Maddie Mihevc & Hannah VanderHoeven

HW3: Hunt Shifting Numbers

Introduction

For this assignment we explored what occurred when we applied a convolutional neural network to the MNIST dataset. Throughout each task we augmented the data in order to make it more challenging for the CNN. The MNIST dataset is a database of handwritten digits and it has a training set with 60,000 samples and a test set of around 10,000 images (LeCun). We utilized TensorFlow Keras in order to develop and train our models. Keras is a high-level API that, "provides essential abstractions and building blocks for developing and shipping machine learning solutions with high iteration velocity" (keras.org). Through the use of Keras we were able to train our model on the MNIST dataset and introduce different augmentations that increased the difficulty for either the training set, the testing set or both.

Task1/Task 2

Traditionally within the MNIST dataset the images were 28 pixels in length on one side and the digit is centered in the image. For task two we were challenged to increase the size of the image to 56 pixels on one side and then to randomly place the 28-pixel image onto the 56-pixel image taking care to make sure it was seamlessly transported. We did this for both the training and the test set. After resizing our images and training our model we received an accuracy of 98.48%. We believe this was due to the fact that we resized both the training and the test set. If we had only resized one, we have seen a slightly lower accuracy, but I speculate the accuracy would be fairly similar since the digit was not augmented too much. It's also worth noting that the accuracy of Task 2 was essentially the same that of Task 1, which was based solely on the 28x28 images without any shifting and reported an accuracy of 98.98%

```
313/313 - 1s - loss: 0.0340 - accuracy: 0.9899
ACCURACY: 0.9898999929428101
```

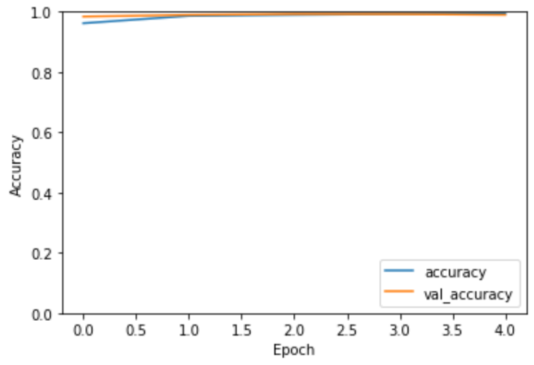


Figure 1 - Task 1

```
625/625 - 2s - loss: 0.0482 - accuracy: 0.9849
ACCURACY: 0.9848999977111816
```

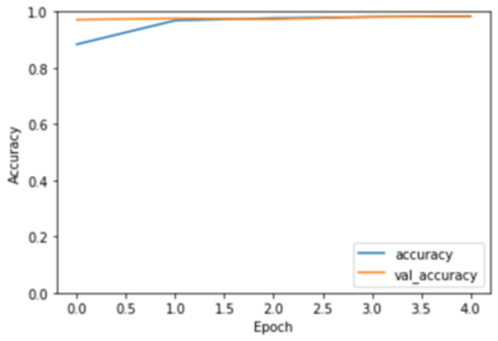
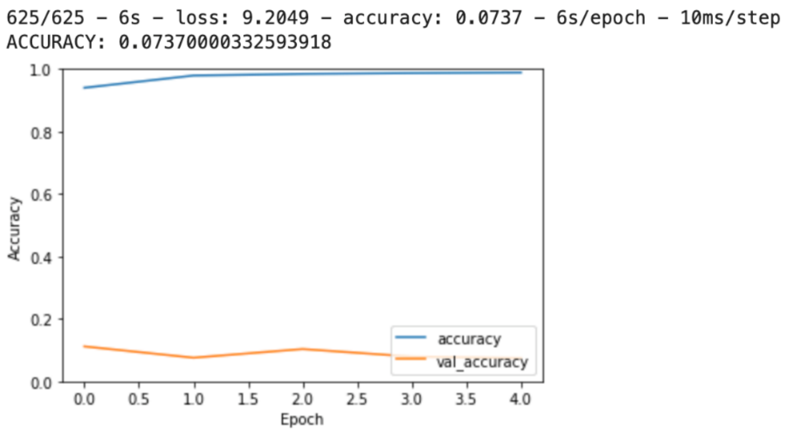


Figure 2 - Task 2

Task 3

For Task 3 we experimented with augmenting the training and the testing datasets in two different ways. For the training dataset we modified it so the digit only appeared in the top half of the image while for the testing dataset we modified it so the digit only appeared in the bottom half of the image. While only the position of the digit was changed the accuracy ended up being less than 10%. This value shows the model is essentially randomly predicting the value each time, statistically leading to success 1 out of 10 tries. This was because the training and the testing datasets looked so different. However, the training accuracy (blue line) was high in the graph because as the CNN continues to train the better it gets at predicting the digit which it matches the training dataset, in other words when the digit appears at the top of the image.

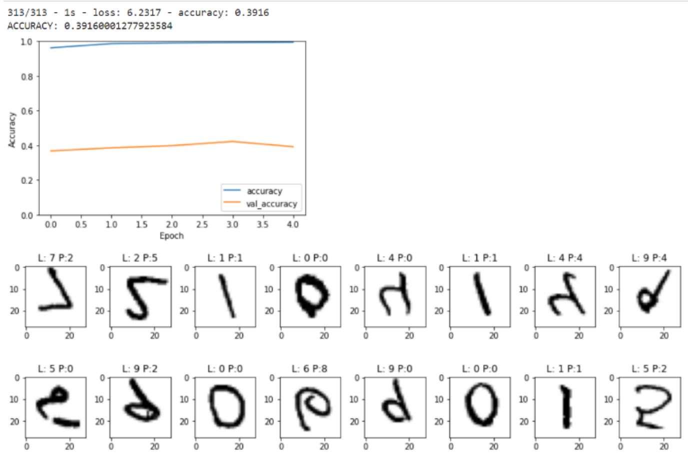


```
625/625 - 6s - loss: 9.2049 - accuracy: 0.0737 - 6s/epoch - 10ms/step
ACCURACY: 0.07370000332593918
```

Figure 3 - Task 3

Task 4

For task four we were allowed to choose our own augmentation variant. We chose to flip the images horizontally, so the image appeared to be upside down. We only applied this only to the testing dataset and left the training dataset as is. The reason we chose to only augment one of the datasets instead of both was to see if the CNN was still able to correctly identify the digits when one of the datasets was facing the opposite direction. We believed that by only augmenting one of the datasets our accuracy would be really poor due to the discrepancies in the images. This turned out to be true but the value was not as low as we expected. We ended up with an accuracy between 30-40%. The reason that our accuracy was higher than 10% which would have indicated that the model was randomly identifying values, could be the fact than 1 and 0 are essentially the same when reflected over the x-axis, thus leading the model to accurately identify those digits. For the other digits in the set it seemed that the model could not always identify the value when they were flipped.



```
313/313 - 1s - loss: 6.2317 - accuracy: 0.3916
ACCURACY: 0.39160001277923584
```

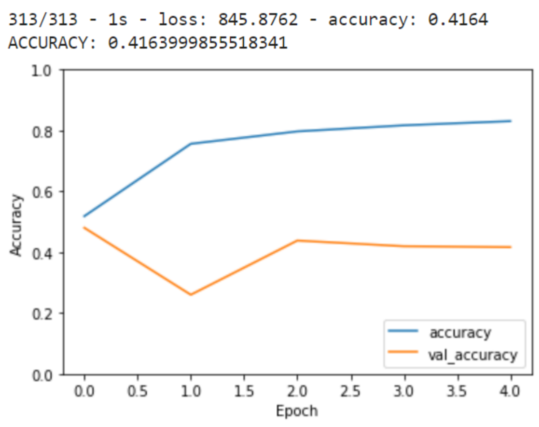
Figure 4 - Task 4

Augmentation/Extra Credit

The second part of this assignment required us to come up with an augmentation strategy to help improve the accuracy of Task 4, and then apply it to the other tasks to see how it would affect the accuracy output. Our strategy involved randomly applying a rotation, zoom and shift to each of the images in the dataset via the “ImageDataGenerator” preprocessor from the “tensorflow.keras.preprocessing.image” library. The range of possible rotation angles for the data was 0-100 degrees, this range was motivated by trying to train using samples where the rotation could potentially match the vertical flip from Task 4. We also applied a random zoom level of 0.5 - 2 and random shift range of 0.25 on the x and y axis hopefully improve the accuracy level in Task 3. It’s worth noting the same augmentation settings were used for each task, but further tweaking to the values could improve/worsen the models further.

Task 1 Augmented

After applying the augmentation to the training dataset for Task 1 we found that is actually decreased the accuracy of the model. The reported accuracy of the model dropped from 99% in the non-augmented data, to 41%. This drastic drop in accuracy shows that augmentation doesn’t always improve a model for the better. In this case, where it is known that the training dataset matches the test dataset so closely it would actually make more sense not apply augmentation at all.

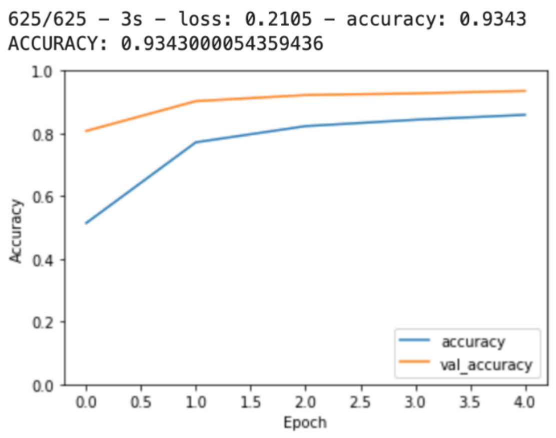


```
313/313 - 1s - loss: 845.8762 - accuracy: 0.4164
ACCURACY: 0.4163999855518341
```

Figure 5 - Task 1 Augmentation

Task 2 Augmented

The random shift incorporated in Task 2 was the same method that was used to randomly shift the values in the augmentation method. Because this, it seems that the zooming and rotation changes actually more of an impact on the accuracy of the model. For this task augmentation brought the accuracy value down to 93%, from the 98% accuracy of the non-augmented model. This is another example, while less extreme, of how augmenting data too much can actually have a negative impact on the prediction accuracy. In this case it makes sense to apply the horizonal and vertical shift, however the zoom and the rotation aren’t that necessary.

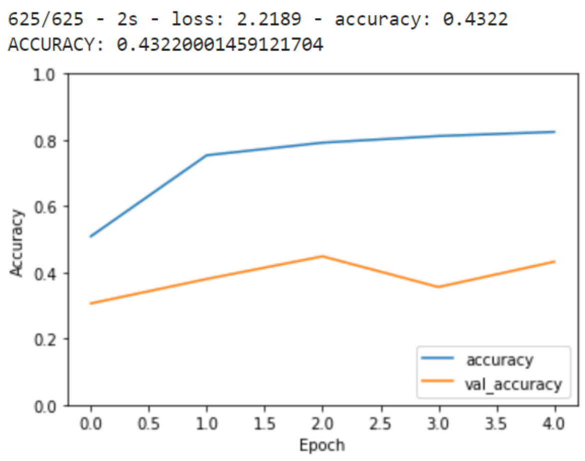


```
625/625 - 3s - loss: 0.2105 - accuracy: 0.9343
ACCURACY: 0.9343000054359436
```

Figure 6 - Task 2 Augmentation

Task 3 Augmented

Augmenting the training data helped improve the accuracy of Task 3. This is likely because the augmentation method shifted both on the x axis and the y axis, while the non-augmented version of this task only shifted on the x axis. Shifting on box axis's makes it possible to train with values more similar to the test dataset, where the images only exist at the bottom of the image. The accuracy of the model improved from 10% to about 43%, when augmentation as applied. The augmentation didn't improve the accuracy to the extent where the model now preforms like the base case in Task 1, however this change does make a good case for using augmentation to improve accuracy. It also begs the question, could tweaks to the augmentation settings such as extending the range of the horizontal, improve the accuracy further?

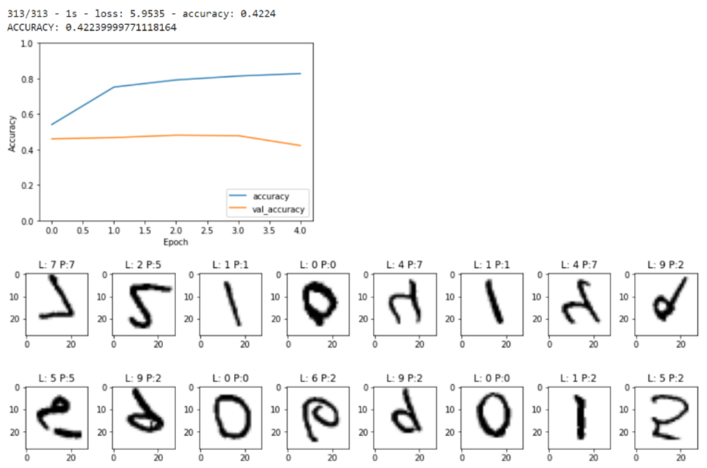


```
625/625 - 2s - loss: 2.2189 - accuracy: 0.4322
ACCURACY: 0.43220001459121704
```

Figure 7 - Task 3 Augmentation

Task 4 Augmented

Augmentation also improved the accuracy of Task 4, however not as drastically excepted. The actually value only jumped a few precent, from 39% before augmentation to 42% when augmentation was applied. We did notice that some values previously identified incorrectly were now reporting the correct value (for example the 7/2 value from the first value in the batch.) However, other values that were wrong in the non-augmented version of Task 4 were still wrong, albeit sometimes with different incorrect values.



```
313/313 - 1s - loss: 5.9535 - accuracy: 0.4224
ACCURACY: 0.42239999771118164
```

Figure 8 - Task 4 Augmentation

Report and Coding Resources

- http://yann.lecun.com/exdb/mnist/
- https://keras.io/about/
- https://medium.com/the-data-science-publication/how-to-augment-the-mnist-dataset-using-tensorflow-4fbf113e99a0
- https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html