Homework 7 **NSC 3270 / 6270**
Due Friday Nov 11, by the end of the day **Fall 2022**
20 points

This assignment builds on our previous classification assignments by asking you to create multi-layer networks that learn to classify naturalistic images of clothing. Much like the dataset of handwritten digits used in an earlier assignment, Fashion-MNIST is a dataset of clothing images consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image, associated with a label from one of 10 classes.

`Homework7.ipynb` supplies code demonstrating how to read in the dataset and display several examples of clothing images. The clothing dataset is structurally similar to that of the MNIST handwritten digits, which means that previously shared code can be repurposed here.

The first part of the assignment asks you to create an autoassociator network that will reduce the dimensionality of the Fashion-MNIST dataset. This will create a set of lower-dimensional patterns that can serve as input to a multi-layer dense network that learns to classify the images. The second part of the assignment asks you to create a convolutional network that learns to classify the original images, without first passing them through an auto-associator.

This assignment has a lot of steps to it! We encourage you to use Piazza to help your classmates along, and we encourage you to look to Piazza for your classmates' tips, tricks, and suggestions. We may reward helpful students with bonus points!

Specific Tasks

Q1a (5 points). Create an autoassociator network that produces a lower-dimensional representation of the Fashion-MNIST images on its hidden layer. As in the earlier assignment with the MNIST handwritten digits, the Fashion-MNIST images will be flattened, converting them from 28x28 images to 784-dimensional patterns.

As we will discuss in the class slides, here are some key details to help you make an autoassociator: You make a densely connected network with a hidden layer and an output layer. You set the output layer to have the same number of units as the input layer. Then when you train the network (using `fit`), you use the training images both as inputs and as the target values for the output layer. If the hidden layer has fewer units than the input layer, this will force the network to create a lower-dimensional version of the training pattern, and then try to recreate that training pattern on the output layer. Note that there's no special "autoassociator" function in Keras. You're using the same Keras commands as in previous assignments, the main difference is in how you set the target values for the output layer.

Train the autoassociator on half of the Fashion-MNIST training images (30,000 of the training patterns). In class, we will discuss the appropriate activation functions for the hidden layer and output layer, to ensure that the autoassociator is approximating Principal Components Analysis (PCA).

It is up to you to determine the appropriate dimensionality of the autoassociator (i.e., the appropriate number of hidden units) as well as the appropriate number of training epochs.

In a markdown cell, justify the number of hidden units by describing what you did to determine the number you selected. If the dimensionality is too small, the classification network (Q1b) will likely perform quite poorly.

Include a plot of `loss` and `val_loss` over training epochs. As you try different values for the number of hidden units and number of training epochs, examine `val_loss` for characteristic signs of over-fitting, and if you see these signs, describe them as part of your markdown justification text.

Include figures that show autoassociator performance when there are an appropriate number of hidden units. Then show what happens when there are too few hidden units. Class slides will review how to show autoassociator performance. You'll take the output vector (which is the network's attempt to recreate the input vector) and reshape it into a 28x28 image, and then display this image. When there are too few hidden units, the recreated image should be blurry. When there are an appropriate number of hidden units, the recreated image should look very similar to the original image. You'll show examples of both of these cases.

Remember, if you're making a "good" autoassociator and a "bad" autoassociator, give the two networks different names. Otherwise, when you create the bad autoassociator, it might overwrite your good network, which you'll be using below.

Q1b (3 points). Use your autoassociator to make reduced-dimensionality training patterns for classification.

Use your trained autoassociator to create a reduced-dimensionality version of the remaining half of the Fashion-MNIST training set (the other 30,000 training patterns). In other words, you will project the remaining half of the Fashion-MNIST images through the network and save the hidden layer activations for each pattern. These are the lower-dimensional representations created by the autoassociator. The class slides will describe how to access the weights and biases from the trained autoassociator to calculate the activation values on the hidden layer for each pattern. Let's call these your *reduced training patterns*.

Now, use these reduced training patterns as the input patterns to train a densely connected multi-layered neural network that learns, via backpropagation, to classify the images as one of the 10 types of clothing.

As in previous assignments, you will determine a number of network properties, and justify these decisions in a markdown cell. Important network properties to figure out: The proper activation function on the hidden layer, and on the output layer (appropriate for a classification problem where output values are 0 and 1). Also determine the proper loss function for a classification network, the number of hidden layers, the number of hidden units on each hidden layer, the number of training epochs, and other settings related to training your network. We encourage you to make posts on Piazza describing things that worked for you, and things that didn't work, and we encourage you to try out settings that your classmates mention in their Piazza posts. But you'll still need to describe your justifications for your choices, and the justification can't be "because someone said it on Piazza"! Describe the architectures and settings you tried along the way and discuss why you chose your final settings.

Check the `loss` and `val_loss` during training to adjust the architecture and parameters of your network to achieve the best performance. Include plots of `loss` and `val_loss` for your final network.

Try to achieve the highest accuracy you can during training (without over-fitting) – make sure you store `accuracy` as a metric.

Q1c (2 points). Take your best network architecture and test it on the set of testing patterns.

Once you have a good network that attains a high accuracy on the training patterns, test your network using the test images and report its performance (using `network.evaluate()`). Note that you'll have to create a set of *reduced testing patterns*, using the same technique you used to create the *reduced training patterns*.

Note that the network should not do as well on the testing patterns as it does on the training patterns. This is normal and good, and we are interested to see how much of a performance drop there is from the training performance to the testing performance.

You should never adjust your network based on the performance with the test images (we will talk about why in class).

Q2 (10 points). Create a convolutional neural network that classifies the Fashion-MNIST images. A convolutional network takes images as input (not flattened images). So for this task, you will use the 28x28 dimensional images as training patterns, not the 784-dimensional flattened vectors or the reduced training patterns.

In the class slides we will provide guidance on how to create a convolutional neural network in Keras. Your task is to find a convolutional neural network architecture that does the best job of classifying the images.

We will show you how to create different kinds of layers in the network: Convolutional layers, max pooling layers, and dense layers. You can explore different architectures that combine these different kinds of layers in different orders. You can also adjust the number of convolutional maps in a layer, the total number of dense layers, and the number of hidden units in the dense layers.

You will want to inspect the val_loss to make sure you are not overtraining the network. Check the `val_loss` and `val_accuracy` during training to adjust the architecture and parameters of your network to achieve the best performance. Make note of any signs of overfitting for particular network architectures (when `val_loss` increases while regular `loss` decreases). Include plots of `loss` and `val_loss` in your notebook.

As in Q1, try to achieve the highest accuracy you can during training (without over-fitting) – make sure you store `accuracy` as a metric.

In a markdown cell, describe the various choices you made and justify them. This should include describing the architectures and settings you tried and discussing why you went with a different architecture and set of settings. As in the previous task, we encourage you to post on Piazza to help your classmates.

Once you have settled on a good network architecture based on training performance, then use `network.evaluate()` on the test images, and report the network's classification performance. As before, the network will do worse on the testing images, and we will be interested to see how much worse.

Unexcused late assignments will be penalized 10% for every 24 hours late, starting from the time class ends, for a maximum of two days, after which they will earn a 0.