# **Programming Language Detection from Images**

# **Introduction**

With the increase of open source programming environments in recent years, the users of these environments have also increased and continue to increase. With increasing interest in software, a lot of source code has become accessible on platforms such as internet, etc. Subsequently, many data mining techniques have been developed through these accumulated resources. However, these techniques are mostly based on textual sources of information. This includes source code, user forums, bug tracking systems, and traceability logs, to name only a few application domains. While each of these types of artifacts require their own heuristics to account for differences in form, function, and vocabulary [1], at the lowest level they share a common textual encoding, regardless of whether they represent natural languages, programming languages, or a combination of both [2]. In addition, there is a significant amount of content available in such formats as is provided in many platforms, such as online coding courses (Udemy, Youtube etc.), where source code is shared as multimedia or video.

Companies that provide online training in software and coding need to know which programming languages are prone by software developers or related persons. Or knowledge of which programming languages work more efficiently with which topics, is an advantage for these companies. Statistical information should be obtained on these and similar issues. In this case, machine learning studies, for text-based data must also be applicable for video or multimedia data. In addition to these firms, when relevant users search on the internet for coding or software, the search results should be correctly associated with the keywords entered by the users and reflected on the results. In this case, as mentioned, machine learning algorithms must be applied to video and multimedia data. As a result, the use of deep learning to classify video and multimedia data after image processing will be a productive step towards this need.

Deep learning is an artificial intelligence function that imitates the workings of the human brain in processing data and creating patterns for use in decision making. Deep learning is a subset of machine learning in artificial intelligence (AI) that has networks capable of learning unsupervised from data that is unstructured or unlabeled. Also known as deep neural learning or deep neural network. Deep learning techniques based on artificial neural networks pioneer machine learning in many fields such as medicine, physics and chemistry [3][4][5].

Based largely on widely studied methods such as backpropagation and gradient descent, deep learning allows ANNs with deep architectures (multiple hidden layers) to be trained using large volumes of input data. This is made computationally feasible with advances in storage and processing hardware, particularly the availability of a graphics processing units (GPU) and scientific computing libraries [6][2]. As a result, deep learning is very suitable for obtaining results from complex data sets such as images efficiently and quickly.

In this study, three different datasets of source code images were classified with Convolutional Neural Network (CNN) which is a deep learning algorithm and the results were comparatively examined.

CNNs use relatively little pre-processing compared to other image classification algorithms. This means that the network learns the filters that in traditional algorithms were hand-engineered. This independence from prior knowledge and human effort in feature design is a major advantage.

# **Literature Review**

In the early 1990s, neural networks had been very popular but had fallen out of favor in research until 2010s. Today, many new models have been introduced with the re-popularization of ANNs. CNN is one of these models.

In 1997, Lawrence et al. developed a method that could be considered as one of the first studies using CNN in the field of face recognition system [7]. In 2015, Li et al., proposed a CNN to solve the problem of face detection caused by large visual variations [8].

Differently from face recognition systems, there are plenty of handwriting or text recognition systems, which are using CNNs. In 2012, Wang et al., used CNN to recognize end-to-end text [9].

On the other hand, CNNs have used in NLP field. In 2014, Kalchbrenner et al., developed a CNN named Dynamic CNN to model natural language sentences [10]. Again in 2014, Kim showed that sentences could be classified using CNNs with little hyperparameter tuning [11].

# **System Architecture**

This project was developed with Python. Pytorch that is the open source machine learning library for python was used. PyTorch provides Tensor computation (like [NumPy](https://en.wikipedia.org/wiki/NumPy" \o "NumPy)) with strong [GPU acceleration](https://en.wikipedia.org/wiki/General-purpose_computing_on_graphics_processing_units). Fast.ai library was used to simplify training neural networks. Pip is a package-management system used to install and manage software packages written in Python and pip was used for this project. CUDA Toolkit that provides a development environment for creating high performance GPU-accelerated applications was used. jupyter notebook was used as program development environment.

In short, the system requirements are as follows;

|  |  |
| --- | --- |
| * Python – 3.6 * Pytorch – 1.1.0 * Fast.ai – 1.0.52 | * Pip – 19.1 * CUDA – 9.0 * Jupyter Notebook |

# **Method**

In this study, two approaches have been tried in this section, these approaches are explained step by step. As shown in the Table 1 below. Existing generic image classification model ResNet-34 is used. Data was loaded, model created and trained with Fastai library.

Table 1. Method stages

|  |  |  |
| --- | --- | --- |
|  | **First Approach** | **Second Approach** |
|  | Find learning rate (lr) | Train final layer |
|  | Train final layer using lr | Train all layers with 1 epoch |
|  | Train all layers using lr | Find learning rate |
|  | Train again final layer using lr | Train all layers using lr |

## **First Approach**

In this approach, the layers are trained according to a learning rate obtained at first. In order, the last layer, all layers and, again, the last layer was trained. the resulting error rate and accuracy values were examined.

1. Find learning rate (lr):

According to the results obtained from the studies, it can be said that the best learning rate is associated with the steepest drop in loss [12]. According to this claim, it is wise to determine the values in the range 1e-3 and 1e-2 as optimum learning rate for the graph in Figure 1.

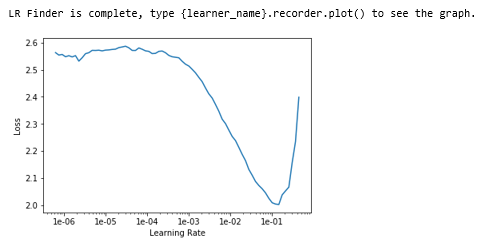


Figure 1. Learning rate graph sample

1. Train final layer using lr:

In the example in Table 2, the final layer was trained in 6 epoch and learning rate was taken as 2e-2 according to Figure 1.

Table 2. Sample training of final layer

|  |  |  |
| --- | --- | --- |
| **Epoch** | **Error Rate** | **Accuracy** |
| 0 | 0.677500 | 0.322500 |
| 1 | 0.630000 | 0.370000 |
| 2 | 0.170000 | 0.830000 |
| 3 | 0.116.250 | 0.883750 |
| 4 | 0.070000 | 0.930000 |
| 5 | 0.078750 | 0.921250 |

1. Train all layers using lr:

To increase the accuracy value or to decrease the error rate, all layers are trained with the learning rate obtained in the first step.

In the example in Table 3, the all layers were trained in 8 epoch and learning rate was taken as 2e-2 according to Figure 1.

Table 3. Sample training of all layers

|  |  |  |
| --- | --- | --- |
| **Epoch** | **Error Rate** | **Accuracy** |
| 0 | 0.066250 | 0.933750 |
| 1 | 0.068750 | 0.931250 |
| 2 | 0.060000 | 0.940000 |
| 3 | 0.056250 | 0.943750 |
| 4 | 0.060000 | 0.940000 |
| 5 | 0.052500 | 0.947500 |
| 6 | 0.055000 | 0.945000 |
| 7 | 0.052500 | 0.947500 |

1. Train again final layer using lr:

Unlike Table 2, in Table 3 the last layer is again trained in 5 epoch. This step was the last part of this approach. finally, the received accuracy values are evaluated.

## **Second Approach**

Unlike the first approach, this approach primarily trains the last layer and all layers with a default lr value. As a result of this process, based on the calibrated model, lr value is found and all layers are re-trained with this lr value.

1. Train final layer: As in the example shown in Table 2, the final layer is trained without the lr value given.
2. Train all layers with 1 epoch: All layers are trained in 1 epoch without the lr value giving.
3. Find learning rate:

lr value is obtained according to the calibrated model. According to Figure 2, optimum lr is determined in the range of 2e-3, 1e-6.

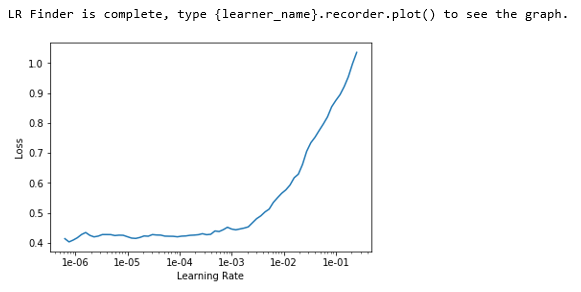


Figure 2. Learning rate graph sample from calibrated model

1. Train all layers using lr: According to the obtained lr value, all the layers are re-trained in 8 epoch.

# **Dataset Description**

3 different data sets were used in this project. The batch size is set to the same and 64 for all data sets. Default values are used to train and validate datasets (train = 0.8, validation = 0.2).

1. Dataset-1:

This data set contains 8 different programming language folders.

|  |  |
| --- | --- |
| * C * C++ * C# * Go | * Python * Ruby * Rust * Java |

This data set was obtained from the codes in the master branches of the most popular titles in these 8 languages in GitHub. The codes are downloaded from master branches and 1Mb of source files each was manually selected then combined to one bundle per programming language. 17-line samples from each of the bundles are rendered into simple white-text-on-black-background images. The image sizes are 299x299. For each programming languages there is 500 images. There is a total data.

1. Dataset-2:

This data set contains 6 different programming language folders.

|  |  |
| --- | --- |
| * C * C++ * C# | * Go * Java * Python |

This is a subset of the Zenodo-ML Dinosaur Dataset [Github] that has been converted to small png files and organized in folders by the language. The images are in the form of white code on black background. The image sizes are 80x80. For each programming languages there is 500 images. There is a total data.

1. Dataset-3:

This data set contains 6 different programming language folders.

|  |  |
| --- | --- |
| * C * C++ * C# | * Go * Java * Python |

This data set was obtained from the same source as the data set 2. that is, this data set consists of images with the same characteristics. Unlike the Dataset-2, there are 10,000 images for each programming language. So there is a total data.

# **Experimental Results and Conclusion**

1. Hyperparameters of 1st Approach:

|  |  |  |
| --- | --- | --- |
| * Final layer:   Learning rate: 2e-2  Epoch: 6 | * All layer:   Learning rate: slice(1e-5, 1e-4)  Epoch: 8 | * Final layer:   Learnin rate: 1e-3  Epoch: 5 |

1. Hyperparameters of 2nd Approach:

|  |  |  |
| --- | --- | --- |
| * Final layer:   Learnin rate: Default  Epoch: 6 | * Train all once:   Learning rate: Default  Epoch: 1 | * All layer:   Learning rate: slice(X, Y)  Epoch: 8 |

The accuracy values obtained by following the steps in 1st approach and 2nd approach from the datasets according to the above given parameters are shown in Table 3 and Table 4. It is observed that the model with the learning rate obtained from the calibrated model gives better accuracy than 1st approach as it is done in 2nd approach.

Table 4. Experimental results for 1st approach

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Dataset** | **Lr** | **Accuracy from Training of Final Layer** | **Lr (Slice)** | **Accuracy from Training of All Layers** | **Lr** | **Accuracy from Training of Final Layer** |
| Dataset-1 | 2e-2 | .91 | (1e-5, 1e-4) | .93 | 1e-3 | .92 |
| Dataset-2 | .92 | .93 | .94 |
| Dataset-3 | .91 | .93 | .92 |

Table 5. Experimental results for 2nd approach

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Dataset** | **Accuracy from Training of Final Layer** | **Accuracy from Training of All Layers** | **Lr** | **Accuracy from Training of Final Layer** |
| Dataset-1 | .67 | .83 | 2e-3, 1e-6 | .96 |
| Dataset-2 | .68 | .83 | 2e-3, 1e-6 | .95 |
| Dataset-3 | .85 | .93 | 1e-4, 1e-6 | .95 |

Accuracy variation according to epoch number was also observed. The accuracy value fluctuating up to a certain number of epoch, after a point, has started to be fixed.

Figure 3. Accuracy change according to epoch number

REFERENCES

[1] E. Linstead, L. Hughes, C. Lopes, and P. Baldi, “Exploring Java software vocabulary: A search and mining perspective,” *Proc. - Int. Conf. Softw. Eng.*, pp. 29–32, 2009.

[2] E. Linstead, A. Atchison, A. Bergh, P. Harnack, and J. Ott, “A deep learning approach to identifying source code in images and video,” *2018 IEEE/ACM 15th Int. Conf. Min. Softw. Repos.*, pp. 376–386, 2018.

[3] M. Raissi, “Deep Hidden Physics Models: Deep Learning of Nonlinear Partial Differential Equations,” *ArXiv e-prints*, vol. 357, pp. 125–141, 2018.

[4] E. Gawehn, J. A. Hiss, and G. Schneider, “Deep Learning in Drug Discovery,” *Mol. Inform.*, vol. 35, no. 1, pp. 3–14, 2016.

[5] P. Lakhani and B. Sundaram, “Deep Learning at Chest Radiography: Automated Classification of Pulmonary Tuberculosis by Using Convolutional Neural Networks,” *Radiology*, vol. 284, no. 2, pp. 574–582, 2017.

[6] J. Nickolls, I. Buck, M. Garland, and K. Skadron, “Scalable Parallel with CUDA,” *Queue*, vol. 6, no. 2, pp. 40–53, 2008.

[7] C. L. Giles, A. C. Tsoi, and A. D. Back, “Face Recognition: A Convolutional Neural-Network Approach,” vol. 8, no. 1, p. 16, 1997.

[8] H. Li, Z. Lin, X. Shen, J. Brandt, and G. Hua, “A Convolutional Neural Network Cascade for Face Detection,” *Lipoprotein Defic. Syndr.*, pp. 17–27, 2015.

[9] T. Wang, D. J. Wu, and A. Y. Ng, “End-to-End Text Recognition with Convolutional Neural Networks,” *Pattern Recognit. (ICPR), 2012 21st Int. Conf. on. [S.l.] [s.n.*, no. Icpr, pp. 3304–3308, 201AD.

[10] N. Kalchbrenner, E. Grefenstette, and P. Blunsom, “A Convolutional Neural Network for Modelling Sentences,” 2014.

[11] Y. Kim, “Convolutional Neural Networks for Sentence Classification,” 2014.

[12] L. N. Smith, “Cyclical learning rates for training neural networks,” *Proc. - 2017 IEEE Winter Conf. Appl. Comput. Vision, WACV 2017*, no. April, pp. 464–472, 2017.