11/5/2024

CS 767 Draft Assignment 3

Jake Giguere <giguere@bu.edu>

Hand in a partial draft of Assignment 3, using the Word template supplied, with (only) the following sections drafted:

1.1, 1.2, and 2.1

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

1. The first change would be to increase the number of epochs. Epochs give more time for the model to learn and classify the images accurately.

2. Adding padding to the images will help the kernel create less bias to a group of pixels.

3. Another option would be to test the kernel size, and see how that impacts learning. The kernel size can be more accurate if smaller, at the cost of longer learning time. Potentially there could be no different from 3x3, 2x2, 1x1

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

With a 3x3 kernel, no padding, and epochs set to 10, the accuracy increased to 80% compared to the original output of 58.12% with 2 epochs

After downloading the colab notebook. I was able to utilize my personal computers 8 cores to thread. The original output was training each epoch at roughly 83 seconds. With threading I was able to train on average 40 sec / epoch.

tf.config.threading.set\_intra\_op\_parallelism\_threads(8)

tf.config.threading.set\_inter\_op\_parallelism\_threads(8)

model.compile(optimizer='adam',

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

metrics=['accuracy'])

history = model.fit(train\_images, train\_labels, epochs=10,

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

airplane430quiet

After adding padding to the Conv2D layers, the accuracy increased to 88.45%. Given 5 more epochs would expect +90% overall accuracy. This increased accuracy is due to each pixel is getting looked at evenly with padding. Without the padding the pixels in the middle will have more bias and have been looped through more than the outer pixels.

model = models.Sequential()

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

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

model.add(layers.Conv2D(64, (3, 3), activation='relu', padding='same'))

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

model.add(layers.Conv2D(64, (3, 3), activation='relu', padding='same'))

With the power of M3 MAX chip and ChatGPT, I was also able to use tf to the fullest capacity.

ChatGPT prompt: *lets take advantage of this M3 Max chip*

ChatGPT responses:

# Enable memory growth to prevent TensorFlow from consuming all GPU memory

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

if gpus:

try:

for gpu in gpus:

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

print("Memory growth enabled for GPUs")

except RuntimeError as e:

print(e)

Also transforming data into a tf data set and using AUTOTUNE to automatically use the optimal number of threads for data loading.

# Convert data to TensorFlow Dataset

train\_ds = tf.data.Dataset.from\_tensor\_slices((train\_images, train\_labels))

test\_ds = tf.data.Dataset.from\_tensor\_slices((test\_images, test\_labels))

# Use parallel data loading and prefetching

AUTOTUNE = tf.data.AUTOTUNE

train\_ds = (

train\_ds

.shuffle(buffer\_size=10000)

.batch(64)

.map(lambda x, y: (tf.expand\_dims(x, -1), y), num\_parallel\_calls=AUTOTUNE)

.prefetch(buffer\_size=AUTOTUNE)

)

test\_ds = (

test\_ds

.batch(64)

.map(lambda x, y: (tf.expand\_dims(x, -1), y), num\_parallel\_calls=AUTOTUNE)

.prefetch(buffer\_size=AUTOTUNE)

)

## 2.1 Give 2-4 requirements for a highly capable, unique application you’ll 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 goal is to develop CNN that can analyze scanned images of handwritten sheet music (inputs) and convert them into digital music notation or MIDI files (output). The CNN should be able to classify musical notes, rests, dynamics, and even more complex symbols like time signatures and clefs. Data is available at <https://github.com/OMR-Research/muscima-pp>, paper written on muscima <https://arxiv.org/abs/1703.04824>.