Character Recognition in Natural Images

Kevin Sharp

Capstone Project 2

Springboard

Data Reference:

T. E. de Campos, B. R. Babu and M. Varma. **Character recognition in natural images.** In *Proceedings of the International Conference on Computer Vision Theory and Applications (VISAPP), Lisbon, Portugal*, February 2009.

What are “Natural” Images?

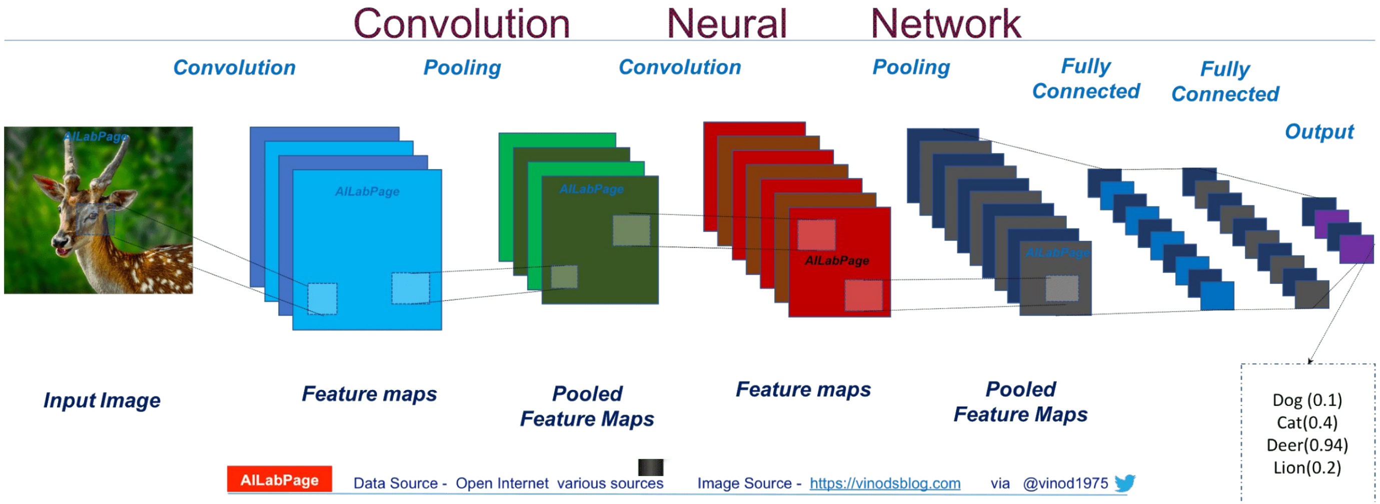
- Photographs taken of objects in the real world
- Letters and numbers may be printed on the surface of such objects
- The characters themselves may be three-dimensional objects
- Recognizing these characters with computer vision presents a more challenging problem than traditional OCR

The Dataset - Chars74K

- Publically available at *http://www.ee.surrey.ac.uk/CVSSP/demos/chars74k/*
- 7700 images spanning 62 classes (digits 0-9, letters A-Z and a-z)
- Authored by T. E. de Campos, B. R. Babu, and M. Varma at Microsoft Research India

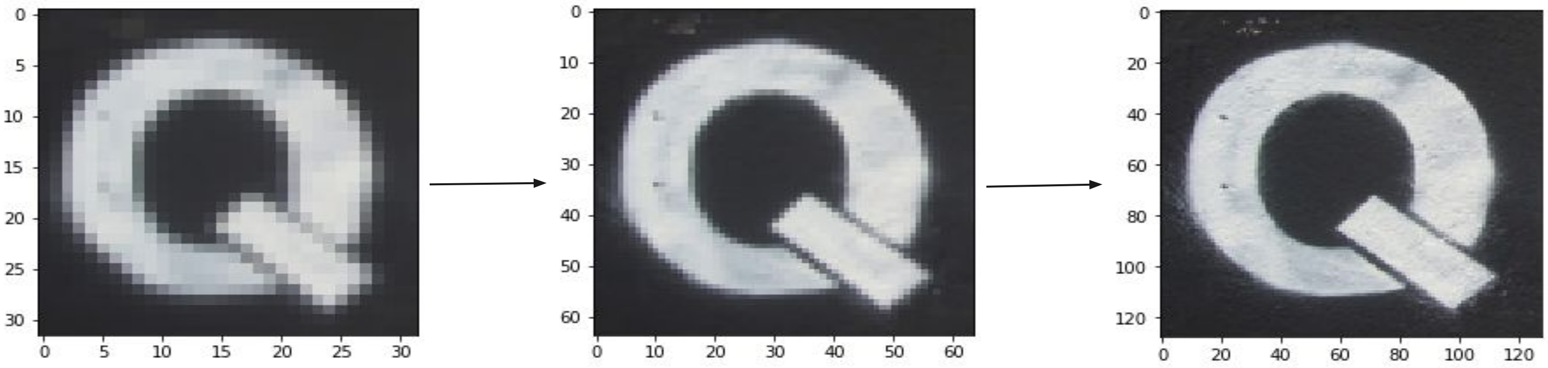
The Approach - Convolutional Neural Networks

- Takes an image as input and outputs predicted class probabilities
- Hidden convolutional, pooling, and fully connected (dense) layers work to extract features and convert these featured into predictions



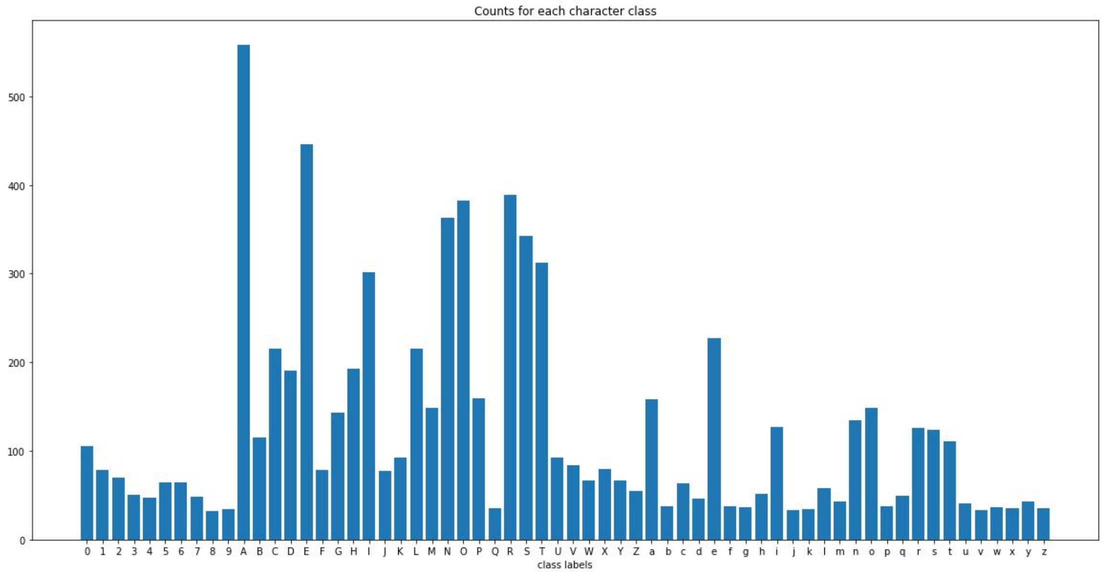
The Approach - Progressive Resizing

- Running multiple iterations of a model on different-sized inputs
- Transfer layers and weights from one iteration to the next (transfer learning)
- Allows model to learn more robust features quickly



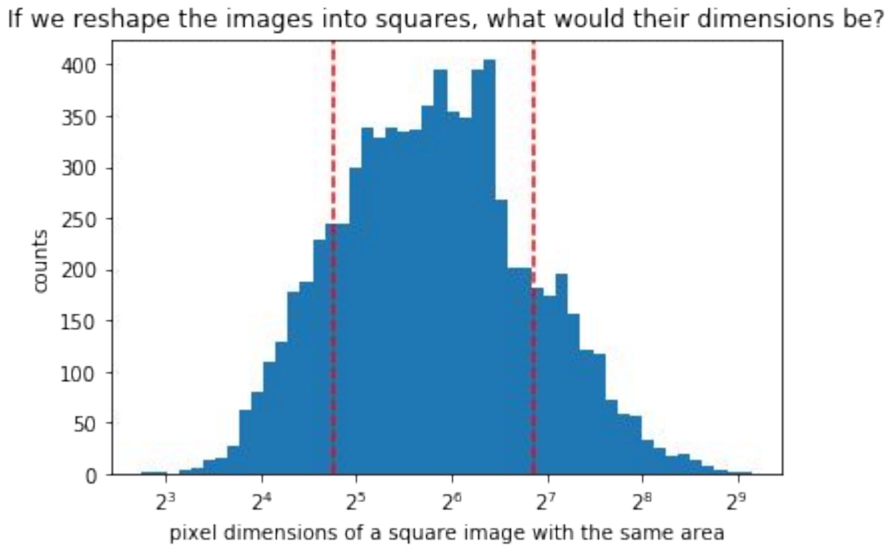
Exploratory Data Analysis - Class Distribution

- Highly imbalanced, range of 526
- Mean of 78
- Median of 124.19
- Class counts will be used to set class weights for the model



Exploratory Data Analysis - Image Dimensions

- The network requires all images to have the same dimensions, so I chose a standardized side length
- With progressive resizing, will use 32x32, 64x64, 128x128



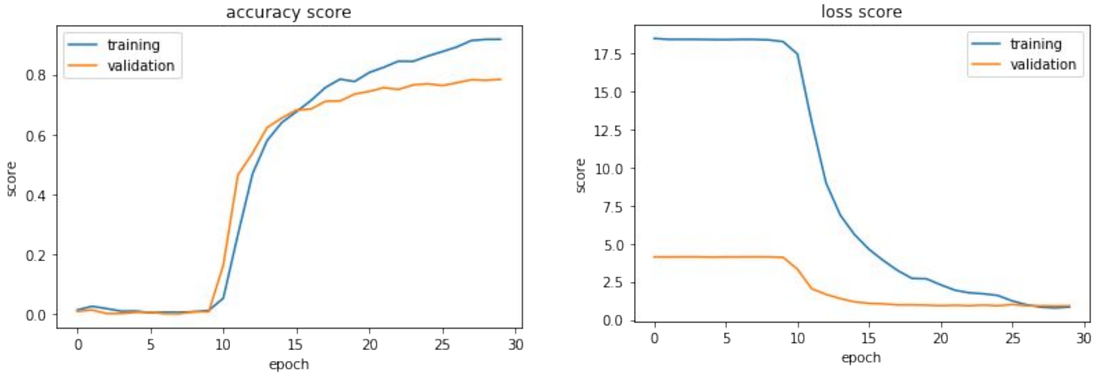
Building and Training the Model - Layers

```python
# CNN layers, 32x32

layers = [
    Conv2D(128, kernel_size=3, padding='same', activation='relu', input_shape=(size, size, 3)),
    MaxPooling2D(padding='same'),
    Dropout(0.2),

    Conv2D(256, kernel_size=3, padding='same', activation='relu'),
    MaxPooling2D(padding='same'),
    Dropout(0.2),

    Conv2D(512, kernel_size=3, padding='same', activation='relu'),
    MaxPooling2D(padding='same'),
    Dropout(0.2),

    Flatten(),
    Dense(2048, activation='relu'),
    Dropout(0.2),

    Dense(62),
    Activation('softmax')]

model_1 = Sequential()
for layer in layers:
    model_1.add(layer)
```

- Three sets of convolution, max pooling, and dropout
- Two fully connected layers
- Softmax activation for classification

Building and Training the Model - Compile and Fit

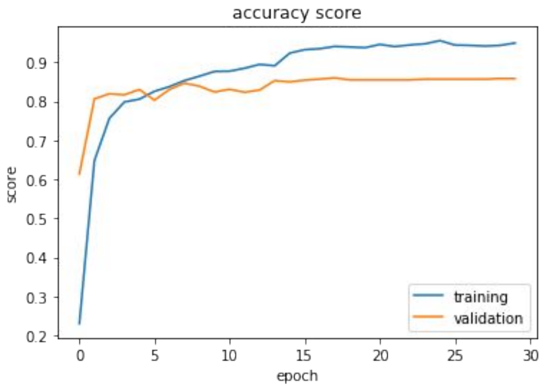
```
# build and run model, 32x32

reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=5, verbose=1, min_delta=0.01)
mc = ModelCheckpoint('best_model_32.h5', monitor='val_loss', mode='min', save_best_only=True)

model_1.compile('adam', 'categorical_crossentropy', ['accuracy'])

model_1.fit(X_train, y_train,
          epochs=30, batch_size=64,
          validation_split=0.25,
          class_weight=class_weights,
          callbacks=[reduce_lr, mc])

plot_model_history(model_1.history.history)
```

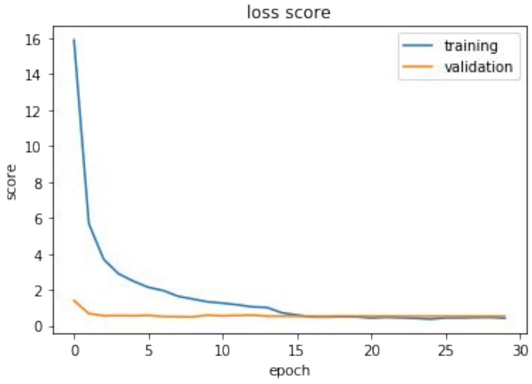


- Checkpoint best model for the next iteration
- Best validation loss - 0.9085
- Best val. accuracy - 78.05%

Building and Training the Model - 2nd Iteration

```
# CNN layers, 64x64

model_2 = Sequential()

model_2.add(Conv2D(64, kernel_size=3, padding='same', activation='relu', input_shape=(size, size, 3)))
model_2.add(MaxPooling2D(padding='same'))
model_2.add(Dropout(0.2))

model_2.add(Conv2D(128, kernel_size=3, padding='same', activation='relu'))

prior = load_model('best_model_32.h5')
for layer in prior.layers[1:]:
    model_2.add(layer)
```

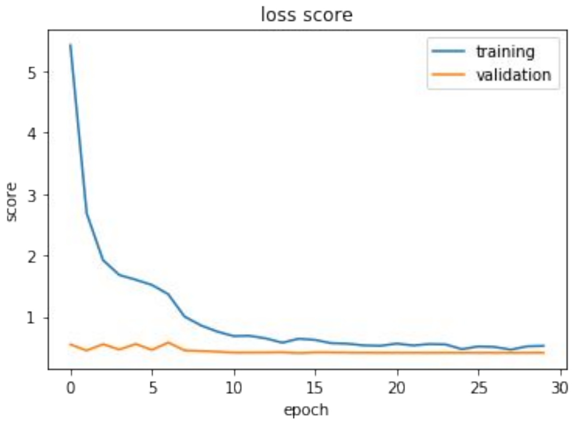


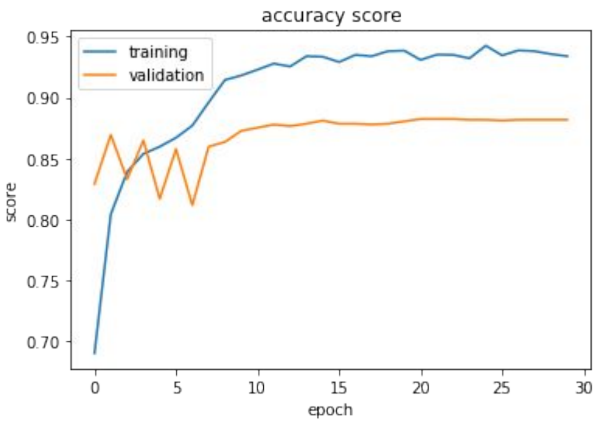


- Uses same compiler and fit parameters as 1st iteration
- Best validation loss - 0.4954 (-0.4131)
- Best val. accuracy - 83.83% (+5.78)

Building and Training the Model - 3rd Iteration

- Layers are added in the same manner as before; compile and fit steps unchanged
- Best validation loss - 0.4096 (-0.0858)
- Best validation accuracy - 88.12% (+4.29)

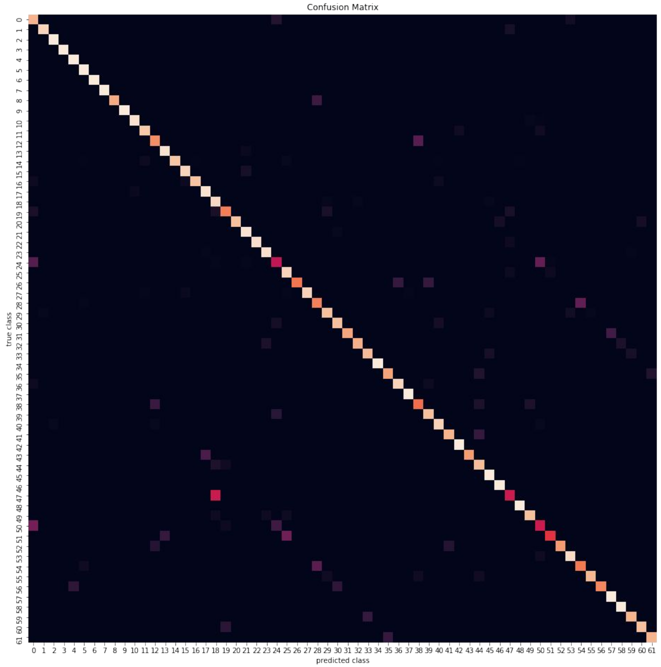




Model Evaluation

```
13/13 [==============================] - 9s 681ms/step - loss: 0.4082 - accuracy: 0.8740
```

- Testing done on 20% of data (holdout)
- Precision - 87.06%
- Recall - 87.96%
- f_1 score - 87.51%
- Two secondary diagonals represent confusion between upper and lowercase letters



Failure Analysis

	true_positives	false_positives	false_negatives	precision	recall	f1
o	15.0	24.0	15.0	0.384615	0.500000	0.442308
l	6.0	4.0	6.0	0.600000	0.500000	0.550000
c	9.0	11.0	4.0	0.450000	0.692308	0.571154
s	18.0	17.0	7.0	0.514286	0.720000	0.617143
p	4.0	2.0	3.0	0.666667	0.571429	0.619048
0	18.0	29.0	3.0	0.382979	0.857143	0.620061
O	36.0	9.0	40.0	0.800000	0.473684	0.636842
J	11.0	3.0	4.0	0.785714	0.733333	0.759524
x	6.0	2.0	1.0	0.750000	0.857143	0.803571
S	50.0	7.0	18.0	0.877193	0.735294	0.806244

- o, c, s, p, O, x, and S appear on both upper-lowercase confusion diagonals
- 0 and l (lowercase L) have visual similarities to other classes
- J has a number of non-standard forms (cursive versus print, etc.)

Recommendations for Improvement

- Data augmentation for under-represented classes
- A full computer vision application would take full scenes, pre-segmented, as input - could this input be used to provide additional context?
    - Adjust probabilities for uppercase versus lowercase
    - Better predictions for number versus letter
    - Which prediction, together with more confident predictions, forms a word found in a pre-loaded dictionary?
- Covers most of the confusion cases found by this model