Thomas Courtney

Artificial Intelligence

Professor Blossom

26 September 2021

Convolutional Neural Networks Assignment

Summary

The assignment called to produce a convolutional neural network using the Keras framework. In this assignment, we look atuse the fashion\_mnist dataset from keras. This dataset contains 60,000 images that are contained in 10 different categories.

Introduction

Convolutional neural networks are one of the newest artificial intelligence frameworks. The framework is mostly used to analyze visual imagery, as produced in the following code. Convolutional neural networks are used commonly in our everyday life. Consider image recognition, image analysis, object detection or face recognition.

For this assignment, we looked at a dataset containing images of different clothing items. For our assignment, the images could be labeled as nine different frequently used clothing items, from t-shirts to ankle boots. In the code below, the user classified all of these items into their corresponding clothing categories.

Body

In the code above, there are three convolutional layers and one fully connected layer. In our final model. Running the model with more convolutional layers adds dimensionality to the model, allowing them to solve complex problems. Removing the connected layer greatly decreases the accuracy of the model, so it is important to have at least one connected layer. Adding more layers may result in overfitting. Below are the summary results of the model.

import pandas as pd

import tensorflow as tf

import tensorflow.keras,os

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

from tensorflow.keras.utils import to\_categorical

from matplotlib import pyplot

from tensorflow.keras.applications.vgg19 import VGG19

import numpy as np

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras import models

from tensorflow.keras import layers

from tensorflow.keras import optimizers

from keras.layers import Dense, Dropout, Activation, Flatten

from keras.layers import Conv2D, MaxPooling2D

import matplotlib.pyplot as plt

conv\_base = VGG19(weights='imagenet', include\_top=False, input\_shape=(150, 150, 3))

import datetime

import keras

import pandas as pd

""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""" Parameters Section

"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""

#defining our global batch size, number of classes, and number of epochs you #want to run in the model

batch\_size = 64

num\_classes = 10

epochs = 30

#tell the program to run the different functions

data\_augmentation = False

#where we save the trainied model, and what we call it.

save\_dir = os.path.join(os.getcwd(), 'saved\_models')

model\_name = 'Courtney\_M4'

# initiate RMSprop optimizer

opt = tf.keras.optimizers.RMSprop(lr=0.0001, decay=1e-6)

""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""" Load Data Section

"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""

# load data, split between train and test sets:

fashion\_data = tf.keras.datasets.fashion\_mnist

(x\_train, y\_train), (x\_test, y\_test) = fashion\_data.load\_data()

#assert x\_train.shape == (20000, 28, 28)

#assert x\_test.shape == (3300, 28, 28)

#assert y\_train.shape == (20000)

#assert y\_test.shape == (3300)

#print training and test independent variables shape

print('x\_train shape:', x\_train.shape)

print('x\_test shape:', x\_test.shape)

print(x\_train.shape[0], 'train samples')

print(x\_test.shape[0], 'test samples')

#print the pictures

r,c=5,5

fig=plt.figure(figsize=(5,5))

for i in range(r\*c):

fig.add\_subplot(r,c,i+1)

plt.imshow(x\_train[i],cmap='Greys\_r')

plt.axis('off')

plt.title(y\_train[i])

# for Paul Blossom's pc

import tensorflow as tf

gpus = tf.config.list\_physical\_devices('GPU')

if gpus:

try:

# Currently, memory growth needs to be the same across GPUs

for gpu in gpus:

tf.config.experimental.set\_memory\_growth(gpu, False)

logical\_gpus = tf.config.experimental.list\_logical\_devices('GPU')

print(len(gpus), "Physical GPUs,", len(logical\_gpus), "Logical GPUs")

except RuntimeError as e:

# Memory growth must be set before GPUs have been initialized

print(e)

# for Paul Blossom's pc

""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""" Pretreat Data Section

"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""

# Convert the vectors into the matrcies

y\_train = tf.keras.utils.to\_categorical(y\_train)

y\_test = tf.keras.utils.to\_categorical(y\_test)

#scale to 0-1

x\_train = x\_train / x\_train.max()

x\_test = x\_test / x\_test.max()

#print the values

print(y\_train)

print(x\_train)

print(y\_test)

print(x\_test)

""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""" Define Model Section

"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""

#see how long the model takes to run

start\_time = datetime.datetime.now()

#initialize the model

model = Sequential()

#set the input layer shape

model.add(tf.keras.layers.InputLayer(input\_shape=(28,28,1)))

model.add(Conv2D(32, (3, 3),

padding='same')) #add the 1st convolutional layer

model.add(Activation('relu')) #set the 1st conv. layer activation function

model.add(MaxPooling2D(pool\_size=(2, 2))) #add the 1st Max Pooling Layer

#model.add(Dropout(.5)) #add dropout to mitigate overfitting

model.add(Conv2D(64, (3,3),

padding='same')) #add the 2nd convolutional layer

model.add(Activation('relu')) #set the 2nd conv. layer activation function

model.add(MaxPooling2D(pool\_size=(2, 2))) #add the 2nd Max Pooling Layer

#model.add(Dropout(.25))

model.add(Conv2D(128, (3,3),

padding='same')) #add the 3rd convolutional layer

model.add(Activation('relu')) #set the 3rd conv. layer activation function

model.add(MaxPooling2D(pool\_size=(2, 2))) #add the 3rd Max Pooling Layer

#model.add(Dropout(.5))

model.add(Flatten()) #Add a Flatten Layer to smooth model

model.add(Dense(128)) #adding a hidden layer to help produce higher acc

model.add(Activation('relu')) #adding the relu activation func to the hidden layer

#model.add(Dropout(.5))

model.add(Dense(64)) #adding a hidden layer to help produce higher acc

model.add(Activation('relu')) #adding the relu activation func to the hidden layer

model.add(Dense(num\_classes)) #initialize the fully connected layer

model.add(Activation('softmax')) #set the connected layer act funtion to softmax

model.summary() #look at the model summary

#reshape model to only include greyscale images

x\_train = x\_train.reshape(x\_train.shape[0], 28, 28, 1)

x\_test = x\_test.reshape(x\_test.shape[0], 28, 28 ,1)

#compile the model with categorical crossentropy loss function, #optimizer will use the RMSprop, and we want to track the accuracy metric.

model.compile(loss='categorical\_crossentropy',

optimizer=opt,

metrics=['acc'])

""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""" Train Model Section

"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""

#use this model when data augmentation is set to false

if not data\_augmentation:

print('Not using data augmentation.')

# Fit the model

history = model.fit(x\_train, y\_train,

batch\_size=batch\_size,

epochs=epochs,

#define the test data

validation\_data=(x\_test, y\_test),

shuffle=True)

#build the model when data augmentation is set to true

else:

print('Using real-time data augmentation.')

# This will preprocess the data in real time

datagen = ImageDataGenerator(

featurewise\_center=False, #make the input mean zero

samplewise\_center=False, # set each sample mean to 0

featurewise\_std\_normalization=False, # divide the inputs and the standard dev of the dataset

samplewise\_std\_normalization=False, # divide each input by its standard dev

zca\_whitening=False, # apply ZCA Whitening

zca\_epsilon=1e-06, #ZCA whitening epsilon

rotation\_range=0, # randomly rotate images between zero and 180 degrees

# random shift of images horizontally

width\_shift\_range=0.1,

# random shift of images vertically

height\_shift\_range=0.1,

shear\_range=0., #range for random shear

zoom\_range=0., #range for random zoom

channel\_shift\_range=0., #range for random channel shifts

#mode for points outside the input boundaries

fill\_mode='nearest',

cval=0., # constant value for the fill mode

horizontal\_flip=True, # horizontally flip images at random

vertical\_flip=False, # vertically flip images at random

# set rescaling factor

rescale=None,

# set the function that could be applied at each input

preprocessing\_function=None,

# format the image

data\_format=None,

# fraction of images reserved for validation

validation\_split=0.0)

# Compute quantities required for feature-wise normalization

datagen.fit(x\_train)

# Fit the model on based on datagen.flow() results

history = model.fit\_generator(datagen.flow(x\_train, y\_train, batch\_size=batch\_size),

epochs=epochs, steps\_per\_epoch=100,

validation\_data=(x\_test, y\_test),

workers=4)

#Save model and weights

if not os.path.isdir(save\_dir):

os.makedirs(save\_dir)

model\_path = os.path.join(save\_dir, model\_name)

model.save(model\_path)

print('Saved trained model at %s ' % model\_path)

"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""

Show output Section

"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""

#stop the timer and print length

stop\_time = datetime.datetime.now()

print ("Required Time For Training:",stop\_time - start\_time)

# print the score of the trained model

scores = model.evaluate(x\_test, y\_test, verbose=1)

print('Test loss:', scores[0])

print('Test accuracy:', scores[1])

#define the variables for each metric

acc = history.history['acc']

val\_acc = history.history['val\_acc']

loss = history.history['loss']

val\_loss = history.history['val\_loss']

#show the number of epochs

epochs = range(len(acc))

#plot the accuracy of both the train and the test dataset

plt.plot(epochs, acc, 'green', label='Training acc')

plt.plot(epochs, val\_acc, 'blue', label='Validation acc')

plt.title('Training and validation accuracy')

plt.legend()

plt.figure()

#plot the loss of both the train and test dataset

plt.plot(epochs, loss, 'green', label='Training loss')

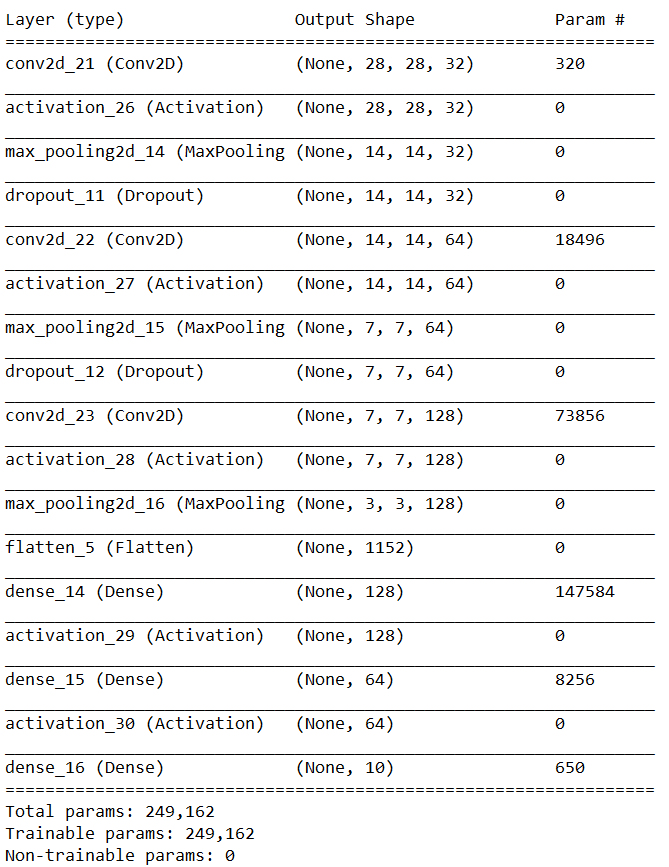
plt.plot(epochs, val\_loss, 'blue', label='Validation loss')

plt.title('Training and validation loss')

plt.legend()

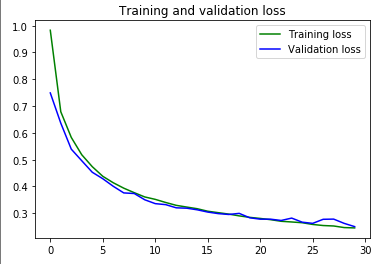
plt.show()

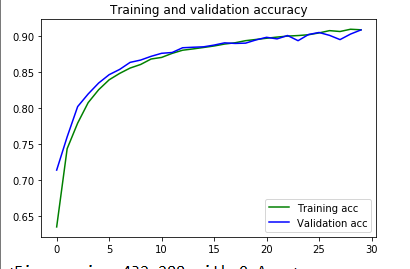
#there is a slight change in performance



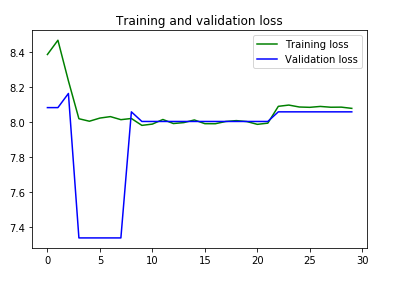
Adding more layers typically results in increased accuracy, and adding additional hidden layers will produce higher accuracy. Users should be wery that they are overfitting their data, so it is important not to add too many layers and it may result in greater bias over time. In the charts below you can see the training and test loss vs training iteration and the classification accuracy. By removing the connected layer, you will see that the accuracy greatly decreases.

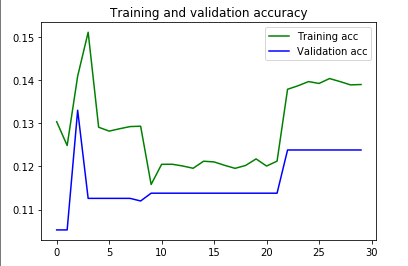
With Connected Layer





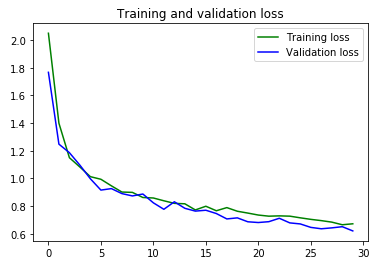
Without Connected Layer

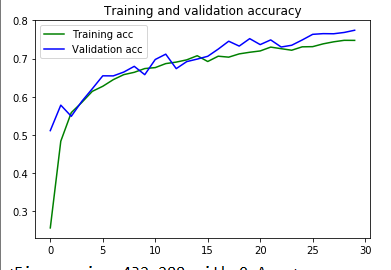




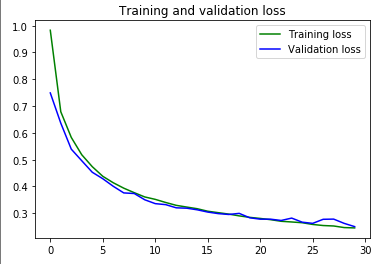
Preprocessing the input by either doing data augmentation on the data or not greatly changes your model. Data augmentation is a data analysis technique that will increase the amount of data by adding modified copies to already existing data. Adding data augmentation acts as a regularizer and helps reduce overfitting. In the charts below you can see the difference between including data augmentation or not. When we use data augmentation, the model’s test loss is 0.6205245852470398 while the test accuracy is 0.7742000222206116. The model takes only two minutes to train. Without using data augmentation, there is a decreased test loss (0.25064441561698914) and an increased test accuracy (0.9063000082969666) but the model took 13 minutes to run. The user is determined to use data augmentation because of the speed.

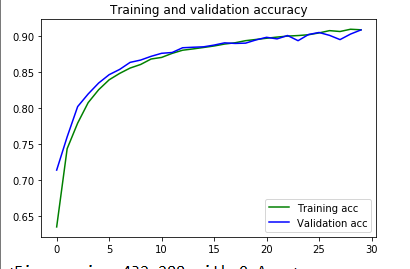
Using Data Augmentation



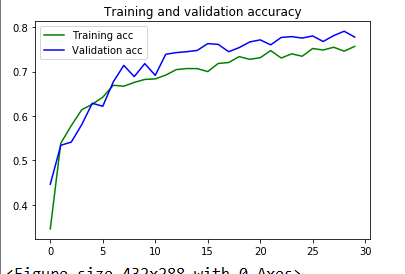


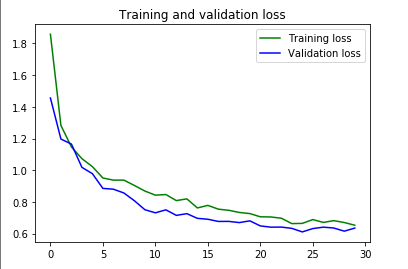
Without Data Augmentation



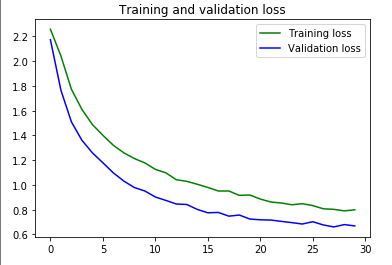
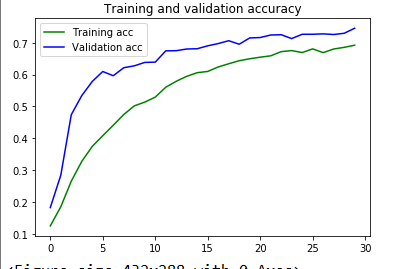


Experimenting with network structure is important. In the final model, we have multiple hidden layers. Choosing the correct amount of layers is vital to the success of the model. Having too few layers will create models that are not sufficiently correct, while having too many may cause overfitting. In the examples below you can see the difference between one hidden layer, a fourth network, and a fifth network. Adding layers certainly will increase the accuracy, but there is fear of overfitting with so many additional layers. Moving forward, I would recommend the user to decrease the amount of models needed. Using only one layer, there is a test accuracy of 0.685699999332428 and a test loss of 0.7846856713294983. This model only took two minutes to run using data augmentation. Dropping down to one layer had some effect on the accuracy of the model in this case. This was also run while using data augmentation, which may be a reason why it does not affect the model that greatly. Compare the visualizations below to previous ones to show that dropping layers has a negative effect on the accuracy of the model.

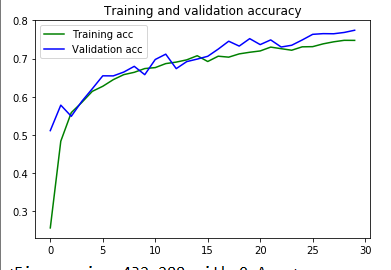


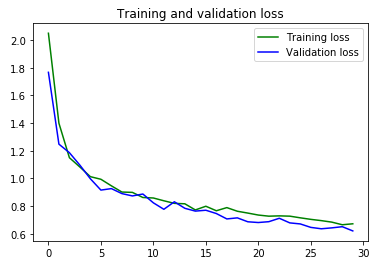


Next we added an additional layer to make a total of four layers. The accuracy ended up being 0.7293000221252441 and the loss was 0.6688282489776611. This added additional time to running the model as well. Since there was little effect on the accuracy and loss, we did not see it valuable to add additional layers. Compared to previous models, adding layers did little to increase the accuracy of the model, while adding a lot of additional time as well.

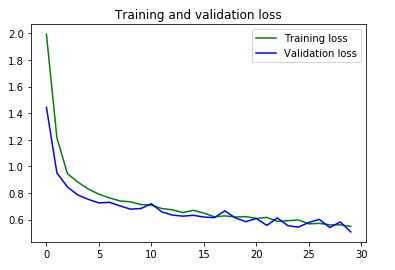
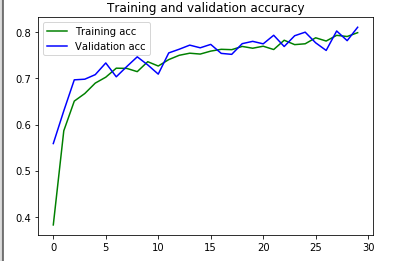


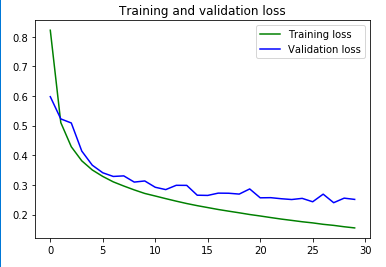
Lastly, we changed the number of drop offs to change the network results. Adding drop will create better performance when used on a larger network, giving the model more of an opportunity to learn independent representations. While using dropouts and data augmentation, the model’s test loss is 0.6205245852470398 while the test accuracy is 0.7742000222206116. The model takes only two minutes to train. Without dropouts it decreased the test loss to 0.5072107315063477 and a test accuracy of 0.8109999895095825. The model did run faster by around 14 seconds

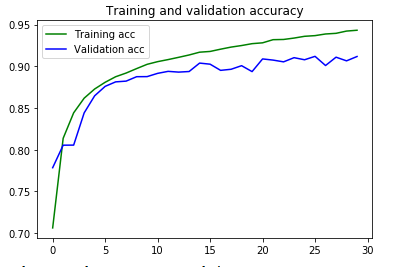
With three dropouts



With no dropouts



In out final model, we decided to not use data augmentation, as our accuracy increased. Additionally, we used the three drop out layers and three augmentation layers. This gave us a result of 0.9118000268936157 test accuracy and a test loss of 0.2510334849357605. This was the highest accuracy and the lowest test loss we found while testing our models. The only reason we considered not to choose this model was because of how long it took. The only reason we considered not to choose this model was because of how long it took. The model takes almost fourteen minutes to run. In our case however, there was no speed element or time crunch, so we decided on the model with the code in an earlier part of this section. The results of the final model are below.



Overall, it is important to tune models over time to get the best model possible. It is also important to remember that the best model possible is not always the most accurate. You can have great models on efficiency (how quickly they derive an answer) as in other real world scenarios speed with less accuracy may be more important. It is also important to remember that tuning models takes time, and it is hard to be certain that you have the best model. Overall, the final model with the code above allows for these results, which are accurate, but are not concerned with overfitting.

Conclusion

Overall, convolutional neural networks make the classification process simple and can speed up processes. This however takes a great amount of computational power to run the model. For the code attached, the user did not have enough graphics space to properly run the models. To do this, the user had to decrease the image size and quality so the computer will run. In the real world where businesses are using computational neural networks, there is no fear that we will not have the correct amount of computational power.

Recommendations

Be cautious when using computational neural networks. No artificial intelligence platform will ever be completely accurate, so please consider before making decisions.

Appendix

The dataset was taken from Tensorflow and loaded into the code above.