Deliverable 3 — More Results

Sheheryar Parvaz

November 13th 2020

1 Changes to the model

To help the model adapt better, I've applied a few augmentations to the training data. Namely, jittering the brightness, contrast, saturation, and hue of the image to simulate darker environments and different colours. Similarly, I'm also applying affine transformations such as rotating and slightly scaling the image.

As to the training, I've added a scheduled learning rate to help the model converge slightly faster.

These do seem to have a small effect on accuracy on the testing set, but not by a significant margin. Unfortunately, the model is still quite ineffective at classifying real world images, so I'm going to continue experimenting. Now that I have a webpage with a camera set up, and greater familiarity with CNNs, I can iterate more effectively. With a webcam, I have found that some of the letters which are quite distinct, such as A, V, W, Y, and a few others, are successfully recognised by the model. I'll try to improve this further, and also display the confidence of other letters to get a better idea.

Unfortunately, since training the model takes quite a bit of time, in the order of more than two or three hours, experimenting with hyperparameters to see what works best is at best quite tedious. Some things I've tried include batch normalisation, dropout, and different output activation functions. I'm still working on getting some of these to reach convergence.

2 Transfer Learning

In researching how to quickly create an object detection model and how to more effectively build a classification model, I've come across *transfer learning* for PyTorch[1]. Since my dataset is very limited, it might be more effective to use an existing feature extractor to create a more general model and iterate more quickly. Additionally, one of the built-in object detection models can also be repurposed for hand detection, however doing this isn't straightforward so I need to look more into it. There are some useful datasets, including[2].

3 Final demonstration

For the demonstration, I've set up a basic flask server which can record using the webcam, and sends a snapshot to the server on a set interval. Additionally, the snapshot

can also be used in the JavaScript in the browser itself. This opens up two possibilities. First, the snapshots can be used by the server. Second, the images can be used in the browser using ONNX.js[3], which is simple to use with PyTorch. These two can be used in conjunction to create a model to first perform object detection, and then object classification, either server- or client-side. Which is done will be decided by further testing. As a fallback, if the object detection fails to work, then I can just have the classification run in a small part of the video feed. Right now, the server evaluates the video every two seconds and prints the result to standard output.

Thus, the final demonstration will be a website with a video playing the webcam feed, and it will display the confidence for the highest matching ASL signed letters. It will do so either by sending the image to the server at a set interval, or by processing the image directly in the browser using ONNX.js[3].

4 Results

```
Accuracy: 98.8% on 18690
 a:  98.77% on  733
 b:  99.27% on  686
 c:  99.34% on  762
 d:  98.51% on  738
 e:  99.27% on  687
 f:  99.43% on  707
 g:  99.31% on  726
 h:  99.00% on  701
 i:  98.91% on  734
 j:  99.36% on  622
 k:  98.91% on  733
 l:  99.58% on  720
 m:  98.66% on  749
 n:  98.69% on  685
 o:  98.41% on  691
 p:  99.72% on  727
 q:  99.72% on  714
 r:  96.90% on  741
 s:  99.35% on  773
 t:  95.44% on  745
 u:  98.84% on  773
 v:  97.21% on  754
 w:  99.45% on  722
 x:  98.53% on  682
 y:  99.60% on  746
 z:  99.84% on  639

Prediction \ Actual
    a   b   c   d   e   f   g   h   i   j   k   l   m   n   o   p   q   r   s   t   u   v   w   x   y   z
a 724   0   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   2  16   0   0   0   0   1   0
b   1 681   2   1   1   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   2   3   0   0   0   0
c   2   0 757   0   0   0   0   0   0   0   1   0   0   0   3   1   0   0   0   0   0   0   0   1   0   0
d   0   0   0 727   0   0   0   0   1   0   0   0   0   0   0   0   0   0   0   1   0   0   0   0   0   0
e   0   0   0   1 682   1   0   0   0   0   0   1   0   1   0   0   0   0   1   0   0   0   0   0   0   0
f   0   0   0   0   0 703   0   0   1   0   0   0   0   0   0   0   1   0   0   0   0   0   0   0   0   0
g   0   0   0   0   0   0 721   3   0   0   0   0   1   0   0   1   0   0   0   0   0   0   0   0   0   0
h   0   0   0   0   0   0   3 694   1   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
i   0   0   0   2   0   0   0   0 726   0   0   0   0   0   0   0   0   0   0   0   0   0   1   0   1   0
j   0   0   0   0   0   0   0   1   0 618   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
k   1   0   1   0   0   1   0   0   0   0 725   0   0   0   0   0   0   1   0   0   1   0   0   1   0   0
l   0   0   0   1   0   0   0   0   0   0   0 717   0   1   0   0   0   0   0   0   0   0   0   0   0   0
m   1   0   0   0   1   0   0   0   0   1   0   0 739   7   0   0   0   0   1   2   0   0   0   0   0   1
n   1   0   0   0   2   0   2   0   2   0   0   0   5 676   1   0   0   0   0   1   0   1   1   0   0   0
o   0   0   0   0   0   0   0   0   0   0   0   1   0   0 680   0   0   0   0   1   0   0   0   0   0   0
p   0   0   0   0   0   0   0   2   0   0   0   0   0   0   1 725   1   0   0   3   0   0   0   0   0   0
q   1   0   0   0   0   0   0   0   0   1   0   0   0   0   0   0 712   0   0   0   0   0   0   0   0   0
r   0   0   0   2   0   0   0   0   2   0   5   0   0   0   1   0   0 718   0   0   5   4   0   5   0   0
s   1   0   0   1   1   0   0   0   1   1   0   0   2   1   3   0   0   0 768   4   0   0   0   1   0   0
t   0   0   0   1   0   0   0   0   0   0   0   0   1   0   1   0   0   0   0 711   0   0   0   0   0   0
u   1   4   0   2   0   0   0   0   0   0   0   1   0   0   0   0   0  21   0   1 764   2   1   2   0   0
v   0   0   0   0   0   0   0   0   0   0   1   0   0   0   0   0   0   0   0   0   1 733   1   0   1   0
w   0   1   0   0   0   1   0   0   0   0   1   0   0   0   0   0   0   0   0   0   0  10 718   0   0   0
x   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1   2   1   0   1   0 672   0   0
y   0   0   0   0   0   0   0   1   0   0   0   1   0   0   0   0   0   0   0   0   0   0   0   0 743   0
z   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   3   0   0   0   0   0 638
```

References

[1] *Torchvision Object Detection Finetuning*. Retrieved 2020–11–13.
`https://pytorch.org/tutorials/intermediate/torchvision_tutorial.html`

[2] *EgoHands*. Retrieved 2020–11–13.
`http://vision.soic.indiana.edu/projects/egohands/`

[3] *ONNX.js*. Retrieved 2020–11–13.
`https://github.com/microsoft/onnxjs`