INFO 371 ProblemSet 7 - Jeremy Dondoyano

See attached ps7.py for the script

The Process

Items denoted (x) are shown in the attempts section, which shows my numbered attempts corresponding to the number in the parenthesis.

I first accidentally ran the model with all features and all files, which took about 25-30 minutes per go, and I also realized that I wasn't reading the correct letters from the file names. After fixing these issues I was able to run the model much quicker and on the correct categories for testing, and I began with (1) using categories: EN+ZN. I played with the first convolutional network, giving it 64 filters, the *strides*, *pool-size*, as well as *size*, *dropout*, and *dense* of the second convolutional layer, and added the *adam* optimizer. This gave me 100% accuracy on both the training and validation matrices, which hinted at overfitting. Then, in (2) I set the first convolutional network to 32 filters, increased the second network's *dropout* to 0.25, and set the *dense* to 560. This gave me optimal results for EN+ZN without signs of overfitting. I also played with adding different categories such as DA, and TH, both of which decreased the accuracy of my model. One thing that you might have noticed was that I changed the *kernel_initializer* to orthogonal starting in (3), which I don't know much about except for the fact that it's supposed to be good for 2D networks, and it increased my accuracies. Next, I then resized the images to 200x200 pixels using the windows PowerToys resize images feature, this process took about 30 minutes for the training images and 6 minutes for the validation images. After resizing the images, I ran the model with the same parameters as (7), which gave me sub-0.55 accuracies for training and validation, and got accuracies of 0.782 and 0.945 for validation and training respectively.

The Caveats

Since I am using a laptop, the results/efficiency of training/testing will not be the most optimized. This being said, I have sixteen gigabytes of ram and a 4-core, 8-thread processor that clocks up to 4.9 GHz, so my system is capable, however, HWMonitor shows the processor reaching temperatures of 100c, which means that it is thermal-throttling during the process. Furthermore, I have an Nvidia GeForce MX230 in my laptop, on which I was able to get CuDNN and TensorFlow GPU to run for GPU acceleration, which helped the speed of the model. I tried to use LSTM and CuDNNLSTM for better results but was not able to figure this out. As stated, I did scale down the data used to 1000 images per category, and the epochs did not seem to be a problem for my computer.

The Final Results

My final results were not as spectacular as I would have liked to see. I tested my model on the languages English, Russian, Thai, and Chinese. Since this includes all but one of the languages in the data, it was harder to get good accuracy. Furthermore, since I was a little bit limited by my computer hardware, I wasn't able to perform as many changes/tests as I would've liked. I settled for trial (10) (I did many more trials with many different changes but none were better than this). This run took 414 seconds to go over 10 epochs of 2000 images in each of the 4 languages. The images, in this case, were cropped to 200x200 as well, and the model had 1.5 million parameters, which is quite a bit.

My first convolutional layer featured 64 filters, with a kernel size of 3, strides set to 3, activation relu, the orthogonal initializer, batch normalization, a pool size of 3, and dropout of 0.8. My second convolutional layer also had 64 filters with a kernel size of 3, orthogonal initializer, activation relu, batch normalization, a pool size of 3, and dropout 0.8. After that, flatten, dense set to 620 with the orthogonal initializer, and relu activation were all added. Moreover, batch normalization, a dropout of 0.8, dense of 100 with softmax activation, and the adam optimizer were also added.

This model gave me a validation accuracy of 0.53519, and a training accuracy of 0.53025, per the confusion matrix:

Predicted Category	EN	RU	TH	ZN
EN	0	9	1987	4
TH	0	0	3798	0
ZN	4	10	1290	2

Attempts:

Bold denotes changed parameters

- (1) Using **10 epochs** with 1000 images from categories EN, and ZN (279 seconds):
    - 1. **Conv2D = 64**, kernel_size=3, **strides=3,** ,img size=500,200 activation=relu,initializer=initializer.HeNormal(), BatchNormalization, **pool_size=3,** dropout=0.25
    - 2. Conv2D = 64, kernel_size=3,initializer=initializer.HeNormal(), activation=relu,BatchNormalization, **pool_size=3**, **dropout=0.2**
    - -flatten, **dense=550** with HeNormal and relu
    - Batchnormalization
    - dropout=0.5
    - dense=2 activation=softmax
    - **optimizer=adam**

  Confusion Matrix Validation accuracy: 1.0
  Confusion Matrix Training accuracy: 1.0
  Overfitting likely

- (2) Using **10 epochs** with 1000 images from categories EN, and ZN,img size=500,200 (250 seconds):
    - 1. Conv2D = 32, kernel_size=3, **strides=3,** activation=relu,initializer=initializer.HeNormal(), BatchNormalization, **pool_size=3,** dropout=0.25
    - 2. Conv2D = 64, kernel_size=3,initializer=initializer.HeNormal(), activation=relu,BatchNormalization, **pool_size=3**, **dropout=0.25**
    - -flatten, **dense=560** with HeNormal and relu
    - Batchnormalization
    - dropout=0.5
    - dense=2 activation=softmax
    - **optimizer=adam**

  Confusion Matrix Validation accuracy: 0.997
  Confusion Matrix Training accuracy: 0.995

- (3) Using **10 epochs** with 1000 images from categories EN, and ZN, img size=500,200 (224 seconds):
    - 1. Conv2D = 32, kernel_size=3, **strides=3,** activation=relu,initializer=**orthogonal**, BatchNormalization, **pool_size=3,** dropout=0.25

    - 2. Conv2D = 64, kernel_size=3,initializer=**orthogonal**, activation=relu,BatchNormalization, **pool_size=3**, **dropout=0.25**
    - -flatten, **dense=560** with **orthogonal** and relu
    - Batchnormalization
    - dropout=0.5
    - dense=2 activation=softmax
    - **optimizer=adam**

  Confusion Matrix Validation accuracy: 0.999
  Confusion Matrix Training accuracy: 1.0
  Overfitting likely

- (4) Using **10 epochs** with 1000 images from categories EN, ZN, **and TH**, img size=500,200 (376 seconds):
    - 1. Conv2D = 32, kernel_size=3, **strides=3,** activation=relu,initializer=**orthogonal**, BatchNormalization, **pool_size=3,** dropout=0.25
    - 2. Conv2D = 64, kernel_size=3,initializer=**orthogonal**, activation=relu,BatchNormalization, **pool_size=3**, **dropout=0.25**
    - -flatten, **dense=560** with **orthogonal** and relu
    - Batchnormalization
    - dropout=0.5
    - dense=2 activation=softmax
    - **optimizer=adam**

  Confusion Matrix Validation accuracy: 0.3903
  Confusion Matrix Training accuracy: 0.3333

- (5) Using **10 epochs** with 1000 images from categories EN, ZN, **and TH**, img size=500,200 (352 seconds):
    - 1. Conv2D = 64, kernel_size=3, **strides=3,** activation=relu,initializer=**orthogonal**, BatchNormalization, **pool_size=3,** dropout=0.25
    - 2. Conv2D = 64, kernel_size=3,initializer=**orthogonal**, activation=relu,BatchNormalization, **pool_size=3**, **dropout=0.25**
    - -flatten, **dense=580** with **orthogonal** and relu
    - Batchnormalization
    - dropout=0.7
    - dense=2 activation=softmax
    - **optimizer=adam**

  Confusion Matrix Validation accuracy: 0.6676
  Confusion Matrix Training accuracy: 0.6686

- (6) Using **10 epochs** with 1000 images from categories EN, ZN, **and TH**, img size=500,200 (352 seconds):
    - 1. Conv2D = 64, kernel_size=3, **strides=3,** activation=relu,initializer=**orthogonal**, BatchNormalization, **pool_size=3,** dropout=0.25
    - 2. Conv2D = 64, kernel_size=3,initializer=**orthogonal**, activation=relu,BatchNormalization, **pool_size=3**, **dropout=0.25**
    - -flatten, **dense=580** with **orthogonal** and relu
    - Batchnormalization
    - dropout=0.7
    - dense=2 activation=softmax
    - **optimizer=adam**

  Confusion Matrix Validation accuracy: 0.6676
  Confusion Matrix Training accuracy: 0.6686

- (7) Using **10 epochs** with 1000 images from categories EN, **and DA**, img size=500,200 (285 seconds):
    - 1. Conv2D = 64, kernel_size=3, **strides=3,** activation=relu,initializer=**orthogonal**, BatchNormalization, **pool_size=3,** dropout=0.25
    - 2. Conv2D = 64, kernel_size=3,initializer=**orthogonal**, activation=relu,BatchNormalization, **pool_size=3**, **dropout=0.25**
    - -flatten, **dense=580** with **orthogonal** and relu
    - Batchnormalization
    - dropout=0.7
    - dense=2 activation=softmax
    - **optimizer=adam**

  Confusion Matrix Validation accuracy: 0.498
  Confusion Matrix Training accuracy: 0.5295

- (8) Using **10 epochs** with 1000 images from categories EN, **and DA**, **img size=200,200** (141 seconds):
    - 1. Conv2D = 64, kernel_size=3, **strides=3,** activation=relu,initializer=**orthogonal**, BatchNormalization, **pool_size=3,** dropout=0.25
    - 2. Conv2D = 64, kernel_size=3,initializer=**orthogonal**, activation=relu,BatchNormalization, **pool_size=3**, **dropout=0.25**
    - -flatten, **dense=580** with **orthogonal** and relu
    - Batchnormalization

    - dropout=0.7
    - dense=2 activation=softmax
    - **optimizer=adam**

  Confusion Matrix Validation accuracy: 0.782
  Confusion Matrix Training accuracy: 0.945

- (9) Using **20 epochs** with 1000 images from categories EN, **RU, TH, ZN**, **img size=200,200** (576 seconds):
    - 1. Conv2D = 64, kernel_size=3, **strides=3,** activation=relu,initializer=**orthogonal**, BatchNormalization, **pool_size=3,** dropout=0.25
    - 2. Conv2D = 64, kernel_size=3,initializer=**orthogonal**, activation=relu,BatchNormalization, **pool_size=3**, **dropout=0.25**
    - -flatten, **dense=580** with **orthogonal** and relu
    - Batchnormalization
    - **dropout=0.7**
    - dense=2 activation=softmax
    - **optimizer=adam**

  Confusion Matrix Validation accuracy: 0.502
  Confusion Matrix Training accuracy: 0.2625

- (10) Using **10 epochs** with 1000 images from categories EN, **RU, TH, ZN**, **img size=200,200** (414 seconds, 1.5 million parameters):
    - 1. Conv2D = 64, kernel_size=3, **strides=3,** activation=relu,initializer=**orthogonal**, BatchNormalization, **pool_size=3, dropout=0.8**
    - 2. Conv2D = 64, kernel_size=3,initializer=**orthogonal**, activation=relu,BatchNormalization, **pool_size=3**, **dropout=0.8**
    - -flatten, **dense=620** with **orthogonal** and relu
    - Batchnormalization
    - **dropout=0.8**
    - **dense=100** activation=softmax
    - **optimizer=adam**

  Confusion Matrix Validation accuracy: 0.535
  Confusion Matrix Training accuracy: 0.53025