Keiland Pullen
CSC 578 sect. 901– Spring 2022
Assignment: HW4

1. Add comments to a code snippet:

1: delta = self.cost_derivative(activations[-1], y) * sigmoid_prime(zs[-1])

Delta is assigned the value from the product of the cost_derivative function times the sigmoid_prime function. The cost_der ivative function takes as its inputs the output_activations and y. The function uses the list of activations stored in the variable "activations" and uses [-1] to select the last item in the list. The cost_derivative would be the "partial c" over the "partial a".   The sigmoid_prime function computes the derivative of the sigmoid function. The sigmoid_prime function would be the "partial a" over "partial z", where the list" zs" contains z vectors and [-1] is the last item in the list.   At the end, the value of delta is equal to (partial C / partial a) · (partial a / partial z).

3: nabla_b[-1] = delta

Nabla_b takes the value of delta for its last bias value. The code defines nabla_b as:
[np.zeros(b.shape) for b in self.biases]
Nabla_b is initialized as a list of biases with 0 for b's shape.

4: nabla_w[-1] = np.dot(delta, activations[-2].transpose())

Similar to nabla_b, the code defines nabla_w as:
[np.zeros(w.shape) for w in self.weights]
Nabla_w is a list of weights with 0 for w's shape. Dot product multiplication is used because delta and activations are vectors. The activations vector is transposed for vector multiplication.

Keiland Pullen
CSC 578 sect. 901– Spring 2022
Assignment: HW4

2. Exercise 2 on the cross-entropy cost function in NNDL3 :

For C = - [y lna + (1 – y) ln (l – y)]

Where y = 0.25, 0.5, 0.8, and for each example y, where y = a, each value (highlighted in yellow) is the minimum.

y = 0.25	a = 0.1	C = - [0.25 ln(0.1) + (1 – 0.25) ln (1 - 0.1)] =	0.65466
y = 0.25	a = 0.2	C = - [0.25 ln(0.2) + (1 – 0.25) ln (1 – 0.2)] =	0.56971
y = 0.25	a = 0.25	C = - [0.25 ln(0.25) + (1 – 0.25) ln (1 – 0.25)] =	0.56233
y = 0.25	a = 0.3	C = - [0.25 ln(0.3) + (1 – 0.25) ln (1 – 0.3)] =	0.56849
y = 0.25	a = 0.4	C = - [0.25 ln(0.4) + (1 – 0.25) ln (1 – 0.4)] =	0.61219
y = 0.25	a = 0.5	C = - [0.25 ln(0.5) + (1 – 0.25) ln (1 – 0.5)] =	0.69314
y = 0.25	a = 0.55	C = - [0.25 ln(0.55) + (1 – 0.25) ln (1 – 0.55)] =	0.74834
y = 0.25	a = 0.7	C = - [0.25 ln(0.7) + (1 – 0.25) ln (1 – 0.7)] =	0.99214
y = 0.25	a = 0.75	C = - [0.25 ln(0.75) + (1 – 0.25) ln (1 – 0.75)] =	1.11164
y = 0.25	a = 0.9	C = - [0.25 ln(0.9) + (1 – 0.25) ln (1 – 0.9)] =	1.75327
y = 0.50	a = 0.1	C = - [0.5 ln(0.1) + (1 – 0.5) ln (1 – 0.1)] =	1.20397
y = 0.50	a = 0.2	C = - [0.5 ln(0.2) + (1 – 0.5) ln (1 – 0.2)] =	0.91629
y = 0.50	a = 0.3	C = - [0.5 ln(0.3) + (1 – 0.5) ln (1 – 0.3)] =	0.78032
y = 0.50	a = 0.4	C = - [0.5 ln(0.4) + (1 – 0.5) ln (1 – 0.4)] =	0.71355
y = 0.50	a = 0.5	C = - [0.5 ln(0.5) + (1 – 0.5) ln (1 – 0.5)] =	0.69314
y = 0.50	a = 0.6	C = - [0.5 ln(0.6) + (1 – 0.5) ln (1 – 0.6)] =	0.71355
y = 0.50	a = 0.7	C = - [0.5 ln(0.7) + (1 – 0.5) ln (1 – 0.7)] =	0.78032
y = 0.50	a = 0.8	C = - [0.5 ln(0.8) + (1 – 0.5) ln (1 – 0.8)] =	0.91629
y = 0.50	a = 0.9	C = - [0.5 ln(0.9) + (1 – 0.5) ln (1 – 0.9)] =	1.20397
y = 0.80	a = 0.2	C = - [0.8 ln(0.2) + (1 – 0.8) ln (1 – 0.2)] =	1.33217
y = 0.80	a = 0.4	C = - [0.8 ln(0.4) + (1 – 0.8) ln (1 – 0.4)] =	0.83519
y = 0.80	a = 0.6	C = - [0.8 ln(0.6) + (1 – 0.8) ln (1 – 0.6)] =	0.59191
y = 0.80	a = 0.8	C = - [0.8 ln(0.8) + (1 – 0.8) ln (1 – 0.8)] =	0.50040
y = 0.80	a = 0.85	C = - [0.8 ln(0.85) + (1 – 0.8) ln (1 – 0.85)] =	0.50943
y = 0.80	a = 0.9	C = - [0.8 ln(0.9) + (1 – 0.8) ln (1 – 0.9)] =	0.54480
y = 0.80	a = 0.95	C = - [0.8 ln(0.95) + (1 – 0.8) ln (1 – 0.95)] =	0.64018

Keiland Pullen
CSC 578 sect. 901– Spring 2022
Assignment: HW4

3. The Learning Slowdown Problem :

I’ve had to read this one a few times to understand it, but I’m still a bit unsure. According to the NNDL text, it isn’t possible to eliminate the “x sub j” term because the cost must be defined as a function of the output activations. Which means that ” x sub j’ will always be produced regardless of which activation function is used. This is seen when taking the partial derivative of the cost function with respect to the weight, the derivative of the weight input “x sub j” will always be produced.

4. TensorFlow / Kera Tutorial analysis :

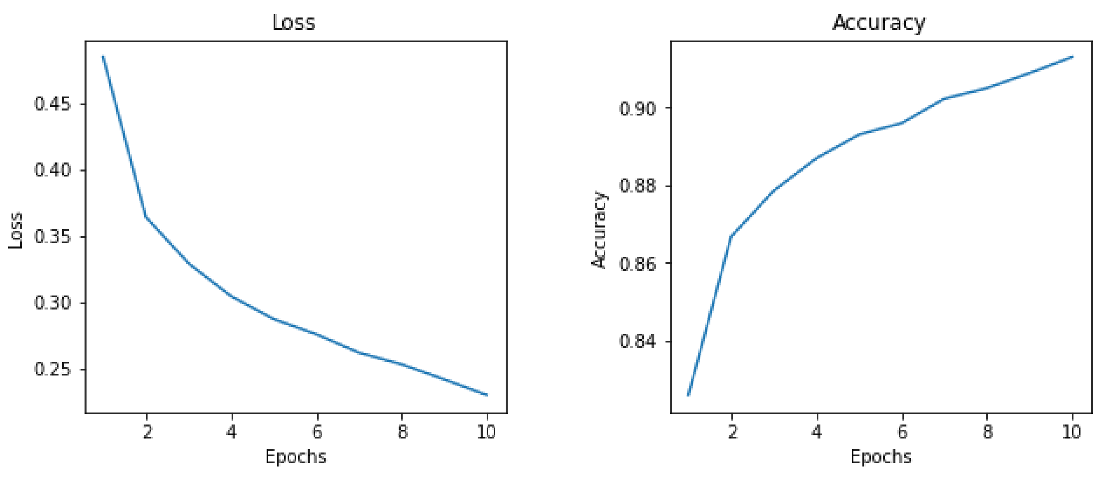
The initial Jupyter notebook was executed to provide a baseline for the model. After the initial run, the initial images were changed to a coat and a sandal. For this run, the Test accuracy was 87% on the test dataset and 91% on the training dataset. For the coat, the accuracy was 92% and for the sandal the accuracy was 100%. I’d like to use the accuracy scores to determine the model’s performance.

For the first experiment, I added an additional hidden layer:

```
tf.keras.layers.Dense(128, activation='relu'),
```

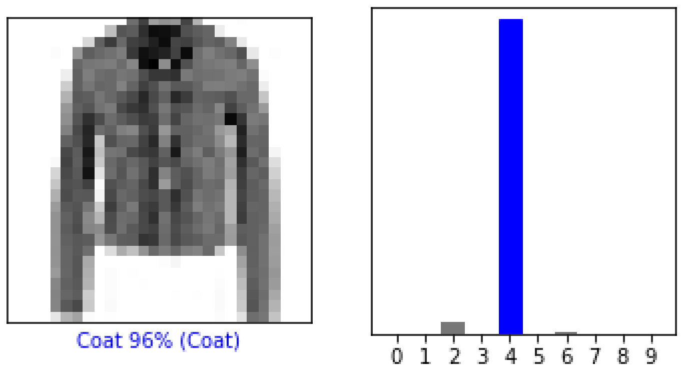
The idea is that the addition of hidden layer will increase the model performance. The test dataset accuracy for the additional hidden layer remained at 87%, while the accuracy for the training dataset was at 91%. However, the prediction for the coat increased to 96% and remained at 100% for the sandal. A third hidden layer was added and the results did not differ. So that hidden layer was removed.

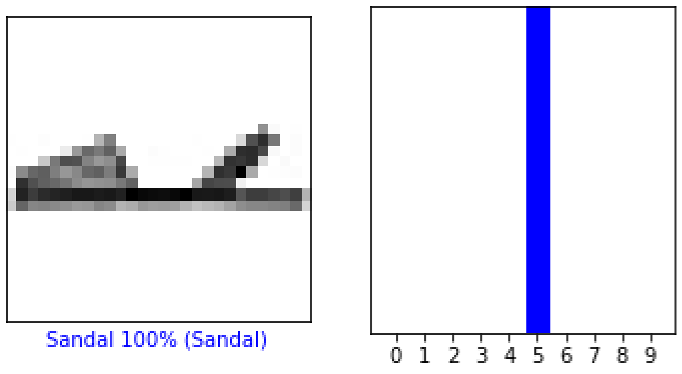
The following are plots for the loss and accuracy of the first experiment:



Keiland Pullen
CSC 578 sect. 901– Spring 2022
Assignment: HW4

The following are the verification predictions of the first experiment:





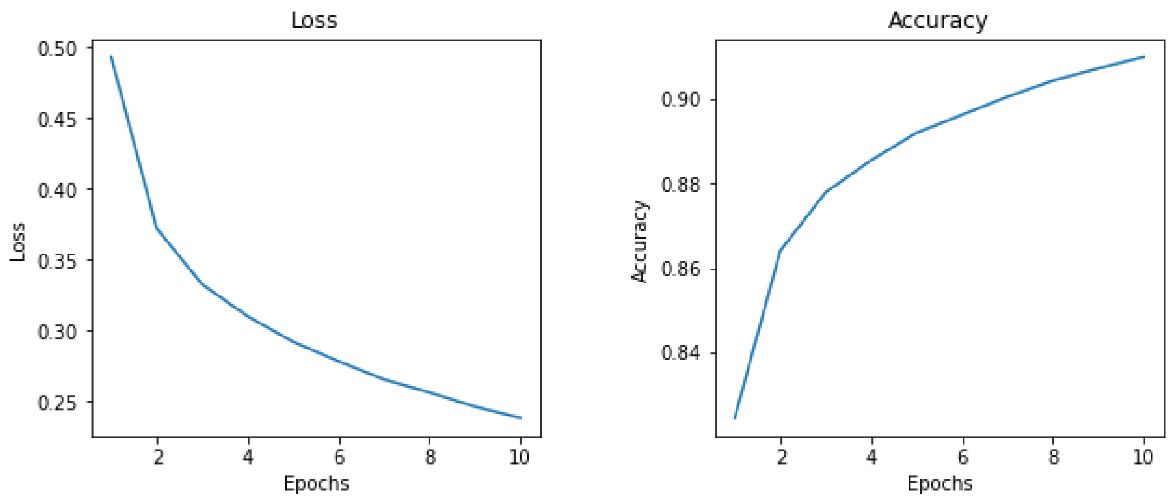
For the second experiment, the number of nodes in both hidden layers were adjusted from the initial value of 128 to 96:

```
tf.keras.layers.Dense(96, activation='relu'),
tf.keras.layers.Dense(96, activation='relu'),
```

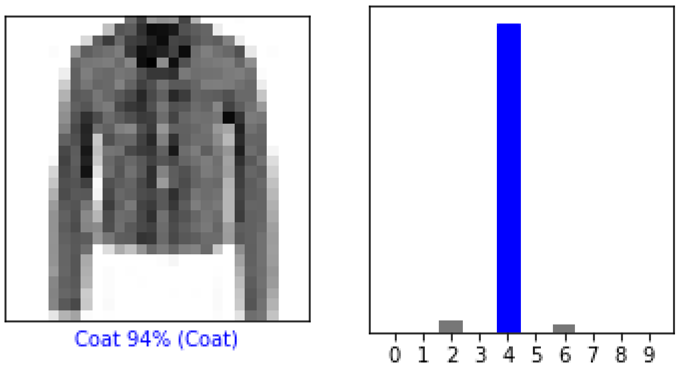
The results of this test were that the test accuracy remained steady at 87% and the training accuracy declined a bit to 90.6%. To verify using the coat, the accuracy was 97% and the accuracy for the sandal remained at 100%.

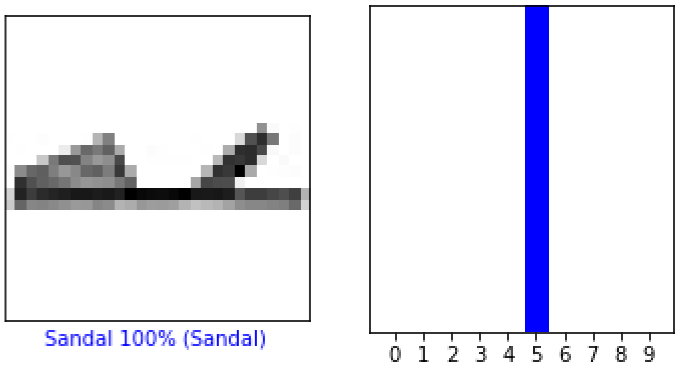
Loss and accuracy plots for experiment 2:

Keiland Pullen
CSC 578 sect. 901– Spring 2022
Assignment: HW4



Verification plots for experiment 2:

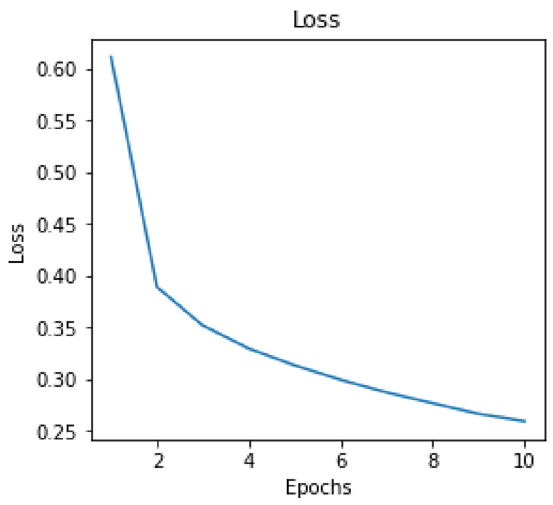


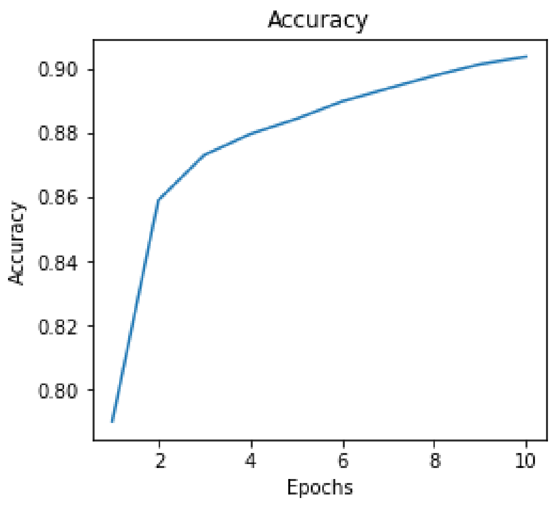


For the third experiment, the activation function was changed from relu to sigmoid. Again, the idea is to increase the accuracy and verification rates. This resulted in the values of 87.9% for the test accuracy and 90.9% for the training accuracy. The verification accuracy of the coat was 97% and that of the sandal remained steady at 100%.

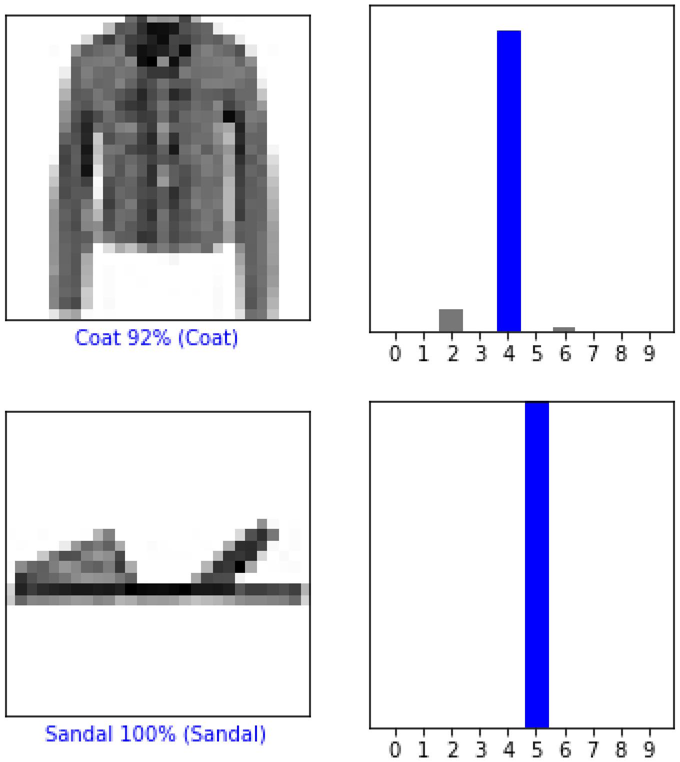
Keiland Pullen
CSC 578 sect. 901– Spring 2022
Assignment: HW4

Loss and accuracy plots for experiment 3:





Verification plot for experiment 3:

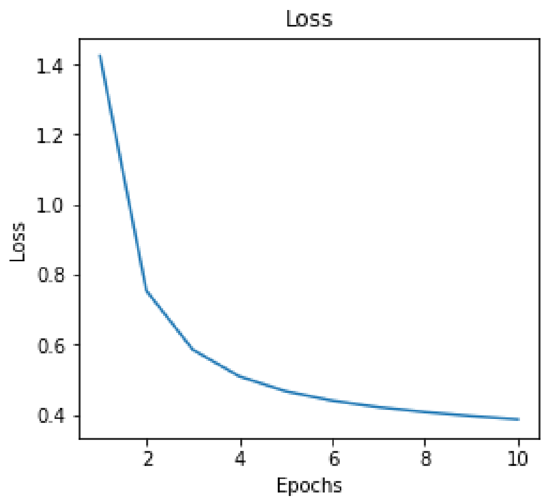


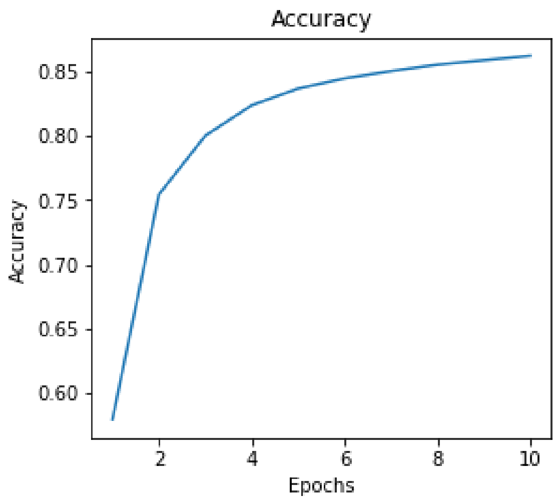
For the fourth experiment, the learning rate was adjusted. The value was changed to .01.

```
opt = tf.keras.optimizers.Adam(learning_rate=0.01)
```

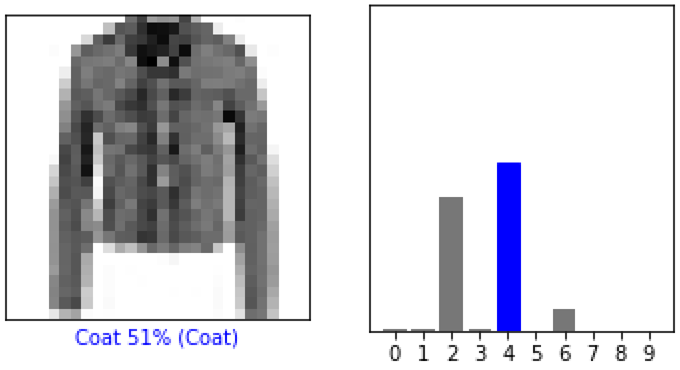
This changed caused the accuracy values to fall for both test and train datasets and the verification data. The value was then changed to 0.1 and the accuracy values dropped with values of 85 % and 86% for the test and training accuracies along with a value of 51% for the coat verification. Strangely, the value for the sandal dropped only to 99%. The values of .001 and .0001 were also tested for the learning rate, and neither produced good results. So the Learning Rate parameter was removed.

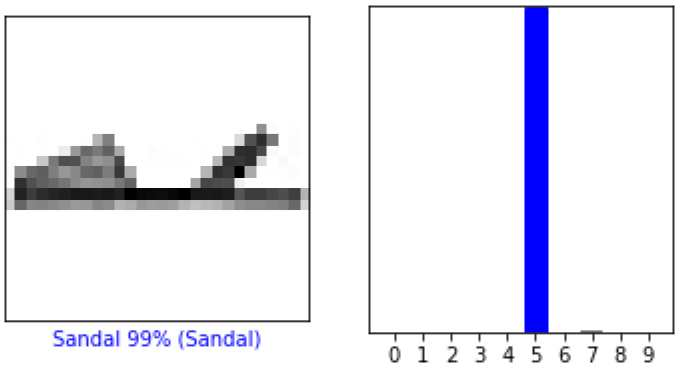
Loss and accuracy plots for experiment 4:





Verification plots for experiment 4:





Keiland Pullen
CSC 578 sect. 901– Spring 2022
Assignment: HW4

Based on the four experiments, it seems that the number of hidden layers has an impact on the accuracy in addition to the activation function. The number of nodes in the hidden layers did not seem to have much of an impact. While the learning rate had a negative impact with this combination of tests, it is possible that it would have a positive impact if combined with other tests. In hindsight, more tests could have been performed to ensure the best parameters. I do wonder if a grid search should have been performed beforehand and then fine tuned the parameters for comparison and contrast.