PS07 Written Report

Raina Scherer

INFO 371, SP23

Overview

This written report contains basically a running report of how I created and tuned my model. I wrote it as I worked, so it should be reasonably complete. There are two preprocessing steps I wrote as separate files as they only need to be run one time, and the model itself. All three should be run from the command line.

In order to replicate my final results, the following steps should be taken:

1. Load the dataset from the Google Drive folder.
2. Unzip it and rename folders so that you have a main "data" folder, which contains "train" and "validation" subfolders.
3. Run `filenames.py` from the command line. This renames each file in a folder by cutting off anything up to the first underscore character. Due to one novel containing an extra underscore, it must be run twice per folder. Once run, each file will start with its language identifier (e.g. "EN").
4. From the command line, run `cropImages.py`. This crops each image in a folder to it's "best" 128x128 section (further information is below). It should be run on both the validation and training image folders.
5. Finally, run the model `language-detector.py` from the command line. Using the optional argument `-p` will create and save a plot of 4 mis-categorized and 4 correctly categorized images for reference.

Preparing and Exploring Data

I have a laptop beefy enough to deal with this type of computation, so I'll be running everything locally (as I have been the whole quarter).

Looking at the data, it seems to be images of text from books in each language (as described). Most have roughly the same amount of information on the page due to the consistent font size. There are a few samples I looked at that are inconsistent and/or likely going to reduce accuracy. For example, one Russian page I looked at that contained 6 words and the chapter heading "IX.".

There are roughly 32,800 training images and 8,000 validation images.

Getting the code to work

I'm using my code from `squares-circles` (lab 08) as a starting point since it does a lot of what I need in terms of getting the images, running from the command line, and a few other features.

The first step in adapting it to run (after switching the directory name) was fixing how it looks at file names. The shape data files from lab 08 all begin with their respective category ("ci", "sq", "cr") which makes it easy to find. For this dataset, the source information is at the start of the file name, which is useful (maybe some authors or sources perform worse than others?) but I could probably do without it, at least for now.

Renaming Data Files

I wrote a little script (contained in `filenames.py`) that renames the image names to start with their respective language code. The results aren't pretty but they preserve the ID code from the original data in case I want to check that information later. It just strips everything before the first underscore character. Right now it has to be run twice because one of the Thai sources has an extra underscore, and an arbitrary number needs to be appended to not lose any files. If I have time I'll fix this to output nicer names but it works well enough!

With this done, it works identically to the `squares-circles` in terms of filenames, where the first two characters of a filename indicate which category it belongs to.

Run 1: Default Params

Once the file names were standardized, the code was able to run successfully.

Output:

confusion matrix (validation)

predicted	DA	EN	RU	TH	ZN
DA	0	604	2	0	413
EN	0	2028	0	0	22
RU	3	1187	14	1	488
TH	0	462	40	87	1310
ZN	0	104	0	0	1202

Validation Accuracy: 0.418

For a first run using adapted code, it could be worse! Looking at some of the miscategorized files, Russian and Danish seem to be giving it issues so far.

Approach Adjustment

In the cats-dogs dataset, we were looking at a dataset of "real" images which were taken by (presumably) regular people of regular cats and dogs. What this means is that there is a level of expected variation in the angle, zoom, movement etc. This variation is captured and used to expand the images we're looking at by adding shear, zoom, and rotation to the images. A similar approach was used to capture variation in the circles-squares data set too.

However, my theory is that this is actually reducing accuracy on the language detection because the images are much more standard. In images of text with a standard font size, there should be very little shear (possibly italics being an exception) and no rotation or zoom. I think that by removing these image manipulations, my accuracy will go up because the characters will stay closer to their "actual", more easily distinguishable form.

Change Made: I commented out the image manipulation lines in `ImageDataGenerator()` except for rescale.

Run 2:

Output:

confusion matrix (validation)

predicted	DA	EN	RU	TH	ZN
DA	607	301	98	3	10
EN	115	1929	3	0	3
RU	166	199	1312	9	7
TH	19	14	8	1846	12
ZN	44	19	6	0	1237

Validation accuracy: 0.870

Looks like my theory was right! 😃 This is helpful for making further adjustments, as it means that my intuition of thinking about how languages differ visually (as images) is a useful approach for improving accuracy.

Next I wanted to increase the image size - I was using 32 x 32 for the squares-circles dataset but looking at the image output it was probably losing a little too much data.

Change Made: Increased target image size from (32 x 32) to (128 x 128).

Run 3:

Output:

confusion matrix (validation)

predicted	DA	EN	RU	TH	ZN

predicted	DA	EN	RU	TH	ZN
DA	693	157	91	50	28
EN	250	1711	65	16	8
RU	38	31	1600	18	6
TH	11	0	3	1884	1
ZN	1	5	31	6	1253

Validation accuracy: 0.896

Comments: The accuracy definitely went up, although not by very much. It also raised the number of parameters from 28,000 to 470,000 so it's arguable if it's a "better" model. The one definite improvement is that it's easier for me to look at the misclassified images - 32 x 32 px doesn't have a lot for me to understand. Also notable is that this boosted the performance of classifying Russian images, as it nearly eliminated instances of Russian being classified as English and Danish.

Tool Adjustments

Now that it's running pretty well, I want to make some changes that will let me tinker with it more effectively.

1. First, I want to add a way to limit the number of images being considered - I can run one epoch in ~40 seconds on the full data set, which isn't bad but it's long for testing smaller changes like refactoring.

To address this I added a command line argument `-image` that allows users to specify a maximum number of training images to consider. This is achieved by just taking a random sample of the filename dataframe for training. The full number of validation images is left in as this step is quite fast. This results in poor accuracy but this argument is just to speed up testing.

Other changes:

- I took out the section that predicts on training data, as it wasn't providing much useful information.
- There was a bug in the plot output which was causing it to display weirdly. I fixed the issue and changed it to save the output to the main folder instead of only showing it, so that I can refer back to the examples.

With those tool changes done, I'm ready to move on to adjusting the code.

Model Adjustments - Preprocessing

The first major change I want to try is cropping the images. I don't want to do a simple resize, because the font height is already standard across images, and any resizing would distort that. Instead, I want to crop each image to a given dimension so that it is standard. Ideally, I want to write a small algorithm so that the image is cropped to it's most information-dense region. This helps avoid getting blank regions and will hopefully maximize the text in each smaller region. Focusing on one good, clear, sample per image is (theoretically) more useful than looking at the entire image since I could keep the important region at a higher resolution. My hope is that this means I can eliminate some of the slower/more computationally heavy parts of the model, and that increased resolution will help the model pick up differences in individual characters - particularly between English and Danish, where ø and o are distinguishing features.

Process: I wrote this as a notebook at first, but the final version is a command line file (`cropImages.py`). Basically, from a given folder it loads in an image, converts it to a tensor, and runs a single layer of average pooling on the image. This is done with a square kernel (default size 128). Strides are set to the same size as the kernel, so each pixel is only looked at as part of one "region". This is much more simple for processing but almost certainly misses out on the best possible region of all regions. The argmin() function is used to find the lowest value in the result of the pooling operation (minimum amount of whitespace) and the image is cropped to that region and saved as a new jpg.

Run 4: Cropped Images, 10 Epochs

Data: Training and Validation cropped to 128x128 px Params: 28,000 Target Image Size: 32 x 32

Output:

confusion matrix (validation)

predicted	DA	EN	RU	TH	ZN
DA	669	322	27	0	0
EN	84	1955	9	0	0
RU	44	159	1486	0	0
TH	0	0	0	1895	1
ZN	1	0	0	0	1303

Validation accuracy: 0.919

Comments: I'm really happy with how this worked! It's significantly faster (10 epochs in ~3 minutes with the full dataset), and the final images for the miscategorized plot are actually discernable. Even with no adjustments to the other layers, it was an improvement in accuracy. Mostly, I am very pleased with the way my cropping functionality works 😃

Run 5:

Now I want to improve the layers given these new, consistent input dimensions. I updated the layer sizes to make more sense, where the first convolution layer has 32 nodes, the second has 16, and the dense layer has 32.

Data: Training and Validation cropped to 128x128 px Params: 14,000 Target Image Size: 32 x 32

Output:

confusion matrix (validation)

predicted	DA	EN	RU	TH	ZN
DA	798	177	42	0	1
EN	229	1798	19	2	0
RU	53	53	1583	0	0
TH	0	0	0	1895	1
ZN	1	0	0	5	1298

Validation accuracy: 0.927

Comments: A change in the right direction! I was able to half the parameters and gain a percentage point in accuracy. Generally this seems to have improved performance for Danish and Russian.

Run 6:

I fiddled with a few parameters - tried removing `max_pooling`, changed a few layer sizes, and altered a couple kernel sizes. Ultimately the changes I kept for this run were increasing the target image size, and changing the first layer to 64 nodes, second to 32, and the dense layer to 16.

Data: Training and Validation cropped to 128x128 px Params: 60,000 Target Image Size: 64 x 64 Epochs: 10

Output:

confusion matrix (validation)

predicted	DA	EN	RU	TH	ZN
DA	624	390	4	0	0
EN	5	2039	4	0	0
RU	4	31	1654	0	0
TH	0	0	0	1896	0
ZN	1	0	0	5	1304

Validation accuracy: 0.950

Run 6.5:

I ran it again with the same parameters just to double check my results.

Output:

confusion matrix (validation)

predicted	DA	EN	RU	TH	ZN
DA	927	50	0	0	1
EN	108	1937	2	1	0
RU	26	10	1651	0	2
TH	0	0	0	1896	0
ZN	1	0	0	5	1304

Validation accuracy: 0.975

Comments:

I'm (obviously) very happy with this and might consider it my best model unless I have a good idea to change things. It has relatively low parameters but manages to categorize 4/5 languages nearly perfectly, with an overall validation accuracy between 0.95-0.975. Danish continues to be difficult to capture so any improvements would need to be geared to helping the CNN with that language specifically.

Technical Explanation

In the PDF, we're asked to comment on the technical side. I made a few mentions to this throughout, but I wanted to include a small section that mentions it directly. I made the financially painful choice to get an M1 Pro Macbook at the start of this year, which felt like overkill until this quarter. I was able to get tensorflow running locally and had things run very smoothly, which enabled me to iterate my model many times to tune the parameters. On the entire dataset, I can run an epoch in ~40 seconds. With my cropped dataset, it's ~25 seconds. I did add functionality to scale down the dataset just to make testing small changes faster, but ultimately it ran pretty well from the start and I didn't need any additional compute resources.

Best (Final) Model

In the end I didn't have any big creative ideas beyond my last model, but I made a couple small changes that proved useful. The biggest impact was adding a pretty aggressive dropout to the first layer. This seemed to help Danish perform a bit better on average. I also increased the pooling_size param to 4 to reduce parameters a little. Overall, these were the main hyperparameters for my final model:

Data: Training and Validation cropped to 128x128 px Params: 40,000 Target Image Size: 64 x 64 Epochs: 30

Output:

confusion matrix (validation)

predicted	DA	EN	RU	TH	ZN

predicted	DA	EN	RU	TH	ZN
DA	796	206	16	0	0
EN	22	2015	11	0	0
RU	6	10	1673	0	0
TH	0	0	0	1896	0
ZN	0	0	0	0	1304

Validation accuracy: 0.966

Comments: It's basically the same as my last run, I think the extra epochs just smoothed it out a little. Overall, it was really fun trying to improve my results!