CSC420H1 L0101, Fall 2018
Assignment 5

Name: JingXi, Hao
Student Number: 1000654188

Due Date: 21st November, 2018 11:59pm

1. **Facial Recognition and Fast Image Retrieval (7 points)**

   In this question you will implement a simplified recognition and retrieval system for images of faces based on the one discussed in class. Suppose you have a dataset of facial images for your employees. You want to build a system that can recognize an employee and let them into the office, or detect a stranger and deny access. The provided pretrained model and data is obtained from https://www.coursera.org/learn/convolutional-neural-networks/home/week/4. Refer to facenet.py for loading the model and data.

   (a) Suppose you have a few (around 5 images) per employee. Is it a good idea to train a neural network to classify an input image into either one of the employees and an "unknown" category? Why or why not?

   Solution:
   This is NOT a good idea becasue of the small size of the dataset. Around 5 images per employee is definitely not enough to train a neural network to classify an input image into either one of the employees and an "unknown" category with accuracy.

   (b) When using a bag of visual words description of images, it is helpful to use a TF-IDF (term frequency inverse document frequency) weighting of each word. Explain the benefit of using TF-IDF over a present/absent representation.

   Solution:
   It is helpful to use a TF-IDF weighting of each word when using a bag of visual words descriptions of images.

   - **Without TF-IDF**
     A bag of visual words descriptions of each image is the histogram analysis of word occurrences in each image. Thus, if certain words appear frequently in most of the images in the database, then these visual words would dominate the vector and have great effect on the similarity computation even though they are non-discriminative. This would cause that our similarity computed does not make much sense.

- **With TF-IDF**
  TF-IDF re-weigh the entries in bag-of-words vectors such that the words frequently occurred in many images are down-weighted. Therefore, the impact of those words on similarity computation is reduced.

(c) For our recognition and retrieval system we want our descriptors to be invariant to changes in pose or lighting. We can use a pre-trained model to recover an embedding for faces that maps images of faces to a 128-dimensional vector. For more details check out FaceNet https://arxiv.org/pdf/1503.03832.pdf. The provided code includes the Keras model and pretrained weights. The model takes in a 96x96 RBG image, and outputs a 128 dimensional embedding. For each image in the saved_faces directory compute and save its embedding. Note that in this case each image has only one descriptor which is the embedding.

Solution:
Please see the detailed code implementation of *compute_and_store_embeddings*() in the file, ***facenet.py***.

(d) For image retrieval, the embeddings have to be clustered. Explain the purpose of this step.

Solution:
The purpose to have the embeddings clustered is to gather visual words together based on the embeddings. Therefore, every time we can map each feature vector for images in database to a visual word by finding the closest visual word (i.e. find the minimum Euclidean distance between each feature vector to each cluster centers).

(e) Using K-means clustering, cluster your saved embeddings into 6 clusters (number of distinct persons in the data). You may use the scikit-learn function for k-means clustering. What should you choose as your visual word representation?

Solution:
Please see the detailed code implementation of *cluster_embeddings*() in the file of, ***facenet.py***.

In this case, we have 6 clusters since there are 6 distinct persons in the data. Then, each cluster center can be chosen as our visual word representation. Thus, we have 6 visual words in total.

(f) Write an inverted index for your saved_faces dataset. For this step, you can assign an integer from 0 to 5 for each distinct visual word. You can save your visual words in a 2D numpy array, using the index to select a specific visual word. You can save it for later use using numpy.save(). Then for each image in saved_faces, find its corresponding visual word (note that you can use the output of K-means for this). Your inverted index can be a dictionary where the key is the index of the word and the values are image filenames.

Solution:

Please see the detailed code implementation of $save\_visual\_words()$ and $build\_inverted\_index()$ in the file of , ***facenet.py***.

(g) For each image in input_faces, you want to find it's matching images from saved_faces. Describe a method to do this.

Solution:

For each image in input_faces, call it query image. The method to find matching images from saved_faces for each query image are described as following.

- First, compute the embeddings for all images in saved_faces and for a query image.
- Then, cluster the embeddings of images in saved_faces to get 6 clusters. Then, We call each cluster center a visual word, which means that we have 6 visual words in this case.
- Assign the embedding of each image in saved_faces to its closest cluster (visual word) by computing the Euclidean distance between the embedding and each cluster center and picking the visual word that offers the smallest distance.
- Build an inverted file index which is a dictionary where the key is the index of the word and the values are image filenames (images from saved_faces).
- Assign the embedding of a query image to its closest visual word by the method mentioned in step 3. In this case, each image only contains one face. Thus, only one visual word assigned to each image. Then, for query image, loop up the visual word in the inverted file index to get a list of images that share the same visual word with it. Note that, in this step, a threshold is used to prevent wrong visual word assignment, which means that if the smallest Euclidean distance is greater than the threshold, then no visual word can be assigned, which means no matching images. Note that for this situation, no need to compute bag-of-word vectors and do TF-IDF weighting, since only one visual word for each image.

(h) Implement your method from part (g). Note that one of the images is a new person and should produce no matches. Your output file should include the input image name, along with the names of all images that matched next to it. Don't worry if there are a few false matches. You will be marked based on your approach rather than having a perfect output.

Solution:

Please see the detailed code implementation of $find\_matching\_images()$ in the file, ***facenet.py***.

Show all matching images for each image in input_faces below.

```
All matching images passed threshold for input_faces/face_image_107.jpg are:

['saved_faces/face_image_4.jpg', 'saved_faces/face_image_6.jpg', 'saved_faces/face_image_7.jpg', 'saved_faces/face_im
age_15.jpg', 'saved_faces/face_image_19.jpg', 'saved_faces/face_image_31.jpg', 'saved_faces/face_image_38.jpg', 'save
d_faces/face_image_47.jpg', 'saved_faces/face_image_50.jpg', 'saved_faces/face_image_58.jpg', 'saved_faces/face_image
_69.jpg', 'saved_faces/face_image_70.jpg', 'saved_faces/face_image_81.jpg', 'saved_faces/face_image_87.jpg', 'saved_f
aces/face_image_95.jpg', 'saved_faces/face_image_96.jpg', 'saved_faces/face_image_112.jpg', 'saved_faces/face_image_1
17.jpg', 'saved_faces/face_image_130.jpg', 'saved_faces/face_image_133.jpg', 'saved_faces/face_image_138.jpg', 'saved
_faces/face_image_143.jpg', 'saved_faces/face_image_144.jpg', 'saved_faces/face_image_149.jpg', 'saved_faces/face_ima
ge_154.jpg', 'saved_faces/face_image_157.jpg', 'saved_faces/face_image_158.jpg', 'saved_faces/face_image_160.jpg', 's
aved_faces/face_image_175.jpg', 'saved_faces/face_image_187.jpg', 'saved_faces/face_image_188.jpg', 'saved_faces/face
_image_190.jpg']

All matching images passed threshold for input_faces/face_image_109.jpg are:

['saved_faces/face_image_3.jpg', 'saved_faces/face_image_22.jpg', 'saved_faces/face_image_24.jpg', 'saved_faces/face_
image_25.jpg', 'saved_faces/face_image_29.jpg', 'saved_faces/face_image_30.jpg', 'saved_faces/face_image_35.jpg', 'sa
ved_faces/face_image_41.jpg', 'saved_faces/face_image_51.jpg', 'saved_faces/face_image_71.jpg', 'saved_faces/face_ima
ge_78.jpg', 'saved_faces/face_image_104.jpg', 'saved_faces/face_image_109.jpg', 'saved_faces/face_image_114.jpg', 'sa
ved_faces/face_image_115.jpg', 'saved_faces/face_image_120.jpg', 'saved_faces/face_image_123.jpg', 'saved_faces/face_
image_124.jpg', 'saved_faces/face_image_125.jpg', 'saved_faces/face_image_131.jpg', 'saved_faces/face_image_139.jpg',
'saved_faces/face_image_146.jpg', 'saved_faces/face_image_166.jpg', 'saved_faces/face_image_168.jpg', 'saved_faces/fa
ce_image_169.jpg', 'saved_faces/face_image_171.jpg', 'saved_faces/face_image_174.jpg', 'saved_faces/face_image_176.jp
g', 'saved_faces/face_image_179.jpg', 'saved_faces/face_image_182.jpg', 'saved_faces/face_image_189.jpg', 'saved_face
s/face_image_193.jpg']

All matching images passed threshold for input_faces/face_image_116.jpg are:

['saved_faces/face_image_1.jpg', 'saved_faces/face_image_2.jpg', 'saved_faces/face_image_5.jpg', 'saved_faces/face_im
age_11.jpg', 'saved_faces/face_image_14.jpg', 'saved_faces/face_image_16.jpg', 'saved_faces/face_image_23.jpg', 'save
d_faces/face_image_36.jpg', 'saved_faces/face_image_46.jpg', 'saved_faces/face_image_48.jpg', 'saved_faces/face_image
_55.jpg', 'saved_faces/face_image_56.jpg', 'saved_faces/face_image_61.jpg', 'saved_faces/face_image_66.jpg', 'saved_f
aces/face_image_79.jpg', 'saved_faces/face_image_88.jpg', 'saved_faces/face_image_116.jpg', 'saved_faces/face_image_1
28.jpg', 'saved_faces/face_image_148.jpg', 'saved_faces/face_image_150.jpg', 'saved_faces/face_image_162.jpg', 'saved
_faces/face_image_164.jpg', 'saved_faces/face_image_172.jpg', 'saved_faces/face_image_177.jpg', 'saved_faces/face_ima
ge_184.jpg', 'saved_faces/face_image_196.jpg']

All matching images passed threshold for input_faces/face_image_119.jpg are:

['saved_faces/face_image_10.jpg', 'saved_faces/face_image_12.jpg', 'saved_faces/face_image_13.jpg', 'saved_faces/face
_image_26.jpg', 'saved_faces/face_image_39.jpg', 'saved_faces/face_image_42.jpg', 'saved_faces/face_image_43.jpg', 's
aved_faces/face_image_53.jpg', 'saved_faces/face_image_59.jpg', 'saved_faces/face_image_63.jpg', 'saved_faces/face_im
age_64.jpg', 'saved_faces/face_image_65.jpg', 'saved_faces/face_image_68.jpg', 'saved_faces/face_image_72.jpg', 'save
d_faces/face_image_73.jpg', 'saved_faces/face_image_75.jpg', 'saved_faces/face_image_85.jpg', 'saved_faces/face_image
_97.jpg', 'saved_faces/face_image_102.jpg', 'saved_faces/face_image_103.jpg', 'saved_faces/face_image_107.jpg', 'save
d_faces/face_image_111.jpg', 'saved_faces/face_image_113.jpg', 'saved_faces/face_image_132.jpg', 'saved_faces/face_im
age_136.jpg', 'saved_faces/face_image_142.jpg', 'saved_faces/face_image_170.jpg', 'saved_faces/face_image_178.jpg',
'saved_faces/face_image_183.jpg', 'saved_faces/face_image_191.jpg']

All matching images passed threshold for input_faces/face_image_126.jpg are:

['saved_faces/face_image_3.jpg', 'saved_faces/face_image_22.jpg', 'saved_faces/face_image_24.jpg', 'saved_faces/face_
image_25.jpg', 'saved_faces/face_image_29.jpg', 'saved_faces/face_image_30.jpg', 'saved_faces/face_image_35.jpg', 'sa
ved_faces/face_image_41.jpg', 'saved_faces/face_image_51.jpg', 'saved_faces/face_image_71.jpg', 'saved_faces/face_ima
ge_78.jpg', 'saved_faces/face_image_104.jpg', 'saved_faces/face_image_109.jpg', 'saved_faces/face_image_114.jpg', 'sa
ved_faces/face_image_115.jpg', 'saved_faces/face_image_120.jpg', 'saved_faces/face_image_123.jpg', 'saved_faces/face_
image_124.jpg', 'saved_faces/face_image_125.jpg', 'saved_faces/face_image_131.jpg', 'saved_faces/face_image_139.jpg',
'saved_faces/face_image_146.jpg', 'saved_faces/face_image_166.jpg', 'saved_faces/face_image_168.jpg', 'saved_faces/fa
ce_image_169.jpg', 'saved_faces/face_image_171.jpg', 'saved_faces/face_image_174.jpg', 'saved_faces/face_image_176.jp
g', 'saved_faces/face_image_179.jpg', 'saved_faces/face_image_182.jpg', 'saved_faces/face_image_189.jpg', 'saved_face
s/face_image_193.jpg']

All matching images passed threshold for input_faces/face_image_600.jpg are:

No matching images.

All matching images passed threshold for input_faces/face_image_97.jpg are:

['saved_faces/face_image_10.jpg', 'saved_faces/face_image_12.jpg', 'saved_faces/face_image_13.jpg', 'saved_faces/face
_image_26.jpg', 'saved_faces/face_image_39.jpg', 'saved_faces/face_image_42.jpg', 'saved_faces/face_image_43.jpg', 's
aved_faces/face_image_53.jpg', 'saved_faces/face_image_59.jpg', 'saved_faces/face_image_63.jpg', 'saved_faces/face_im
age_64.jpg', 'saved_faces/face_image_65.jpg', 'saved_faces/face_image_68.jpg', 'saved_faces/face_image_72.jpg', 'save
d_faces/face_image_73.jpg', 'saved_faces/face_image_75.jpg', 'saved_faces/face_image_85.jpg', 'saved_faces/face_image
_97.jpg', 'saved_faces/face_image_102.jpg', 'saved_faces/face_image_103.jpg', 'saved_faces/face_image_107.jpg', 'save
d_faces/face_image_111.jpg', 'saved_faces/face_image_113.jpg', 'saved_faces/face_image_132.jpg', 'saved_faces/face_im
age_136.jpg', 'saved_faces/face_image_142.jpg', 'saved_faces/face_image_170.jpg', 'saved_faces/face_image_178.jpg',
'saved_faces/face_image_183.jpg', 'saved_faces/face_image_191.jpg']
```

(i) So far the task was to find corresponding images where there is only one person per image. Suppose you hold an office party and take lots of pictures. Your employees want to find pictures of themselves, even when taken with other people. Describe a more general retrieval system based on the idea of face embeddings. What are the challenges in this case? You may use drawings/diagrams to support your explanation.

Solution:

The steps of a more general retrieval system based on the idea of face embeddings are listed as following:

- First, need to do face detection by drawing bound box around each face shown in the image. Then, crop by the bounding box, which is a query image.
- Compute the embeddings for all images in the database and for a query image.
- Then, cluster the embeddings of images in the database to get k clusters. Then, We call each cluster center a visual word, which is a vector that lives in the same dimensional space as the descriptors.
- Assign every embedding for images in the database and for a query image to its closest cluster by computing the Euclidean distance between embedding and cluster center.
- Build an inverted file index which is a dictionary where the key is the index of the word and the values are image filenames (images from the database).
- For a query image, look up all the visual words in the inverted file index to get a list of images that share at least one visual word with it.
- Compute a bag-of-words vector for each retrieved image and query and do re-weight using TF-IDF.
- Compute similarity between query BoW vector and all retrieved image BoW vectors. Sort similarities from highest to lowest. Take top K most similar images.
- Make spatial verification on top K retrieved images and remove those with less inliers.

2. **Deep Learning (5 points)**

In this question you will experiment with training and evaluating an image classifier on the fashion-MNIST dataset (because MNIST is too easy).

Refer to https://www.tensorflow.org/tutorials/keras/basic_classification to learn how to interact with the dataset using tensorflow or keras.

Refer to https://pytorch.org/docs/master/torchvision/datasets.html#fashion-mnist if you prefer to use pytorch.

For free GPU access you can use https://colab.research.google.com/ to speed up training.

(a) Describe the vanishing gradient problem. What are some ways to mitigate the vanishing gradient problem?

Solution:

- First, we describe the vanishing gradient problem. The vanishing gradient problem is a difficulty occurred during the neural network training with gradient based learning methods and backpropagation.This problem would cause that the gradient tends to be vanishingly small when move backward through the hidden layers, which implies that the neurons in the earlier layers have less learning ability than the neurons in the later layers. This problem would prevent the weight from changing its value. Also, this problem may stop training completely in the worst case.

- Then, we show the ways to mitigate the vanishing gradient problem below. The rectified linear unit can be employed to tackle this problem. Also, we can use a fast GPU that can perform more steps.

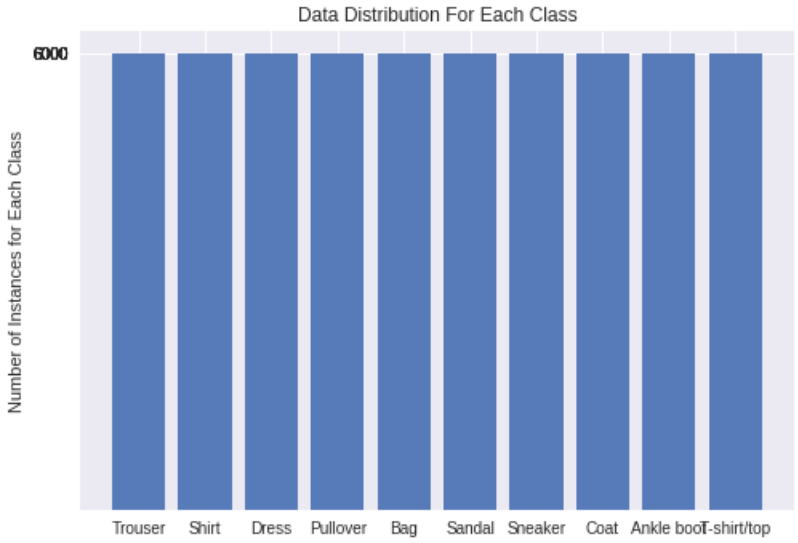
(b) Why do CNN's include max pooling layers?

Solution:
We use max pooling layers in CNN to get invariance to small shifts in position. Max pooling layers are able to effectively reduce the spatial size of the representation, reduce the number of parameters and the amount of computation, as well as control overfitting.

(c) After loading and preprocessing the data:

i. Show a bar graph of the number of instances in each class.

Solution:
Show the bar graph of the number of instances in each class below. Notify that for each class, we have the same number of instances, which is 6000.



ii. Split your data into a training set and validation set. Something like 70-30 is fine.

Solution:
Please see the detailed code implementation for this question in the file, ***csc420_a5_q2.ipynb***.

(d) Build a model using keras, tensorflow or pytorch. Think about your choice of activation functions, number of layers . . . etc What should the activation at the output be? Don't worry if you don't

initially have an idea about the hyperparameters of your model, the point is to try different things and experiment.

Solution:

Please see the detailed code implementation for this question in the file, ***csc420_a5_q2.ipynb***.

The model chosen is show below and it is compiled with Adam optimizer with learning rate equals to 0.0004, batch size equals to 32, and cross-entropy as loss function. This model obtained 90.37% test set accuracy.

```
model = keras.Sequential([
    keras.layers.Conv2D(filters=64, kernel_size=3, strides=1, activation='relu', input_shape=(28,28,1)),
    keras.layers.MaxPooling2D(pool_size=2),
    keras.layers.Conv2D(filters=32, kernel_size=3, strides=1, activation='relu'),
    keras.layers.Conv2D(filters=32, kernel_size=3, strides=1, activation='relu'),
    keras.layers.MaxPooling2D(pool_size=2),
    keras.layers.Flatten(),
    keras.layers.Dense(128, activation=tf.nn.relu),
    keras.layers.Dense(10, activation=tf.nn.softmax)
])
```

```
model.compile(optimizer=keras.optimizers.Adam(lr=0.0004),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

(e) Define your loss function in terms of your model's prediction and ground truth label. Write it down in the pdf.

Solution:

Here we use cross-entropy as our loss function. After loading and preprocessing the data, we can convert each label expression in labels vector into $1 - of - K$ encoding, where is a 1 by 10 vector with one of the entries is 1 and the rest are 0 instead of using an integer representation for each label. Let N denote the total number of images in the set. Then, our ground truth labels vector, y_{true}, for each set would be a N by 10 matrix. Then, let our model's predictions vector on the set be y_{pred} with dimension of N by 10. Each row in y_{pred} is prediction for an image in set and each entry in a prediction describe the "confidence" of the model that the image corresponds to each of the 10 different classes. Then, for each prediction and its corresponding ground truth label, the loss function is defined as

$$-(y_{true}^{(i)} log(y_{pred}^{(i)}) + (1 - y_{true}^{(i)}) log(1 - y_{pred}^{(i)}))$$

(f) Write a training loop to train your model (you may use Adam as your optimizer, but feel free to experiment). Credit is given for plotting the training loss and validation loss, as well as the training and validation accuracy. What is your choice of batch size? When should you stop training? You may change your initial architecture and hyperparameters if you don't like your results. Try to get over 88% test set accuracy. In your report, make a diagram of your final architecture, learning rate, batch size, number of epochs trained and final training/validation loss and accuracy. Include your plots in your report.

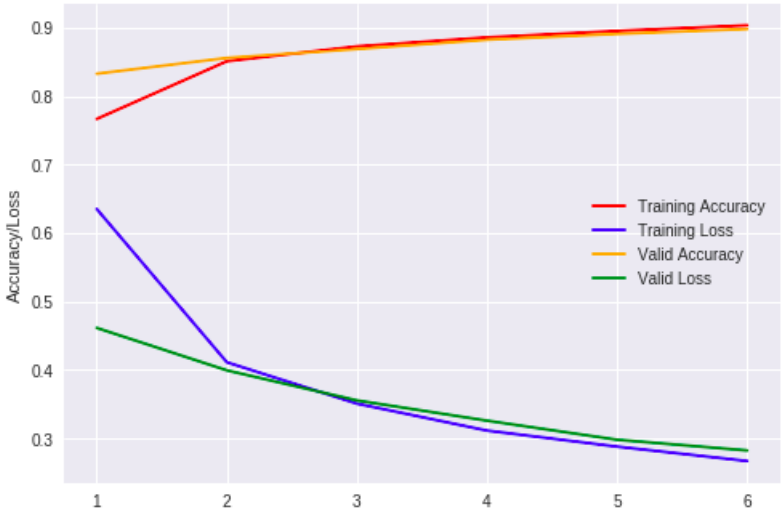
Solution:

Please see the detailed code implementation for this question in the file, ***csc420_a5_q2.ipynb***.

Keep execute the training loop to train our model until the accuracy increased for the validation set becomes negative. Then, we show a table below, listing the learning rate, batch size, number of epochs trained, and final training/validation loss and accuracy.

	Values
Learning Rate	0.0004
Batch Size	32
Number of Epochs Trained	6
Final Training Loss	0.2670108767407281
Final Training Accuracy	0.9036190476190477
Final Validation Loss	0.28244575752152334
Final Validation Accuracy	0.8982222222222223

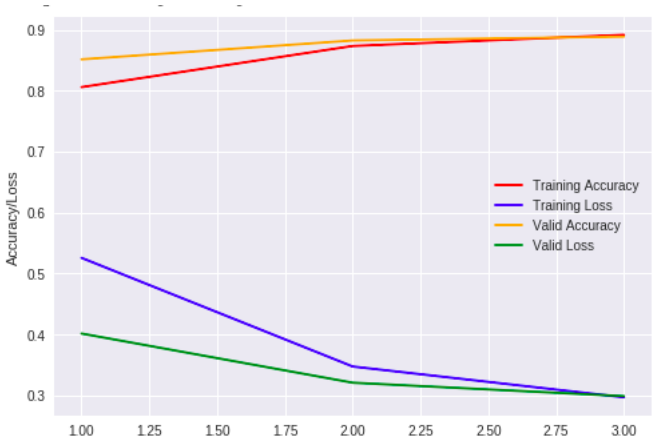
The test set accuracy achieved is 89.49%. We show the plot for train/validation loss and accuracy below.



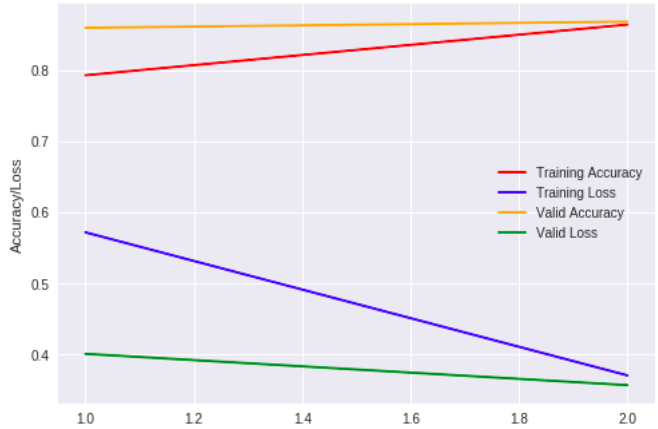
(g) Plot the training and validation curves (loss and accuracy) for different batch sizes. Try 8, 16, 32, 64. Include the plots in your report and comment on your results.

Solution:

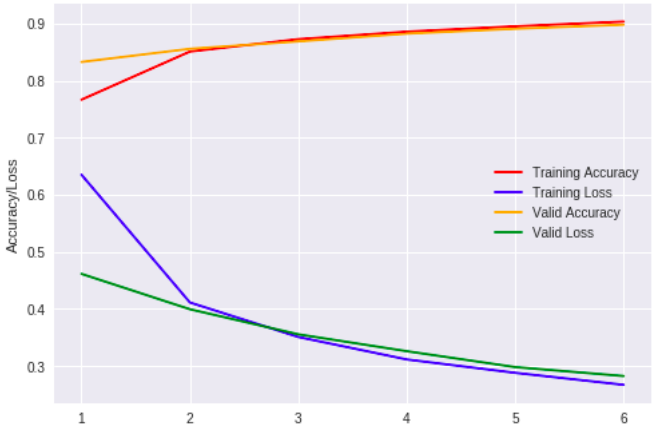
Show the plot with batch size equals to 8.



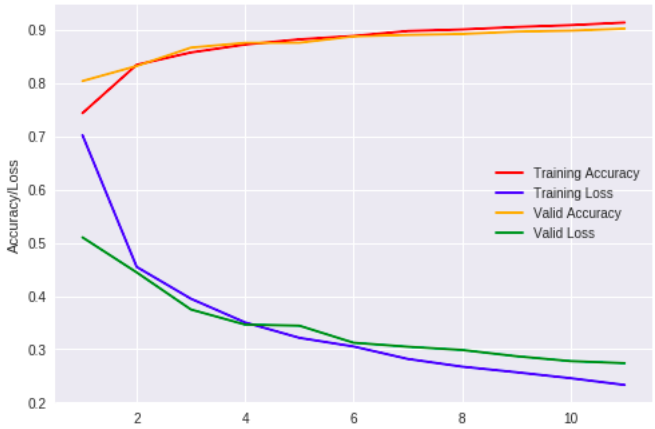
Show the plot with batch size equals to 16.



Show the plot with batch size equals to 32.



Show the plot with batch size equals to 64.



The batch size defines the number of samples that will be propagated through the network. Based on the plots, the larger the batch size, the more iterations to train the neural network. Also, the larger the batch size, the less the train/validation loss and the larger the train/validation accuracy.