Tarea 2: ConvNets (parte práctica)

Fecha de entrega: 25 de septiembre, 23:59

Profesor: Pablo Estévez V.
Auxiliar: Ignacio Reyes J.
Semestre: Primavera 2019

Tarea 2: ConvNets (parte práctica)

1. Dropout

Sol:

Observando los gráficos de accuracy y loss para ambos casos, se verifica en el caso de Dropout con probabilidad 1.0 el sobreajuste, porque aunque el gráfico converge en un Loss de entrenamiento más pequeño que el uso de Dropout, el Loss de validación empeora a medida que ocurren más iteraciones.

Así que elijo el valor Dropout = 0.5 ya que converge a un Loss y accuracy de validación menores.

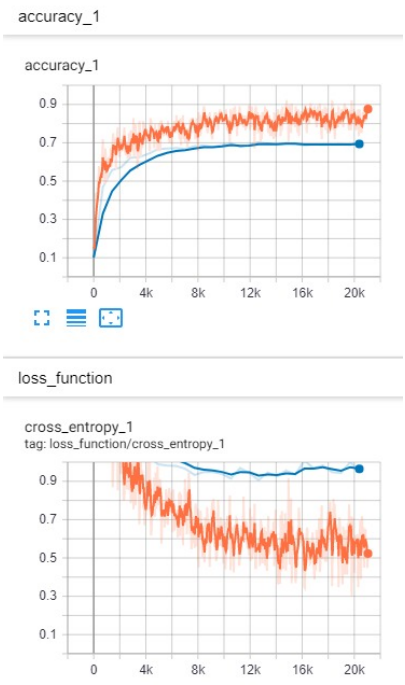

Figura 1: gráficos de accuracy y loss para Dropout=0.5

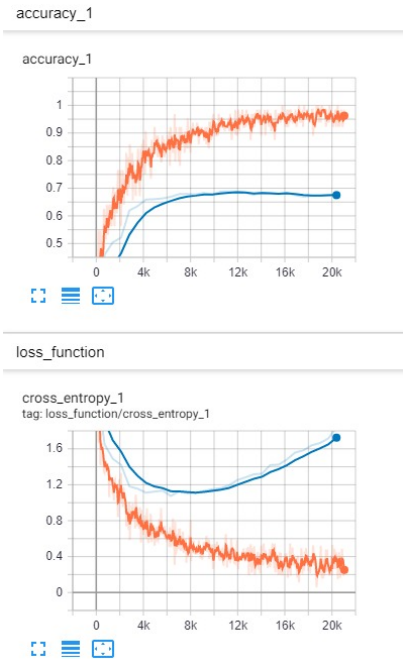

Figura 2: gráficos de accuracy y loss para Dropout=1

2. Impacto del número de capas

Sol:

Mirando el gráfico de accuracy, resulta que converge a un valor cercano al 65 %. Además, el error de la función L'oss es aproximadamente 1.1 para la validación. Peor que n = 2 capas de convolución. Esto se debe a que solo usamos 1 filtro para obtener mapas de características, con la base de datos que tenemos no se puede inferir mucho.

Para el caso de n = 3 capas convolucionales, ya hay una ligera mejora en la accuracy pero no muy significativa. aproximadamente 3 % de diferencia. Esto se debe a que cuantas más capas tenga, más podrá inferir al aprender las diferentes partes de la imagen y los mapas de características de cada capa.

Elijo n = 3 capas de convolución porque tiene una mejor precisión y la diferencia de error en la función Loss, de entropía cruzada, no es significativa en comparación con n = 2 capas de convolución.

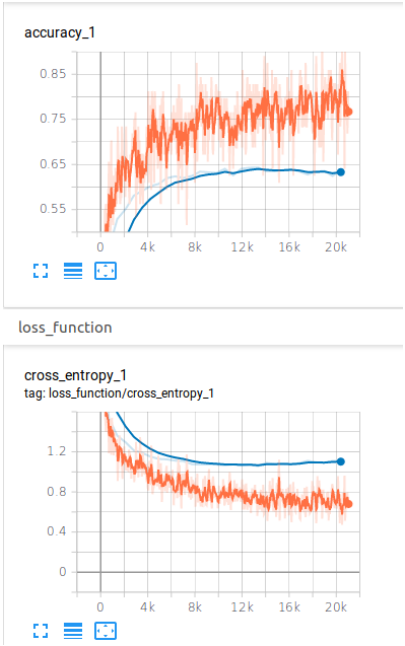

Figura 3: gráficos de accuracy y loss para n=1 capas de convolución

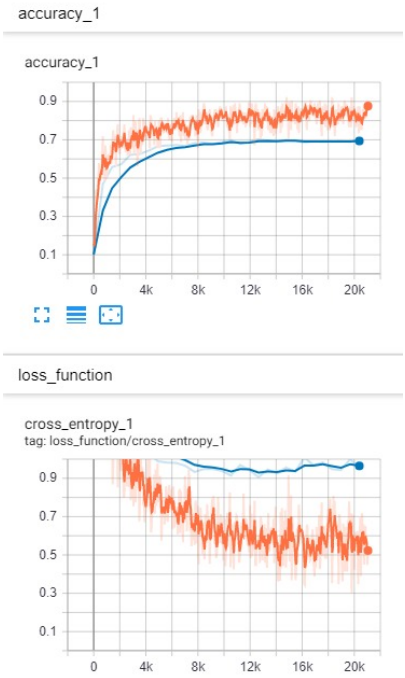

Figura 4: gráficos de accuracy y loss para n=2 capas de convolución

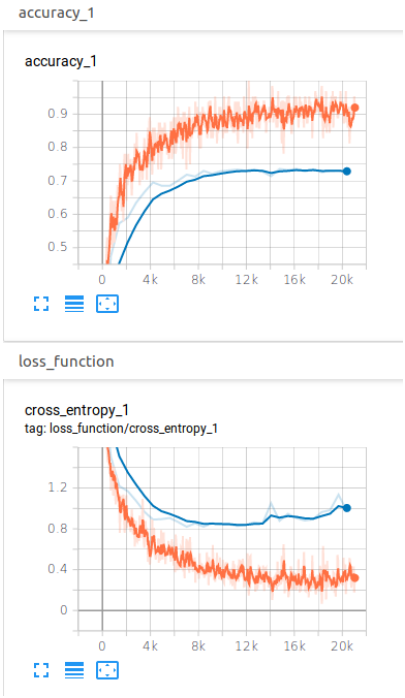

Figura 5: gráficos de accuracy y loss para n=3 capas de convolución

3. Comparación con MLP

Sol:

Comparando los valores de tiempo transcurrido resulta que con solo fully connected es más lento y se necesitan más iteraciones, 35k iteraciones para entrenar.

```
Time elapsed 1.64 minutes
********************************
Testing set accuracy @ epoch 22 (best validation acc): 0.6951
********************************
```

Figura 6: Time Elapsed con capas de convolución

```
Time elapsed 1.76 minutes
********************************
Testing set accuracy @ epoch 18 (best validation acc): 0.5000
********************************
```

Figura 7: Time Elapsed sin capas de convolución

Además, al verificar los valores de precisión también muestra un sobreajuste, porque la red tiene una buena precisión, mas de 90 %, para la base de datos de entrenamiento pero no para la validación, obteniendo un resultado del 50 %.

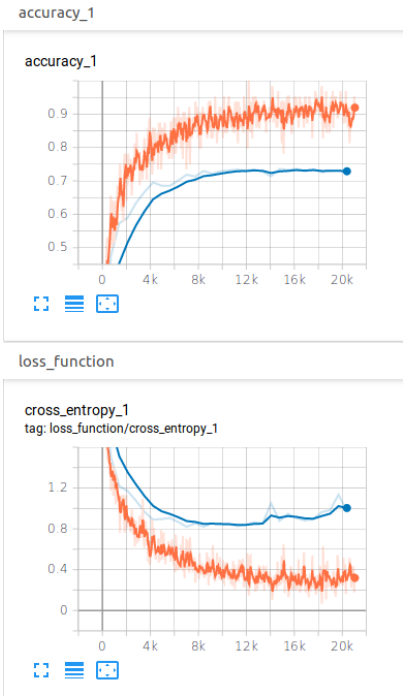

Figura 8: gráficos de accuracy y loss para n=3 capas de convolución

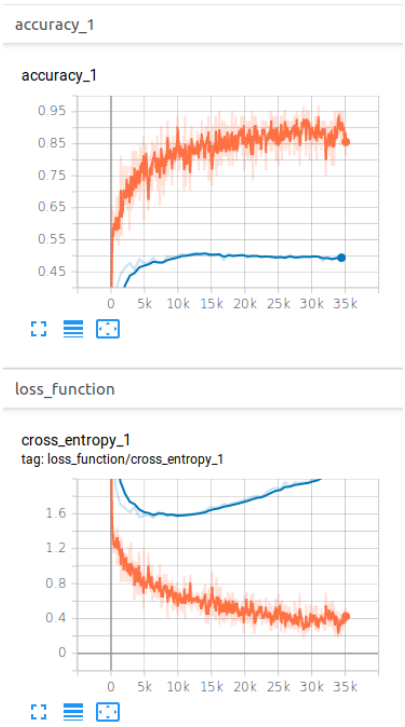

Figura 9: gráficos de accuracy y loss solo con fully connected

4. Data Augmentation

   Sol: Mediante la observación de ambos gráficos, tanto entropía cruzada como accuracy, hay un empeoramiento en los resultados del entrenamiento, pero hubo una mejora ligeramente significativa en los resultados de validación. Esto por el mismo número de iteraciones.

   Esta es una técnica para combatir el sobreajuste. Data Augmentation es una estrategia que permite aumentar significativamente la diversidad de datos disponibles para entrenar los modelos, sin recopilar datos nuevos.

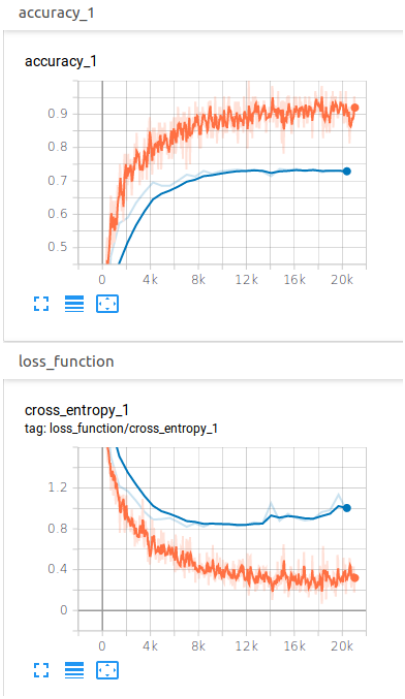

Figura 10: gráficos de accuracy y loss sin Data Augmentation (3 capas convolucionales)

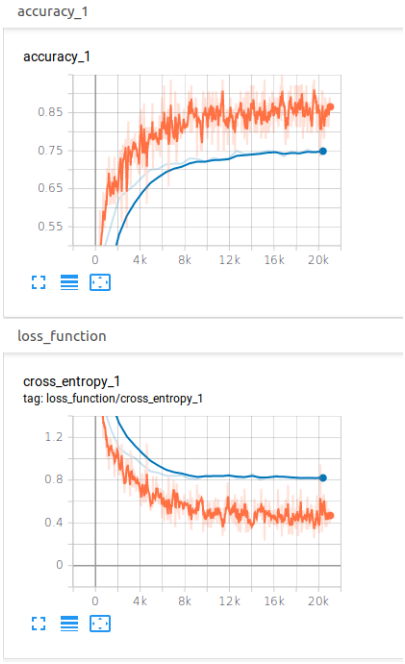

Figura 11: gráficos de accuracy y loss con Data Augmentation (3 capas convolucionales)

Parte de programación: ¡¿No hay fully-connected?!

Sol: Para lograr el objetivo, agregué la seguinte función que define la capa convolucional.

```
def final_conv_layer(input_tensor, kernel_shape, layer_name):

    weights = tf.get_variable("weights", kernel_shape,
                              initializer = tf.contrib.layers.xavier_initializer_conv2d())
    biases = tf.get_variable("biases", [kernel_shape[3]],
                             initializer=tf.constant_initializer(0.05))

    tf.summary.histogram(layer_name + "/weights", weights)
    tf.summary.histogram(layer_name + "/biases", biases)


    conv = tf.nn.conv2d(input_tensor, weights,
                        strides = [1, 1, 1, 1], padding='SAME')

    out_conv = conv + biases

    return out_conv
```

Figura 12: función que define la capa convolucional solicitada

En el modelo agregué la siguiente llamada de la función que define la red convolucional solicitada, con filtro 1x1, 10 feature maps, sin función de activación y sin Pooling. Para obtener un vector de salida de dimensión 10 se hizo un average pooling de 4, para obter los promedios de la feature map.

```
#
previous_n_feature_maps = n_filters_convs[2]
n_filters = 10
layer_name = 'match_layer'
with tf.variable_scope(layer_name):
     conv_out = match_layer(
         pool_out,
         [1, 1, previous_n_feature_maps, n_filters],
         layer_name)

average = tf.nn.avg_pool(conv_out, ksize=[1,4,4,1], strides=[1,4,4,1], padding='SAME')
model_output = tf.squeeze(average)
```

Los gráficos resultantes de la entropia cruzada y accuracy fueron:

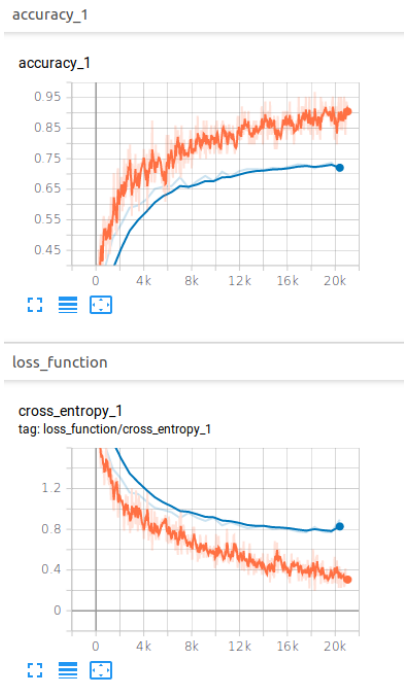

Figura 13: gráficos de accuracy y loss sin fully connected

Los gráficos resultantes fueron similares, se puede decir que resuelve el problema.

Calculo de pesos/parametros y espácio en la memoria (1 peso= 4 bytes)

Image(RGB): [32x32x3] weights: 0
CONV3-16: [32x32x16] weights: ((5*5+1)* 3)*16 = 1248 pesos , memoria=4992 bytes
maxpool2: [16x16x16] weights: 0
CONV3-32: [16x16x32] weights: ((5*5+1)* 16)*32 = 13312 pesos , memoria=53248 bytes
maxpool2: [8x8x32] weights: 0
CONV3-64: [8x8x64] weights: ((5*5+1)* 32)*64 = 53248 pesos , memoria=212992 bytes
maxpool2: [4x4x16] weights: 0

Número total de parámetros o pesos = 67808, memoria= 271232 bytes.

Caso sin fully connected:
CONV1-64: [1x1x64] weights: ((1*1+1)* 64)*10 = 1280 pesos , memoria=5120 bytes

Número total de parámetros o pesos = 69088, memoria= 276352 bytes.

Caso sin fully connected:
FC-50: [1x1x4096] weights: (1024+1)*(50+1) = 52275 pesos , memoria=209100 bytes
FC-10: [1x1x1000] weights: (50+1)*(10+1) = 561 pesos , memoria=2244 bytes

Número total de parámetros o pesos = 120644, memoria= 482576 bytes.

Para medir los tiempos, se hizo el siguiente bloque de código en el entrenamiento:

```
#AQUI REGISTO HASTA 10 TIEMPOS DE ITERACION DE CADA EPOCH
if(i<10):
  t_end = time.time()
  t_aux.append((t_end - t_prev)/703)
  t_prev=t_end
  i+=1
```

Resultados para sin fully connected:

```
print('media(segundos):',np.mean(t_aux))#/(60.0*60.0))
print('std(segundos):',np.std(t_aux))#/(60.0*60.0))

k=1
for t in t_aux:
  seg=t#/(60.0*60.0)
  print('epoch: ', k, '  -----   tiempo de iteracion= ', seg, 'segundos')
  k+=1
```

```
media(segundos): 0.004407301701997456
std(segundos): 0.00023149206754585153
epoch:  1   -----   tiempo de iteracion=  0.005055006333808302 segundos
epoch:  2   -----   tiempo de iteracion=  0.004557570556488688 segundos
epoch:  3   -----   tiempo de iteracion=  0.00439538928556917 segundos
epoch:  4   -----   tiempo de iteracion=  0.00427023581047655 segundos
epoch:  5   -----   tiempo de iteracion=  0.004275571570796614 segundos
epoch:  6   -----   tiempo de iteracion=  0.004277747182723978 segundos
epoch:  7   -----   tiempo de iteracion=  0.004264274667710024 segundos
epoch:  8   -----   tiempo de iteracion=  0.004316221768961183 segundos
epoch:  9   -----   tiempo de iteracion=  0.004321327250169996 segundos
epoch:  10   -----   tiempo de iteracion=  0.004339672593270054 segundos
```

Resultados para con fully connected:

```
[19] print('media(segundos):',np.mean(t_aux))#/(60.0*60.0))
     print('std(segundos):',np.std(t_aux))#/(60.0*60.0))

     k=1
     for t in t_aux:
       seg=t#/(60.0*60.0)
       print('epoch: ', k, '  -----   tiempo de iteracion= ', seg, 'segundos')
       k+=1
```

```
media(segundos): 0.005317982467445167
std(segundos): 0.0003583721616738953
epoch:  1   -----   tiempo de iteracion=  0.006347315409783788 segundos
epoch:  2   -----   tiempo de iteracion=  0.005437616603303261 segundos
epoch:  3   -----   tiempo de iteracion=  0.00530069515342224 segundos
epoch:  4   -----   tiempo de iteracion=  0.005212512837026061 segundos
epoch:  5   -----   tiempo de iteracion=  0.00521069977056251 segundos
epoch:  6   -----   tiempo de iteracion=  0.005240373557187076 segundos
epoch:  7   -----   tiempo de iteracion=  0.005132741643219211 segundos
epoch:  8   -----   tiempo de iteracion=  0.005093871275357807 segundos
epoch:  9   -----   tiempo de iteracion=  0.005144862328281104 segundos
epoch:  10   -----   tiempo de iteracion=  0.005059136096308614 segundos
```

Como se puede ver en los resultados y el número de parámetros, el problema también se puede resolver sin el uso de capas fully connected, con menos tiempo y menos parámetros, ocupando menos memoria. Esto se logra porque el vector tendrá valores promediados de los valores del mapa de características de redes convolucionales, que han aprendido sobre ciertas porciones de la imagen.