UNIVERSITY OF CALIFORNIA, BERKELEY

Homework 6: CS 189

Oscar Ortega

July 16, 2021

1 DATA VISUALIZATION

1. a) Of the subjects seen in the visualization, it appeared to be the case that the key frame annotations seemed pretty accurate for the most part and reflected the movement they were supposed to visualize.

   b) However, for some of the images seen, it appeared as though the images were rotated and as a result, the key-point annotations would appear off. Even when we don't consider the images that had rotated output images, there is greater variance in the key point data, just due to the differences in how people performed the respective poses.

2. In terms of the overall variety in data, one should notice that there was great differences in the lighting, and it also happened to be the case that some people took the images horizontally instead of vertically. The differences in how the subjects performed the poses introduced a great amount of variance in the locations of the key-points. The aspect ratio in terms of the subjects also varied depending on how close one took the videos of themselves.

3. a) It appears to be the case that for the most part, the bounding boxes of the key point annotations would provide good estimations for the people's locations, although in exercises that included arm raises, the images would crop parts or the arms and tops of the heads.

   b) It appears to be the case that for different poses (like the squat), it will be probably harder to detect the movements than with the keyframes. Some of the exercieses when looking at the random frames look like they will be harder to distinguish than the others.

4. After applying the transformation to normalize the Neck Key-Points, there was much less variance in the locations of the respective keys.

2 Modular Fully-Connected Neural Networks

2.1 Activation Functions

- 2.1.1

  1. I used the first principle's method to compute my gradients for W and X:
     If we consider the affine function as a function of W, X and b respectively:

$$f(W+\Delta) = (W+\Delta)X + b = WX + \Delta X + b \tag{2.1}$$

$$\nabla_W = X^T \tag{2.2}$$

$$f(X+\Delta) = W(X+\Delta) + b = WX + W\Delta + b \tag{2.3}$$

$$\nabla_X = W^T \tag{2.4}$$

  The derivative with respect to b_i is below:

$$\frac{\partial f}{\partial b_i} = 1 : \forall i \tag{2.5}$$

$$\nabla_b = \mathbb{1} \in \mathbb{R}^n \tag{2.6}$$

- 2.1.2:

  - Consider a matrix $A \in \mathbb{R}^{m,n}$: Relu(A) $\begin{cases} \text{ReLu}(A[i,j]) = A[i,j] : A[i,j] \geq 0 \\ \text{ReLu}(A[i,j]) = 0 : \text{else} \end{cases}$

$$\text{ReLu}'(A) = \begin{cases} \text{Relu}'(A[i,j]) = 1 : A[i,j] \geq 0 \\ \text{Relu}'(A[i,j]) = 0 : \text{else} \end{cases}$$

  The linear activation function would never have a vanishing gradient, as the derivative is constant over its domain. The ReLU function suffers from this problem much less, and both tanh and sigmoid activation functions experience this problem. In the case of the sigmoid activation function, which is a monotonically increasing function over its domain, its important to note that because the function maps numbers from the set of reals to the set of reals in [0,1], the derivatives of the function will always be quite small. Tanh will also experience similar behavior. If you have inputs that are quite large in absolute value, the odds that a sigmoid or tanh unit will begin to suffer from the vanishing gradient problem is very high.

- 2.1.3:
  There are two cases when computing the loss for the soft-max derivative. You can either compute the loss with respect to an incorrect activation or with respect to a correct activation.

  Let s_{nc} correspond to the activation of an incorrect label:

$$\begin{aligned}
\frac{\partial E}{\partial s_{nc}} &= \frac{\partial - \log(t_c)}{\partial t_c}\frac{\partial t_c}{\partial s_{nc}} && (2.7)\\
&= \frac{-1}{t_c}\frac{\partial t_c}{\partial s_{nc}} && (2.8)\\
&= \frac{-1}{t_c}\frac{\partial e^{sc}(\sum_{k=1}^{C} e^{sk})^{-1}}{\partial s_{nc}} && (2.9)\\
&= \frac{-1}{t_c} - e^{s_c}e^{s_{nc}}(\sum_{k=1}^{C} e^{sc})^{-1}(\sum_{k=1}^{C} e^{sc})^{-1} && (2.10)\\
&= t_c && (2.11)
\end{aligned}$$

Let s_c correspond to the activation of the correct label.

$$\begin{aligned}
\frac{\partial E}{\partial s_c} &= \frac{\partial - \log(t_c)}{\partial t_c}\frac{\partial t_c}{\partial s_c} && (2.12)\\
&= \frac{-1}{t_c}\frac{\partial t_c}{\partial s_c} && (2.13)\\
&= \frac{-1}{t_c}(t_c - (t_c)^2) && (2.14)\\
&= t_c - 1 && (2.15)
\end{aligned}$$

2.2 Two Layer Networks

1. Some things I noticed when exploring with the hyper-parameters is that setting the learning rate past 10^{-4}, would make it so accuracy would not increase beyond 15 percent, and it was also the case, that if I added more than about 50 nodes per layer, there wasn't a dramatic increase in the accuracy. I ended up using two layers with 150 nodes each. My hyper-parameters are as follows:
   - 'lr decay': .99
   - 'num epochs': 100
   - 'batch size': 100
   - 'learning rate': 1e-5
   - 'weight scale': .01

2.3 Multi-layer Networks

2. I found that adding more layers in general did not introduce an enormous gain in accuracy, which tended to all be at around 86-88 percent. My other hyper-parameters were the same and found the best results with 3 layers of 30, 20, and 20 neurons respectively.

3 Convolution and Backprop Revisited

Credit to Ryan Chan on definition of Kronecker Delta function:

1. Because we define the derivative approximation of the image I as $(I[t+1]-[I[t-1])/2$. We want it to be the case that our mask G takes on half of the value of the image at location $[t+1]$ multiplied by negative one, and the value at location $[t-1]$ multiplied by 0.5, our mask would look like the following:

$$G = \begin{bmatrix} -\frac{1}{2} & 0 & \frac{1}{2} \end{bmatrix}$$

   This can be expressed as the following:

$$G = -\frac{1}{2}\delta(t-1) + \frac{1}{2}\delta(t+1)$$

2. Here are my codes:

Algorithm 1 ConvolveXY(Image I, int x, int y): Convolves at location x,y

```
$c = 0$
for $a = 0, \ldots, w-1$ do
    for $b = 0, \ldots, h-1$ do
        for m $\in \{R, G, B\}$ do
            $c = c + I[x+a, y+b] \cdot m[x, y]$
        end for
    end for
end for
return $c$
```

Algorithm 2 Convolve(Image I, int S): Convolves an Image with stride s

```
intitialize $C \in \mathbb{R}^{(1+(W-w)/s),(1+(H-h)/s)}$
for $[x, y] \in \{1, 2, \ldots, (W-w)/s\} \times \{1s, 2s, \ldots, (H-h)/s\}$ do
    $C[x, y] = $ convolveXY$(I, sx, sy)$
end for
return $C$
```

3. Extrapolating the from part a, what we want is a mask m that returns high value when there is a vertical edge, in other-words, when there is a large change or gradient in the vertical direction. We can proceed to stack our original mask in part a to end up with the following:

$$G = \frac{1}{2}\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

   The transpose of this would be able to detect horizontal edges.

4.

$$\frac{\partial L}{\partial G_c[x,y]} = \sum_{i,j} \frac{\partial L}{\partial R[i,j]} \frac{\partial R[i,j]}{\partial G_c[x,y]} \tag{3.1}$$
$$= \sum_{i,j} \frac{\partial L}{\partial R[i,j]} I_c[i+x, j+y] \tag{3.2}$$

$$\frac{\partial L}{\partial I_c[x,y]} = \sum_{i,j} \frac{\partial L}{\partial R[i,j]} \frac{\partial R[i,j]}{\partial I_c[x,y]} \tag{3.3}$$
$$= \sum_{i,j} \frac{\partial L}{\partial R[i,j]} G_c[i+x, j+y] \tag{3.4}$$

5. To perform max pooling, we replace the double summation in the convolution operation with a max and remove any sort of mask. Our expression is as follows:

$$\text{R[i,j]} = \max\{I[(i\cdot s)+a, (j\cdot s)+b] : a, b \in \{0,\ldots,w-1\} \times \{0,\ldots,h-1\}\}$$

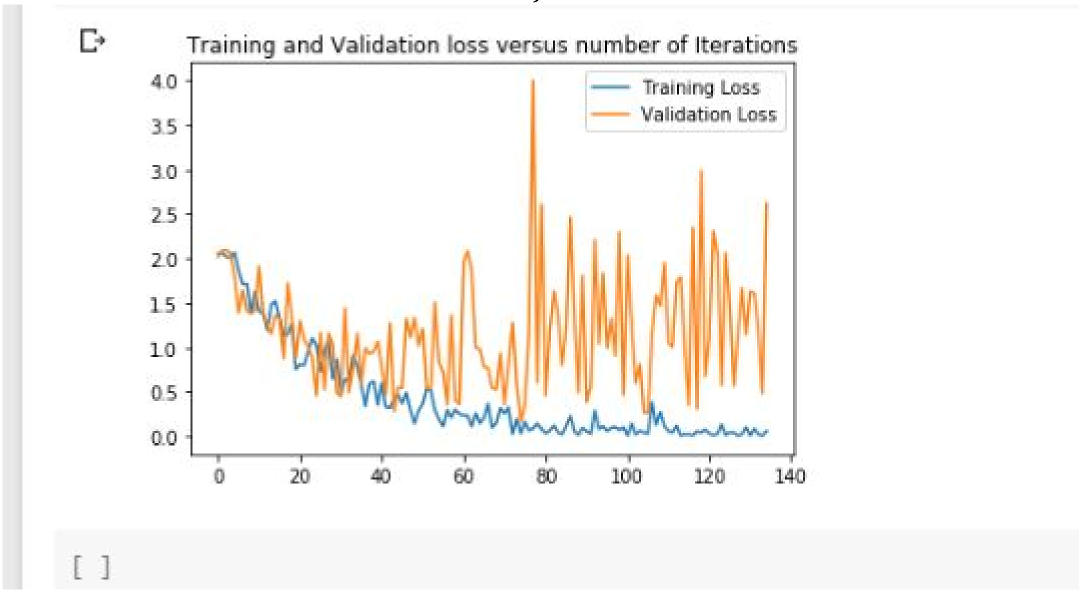
6. We can express the gradient with respect to the pool layer as the following:

$$\frac{\partial L}{\partial \text{pool}[i,j]} = \frac{\partial L}{\partial \text{out}} \frac{\partial \text{out}}{\partial \text{pool}} \tag{3.5}$$
$$= \frac{\partial L}{\partial \text{out}} \mathbb{1}(out[i,j] = \max[i,j] : \forall i,j) \tag{3.6}$$

4 Convolutional Neural Networks (CNNs)

1. On jupyter notebook

2. CNN.JPG



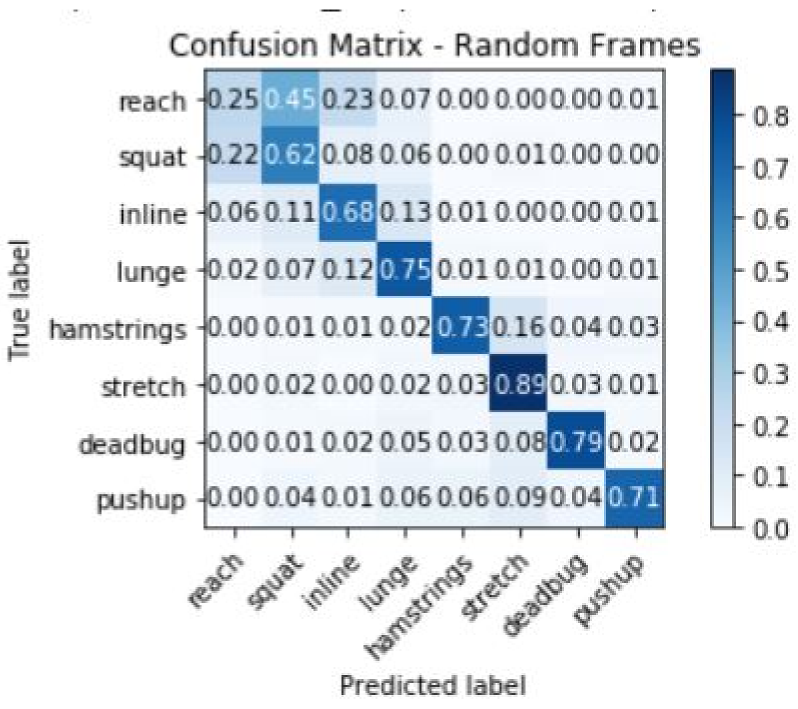
3. If we replaced the convolutional layers of our neural net with fully connected layers, the number of parameters in our model would increase dramatically. My first layer would now have $4*112*56$ parameters, the second layer would have $16*54*26$ parameters, the third layer would have $32*26*12$ parameters,the fourth layer would have $64*12*5$ parameters, and the fifth layer would have $128*5*1$ parameters.

4. on jupyter notebook

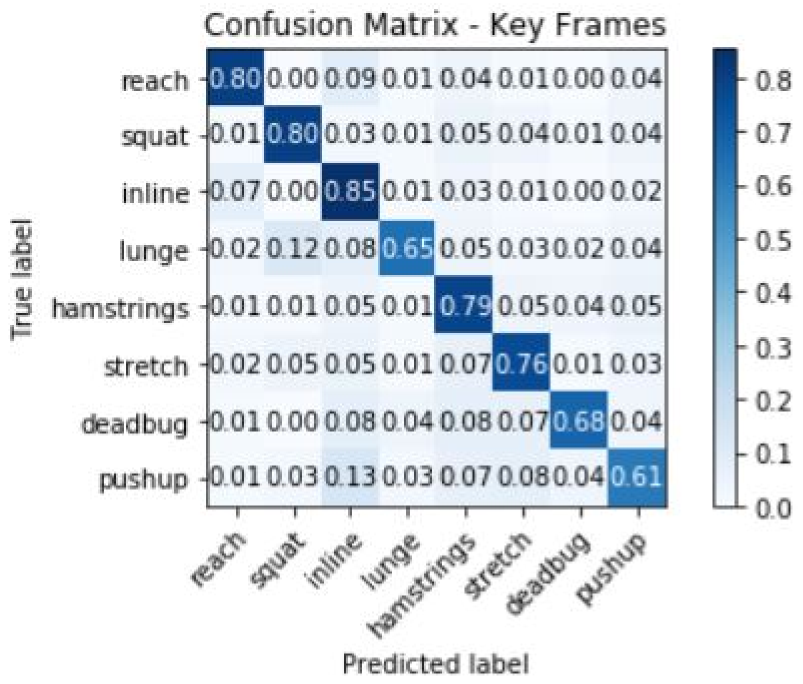
5. Here are the tables:

Random Frames Overall Accuracy – .704629	
images	0.7046
reach	0.2453
squat	0.6209
inline	0.6805
lunge	0.7525
hamstrings	0.7316
stretch	0.8858
deadbug	0.7886
pushup	0.7072

Key Frames Overall Accuracy - .74358	
reach	0.8
squat	0.8
inline	0.8533
lunge	0.64666
hamstrings	0.78666
stretch	0.76
deadbug	0.68
pushup	0.61333



6.



When comparing the confusion matrix for the random feature model, it turns out that reaches and squats were very likely to be misclassified as one another, in fact 45 percent of those labels' total misclassifications were because the model predicted one when it was supposed to be the other. Reach would be mistaken for inline a lot as well. I believe this is because these exercises were both symettric and didn't have a left and right component. It's interesting to note that the same thing did not happen with the keyframes.

7. Kaggle Username : oortega - Total Accuracy test dataset : 0.70832