Neural Networks. Delivery for R&I

Diego Palacios, Arturo del Cerro, Nil Bellmunt
NIU 1594283, 1593930, 1588664

Universitat Autònoma de Barcelona

July 24, 2021

Overview

Descriptive analysis of the dataset

- In order to see if each vowel and accent is uniformly distributed we analyze their frequencies

Vowel	a	e	i	o	u
Frequency	20.45%	20.8%	19.1%	20%	19.65%

Table: Vowel frequency

Accent	á	à	ä	â	ā	no accent
Frequency	18.8%	15.6%	16.05%	16.85%	15.4%	17.3%

Table: Accent frequency

- We also check that the independent frequencies of all 30 labels are similar.
- There are images that don't display correctly the accent mark, either they don't show it, or show a square instead:

- We decide to keep these images (6.9% of the dataset) in the dataset, because these cases always belong to the 'bar' accent mark, and in the case where no accent is displayed, the letter on the right tends to be shifted to the left. Therefore a Neural Network can learn this.

Preprocessing techniques

- Convert to **black and white**: Color does not provide useful information.
- **Invert**: Invert image so background has value 0. This is done so we can naturally set the padding in convolutions to have value 0.
- **Scale**: Scale image such that the larger pixel has value 1.

```
def image_preprocess(image):
    image = tf.io.read_file(image)
    image = tf.io.decode_png(image, channels=1) # read in black and white
    image = tf.image.convert_image_dtype(image, tf.float32) # between 0 and 1
    image = 1 - image # invert colors so background is 0
    image = image / tf.reduce_max(image) # normalize to [0, 1]
    image.set_shape([128, 128, 1]) # set shape because it is not automatically
                                   # infered at compile time. Assume 128x128.
    return image
```

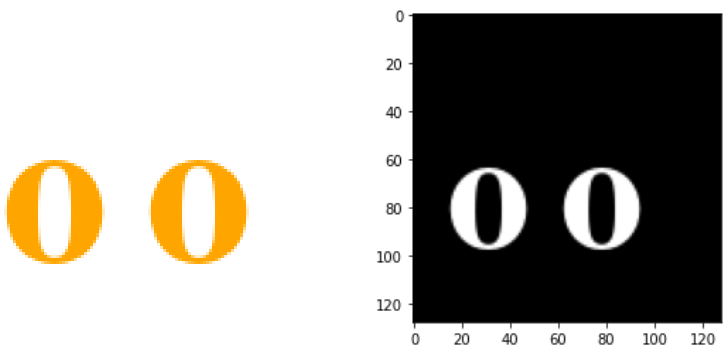


Figure: Original and preprocessed image

For all the experiments we use 75% of the dataset for training and 25% for validation. The split is the same across all experiments to have consistent results.

Experiments: Simple model

We tried a first Model (initial guess) with 3 layers:

- 3x3 convolution (no padding), 32 filters, ReLU
- Dense, 128 neurons, ReLU
- Dense, 30 neurons, SoftMax

The model is trained with:

- **Adam** optimizer, because of its fast convergence
- **Batchsize 32**, which gives 47 steps per epoch
- **8 epochs**, enough for convergence
- Categorical **cross-entropy** loss function

$\approx$ 60% accuracy. Huge number of parameters (65M) due to dense layer after the first convolution. Huge overfitting due to the large number of parameters and simplicity of the model.

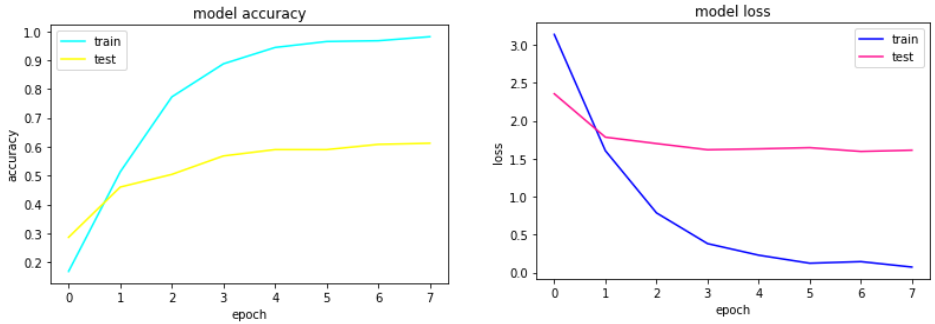


Figure: Accuracy and loss from simple model

Experiments: Convolutional model

A more sophisticated model with 5 convolutions:

- 3x3 convolution (no padding), 32 filters, 2x2 max pooling, ReLU
- 4 more convolutions as above with 64, 128, 256 and 512 filters
- Dense, 30 neurons, SoftMax

The number of filters are doubled at each convolution as its common practice. To keep more or less constant the number of calculations at each layer, 2x2 max pooling is used after the convolutions to reduce the dimensions of the image channels.
We use the same hyperparameters as in the previous experiment.
97% accuracy and $1.6M$ parameters, clearly improving the previous model.

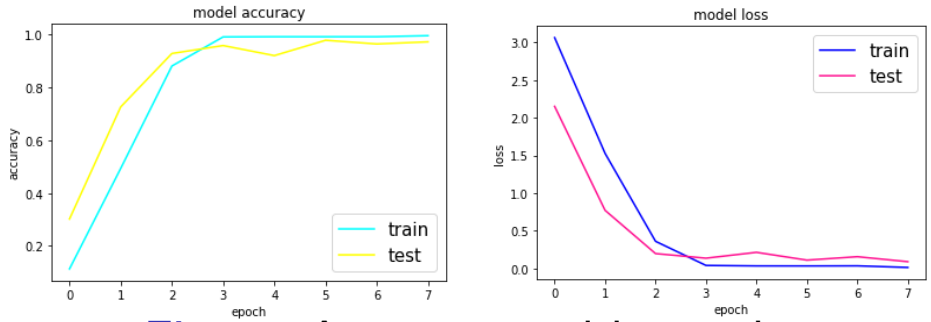


Figure: Accuracy and loss values

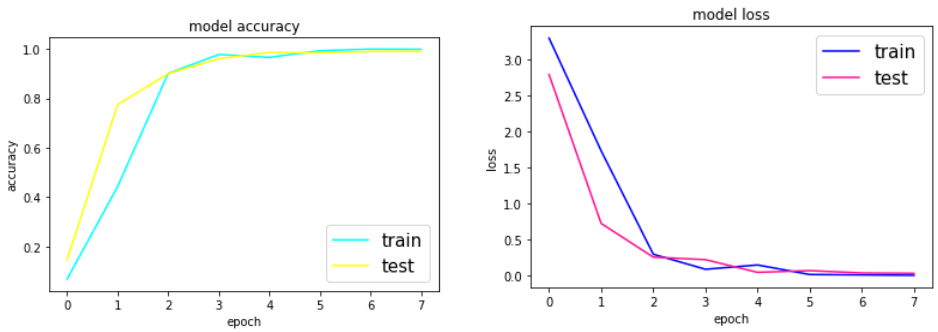
A similar model but with padding = 'same' instead of 'valid' reaches a 98% accuracy. This model will be the **base model**.

Experiments: Global pooling

We now use a Global Pooling layer after the last convolution of the base model to reduce the number of parameters in the dense layer and reduce overfitting:

- **Global Average Pooling**
  From the base model we add at the end a GAP layer instead of the max pooling and flatten layers. Convergence is quite slower but gets a higher accuracy (99.59%). Here we had to take 16 epochs instead of 8 to ensure convergence.

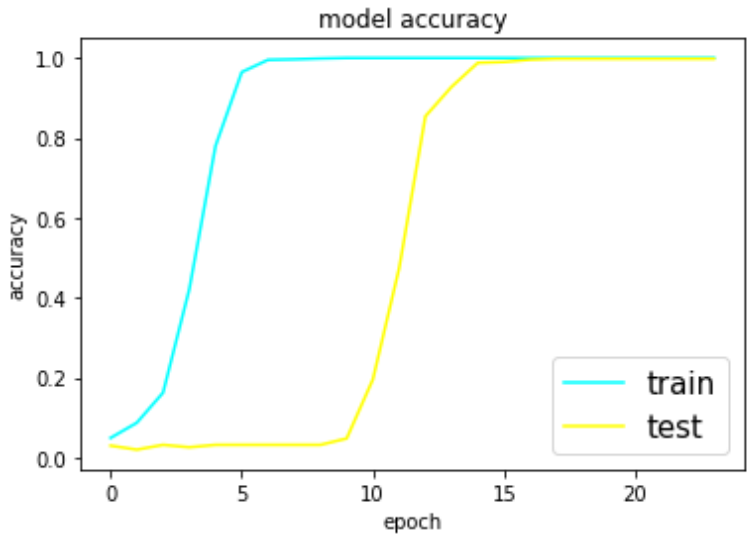
- **Global Max Pooling**
  Same but using a GMP layer. Gets a 99.4% accuracy but training is much faster. In order to deal with that accuracy loss we add an extra dense 4096 layer and get an accuracy and loss values of 99.6% and 0.0155 respectively. We decide to continue with this model because it is faster.

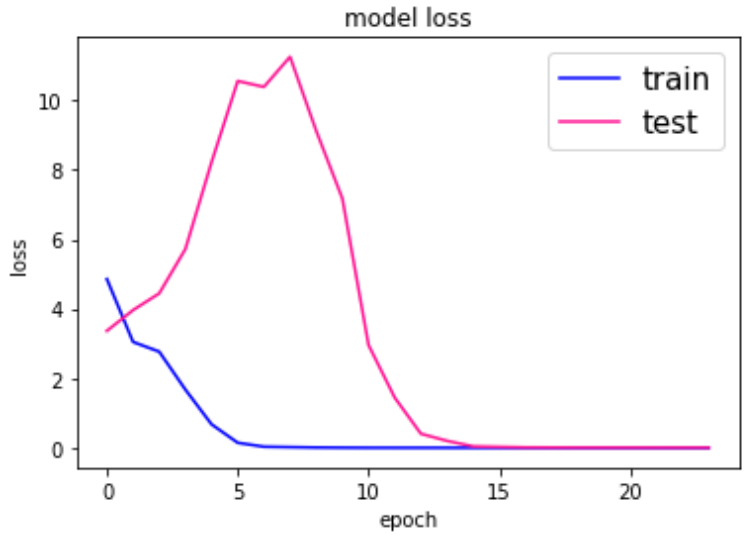


Experiments: Batch normalization

We use batch normalization to stabilize the learning process. Model structure:

- 5 convolutions as in the baseline model, but with batchnormalization
- Global Max Pooling
- Dense, 4096 neurons, ReLU
- Dense, 30 neurons, SoftMax





Batch normalization produces overfitting at the beginning but then starts to generalize quite fast ending with an accuracy of 99.8%. We had to train during 24 epochs to ensure convergence.

Discarded experiments

Here we summarize some other experiments that did not improve the behavior of the network:

- **Strides**: using stride=2 in the convolutions instead of 2x2 max pooling accelerates the training, but the performance is degraded. This is why we decided to use max pooling instead of strides.
- **Spatial Dropout**: We tried different amounts of spatial dropout after the convolutions to have more regularization, however this did not improve the results because we already had a good accuracy.
- **Data augmentation**: Despite data augmentation being a great tool for improving accuracies by reducing overfitting, we decided not to implement it because we already had good accuracy and data augmentation degraded it. Also, the images that are more commonly misclassified are the ones where the font is in uppercase or with boxes around the letters. This images are difficult to recreate and also represent a very small fraction of the dataset.

Final Model: Two Headed Network

- Looking for a better and more elegant model, we developed our final model following the path of the previous model with the difference that it now consists of a 2 headed network: one for classifying vowels and the other for classifying accents.
- By predicting the vowels and accents separately, we believe the network overfits less because now there are more images per class, and each head of the network is more specialized.
- This model achieves 100% accuracy on the validation set that we have been using in all experiments. To have a more precise prediction of the accuracy, we have done a 5 folds cross-validation and obtained that the real accuracy is close to **99.7%**.

Final Model: Structure and Prediction

The structure of the final model is:

- 3x3 convolution (padding='same'), 32 filters, BatchNorm, 2x2 max pooling, ReLU
- 4 more convolutions as above with 64, 128, 256 and 512 filters
- Global Max Pooling
- Dense, 4096 neurons, ReLU
- **Head 1**: Dense, 5 neurons, SoftMax
- **Head 2**: Dense, 6 neurons, SoftMax

For the final predictions, we make an ensemble of the 5 models trained in the crossvalidation. The ensemble is done by averaging the output probabilities. We do the ensemble to reduce the variability of a single model, and ensure the accuracy is close to the predicted with cross validation. We don't expect the accuracy to increase because the models are highly correlated (they are train with almost the same data).

Hyperparameters

For our final model we use:

- Optimization parameters
  - **32 epochs**, to ensure convergence.
  - Use of **Adam optimizer** because it converges fast, with the default values of TensorFlow. Adam is a stochastic gradient descent method based on adaptive estimation of first-order and second-order moments.
  - Mini-batch size: 32 to train the model and 16 to test it.
  - Loss: Sum of the categorical **cross-entropy** of the two heads.
- Model specific hyperparameters
  - Activation functions: **ReLU**, because it does not have gradient problems and computation is fast.
  - 8 layers depth (5 convolutions + GMP + 2 Denses)
  - Convolution number of filters: 32, 64, 128, 256 and 512
  - Convolution kernel size: 3x3
  - Padding with zeros such that the output of the convolution has the same shape as the input.
  - Pooling type: 2x2 max pooling
  - Hidden dense size: 4096
  - Batch normalization: TensorFlow default values

Accuracy

Summarizing the accuracies of the models we developed we can check that

	parameters	layers	loss function	test accuracy
Simple model	65,032,414	3	1.7260	59.4%
Convolutional model	1,629,470	6	0.0889	98.0%
Global max pooling	1,583,390	7	0.0155	99.6%
Batch normalization	3,796,126	8	0.0080	99.8%
Final model [1]	3,718,283	8	0.0061	100.0%
Final model [2]	3,718,283	8	0.0216	**99.7%**

- Justification: The experiments were done with the same train-validation split, and only at the end we used cross-validation to avoid implicit overfiting, and to have an accurate estimation of the accuracy.
- Needs for accuracy improvements:
  Our final model gets almost a 100% accuracy. The only ones that the models may get wrong are the letters with boxes or the fonts in uppercase, which represent a very small number of the dataset. Therefore we conclude we would need a larger dataset that has more of this types of samples to improve accuracy. We do not need more computer power, because using a cloud GPU in vast.ai, we were able to do the crossvalidation in half an hour. The price of the GPU was only 0.10$/hour.

[1] Measured with no cross-validation

[2] Measured with 5 folds cross-validation

Conclusions

- We have gain experience working with TensorFlow to create predictive models, and using cheap cloud GPUs in vast.ai website to train models faster (we could have also done the cross-validation in our laptop, but it would have taken some hours).
- The given exercise was not hard for the networks to understand due to the dataset being quite easy (well displayed letters with clear background and no noises). The biggest problems were with the bar accent mark.
- Although the bar accent mark was not always correctly displayed in the image, the network has still been able to identify it in most cases.
- This model would provably work good as well in a harder exercise with more diversity of symbols and accents.
- We thought of using more complex models, such as ResNet, but we did not do it because the final model had great performance.
- With the final model, we correctly classify almost all images. Although the model gives 99.7% accuracy with cross-validation, it is more optimal (to maximize the grade of this project) to say that it will have $\mathbf{99 \pm 1\ \%}$ accuracy on the testset because we obtain the same grade if the real accuracy is at a distance equal or smaller than 1% from prediction.
- Being this a real paper and not an exercise, and having to give a single value for the accuracy instead of an interval, we would not have changed to 99% and had kept the 99.7% accuracy prediction.