Alessandro Allegranzi

BU MET CS 767

Assignment 3

5/27/2024

Table of Contents

[1. Modification of code to attempt improvement (2 page max) 2](#_Toc167383165)

[1.1 Description of changes and reason they *could reasonably be* an improvement (at most one page) 2](#_Toc167383166)

[1.2 Comparison of the result with the original output, with explanation 2](#_Toc167383167)

[1.3 URL of your Colab code 6](#_Toc167383168)

[2. Your CNN Application (3 pg max) 6](#_Toc167383170)

[2.1 Give 2-4 requirements for an application that you will implement 6](#_Toc167383171)

[2.2 Sample I/O 7](#_Toc167383173)

[2.3 The CNN Architecture 8](#_Toc167383175)

[2.3 Key code 8](#_Toc167383177)

[2.4 URL of your Colab code 8](#_Toc167383179)

[References 9](#_Toc167383181)

[Evaluation 10](#_Toc167383182)

[Appendix 1 10](#_Toc167383183)

[Appendix 2… 10](#_Toc167383184)

5/27/2024

MET CS 767 Assignment 3: CNN’s

*Alessandro Allegranzi*

Applied machine learning—after architecture selection—is largely a process of selecting parameter values. This requires an understanding of the parameters, and a systematic approach to dealing with parameter values. One purpose of this assignment is to give you experience with this. The other is to give you practice creating a deep convolutional neural network.

The overall instructions are the same as in prior assignments.

# Modification of code to attempt improvement (2 page max)

Copy the implementation [here](https://colab.research.google.com/drive/1Yg-NXKlYzfvv9jI2MCxWZErovpR57_YC?usp=sharing) to your Google drive. Systematically modify the parameter values, attempting to improve the output, and report the results as below. Since the accuracy of the given implementation is already high, consider reducing the size of the CIFAR training set—or substituting parts of it so that the baseline implementation leaves more percentage room for improvement. If it adds to clarity, describe and explain changes that make the result worse.

## 1.1 Description of changes and reason they *could reasonably be* an improvement (at most one page)

## 

**Making model less effective:**

Use a simpler activation function:

model.add(layers.Conv2D(32, (3, 3), activation='linear', input\_shape=(32, 32, 3)))

# Next layer: max over 2x2 non-overlapping squares

model.add(layers.MaxPooling2D((2, 2)))

# Typically increase kernel count with depth (to 64 here).

model.add(layers.Conv2D(64, (3, 3), activation='linear'))

model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(64, (3, 3), activation='linear'))

Increase learning rate:

model.compile(optimizer=tf.keras.optimizers.Adam(learning\_rate=0.1),

loss=tf.keras.losses.SparseCategoricalCrossentropy(from\_logits=True),

# how often predictions equal labels ("https://keras.io/api/metrics/")

metrics=['accuracy'])

history = model.fit(train\_images, train\_labels, epochs=2, #?? SET BACK TO 10

validation\_data=(test\_images, test\_labels))

The above changes lowered the model accuracy to under 40%, which has more room to improve compared to the original.

print(test\_acc)

0.37790000438690186

**Making model more effective:**

Increase number of kernels, add batch normalization, and add layers:

model.add(layers.Conv2D(64, (3, 3), activation='relu', input\_shape=(32, 32, 3)))

model.add(layers.BatchNormalization())

# Next layer: max over 2x2 non-overlapping squares

model.add(layers.MaxPooling2D((2, 2)))

# Typically increase kernel count with depth (to 64 here).

model.add(layers.Conv2D(128, (3, 3), activation='relu'))

model.add(layers.BatchNormalization())

model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(256, (3, 3), activation='relu'))

model.add(layers.BatchNormalization())

model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Flatten())

model.add(layers.Dense(64, activation='relu'))

model.add(layers.BatchNormalization())

model.add(layers.Dropout(0.5))

model.add(layers.Dense(32, activation='relu'))

model.add(layers.BatchNormalization())

model.add(layers.Dropout(0.5))

model.add(layers.Dense(10))

Model: "sequential\_8"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

conv2d\_31 (Conv2D) (None, 30, 30, 64) 1792

batch\_normalization\_26 (Ba (None, 30, 30, 64) 256

tchNormalization)

max\_pooling2d\_24 (MaxPooli (None, 15, 15, 64) 0

ng2D)

conv2d\_32 (Conv2D) (None, 13, 13, 128) 73856

batch\_normalization\_27 (Ba (None, 13, 13, 128) 512

tchNormalization)

max\_pooling2d\_25 (MaxPooli (None, 6, 6, 128) 0

ng2D)

conv2d\_33 (Conv2D) (None, 4, 4, 256) 295168

batch\_normalization\_28 (Ba (None, 4, 4, 256) 1024

tchNormalization)

max\_pooling2d\_26 (MaxPooli (None, 2, 2, 256) 0

ng2D)

flatten\_1 (Flatten) (None, 1024) 0

dense\_3 (Dense) (None, 64) 65600

batch\_normalization\_29 (Ba (None, 64) 256

tchNormalization)

dropout\_2 (Dropout) (None, 64) 0

dense\_4 (Dense) (None, 32) 2080

batch\_normalization\_30 (Ba (None, 32) 128

tchNormalization)

dropout\_3 (Dropout) (None, 32) 0

dense\_5 (Dense) (None, 10) 330

=================================================================

Total params: 441002 (1.68 MB)

Trainable params: 439914 (1.68 MB)

Non-trainable params: 1088 (4.25 KB)

Using RMSProp as the optimizer function [2], increase epochs to 8, and use early stopping and model checkpoint callback functions. Early stopping stops the training if the trigger sets off, in this case I’ve set it to stop if the validation loss stops improving for 3 epochs. Model checkpoint saves the best model in case the training is interrupted [3]:

model.compile(optimizer=tf.keras.optimizers.RMSprop(),

loss=tf.keras.losses.SparseCategoricalCrossentropy(from\_logits=True),

# how often predictions equal labels ("https://keras.io/api/metrics/")

metrics=['accuracy'])

early\_stopping = EarlyStopping(monitor='val\_loss', patience=3)

model\_checkpoint = ModelCheckpoint('best\_model.h5', monitor='val\_loss', save\_best\_only=True)

history = model.fit(train\_images, train\_labels, epochs=8, #?? SET BACK TO 10

validation\_data=(test\_images, test\_labels), callbacks=[early\_stopping, model\_checkpoint])

The accuracy improved considerably to over 74%. The above configs were the best accuracy model I could make with testing.

## 1.2 Comparison of the result with the original output, with explanation

Original Model (without modifications to make it less effective):

313/313 - 4s - loss: 1.0287 - accuracy: 0.6383 - 4s/epoch - 12ms/step

A graph with a line and a line

Description automatically generated with medium confidence

Accuracy: 0.6287000179290771

Model After Improvements:

313/313 - 10s - loss: 0.8611 - accuracy: 0.7460 - 10s/epoch - 32ms/step

A graph with a line and a line

Description automatically generated

Accuracy: 0.7459999918937683

The model stopped training after the 6th epoch due to earlyStopping, so the callback function proved useful to save resources. The model improved because the additional kernels, neurons, and layers helped the model learn more complex patterns. Batch normalization also improved results [1]. The additional training epochs also gave the model more opportunities to learn from the data. The change to using the RMSprop optimizer function may or may not have made a substantial difference. In testing the epoch and neuron increase alone achieved similar results.

Example output:

# Testing out the model.

# Preprocess the image

img\_tensor = tf.keras.preprocessing.image.img\_to\_array(test\_images[1])

img\_tensor = np.expand\_dims(img\_tensor, axis=0)

img\_tensor /= 255.

# Classify the image

predictions = model.predict(img\_tensor)

predicted\_class = np.argmax(predictions)

print(f'The model predicts that the image belongs to class {class\_names[predicted\_class]}.')

1/1 [==============================] - 0s 60ms/step

The model predicts that the image belongs to class cat.

## 1.3 URL of your Colab code

[Google Colab Code](https://colab.research.google.com/drive/1ZbQZrwNBqeinQexXPvIlcdtetL6X3eeX?usp=sharing)

### >>AI generation for section 1 (or check: I did not use AI generation here \_X\_). Please collapse this.

PARAGRAPH DESCRIBING YOUR VALUE ADDED TO AI-GENERATED MATERIAL

Your response replaces this.

YOUR PROMPT SEQUENCE

[1] Your first prompt replaces this.

[2]

Your response replaces this.

# 

# 2. Your CNN Application (3 pg max)

## 2.1 Give 2-4 requirements for an application that you will implement

## These describe *what* functionality your application will provide for the user, including the nature of inputs and outputs. This section should not include *how* you will design or code the application.

The Fashion Classifier CNN

**Requirements:**

1. **Data Loading and Preprocessing:** The application should load the Fashion MNIST dataset from Keras datasets.
2. **Model Building:** The application should build a CNN model ready to train for classification.
3. **Training:** The application should train the model using the training data.
4. **Evaluation and Prediction:** The application should evaluate the model's performance on the test data. It should also be able to take an input image, preprocess it, and use the trained model to predict the fashion item in the image.

**Input:**

The input to the application is the Fashion MNIST dataset, which consists of 60,000 28x28 grayscale images of 10 fashion categories, along with a test set of 10,000 images. Each image is represented as a 28x28 array with pixel values ranging from 0 to 255 [4].

Once the model is trained, the input will be any fashion item image converted into the 28x28 array with pixel values ranging from 0 to 255.

**Output:**

The output of the application is the classification of the input images. For each input image, the application should output a list of probabilities for the 10 fashion categories. The category with the highest probability is the model's prediction for the input image.

Once the application is trained, the output can be applied to any new fashion item image.

## 2.2 Sample I/O

## Give three varied input/outputs pairs for your implemented application.

Input: A 28x28 grayscale image of a T-shirt/top.

Output: The label 0 (T-shirt/top).

Input: A 28x28 grayscale image of a Sneaker.

Output: The label 7 (Sneaker).

Input: A 28x28 grayscale image of a Bag.

Output: The label 8 (Bag).

## 2.3 The CNN Architecture

## Show your architecture in one or more annotated figures.

The CNN architecture starts with a convolutional layer with 32 filters of size 5x5, followed by a batch normalization layer and a max pooling layer. This pattern of a convolutional layer, batch normalization, and max pooling is repeated three more times, with the number of filters in the convolutional layers increasing to 64, 64, and 128 respectively. After the convolutional layers, the model has another convolutional layer with 128 filters, followed by a flatten layer to reshape the 3D outputs to 1D. Then, there are two dense (fully connected) layers with 64 and 32 neurons respectively, each followed by batch normalization and a dropout layer with a rate of 0.5 for regularization. The model ends with a dense layer with 10 neurons and a softmax activation function because we have 10 classes we are training to identify. The 'relu' activation function is used in all convolutional and dense layers except the last one, and the 'he\_normal' initializer is used to initialize the weights. Below is a model summary.

Model: "sequential\_3"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

conv2d\_13 (Conv2D) (None, 28, 28, 32) 832

batch\_normalization\_7 (Bat (None, 28, 28, 32) 128

chNormalization)

max\_pooling2d\_11 (MaxPooli (None, 14, 14, 32) 0

ng2D)

conv2d\_14 (Conv2D) (None, 14, 14, 64) 18496

batch\_normalization\_8 (Bat (None, 14, 14, 64) 256

chNormalization)

max\_pooling2d\_12 (MaxPooli (None, 7, 7, 64) 0

ng2D)

conv2d\_15 (Conv2D) (None, 7, 7, 64) 36928

batch\_normalization\_9 (Bat (None, 7, 7, 64) 256

chNormalization)

max\_pooling2d\_13 (MaxPooli (None, 3, 3, 64) 0

ng2D)

conv2d\_16 (Conv2D) (None, 3, 3, 128) 73856

batch\_normalization\_10 (Ba (None, 3, 3, 128) 512

tchNormalization)

max\_pooling2d\_14 (MaxPooli (None, 1, 1, 128) 0

ng2D)

conv2d\_17 (Conv2D) (None, 1, 1, 128) 147584

flatten\_3 (Flatten) (None, 128) 0

dense\_9 (Dense) (None, 64) 8256

batch\_normalization\_11 (Ba (None, 64) 256

tchNormalization)

dropout\_6 (Dropout) (None, 64) 0

dense\_10 (Dense) (None, 32) 2080

batch\_normalization\_12 (Ba (None, 32) 128

tchNormalization)

dropout\_7 (Dropout) (None, 32) 0

dense\_11 (Dense) (None, 10) 330

=================================================================

Total params: 289898 (1.11 MB)

Trainable params: 289130 (1.10 MB)

Non-trainable params: 768 (3.00 KB)

## 2.3 Key code

## Provide snippets of the essential core code of your implementation.

Data setup:

# https://keras.io/api/datasets/fashion\_mnist/

(train\_images, train\_labels), (test\_images, test\_labels) = datasets.fashion\_mnist.load\_data()

# Normalize pixel values to be between 0 and 1 like we did in assignment 3 part 1.

train\_images, test\_images = train\_images / 255.0, test\_images / 255.0

Model Definition:

model = tf.keras.Sequential([

# Setting up he main pattern of convolutional layer + batch normalization

# + pooling. Starting with a larger kernel size in the first layer.

DefaultConv2D(filters=32, kernel\_size=5, input\_shape=(28, 28, 1)),

tf.keras.layers.BatchNormalization(),

tf.keras.layers.MaxPooling2D(),

DefaultConv2D(filters=64),

tf.keras.layers.BatchNormalization(),

tf.keras.layers.MaxPooling2D(),

DefaultConv2D(filters=64),

tf.keras.layers.BatchNormalization(),

tf.keras.layers.MaxPooling2D(),

DefaultConv2D(filters=128),

tf.keras.layers.BatchNormalization(),

tf.keras.layers.MaxPooling2D(),

DefaultConv2D(filters=128),

# Dense layers. Also adding in Dropout for more efficient training.

tf.keras.layers.Flatten(),

tf.keras.layers.Dense(64, activation='relu',

kernel\_initializer='he\_normal'),

tf.keras.layers.BatchNormalization(),

tf.keras.layers.Dropout(0.5),

tf.keras.layers.Dense(32, activation='relu',

kernel\_initializer='he\_normal'),

tf.keras.layers.BatchNormalization(),

tf.keras.layers.Dropout(0.5),

tf.keras.layers.Dense(10, activation='softmax')

])

Model Training:

# Compiling with RMS prop as the optimization function.

model.compile(optimizer=tf.keras.optimizers.RMSprop(),

loss=tf.keras.losses.SparseCategoricalCrossentropy(),

metrics=['accuracy'])

# Adding 2 callback functions to save the most effective model and to short circuit

# the training if epochs stop improving the model after 2 epochs.

# https://keras.io/api/callbacks/early\_stopping/

# https://keras.io/api/callbacks/model\_checkpoint/

early\_stopping = EarlyStopping(monitor='val\_loss', patience=2)

model\_checkpoint = ModelCheckpoint(FILE\_PATH, monitor='val\_loss', save\_best\_only=True)

history = model.fit(train\_images, train\_labels, epochs=6,

validation\_data=(test\_images, test\_labels), callbacks=[early\_stopping, model\_checkpoint])

The model reached over 91% accuracy on one training run, which is quite good.

313/313 - 6s - loss: 0.2860 - accuracy: 0.9132 - 6s/epoch - 19ms/step

model accuracy: 0.9132000207901001

A graph with a line

Description automatically generated

Sample output:

# Testing out the model.

# Preprocess the image

img\_tensor = tf.keras.preprocessing.image.img\_to\_array(test\_images[1])

img\_tensor = np.expand\_dims(img\_tensor, axis=0)

img\_tensor /= 255.

# Classify the image

predictions = model.predict(img\_tensor)

predicted\_class = np.argmax(predictions)

print(f'The model predicts that the image belongs to class {class\_names[predicted\_class]}.')

print(f'The unformatted result with probability for all classes {predictions}')

1/1 [==============================] - 0s 26ms/step

The model predicts that the image belongs to class Sneaker.

The unformatted result with probability for all classes [[1.06928665e-02 9.08407310e-06 1.35429460e-03 3.30683921e-04

9.03731387e-04 9.10533301e-04 1.55516504e-03 9.73287463e-01

9.50120389e-03 1.45498291e-03]]

## 2.4 URL of your Colab code

[Colab Notebook Link](https://colab.research.google.com/drive/18C9is4DnrZr9w1wJ-ml4OlicbEsEFyr8?usp=sharing)

### >>AI generation for section 2 (or check: I did not use AI generation here \_\_). Please collapse this.

PARAGRAPH DESCRIBING YOUR VALUE ADDED TO AI-GENERATED MATERIAL

Similar to assignment 2, I used an AI prompt to generate some application ideas. I chose

The fashion item suggestion.

YOUR PROMPT SEQUENCE

[1] Given the template notebook, can you recommend a few ideas for another CNN application to implement that uses datasets available in Keras? Please list 4 requirements describing the functionality and describe the nature of the input and output to the application.

AI Answer

Sure, here are a few ideas for Convolutional Neural Network (CNN) applications using datasets available in Keras:

1. **\*\*Handwritten Digit Recognition:\*\***

- **\*\*Functionality:\*\*** The application should be able to recognize handwritten digits (0-9) from images.

- **\*\*Input:\*\*** The input to the application will be images of handwritten digits. You can use the MNIST dataset available in Keras.

- **\*\*Output:\*\*** The output will be the digit (0-9) that the model predicts the image represents.

2. **\*\*Fashion Item Classification:\*\***

- **\*\*Functionality:\*\*** The application should classify images of fashion items into one of 10 categories (T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, Ankle boot).

- **\*\*Input:\*\*** The input to the application will be images of fashion items. You can use the Fashion MNIST dataset available in Keras.

- **\*\*Output:\*\*** The output will be the category that the model predicts the image belongs to.

3. **\*\*CIFAR-10 Image Classification:\*\***

- **\*\*Functionality:\*\*** The application should classify color images into one of 10 categories (airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck).

- **\*\*Input:\*\*** The input to the application will be color images. You can use the CIFAR-10 dataset available in Keras.

- **\*\*Output:\*\*** The output will be the category that the model predicts the image belongs to.

4. **\*\*Sentiment Analysis from Text:\*\***

- **\*\*Functionality:\*\*** The application should predict the sentiment (positive or negative) of a movie review.

- **\*\*Input:\*\*** The input to the application will be text of movie reviews. You can use the IMDB dataset available in Keras. Note that while this is not a typical application for CNNs, they can be used for this task by treating the text as a 1D "image".

- **\*\*Output:\*\*** The output will be the sentiment (positive or negative) that the model predicts for the review.

Each of these applications would involve training a CNN on the respective dataset, and then using the trained model to make predictions on new data. The exact architecture of the CNN would likely be different for each application, and would need to be determined through experimentation.

# References

You are welcome to use the work of others—but only if you clearly indicate what work is theirs. Failure to do so is plagiarism. Each of your references should occur within the text. For example, [1] should occur below *and* within the body of your response at the relevant location. Include specific sections of the textbooks if used directly.

[1] Geron, Aurelien. *Hands-On Machine Learning with Scikit-learn, Keras, & Tensorflow Third Edition*. “Chapter 14: Deep Computer Vision Using Convolutional Neural Networks”. O’Reilly Media, Inc. October, 2022

[2] *RMSProp*. Keras Docs. https://keras.io/api/optimizers/rmsprop/

[3] *Early Stopping*. Keras Docs. <https://keras.io/api/callbacks/early_stopping/>

[4] *Fashion MNIST Dataset*. Keras Docs. https://keras.io/api/datasets/fashion\_mnist/

# Evaluation



# Appendix 1

…

# Appendix 2…