10e-3，动量系数为0.25，平均损失为57.65，效果如下：


17 26
22 15
29 38
22 24
25 18
25 29