James Landry

Abdulmateen Adebiyi

CS 6364.003 Artificial Intelligence

December 13, 2019

FashionMNIST Image Classification using Glimpse Actions

**Abstract**

I n this work, we applied recurrent neural network for image classification using glimpse action. Glimpse of images are now used because it reduces the total number of data needed for the image classification. Glimpse also ignore irrelevant part of the image. The overall idea of glimpses comes from human-eye perception and how humans can recognize objects by quick glimpses. Contours of images can give enough information for a human to accurately classify an object. We didn’t make use of convolution neural network for our glimpse classification because it is more computationally expensive and it scales linearly with the number of pixels. In our evaluation, we recorded a training accuracy of 80.05% on 47 Epochs and bets validation accuracy of 85.6% in 46 Epochs

1. **Introduction**

The idea of using glimpse action in image classification is from how the human eye works. It shows how the retina works by paying attention to a particular part of an object and blanking the rest out. If a person is looking at a building all other part of the building might be blurred and the eye only focus on the garden or any other part of the building.

Our model uses the idea such that our model only focuses on the interesting part of the object and use lower attention for the other part of the object. The model is then able to reduce the number of data needed in focusing on less interesting part of the image.

With the success story of convolution neural network in Image classification and object detection, images are now being classified seamlessly. Convolution neural network has also recorded very high accuracy in image classification. These successes come with some bottleneck which are the training with a large amount of data, running on multiple GPU.

These drawbacks have made researchers working on image classification to find a way in which the data that are needed for image classification can be reduced. The result is leveraging on how the human eye view objects. They noticed that the human eye does not process the entire object or a particular scene at once rather it focuses on a particular space to use the necessary object that is required at a particular time. Using this approach for our model will result in lower computational costs. It reduces the number of objects to be processed by the model and it also reduces the task complexity.

Our model uses a recurrent neural network that uses glimpses during the training process to quickly try and train the network. The network splits the set into training and validation sets

The recurrent model processes the input in a sequential way. It picks the different part one at a time for processing. It then combines the input to build an internal representation. A recurrent model was used because it has the ability to process the next image in a sequence. Our next object will always be the new part of the image that has the viewer’s attention.

1. **Data**

In the beginning of the project, the EMNIST dataset was a candidate for classification. The EMNIST dataset is a set of handwritten characters gathered from NIST Special Database 19 and converted to 28 x 28 pixel images[1]. Glimpse Actions on an image of that size would fair well on this dataset, as there is minimal noise in the image.

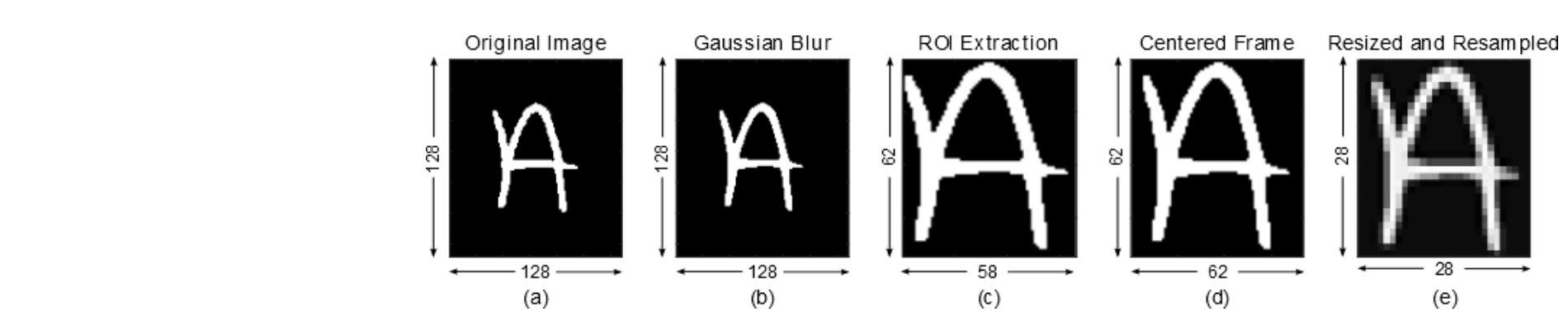


Figure (1): Example of EMNIST conversion

In the implementation of the project, pytorch torchvision library was used. Torchvision can fetch the EMNIST dataset. However, the link to the dataset was broken inside of the library. The link broke recently, within the last few months. The dataset would not cooperate after hours of manipulation of the library with alternate possible links.

Therefore, a solution needed to be found. Since the EMNIST was not available, another dataset within torchvision could be a possible dataset. Inside torchvision, the FashionMNIST dataset could be found. The FashionMNIST dataset is similar to the EMNIST dataset, but instead of handwritten characters, the dataset contains images of clothing[2].

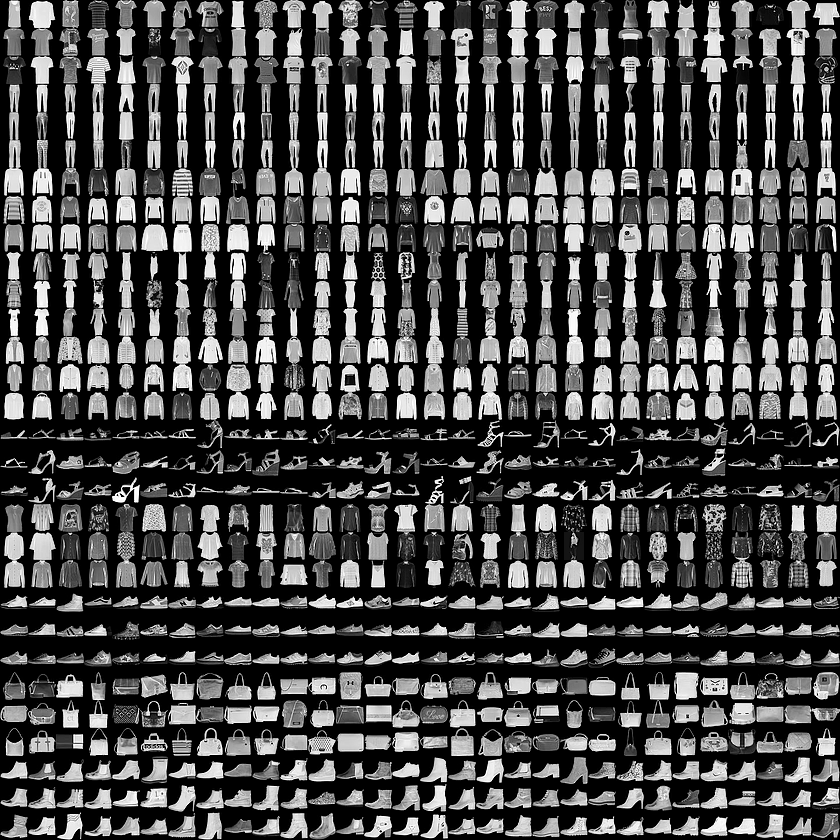


Figure (2): FashionMNIST example

These images are more difficult to classify, but it proposes a challenge for the project. FashionMNIST did indeed load properly. FashionMNIST is comprised of 60,000 training images and 10,000 test images. These images are spread across 10 classes, as to create a level of complexity other than whether the image is correct in a binary sense. It introduces the complexity of what each image is, instead of focusing on a single class.

1. **Experiment/Implementation**

For experiment and implementation, a model was found online. We are borrowing and adapting Kevin Zakka’s Recurrent Attention Model [3]. The model was initially set to run with the base MNIST dataset. Many variables were taken into consideration for the model. For example, how to tune hyperparameters and how long to train the model. The solution for this was a configuration file. This configuration file in the model details many hyperparameters and other details needed for successful implementation.

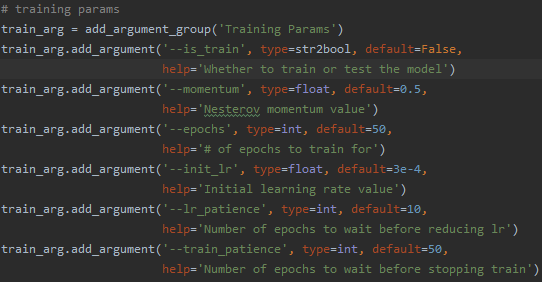


Figure (3): Screenshot of Training Parameters

In Figure 3, the training parameters are seen. There is a parameter for whether the program runs in training mode or testing mode. There are parameters for Nesterov momentum, number of epochs to run, initial learning rate, number of epochs to wait before reducing learning rate (if needed), and the number of epochs before stopping if the results start to stagnate. All of these values can be changed to affect results in training.

The next figure are the data parameters, used during the loading of the dataset. There are parameters for the proportion of training set for validation, number of images in a batch, number of subprocesses used for loading data, whether to shuffle the training and validation sets, and whether or not to show a sample set of images when the dataset is loaded.

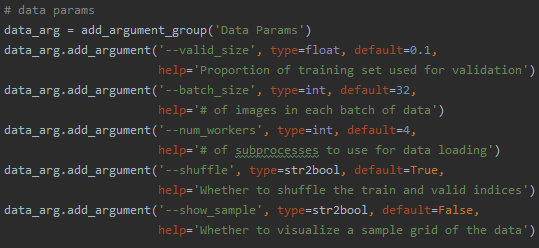


Figure (4): Screenshot of Data Parameters

In Figure 5 below, core network parameters are laid out. It is a simple set of two parameters: number of glimpses that will be used, and the hidden size of the Recurrent Neural Network.

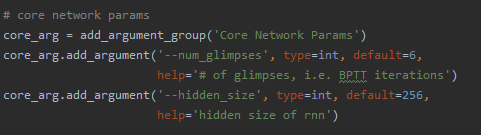


Figure 5: Screenshot of Core Network Parameters

Next figure contains the Glimpse Network parameters. These parameters how glimpses are taken. The first three parameters deal directly with the actual glimpses. The first is the size of the extracted patches at high resolution. In this case, each patch is 8 x 8 pixels in size. The second is the scale of successive patches, and the third is the number of patches per glimpse. The final two parameters deal with the hidden size of the loc and glimpse fully-connected layers in the network.

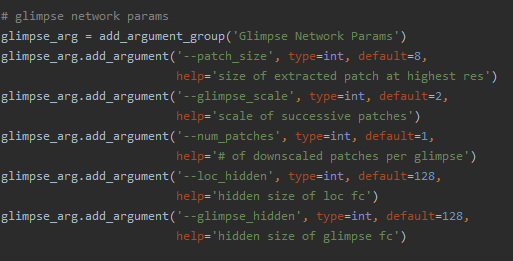


Figure 6: Screenshot of Glimpse Network Parameters

The last set of parameters that will be discussed are the miscellaneous parameters. Within the figure below, a few of these are directory parameters that will be used to save data. For instance, --data\_dir is the parameter for the file folder that will hold the data once it is downloaded. In this case of this project, data is stored to ./data. In order to expand the capability of the model, the option to run on a GPU is offered. Given the nature of the hardware used, this would be kept off, as the GPUs of the devices would not be able to withstand the model.



Figure 7: Screenshot of the Miscellaneous Parameters

In order to report progress in real time, Tensorboard is used for visualization. It reports the epoch, the current percentage of the training, along with time elapsed/estimated time remaining in the epoch.

1. **Conclusions**

Testing the model worked right away once the dataset was switched to FashionMNIST. With the settings seen above, running on a Toshiba Satellite L55W-C5236X laptop with 2.2 GHz dual core processing, the training ran for over an hour and a half, with and average of a little over one and a half minutes. The best epochs for training and validation accuracy can be seen below.

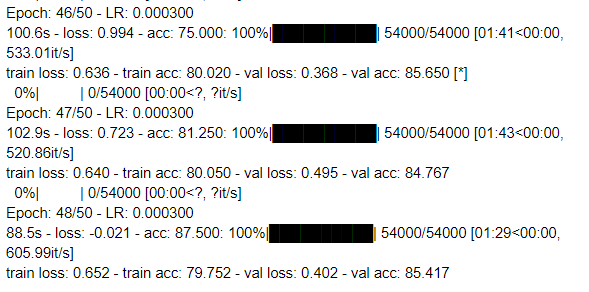


Figure 8: Screenshot of copied results from PyCharm

As seen in the image above, the best epoch for validation accuracy was Epoch 46, with an accuracy of 85.650%. The best epoch for training accuracy was the following epoch at 80.050%. These numbers are not terrible, but they can be improved. Hyperparameter tuning can be time consuming, especially if there is an increase in the number of epochs. In the model’s description, it was stated that paper accuracy of the MNIST dataset can be found within 30 epochs, even though it was initially set for 200 epochs.

After training was completed, the training parameter was turned off, and the model was run again. This time, it ran the test set of 10,000 images. The results are as follows:



Of the 10,000 images, the model trained with the parameters set correctly identified 8,433 images. For the nature of the Fashion MNIST dataset 84.33% is decent. In the testing module, I added a line to print metrics at each training interval. With the way the model is set, having a full summary with ROC and Precision Recall Curve proved to be difficult. Here is the first test interval metric.

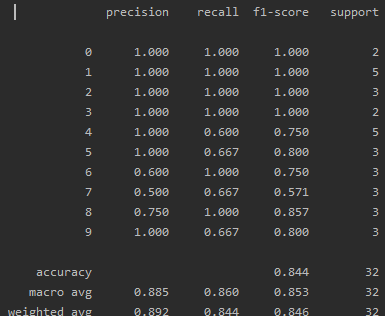


Figure (9): Screenshot of First Test Interval Metrics

In the very first batch of test images around 88-89% of the images were correctly identified. This trend continued roughly more or less, until testing was complete. There are over 100 metric groups like this one.

1. **References**

[1] Crawford, Chris. “EMNIST (Extended MNIST).” *EMNIST (Extended MNIST)*, Kaggle,

https://www.kaggle.com/crawford/emnist.

[2] Zalandoresearch. “Zalandoresearch/Fashion-Mnist.” FashionMNIST, 10 Aug. 2019,

https://github.com/zalandoresearch/fashion-mnist.

[3] Kevinzakka. “Kevinzakka/Recurrent-Visual-Attention.” GitHub, 12 Apr. 2018,

https://github.com/kevinzakka/recurrent-visual-attention.