Cheryl Ngo
11/24/2019

2. This network uses 3 convolutional layers, 2 max pooling layers, and 2 dense layers.
    a. I started by separating my data into training and test data. I also normalized the data by dividing by 255 and subtracting .5. My first approach involved three convolutional layers, two max pooling layers, one layer to flatten, two dense layers with relu activation and one dense layer with softmax activation. I used adam as the optimizer with categorical crossentropy as the loss and accuracy as the metric. The units in each convolutional layer was 32, 64, and 128 units with 512 and 128 units in the dense layers. The model showed overfitting after only a few epochs. I experimented with putting different units in one of each of the convolutional layers and did not achieve better metrics. I tried putting comparatively few hidden units in the first to dense layers (10-40) which resulted in an accuracy around 85%. Then, I changed the number of nodes in my first dense layer to 448 and 112 in the second dense layer. The resulting graphs showed slightly improved accuracy.

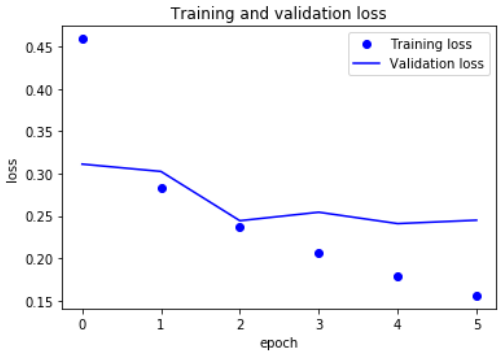
Layer (type)	Output Shape	Param #
conv2d_264 (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d_255 (MaxPoolin	(None, 13, 13, 32)	0
conv2d_265 (Conv2D)	(None, 13, 13, 64)	18496
max_pooling2d_256 (MaxPoolin	(None, 6, 6, 64)	0
conv2d_266 (Conv2D)	(None, 6, 6, 128)	73856
max_pooling2d_257 (MaxPoolin	(None, 3, 3, 128)	0
flatten_87 (Flatten)	(None, 1152)	0
dense_179 (Dense)	(None, 448)	516544
dense_180 (Dense)	(None, 112)	50288
dense_181 (Dense)	(None, 10)	1130

Total params: 660,634
Trainable params: 660,634
Non-trainable params: 0

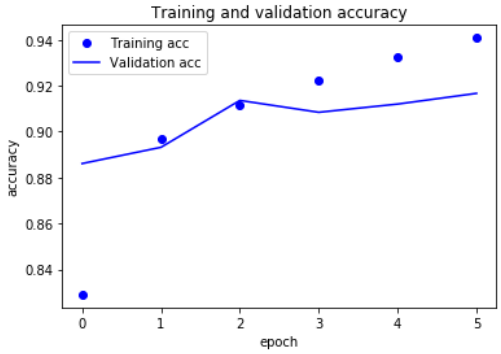
    b. Graphs

Cheryl Ngo
11/24/2019

i.



ii.



c. This model had a problem with overfitting even when training through a minimal number of epochs due, in part, to the many layers. Most models that I used had a test accuracy of 80-90%. My final model resulted in a test accuracy of 91.23%. It took 2 minutes and 51 seconds to train the model.

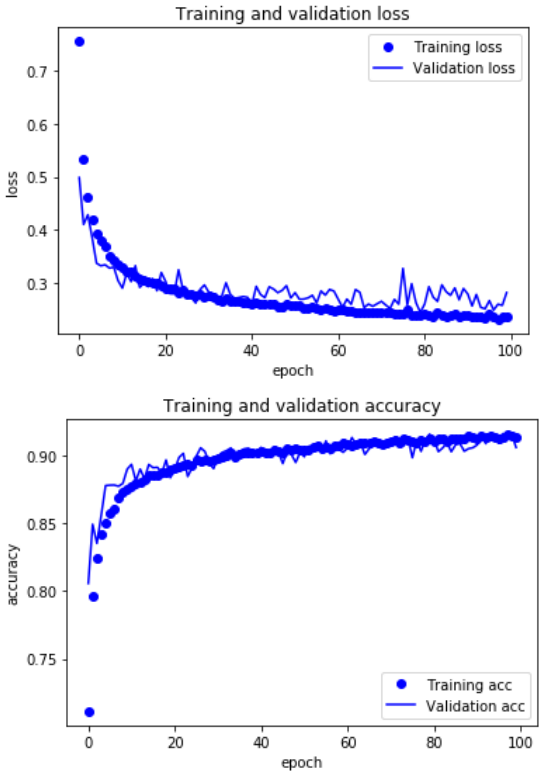
3. This network uses data preprocessing to help the model generalize.
   a. The data augmentation model used the same structure as the first model but added a step with ImageDataGenerator. I also used the image generator to add a reshape step which divided the data by 255. I started with a rotation range of 20 and .1 for all other values and then tweaked the values until I reached my final choice of 25 for the rotation range and .15 for all other values trained through 100 epochs.

Layer (type)	Output Shape	Param #
conv2d_345 (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d_336 (MaxPoolin	(None, 13, 13, 32)	0
conv2d_346 (Conv2D)	(None, 13, 13, 64)	18496
max_pooling2d_337 (MaxPoolin	(None, 6, 6, 64)	0
conv2d_347 (Conv2D)	(None, 6, 6, 128)	73856
max_pooling2d_338 (MaxPoolin	(None, 3, 3, 128)	0

flatten_115 (Flatten)	(None, 1152)	0
dense_263 (Dense)	(None, 448)	516544
dense_264 (Dense)	(None, 112)	50288
dense_265 (Dense)	(None, 10)	1130

Total params: 660,634
Trainable params: 660,634
Non-trainable params: 0

b. Graphs



c. The data augmentation process really helped the model avoid overfitting. The test and training loss and accuracy are much closer than in the original model even when being trained for 100 epochs. This is a dramatic difference from the first model. The data augmentation improves the ability of the model to generalize results from the training data and apply what it learns to the test data. It took 49 minutes and 3 seconds to train the model. The test accuracy slightly exceeded the first model at 91.54%

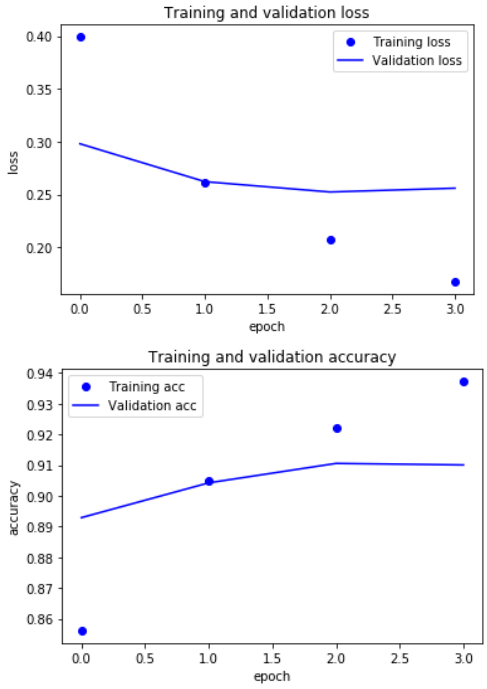
4. This network has one hidden convolutional layer.
   a. This model uses one convolutional layer with 32 units with a max pooling layer after. The dense layers have the same specifications as the first model. I tried reducing the nodes in the densely connected layers to 448 and 112 but found that

this resulted in a slightly lower accuracy, so I decided to keep the number of nodes from the first model and train through 4 epochs.

Layer (type)	Output Shape	Param #
conv2d_53 (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d_53 (MaxPooling	(None, 13, 13, 32)	0
flatten_33 (Flatten)	(None, 5408)	0
dense_64 (Dense)	(None, 26)	140634
dense_65 (Dense)	(None, 10)	270

Total params: 141,224
Trainable params: 141,224
Non-trainable params: 0

b. Graphs



c. This model performed similar to my original model. My final one-convolutional-layer model resulted in 90.81% accuracy on the test data and took 1 minute and 57 seconds to train. Depth is an important factor in the model as one convolutional layer was not able to learn as much as the three convolutional layers; however, having the three layers did also result in easy overfitting. While this was not an improvement over the original model, the decrease in processing

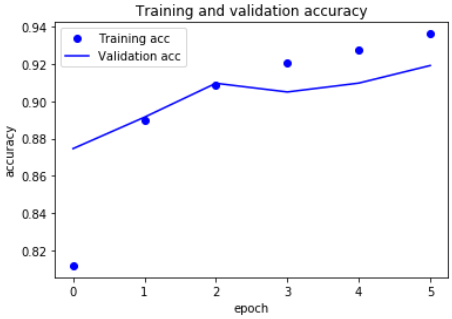
time could make this model a serious contender for certain applications where decent accuracy is needed from a less expensive model.

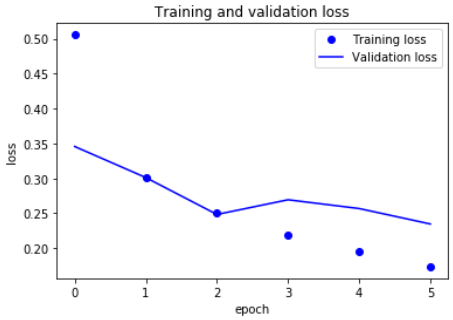
5. This network adds another convolutional layer and another max pooling layer
   a. I added another convolutional layer with 128 units and a max pooling layer to the original model. I trained the model through 6 epochs.

Layer (type)	Output Shape	Param #
conv2d_302 (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d_293 (MaxPoolin	(None, 13, 13, 32)	0
conv2d_303 (Conv2D)	(None, 13, 13, 64)	18496
max_pooling2d_294 (MaxPoolin	(None, 6, 6, 64)	0
conv2d_304 (Conv2D)	(None, 6, 6, 128)	73856
max_pooling2d_295 (MaxPoolin	(None, 3, 3, 128)	0
conv2d_305 (Conv2D)	(None, 3, 3, 128)	147584
max_pooling2d_296 (MaxPoolin	(None, 1, 1, 128)	0
flatten_101 (Flatten)	(None, 128)	0
dense_221 (Dense)	(None, 448)	57792
dense_222 (Dense)	(None, 112)	50288
dense_223 (Dense)	(None, 10)	1130

Total params: 349,466
Trainable params: 349,466
Non-trainable params: 0

   b. Graphs





c. This model performed slightly better to my original model at 91.61% accuracy. It took 2 minutes and 55 seconds to train which is slightly longer than the original model. Overall, I would not consider this to be a significant improvement over the first model.

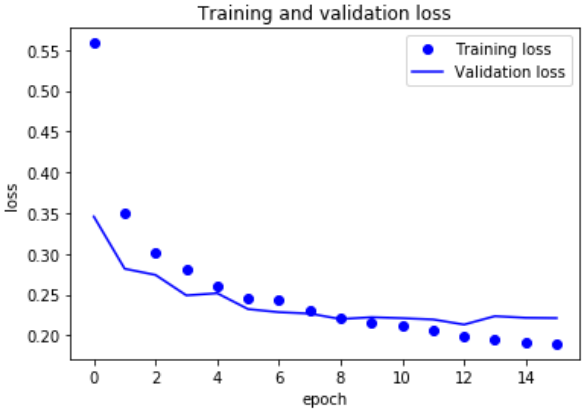
6. This network adds dropout to reduce overfitting.
   a. For this network, I used the same specifications as the three-layer model, but I added a dropout layer with a rate of .2 after each max pooling layer and the first two dense layers. This model was trained through 15 epochs.

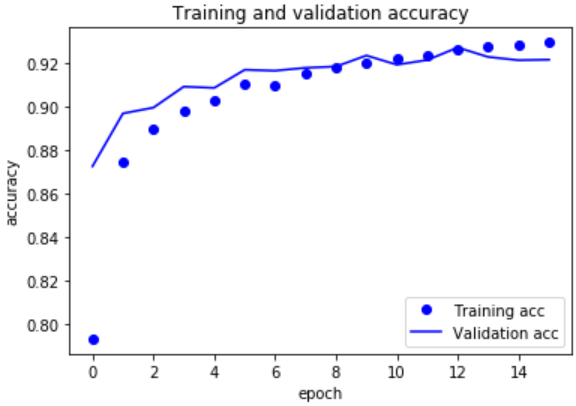
Layer (type)	Output Shape	Param #
conv2d_249 (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d_240 (MaxPoolin	(None, 13, 13, 32)	0
dropout_38 (Dropout)	(None, 13, 13, 32)	0
conv2d_250 (Conv2D)	(None, 13, 13, 64)	18496
max_pooling2d_241 (MaxPoolin	(None, 6, 6, 64)	0
dropout_39 (Dropout)	(None, 6, 6, 64)	0
conv2d_251 (Conv2D)	(None, 6, 6, 128)	73856
max_pooling2d_242 (MaxPoolin	(None, 3, 3, 128)	0
dropout_40 (Dropout)	(None, 3, 3, 128)	0
flatten_82 (Flatten)	(None, 1152)	0
dense_164 (Dense)	(None, 448)	516544
dropout_41 (Dropout)	(None, 448)	0
dense_165 (Dense)	(None, 112)	50288

```
dropout_42 (Dropout)         (None, 112)               0
_________________________________________________________________
dense_166 (Dense)            (None, 10)                1130
=================================================================
Total params: 660,634
Trainable params: 660,634
Non-trainable params: 0
```

Graphs

b.





c. I tried various quantities for the dropout rate but found that a consistent .2 worked well. I thought that varying the dropout rate throughout the model might be useful for maintaining more control over the network, but this method did not break 91% accuracy. Simply adding the dropout layer at rate .2 and training for 16 epochs resulted in increased accuracy to 92.01%. This model took 7 minutes and 28 seconds to train. Dropout is very important as it helped mitigate overfitting which is evident in the graphs of the three convolutional layer model and the dropout model when trained for an extended number of epochs. The three-layer without dropout began overfitting after only a handful of epochs, and using dropout resulted in the highest test accuracy of this experiment.

7. For this problem, changing the number of layers in the model had a small impact on the accuracy of the model. Some techniques to manage overfitting and increase generalization are data augmentation and dropout—both of which made improvements upon my original model in exchange for increased processing time. I'm confident that the models including data augmentation and dropout are more robust than the others

because they are able to train for more epochs and spend more time effectively learning. For further testing, I would try a model that includes both dropout and data augmentation with three convolutional layer model.

8. Appendix: Code

```
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Code Flow
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
'''
The true/false variables in the Variables Used section are to be manipulated
to determine what type of model should be trained. For each model, the data
loaded and split into training and test data. The image shape is saved for
use as parameters in the model. According to the true/false variables set,
the parameters number of epochs for which the model is trained is varied.
The data is reshaped to be represented as sparse, binary vectors.  After this,
the data may be augmented.

Depending on the true/false parameters specified, the convolutional model is
built with one or many convolutional and max pooling layers and possible
dropout followed by two dense layers with relu activation and a final dense
layer with 10 units and softmax activation.

The model is then trained for the specified number of epochs and the
results in terms of the loss function and accuracy are plotted on a graph
for both the training and test data.  The start and stop time for the model
training are recorded and the duration is printed along with the final test
loss and test accuracy.
'''
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Import Libraries Section
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
import matplotlib.pyplot as plt
from keras import models
from keras import layers
from keras.layers import Dropout
import datetime
from keras.utils import to_categorical

from keras.preprocessing.image import ImageDataGenerator
import numpy as np

""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Variables Used
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
#Set as True if the model should include data augmentation
dataAugmentation=True
#Set as True if the model has at least three convolutional layers (no dropout)
threeConvLayer=True
#Set as True if the model has four convolutional layers (no dropout)
```

```
fourConvLayer=False
#Set as True if the model has dropout (three convolutional layers)
dropOut=False

'''
num_nodes (first and second) are used to set the number of hidden units in the
    two convolutional layers
num_epochs is used to determine how many epochs the model should be trained for
size_batch is used to set the number of samples used to train the network
    at a time
verbose_setting is used to generate output that allows the user to visualize
    the progress of the training
num_classes is used to signify that there are 10 classes in the data

train/test_images are used to store the normalized input vector
train/test_labels are used to store the model factors
train/test_number_images, train/test_x_image_size, and
    train/test_y_image_size are used in the convolutional layers to specify
    the shape of the inputs
train_datagen is used to perform data augmentation

acc, val_acc, loss, and val_loss are used to store the results of the model
    training for use in a plot to compare loss and metrics for the training and
    test data
'''
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Objective
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
'''The goal of this program is to create a ConvNet that accurately classifies
data trained from the MNIST fashion dataset.  The code is split into
sections and subtasks are commented out and classified in the applicable
section
'''
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Load Data Section
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
#Set a seed to allow for replication
np.random.seed(7)

from keras.datasets import fashion_mnist
#Split into test and training data
(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()

#Capture the time to allow us to calculate training duration
start_time = datetime.datetime.now()
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
```

Cheryl Ngo
11/24/2019

Parameters Section

```
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
#Set a variable to represent the first dimension--number of images
train_number_images = train_images.shape[0]
test_number_images = test_images.shape[0]

#Set a variable to represent the second dimension--pixel value length
train_x_image_size = train_images.shape[1]
train_y_image_size = train_images.shape[2]

#Set a variable to represent the third dimension--pixel value width
test_x_image_size = test_images.shape[1]
test_y_image_size = test_images.shape[2]

#Set the number of hidden units in the first dense layer
num_nodes_first = 448
#Set the number of hidden units in the second dense layer
num_nodes_second=112 #128

#Train the dropout model through 16 epochs
if dropOut:
   num_epochs=16
#Train the data augmentation model through 8 epochs
elif dataAugmentation:
   num_epochs=100
#Train the four convolutional layer model through 6 epochs
elif fourConvLayer:
   num_epochs=6
#Train the three convolutional layer model through 6 epochs
elif threeConvLayer:
   num_epochs=6 #Four conv layer trained through 15 epochs
#Train the one convolutional layer model through 4 epochs
else:
   num_epochs=4

#Set the number of samples processed before the model is updated as 30
size_batch = 30

#Return a visual progress indicator
verbose_setting = 1

#Set the number of classes as 10 since there are 10 types of clothing
num_classes = 10
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
```

Pretreat Data Section

```
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
```

```
#Reshape the training images
train_images = train_images.reshape((train_number_images, train_x_image_size ,
train_y_image_size, 1))
#Convert training images to a float
train_images = (train_images.astype('float32'))

#Reshape the test images
test_images = test_images.reshape((test_number_images, test_x_image_size ,
test_y_image_size, 1))
#Convert test images to a float
test_images = (test_images.astype('float32'))

#Use one hot encoding to transform the integer into a catagorical variable (binary vector)
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)

#Perform if the model is data augmentation
if dataAugmentation:
   #Create a data generator for the training data
   train_datagen = ImageDataGenerator(
      #Rescale the images
      rescale=1./255,
      #Include copies of the images rotated by up to 40 degrees
      rotation_range=25,
      #Shift the image horizontally by up to 25% of width
      width_shift_range=0.15,
      #Shift the image vertically by up to 25% of width
      height_shift_range=0.15,
      #Set the shear angle intensity
      shear_range=0.1,
      #Set the amount of zoom
      zoom_range=(.9,1.1),
      horizontal_flip=True,)
   # The validation data does not get augmented
   test_datagen = ImageDataGenerator(rescale=1./255)

   #Augment the training data
   train_generator = train_datagen.flow(
         train_images, train_labels,
         batch_size=32)
   #Do not augment the validation data
   validation_generator = test_datagen.flow(
         test_images, test_labels,
         batch_size=32)

else:
```

```
    #Rescale test and training images to [0,1]
    train_images = (train_images / 255)-.5
    test_images = (test_images/ 255)-.5

""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Define Model Section
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
#Start the sequential model
network = models.Sequential()

#Add a convolutional layer of 32 hidden units with relu activation
network.add(layers.Conv2D(32, (3, 3), activation='relu',
input_shape=(train_x_image_size , train_y_image_size, 1)))
#Add a max pooling layer to reduce dimensionality
network.add(layers.MaxPooling2D((2, 2)))
#Add dropout to reduce overfitting
if dropOut:
    network.add(Dropout(0.2))

if threeConvLayer:
    #Add a convolutional layer of 64 hidden units with relu activation
    network.add(layers.Conv2D(64, (3, 3), activation='relu',
input_shape=(train_x_image_size , train_y_image_size, 1),padding='same'))
    #Add a max pooling layer to reduce dimensionality
    network.add(layers.MaxPooling2D((2, 2)))
#Add dropout to reduce overfitting
    if dropOut:
        network.add(Dropout(0.2))

    #Add a convolutional layer of 128 hidden units with relu activation
    network.add(layers.Conv2D(128, (3, 3), activation='relu',
input_shape=(train_x_image_size , train_y_image_size, 1),padding='same'))
    #Add a max pooling layer to reduce dimensionality
    network.add(layers.MaxPooling2D((2, 2)))
#Add dropout to reduce overfitting
    if dropOut:
        network.add(Dropout(0.2))

if fourConvLayer:
    #Add a convolutional layer of 128 hidden units with relu activation
    network.add(layers.Conv2D(128, (3, 3), activation='relu',
input_shape=(train_x_image_size , train_y_image_size, 1),padding='same'))
    #Add a max pooling layer to reduce dimensionality
    network.add(layers.MaxPooling2D((2, 2)))

#Reshape the tensor to one dimension
```

```
network.add(layers.Flatten())

#Add a dense layer of num_nodes_first hidden units with relu activation
network.add(layers.Dense(num_nodes_first, activation='relu'))
#Add dropout to reduce overfitting
if dropOut:
   network.add(Dropout(0.2))

#Add a dense layer of num_nodes_second hidden units with relu activation
network.add(layers.Dense(num_nodes_second, activation='relu'))
if dropOut:
   network.add(Dropout(0.2))

#Add the final hidden layer used to classify the images in one of num_classes classes
network.add(layers.Dense(num_classes, activation='softmax'))

#Set the compiler to have the adam optimizer with categorical crossentropy loss and
accuracy as the metric
network.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

#View a detail summary of the model structure
network.summary()

""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Fit Model Section
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
#Use fit_generator to train the model if data augmentation was used
if dataAugmentation:
   history=network.fit_generator(train_generator, steps_per_epoch=len(train_images)/32,
epochs=num_epochs, validation_data=validation_generator,
validation_steps=len(test_images)/32)
#Use .fit to train the model
else:
   history = network.fit(train_images, train_labels, epochs=num_epochs,
batch_size=size_batch, verbose = verbose_setting, validation_split=0.2)
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
Show output Section
""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
#Save test loss and accuracy from the non-data augmentation results
if not dataAugmentation:
   test_loss, test_acc = network.evaluate(test_images, test_labels)
   #Print the test loss
   print('test_loss:', test_loss)
   #Print the test accuracy
```

```
    print('test_acc:', test_acc)

#Save the stop time to calculate the run time
stop_time = datetime.datetime.now()
#Print the run time
print ("Time required for training:",stop_time - start_time)

#Save the test accuracy of the model per epoch to plot
acc = history.history['acc']
#Save the validation accuracy of the model per epoch to plot
val_acc = history.history['val_acc']
#Save the test loss of the model per epoch to plot
loss = history.history['loss']
#Save the validation loss of the model per epoch to plot
val_loss = history.history['val_loss']
#Save the number of epochs for use in our metric plot
epochs = range(len(acc))

#Create a plot of training accuracy
plt.plot(epochs, acc, 'bo', label='Training acc')
#Add to the plot validation accuracy
plt.plot(epochs, val_acc, 'b', label='Validation acc')
#Set a title for the plot
plt.title('Training and validation accuracy')
#Set a y-axis label for the plot
plt.ylabel('accuracy')
#Set an x-axis label for the plot
plt.xlabel('epoch')
#Put a legend on the plot
plt.legend()

#Create figure
plt.figure()
#Create a plot of training loss
plt.plot(epochs, loss, 'bo', label='Training loss')
#Add to the plot validation loss
plt.plot(epochs, val_loss, 'b', label='Validation loss')
#Set a title for the plot
plt.title('Training and validation loss')
#Set a y-axis label for the plot
plt.ylabel('loss')
#Set an x-axis label for the plot
plt.xlabel('epoch')
#Put a legend on the plot
plt.legend()
```

```
#Show the plots
plt.show()
```