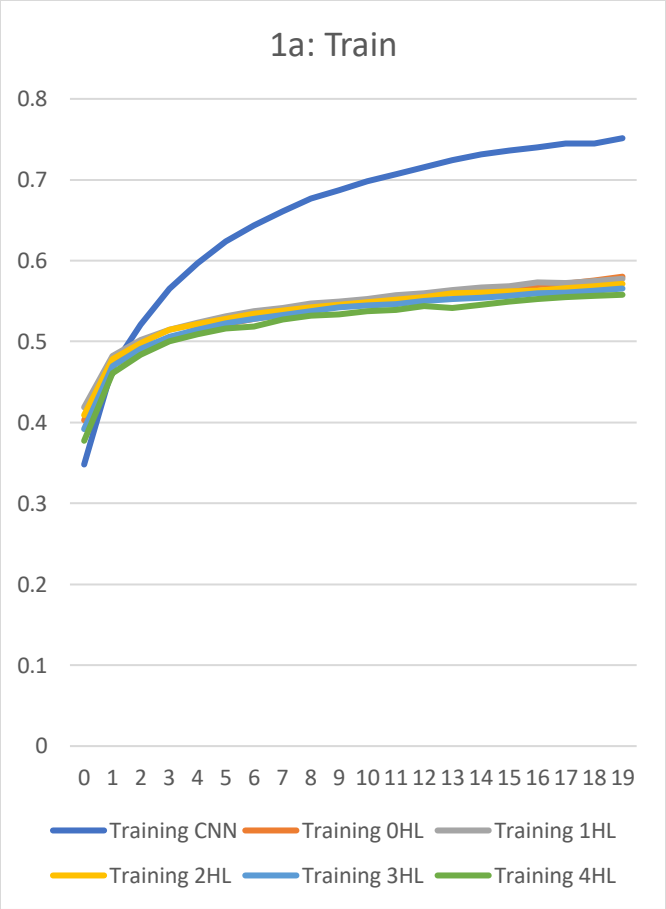
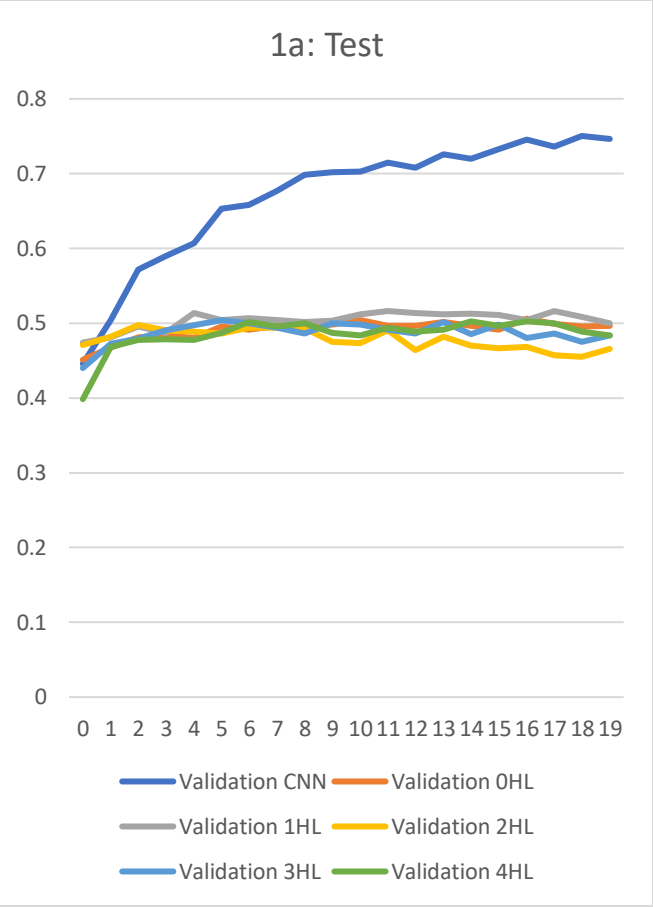
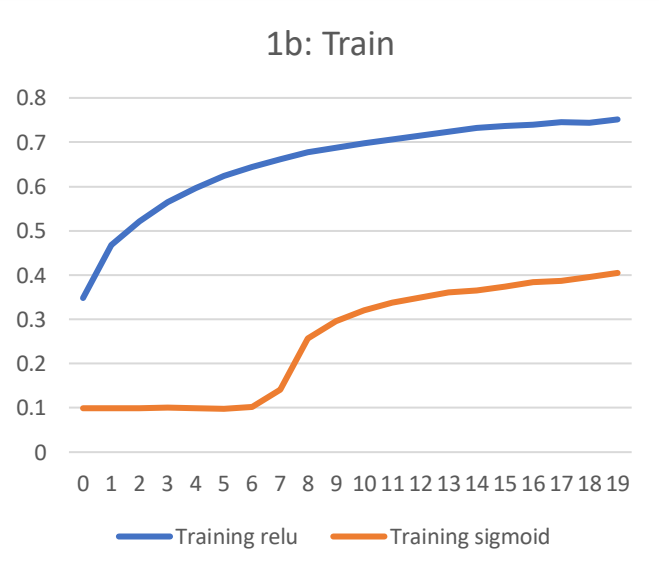
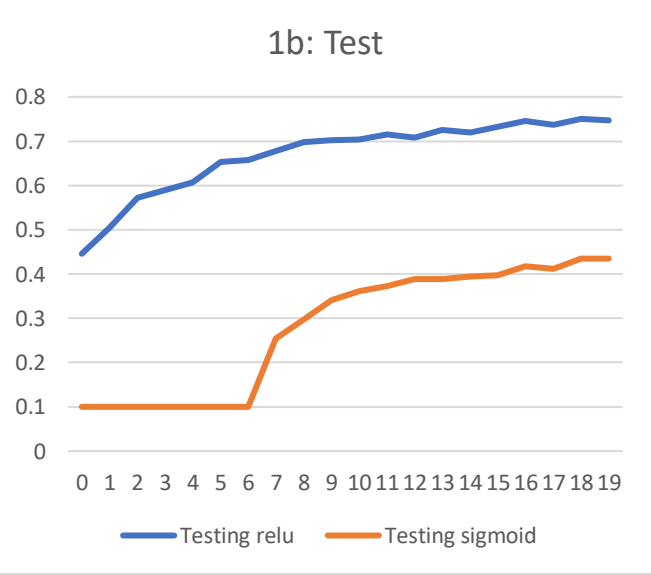
CS680 Assignment 4 `

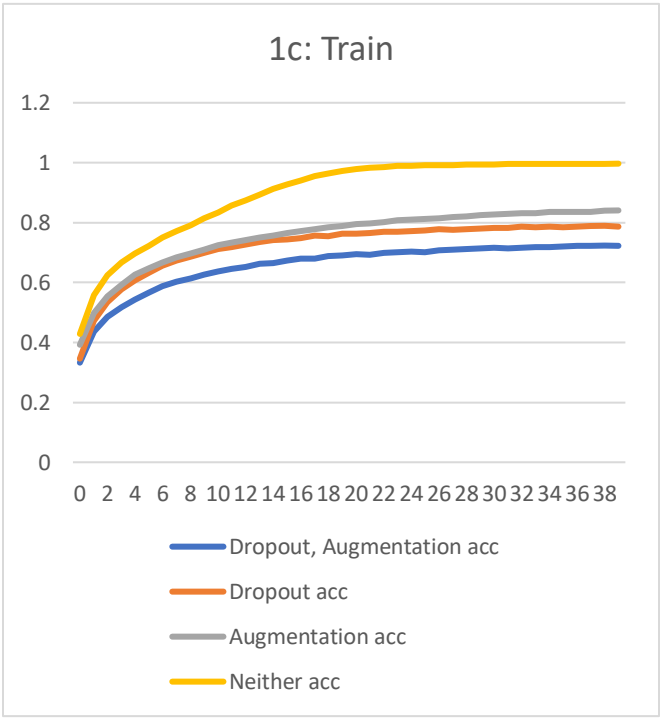
Kilby Baron 20773955

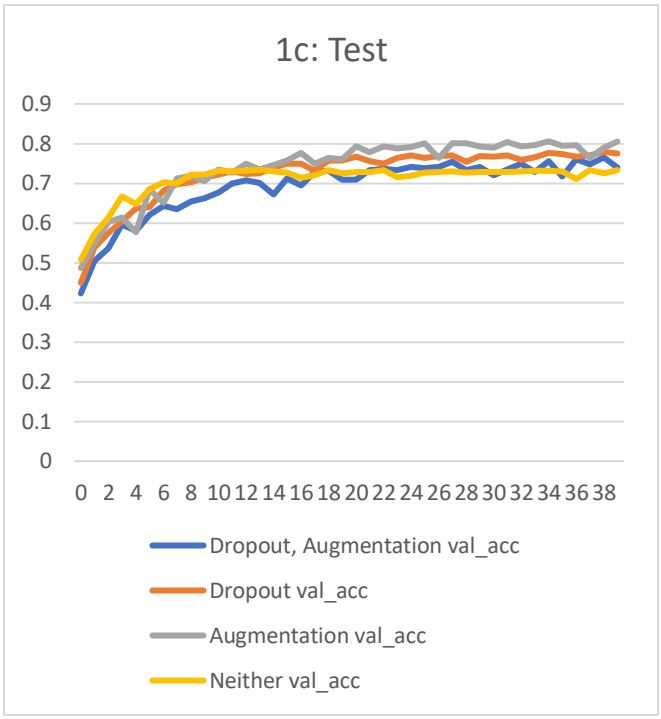


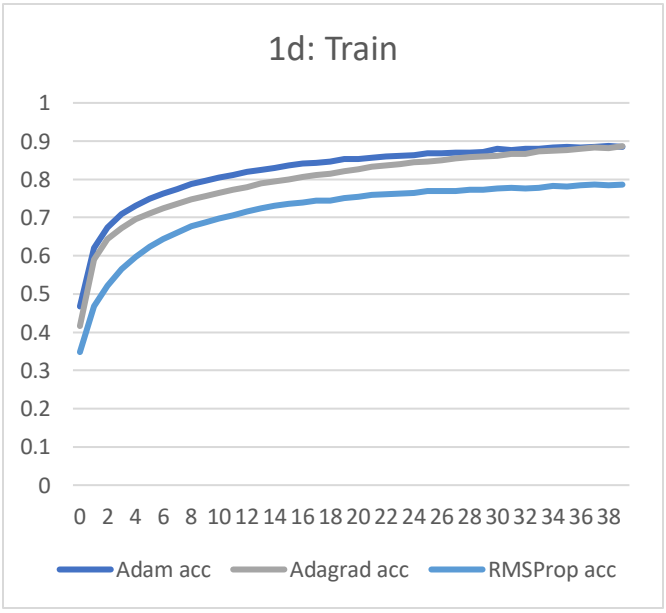


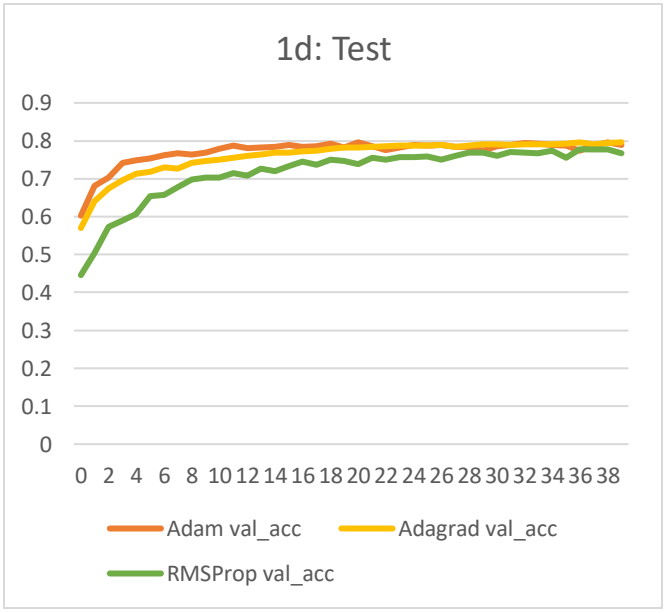


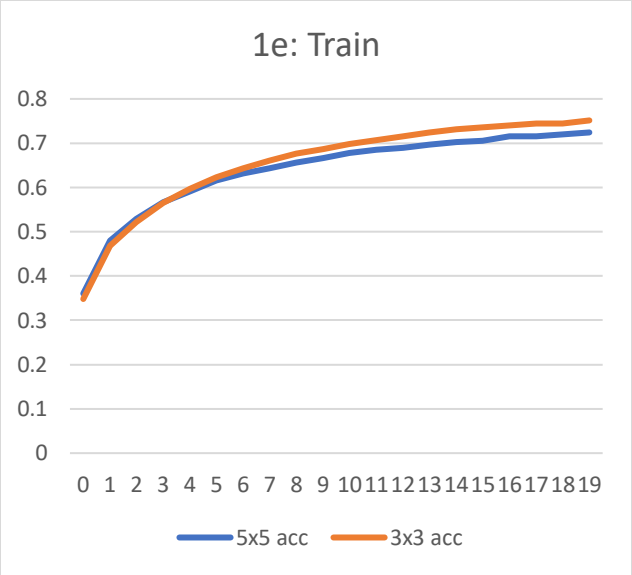


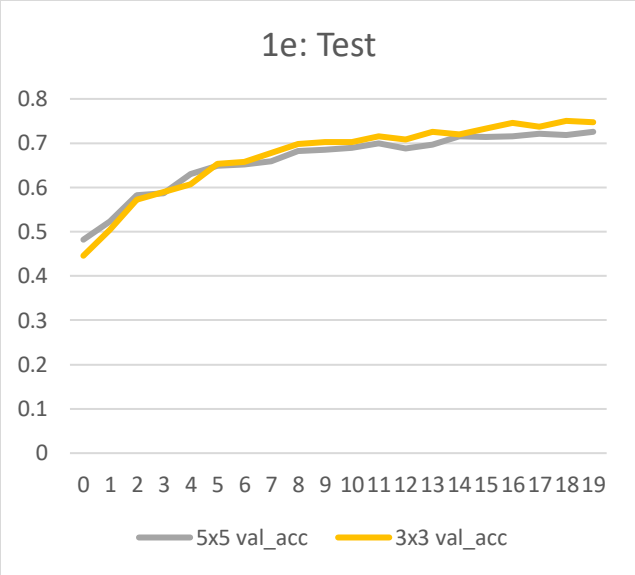

1c: Train
1.2
1
0.8
0.6
0.4
0.2
0
0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38
Dropout, Augmentation acc
Dropout acc
Augmentation acc
Neither acc


1c: Test
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38
Dropout, Augmentation val_acc
Dropout val_acc
Augmentation val_acc
Neither val_acc


1d: Train
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38
Adam acc
Adagrad acc
RMSProp acc


1d: Test
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38
Adam val_acc
Adagrad val_acc
RMSProp val_acc





Discussion

1a:

In both the train and test sets, the accuracy of the convolutional neural network is much better than all of the simple dense neural networks, which all perform similarly. The CNN is able to classify images so much better than simple dense neural networks because the CNN is able to break down the images into their basic features quickly and effectively. For example, a CNN can perform edge detection on an image using certain convolution filters, and summarize the edges succinctly using a pooling layer. Subsequent layers may then use the edges to detect simple shapes, and these shapes can then be used to detect even more detailed features. Being able to summarize the most distinctive aspects of images in this way using a handful of layers allows the CNN to classify images very effectively.

In comparison, the simple dense neural networks cannot detect the images' important features nearly as effectively. They are looking at each image as an array of pixels and looking for patterns but have to deal with a lot of noise. Therefore, they learn much more slowly, and after 20 epochs the number of hidden layers does not have a significant impact.

1b:

In both train and test sets, the sigmoid function is less accurate than the relu function. This is because the sigmoid function transforms most large values to 1 and most small values to zero, with only a small area of sensitivity in the middle.

I suspect that the accuracy for the sigmoid CNN increases rapidly around the 7^{th} epoch because this is where the CNN found the sigmoid function's "sweet spot". However, from the tenth epoch onward, the sigmoid CNN's accuracy increases at about the same rate as the relu CNN, just ~0.3 points lower.

After 20 epochs, the test accuracy of the sigmoid CNN is ~0.44, while the relu CNN's accuracy is ~0.75. This is because the limited sensitivity of the sigmoid function causes each layer to lose more information than the relu function, which makes it more difficult to adapt weights to extract features.

1c:

Dropout works by randomly dropping some units from the network when training, which can help reduce the risk of overfitting due to the high expressivity of relu units. Data augmentation is another technique to reduce overfitting by making reasonable modifications to training data to make new data. This increases the amount of data in the training set, and adds variation to the data. The data augmentation that was preset in the assignment randomly shifted images and randomly flipped images horizontally. This is reasonable for the categories of images being classified for this assignment, since none of the classes have a required horizontal orientation.

The model that used neither dropout nor data augmentation was the most accurate on the training set. This is because there was less variation in the images, so the model was able to learn to recognize those specific images really well. Adding either dropout or data augmentation reduced this accuracy by over 0.2 after 40 epochs. Dropout reduced the training accuracy because randomly dropped half of the units at each layer forced the model to train different subnetworks of units at every iteration. Data augmentation reduced the training accuracy because there were more variations of each type of image, so the model required more time to learn to recognize each variation. For these reasons, using both dropout and data augmentation was the least accurate on the training set, reducing the training accuracy by about 0.25 after 40 epochs.

On the test data, the dropout and data augmentation models performed the best. The data augmentation model did the best because it had more variations of each image to train on, which reduced overfitting and helped it recognize new images of each class. The dropout model performed second best because it did not rely on any small subset of units to classify the images, which also reduced overfitting and allowed it to identify new images more successfully. Using both data augmentation and dropout was not as successful as using only one of the two after 40 epochs. This is likely because the model required more epochs to increase its accuracy. Finally, the model that used neither overfitting solution was the least successful on the test set because it was less equipped to deal with variations in the new images.

1d:

Gradient descent tends to zigzag as it approaches the global minimum, which slows down convergence. Optimizers such as Adagrad, RMSProp, and Adam speed up gradient descent by reducing this zig-zagging. Adagrad does this by adjusting the length of the step in each direction separately using the sum of squares of the gradient. The problem with Adagrad is that the step size will approach zero because the sum of squares is always increasing. RMSProp is an optimizer that is very similar to Adagrad, but it addresses the decaying learning rate by dividing by the RMS instead of the sum of squares. A problem with RMSProp is that it lacks momentum, so Adam is another improved optimizer that induces momentum by replacing the gradient at each step with it's moving average. Ultimately, all of these

methods of gradient descent would converge at the same point given enough iterations, but I would expect to see Adam converge faster, followed by RMSProp, and finally adagrad.

Interestingly, in both the train and test sets, while Adam converged the fastest in this assignment, Adagrad converged faster than RMSProp. This is because Adam and Adagrad used default parameters, while RMSProp did not. In particular, RMSProp used a learning rate of 0.0001 and a decay of 1e-6, while the other optimizers had learning rates of 0.01 and 0.001, and decay rates of 0. When I set the parameters of RMSProp to the default settings, it performed closer to what I expected.

1e:

In both the train and test sets, using 3x3 filters with sets of 2 convolution layers is slightly more accurate than using 5x5 filters with one convolution layer. This is likely because using more convolutional layers allows the CNN to detect higher level features which allows it to classify the images more accurately; however, this difference in accuracy is marginal. The 5x5 filter is has a larger receptive field which can capture more basic image components.

2.

Note: My interpretation of Q2 is that the foreground object can be less than 10 pixels from the border after translation, because this is what Professor Poupart said on Piazza.

To answer both of these questions, I provide counter examples with the following specifications:

Input: 21x21 matrix of zeros with a single 1 in the center to represent the foreground image.

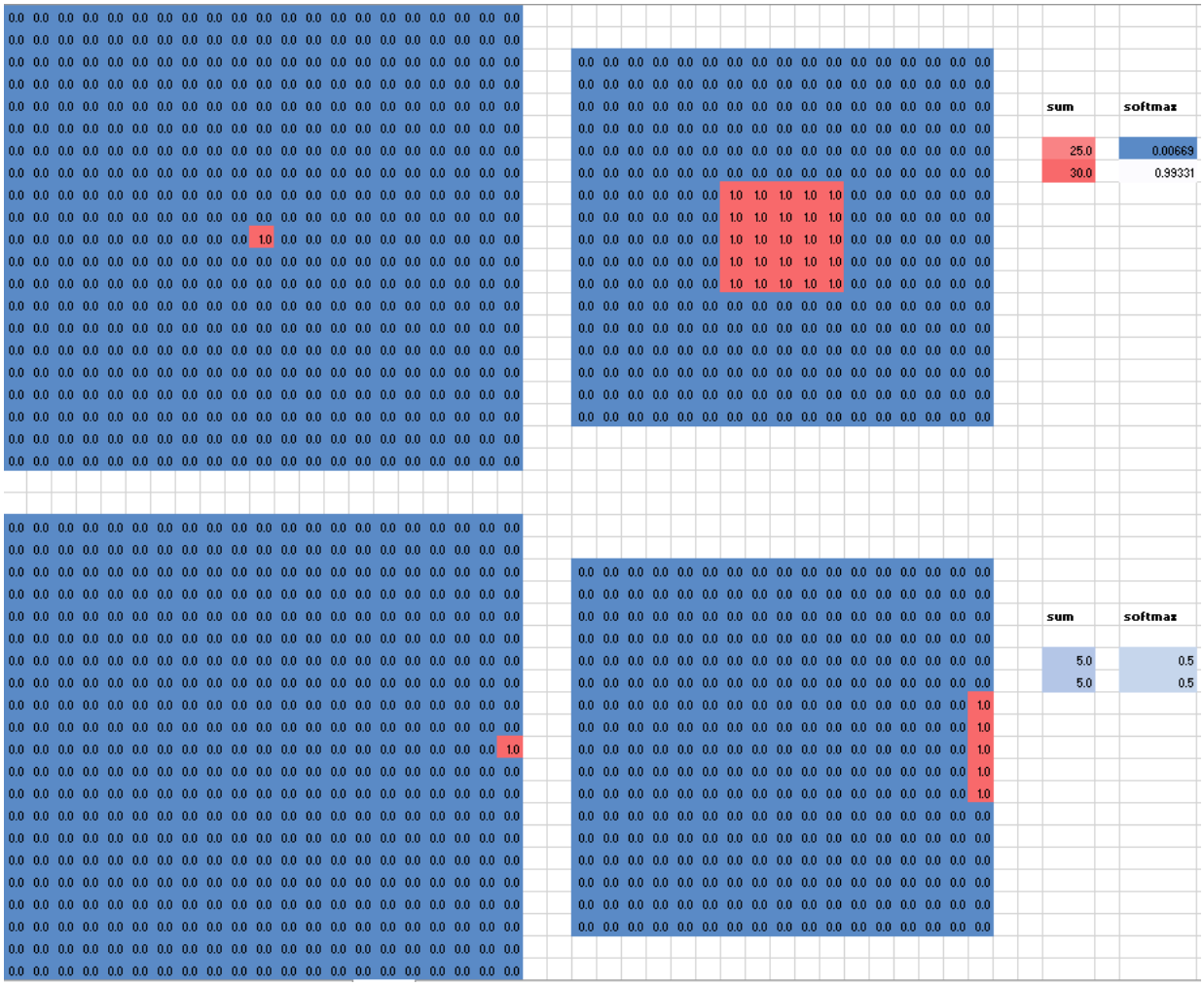
Convolution Filter: 5x5 matrix of ones

Activation functions: Relu

Number of output units: 2

Weights from last hidden layer to output units: All 1s except the the center column's weights are 2s for the second unit

a) The example below shows the input image, the output of the convolutional layer, and the output of the softmax layer. The "sum" column shows the sum of weighted inputs to the softmax units before the activation function. Translating the foreground image by 10 pixels causes 1/5 of the convolutions to capture the image, which dramatically changes the output.



b) In this example, the convolutional output has “same” padding so that more max pooling can be done. Again, the convolutions only capture the image 1/5 of the time after translation, which causes the softmax output to be different.