```
126     @staticmethod
127     def backward(ctx, grad_output):
128         classes, indices, gamma, weight = ctx.saved_variables
129         eps = 0.00001
130 
131         #get one_hot representation of indices
132         one_hot = torch.cuda.ByteTensor(classes.size()).zero_()
133         one_hot.scatter_(1, indices.data.unsqueeze(1), 1)
134         one_hot = Variable(one_hot)
135 
136         #calc softmax and logsoftmax
137         probs = F.softmax(classes, 1)
138         probs_mask = probs[one_hot].unsqueeze(1)
139         logs_mask = (probs_mask+eps).log()
140 
141         #get weights into the right shape
142         weights = torch.index_select(weight, 0, indices).unsqueeze(1)
143 
144         #gradient derived by hand, CE is when focal_change == 1
145         focal_factor = torch.pow((1-probs_mask), gamma-1) * (1 - probs_mask - gamma * logs_mask * probs_mask)
146         grad = weights * (probs - one_hot.float()) * focal_factor
```

github.com/nikitazozoulenko

```
 94 class FocalLoss(autograd.Function):
 95     @staticmethod
 96     def forward(ctx, classes, indices, gamma, weight):
 97         """Compute the focal loss with indices
 98         Args:
 99             classes:    (Variable) Shape: [num_bboxes, C+1] Tensor with C+1 classes, (NOT SOFTMAXED)
100             indices:    (Variable) Shape: [num_bboxes]      Tensor with GT indices, 0<= value < C+1
101             gamma:      (Variable) Shape: [1]               The exponent for FL
102             weight:     (Variable) Shape: [C+1]             The "alpha" described in the paper, weight per class
103         Return:
104             focal_loss: (Variable) Shape: [1] Focal loss
105         """
106         ctx.save_for_backward(classes, indices, gamma, weight)
107         eps = 0.00001
108 
109         #get one_hot representation of indices
110         one_hot = torch.cuda.ByteTensor(classes.size()).zero_()
111         one_hot.scatter_(1, indices.unsqueeze(1), 1)
112 
113         #calc softmax and logsoftmax
114         probs = F.softmax(Variable(classes), 1).data
115         probs = probs[one_hot]
116         logs = (probs+eps).log()
117 
118         #get weights into the right shape
119         weights = torch.index_select(weight, 0, indices)
120 
121         #calculate FL and sum
122         focal_loss = -weights * torch.pow((1-probs), gamma) * logs
123         #return torch.mean(focal_loss, 0)
124         return focal_loss
125 
```