Towards Fine-Tuning of VQA Models in Public Datasets

Miguel E. Ortiz[1], Luis M. Bergasa[1], Roberto Arroyo[2], Sergio Álvarez[2], Aitor Aller[2]

[1] Electronics Department, University of Alcalá (UAH), Spain
eduardo.ortiz@edu.uah.es, luism.bergasa@uah.es
[2] Nielsen, Spain
{roberto.arroyo,sergio.alvarezpardo,aitor.allerbeascoechea}@nielsen.com

Abstract. This paper studies Visual Question Answering (VQA) topic, which combines Computer Vision (CV), Natural Language Processing (NLP) and Knowledge Representation Reasoning (KRR) in order to automatically provide natural language responses to questions asked by users over images. A revision of the state of the art for this technology is done, and among the different approaches we select the model known as Pythia to build upon it, because it is one of the most popularized and successful approaches in VQA Challenge. We study the original Pythia implementation with the goal of finally applying it to eCommerce use cases. Recently, an exhaustive breakdown was done to the Pythia code by Facebook AI Research. We choose to use this more current framework after confirming that the two implementations had the same characteristics. We present the different modules of the FAIR implementation and how to train the model, proposing some improvements regarding the baseline. Different fine-tuned models are trained and with the best one an accuracy of 66.22% is obtained for the test set of the VQA-v2 dataset. An exhaustive revision of the quantitative results achieved for the most important experiments and some qualitative results for the best trained model are discussed.

Keywords: Computer Vision, Natural Language Processing, Knowledge Representation & Reasoning, Visual Question Answering, Artificial Intelligence.

1 INTRODUCTION

Recently, there is a renewed enthusiasm in multi-discipline Artificial Intelligence (AI) research problems. Part of this excitement stems from a belief that multi-discipline tasks are a step towards solving AI. What makes for a compelling "AI-complete" task? In [1], it is stated that in order to spawn the next generation of AI algorithms, an ideal task should require multi-modal knowledge beyond a single sub-domain and should have a well-defined quantitative evaluation metric to track progress. Within this context, Visual Question Answering (VQA) is a recently popularized topic, where different types of knowledge are combined. Mainly, VQA is based on Computer Vision (CV), Natural Language Processing (NLP) and Knowledge Representation Reasoning (KRR). VQA can be defined as a system that takes as input an image and a free-form, open-ended, natural-language question about the image and produces a natural-language output answer, as depicted in the examples presented in Fig.1.

Fig. 1: Examples of inputs and outputs for a standard VQA approach

There are different methods in the state of the art able to provide solutions to the different use cases related to the generation of automatic answers to questions performed over images. Most important ones are summarized in Table 1. For a deeper explanation, we refer the readers to the original paper of each method.

Table 1: Summary of VQA state of the art methods.

Method	Reference	Datasets and accuracies
Vanilla VQA	[1]	54.06 (VQA [1]).
Stacked Attention Networks	[2]	58.90 (DAQUAR [3]).
Faster R-CNN in VQA	[4]	63.15 (VQA-v2 [1]).
Neural-Symbolic VQA	[5]	99.80 (CLEVR [6]).
FVTA	[7]	66.90 (VQA-v2 [1]).
Bottom-up and Top-Down	[8]	65.32 (VQA-v2 [1]).
Pythia	[9]	72.27 (VQA-v2 [1]).
Differential Networks	[10]	68.59 (VQA-v2 [1]).
LoRRA	[11]	69.21 (VQA-v2 [1]), 27.63 (TextVQA [11]).

Most of the methods have been validated in the VQA-v2 dataset. This dataset contains 204,721 images from the MS COCO [12] and other 50,000 images from some abstract scenes. The MS COCO dataset has images depicting diverse and complex scenes that are effective at eliciting compelling and diverse questions. The dataset collects as well some realistic abstract scenes to enable research focused only on the high-level reasoning required for VQA by removing the need to parse real images. The dataset contains more than 1M questions with around 10M answers and it can be accessed from [13].

Some of the methods depicted in Table 1 provide open-source implementations. One of the most popular is the provided by Pythia [9]. This is a modular framework for vision and language multimodal research built on top of PyTorch [14]. This implementation is available in GitHub [15].

With this background, this paper introduces a VQA solution that builds upon Pythia and the fine-tuning of derived models. The VQA-v2 dataset is used to compare results with respect to other approaches of the state of the art using standard questions. The final goal of our solution is being subsequently applied in VQA use cases focused on eCommerce.

2 VQA IMPLEMENTATION

In this section, we describe the VQA approach developed with the aim of being subsequently applied to solve eCommerce use cases. Firstly, we do an overview of Pythia's method, which is the baseline of our proposal. Then, we present the implementation details of the method, explaining the different modules of its architecture. We finish highlighting the contributions regarding Pythia method.

2.1 Pythia's Overview

Pythia was the winner architecture in the VQA Challenge 2018 [9]. This model is based on the Bottom-up and Top-Down attention model [8], which is a network that processes the image and text separately, adding attention to image processing to get better features for the final classification. Pythia provides some changes over this model to increase its accuracy in the VQA-v2 dataset. The main improvements that Pythia introduces in the Bottom-up and Top-Down attention model are:

- **Weight normalization with ReLU.** This new method replaces the gated hyperbolic tangent activation in the image attention module for a weight normalization followed by ReLU to reduce computation charge. Features concatenation is also replaced by element-wise multiplication. With these improvements, the accuracy is increased to 66.91 % according to the Pythia paper.
- **Learning Schedule.** Pythia researchers decided to reduce the batch size, typically fixed to 512, and make use of the learning rate scheduler to first linearly increase the initial learning rate from 0.002 to 0.01, a process known as warm-up, and then reduce it every some thousand iterations. With this improvement they obtained an accuracy of 68.05 %.
- **Detectron and Fine-Tuning.** In this phase, a fine-tuning of the Detectron output (a detector based on feature pyramid network) [16] is made, which uses ResNeXt as backbone together with two fully connected layers (FC6 and FC7) for region classification. In this case, fine-tuning occurs at layer FC7. This boosted accuracy to 68.49%.
- **Data Augmentation.** Additional data is added from Visual Gnome and Visual Dialog datasets and mirroring is done to the images in the VQA-v2 dataset. The number of training iterations is also increased. With all this the accuracy is increased to 69.24 %. Grid feature and 100 objects. Grid level features are added to the bottom-up features, this is because the first features does not represent all the necessary information. These new features are obtained from ResNet152. Object-level features and grid-level features are separately fused with features from questions and

then are concatenated to fed to classification. In addition, adaptive object proposals (10 to 100) are no longer used and a fixed number of 100 object proposals for all images are used instead. With these two upgrades the accuracy increased to 70.01%.

– **Ensemble.** Two types of model ensembles are made. The first one uses the best single model and trains the same network with different seeds, and finally averages the predictions from each model to obtain an accuracy of 70.96%. In the second case, several trained models are used. These models are trained with different Detectron models and with/without data augmentation. This approach gives 72.18 % of accuracy.

A summary of the Pythia improvements on VQA-v2 are shown in Table 2. However, these improvements, present in the Pythia framework, except for the ensembles, report an accuracy of 66.7 % according to the current Facebook AI Research (FAIR) technical reference [15]. In this framework, all the upgrades mentioned before are codified and validation is carried out over the same dataset (test-dev). This accuracy difference can be due to the implementation of Pythia or the features are not exactly the same, or due to the numbers presented in the original implementation are overvalued. As our numbers are closer to the provided by FAIR and these numbers are more up to date, we will take as baseline for comparing the performance provided by FAIR.

Table 2: Pythia improvements on VQA-v2.

Model	test-dev
Bottom-up and Top-down	65,32
Weight normalization with RELU	66,91
Learning Schedule	68.05
Detectron & Fine-tunning	68,49
Data Augmentation	69.24
Grid feature & 100 bboxes	69.81
Ensemble	72.18

2.2 Implementation Review

As previously described, Pythia includes modules classified in three main sections: CV, NLP and KRR. Each of these sections can be seen working together in Fig.2. CV section is represented by Features and ImageEncoder modules. Questions are processed in NLP modules, which are WordEmbedding and TextEmbedding. KRR section receives the CV and NLP outputs in ImageEmbedding module to generate new highlighted features. Finally, text and image attention features are fused in the ModalCombineLayer module, which loads the data in the ClassifierLayer to get the final scores of answers.

As it can be seen, the implementation of Pythia does not perform the extraction of features, so it's necessary to obtain them from some external models in a previous stage. Pythia just processes as inputs the features and the questions of the subject.

Hereafter, main modules that make up the Pythia model will be explained.

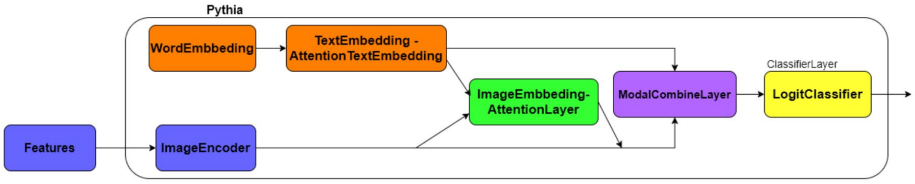

Fig. 2: Full Pythia Model.

The Pythia implementation receives as inputs the features of the MS COCO images from two different networks: Detectron and ResNet152. These features are the inputs of the first module of Pythia implementation (ImageEncoder).

ImageEncoder (CV) is a module that processes the last layer of the Detectron network, FC7 (see Fig.3). Here, Pythia loads the features of the FC6 layer of Detectron and trains FC7 with the rest of modules. FinetuneFasterRcnnFpFc7 module is a fully connected net that is in charge with the FC7 training process. For the grid features (coming from ResNet152), Identity is used. This module just sends the input to the output without any arithmetic operation. In case any other operation is needed, Identity module can be changed by the corresponding one. The same strategy can be done with FinetuneFasterRcnnFpnFC7. This procedure gives the possibility of choosing the feature extractor model that best fits to the specific application. In general, ImageEncoder works as a bridge between the features of the image (CV section) and Pythia, where we can add layers to the features that we receive if it is needed.

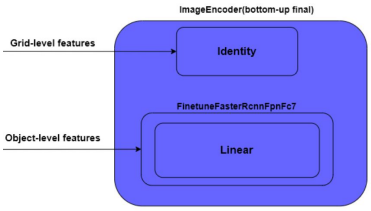

Fig. 3: Computer Vision section.

To process the questions (NLP section) two modules are used: WordEmbedding and TextEmbedding-AttentionTextEmbedding, as we depict in Fig. 4. WordEmbedding uses Glove to represent the features of the input text words. Words are represented as a vector of indexes, which are obtained from a vocabulary of 100k words. That means that it is necessary to transform each word of the text to a one-hot vector. The second module is an architecture that highlights the most relevant features of the whole text, and is where the Attention process is applied. As in most text processors, the first step is to use a LSTM (Long Short-Term Memory) net to extract features of the text, it must be an RNN (Recurrent Neural Network) like LSTM or GRU (Gated Recurrent Unit) because they keep information of the previous inputs, that is, the previous words of the current question. Then, outputs of the LSTM are forward to a CNN with the purpose of getting a better understanding of the features. In vanilla NLP, a global max pool is applied

to the output of the RNN. In VQA, it is necessary to increase the complexity of the model to get better features. To obtain these improved features the model uses Dropout, Convolutional, ReLu and Softmax layers. Dropout is a layer to avoid overfitting by disabling a percentage of the features. Features to be disabled are randomly selected in each iteration of the training. The role of the Convolutional layer is similar to the seen in the CV section, that is, to extract features with more relevance that can better represent the question. At the end of this model, a Softmax layer is used to obtain the probabilities of the features. The features with high probabilities are the ones that have more impact in the current question.

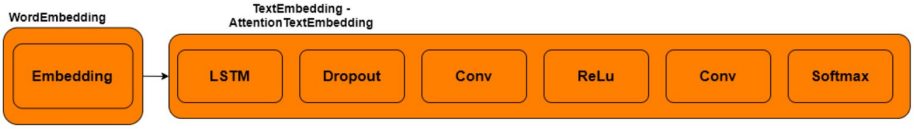

Fig. 4: Natural Language Processing module.

Again, Pythia framework lets the programmer change the Embedding and the AttentionTextModel models with different approaches if it is needed.

The output of ImageEncoder and AttentionTextEmbedding are processed by ImageEmbedding - AttentionLayer. It is in this layer where the KRR process is done and the TopDownAttention module is in charge with this process (see Fig. 5). This is the module where Pythia's first improvement with respect to Bottom-up and Top-Down attention model is carried out. Hyperbolic tangent is replaced by weight normalization followed by ReLu to reduce computation and the concatenation of the image and text features is replaced by an element-wise multiplication. These upgrades can be found in the NonLinearElementMultiply module that receives both features. New features are received for the LinearTransform layer which performs a linear with weight normalization operation. Finally, Softmax layer is applied to the output of LinearTransform to weight the features with probabilities from 0 to 1.

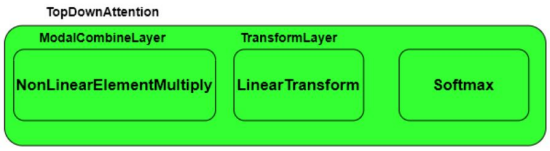

Fig. 5: Knowledge Representation and Reasoning module.

After obtaining the features with attention from images and text, they are merged using the NonLinearElementMultiply module depicted in Fig. 6. In this module, ReLuWithWeightNormFC performs a Linear with weight normalization followed by a ReLu layer where the Linear module is just a fully connected layer. This operation is applied to image, text and context features. Context feature is only used in the LoRRa model [11]. The output is element-wise multiplied in order to get image-question and

question-context features, and finally these features are concatenated. At the end of the module there is a Dropout layer to avoid overfitting.

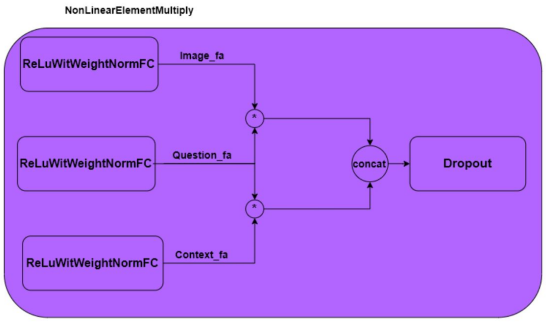

Fig. 6: Fusion module.

The output of the fusion module is introduced as input in the last Pythia module, the LogitClassifier, as we can see in Fig. 7. The scores of the answers are obtained in this module. Inside the LogitClassifier the input is connected to two different branches, one for text features and other for image ones. Each branch is formed by modules that we already know: ReLuWithWeightNormFC plus Linear. The difference between these two layers is the hidden dimension that returns ReLuWithWeightNorm. For text the dimension is smaller than for image. This is done to get the best possible features related to image and text. After this, the Linear layer returns the same two feature vectors of same size. Finally, both vectors are added to get the final output of the model.

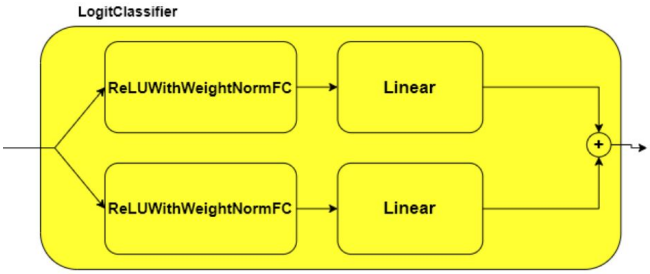

Fig. 7: Classifier module.

2.3 Contributions

Pythia's framework provides a configuration file to create new models and run it in an easy way. It offers a lot of modules that can be used for established applications or for new research. In this section we highlight the modifications introduced on the Pythia's baseline code in order to reach the best performance results.

- Pythia does not provide an user friendly representation of the performance metrics like accuracy or loss. Instead the framework offers a simple log file with all performance information. We have implemented a module that parses the configuration file and gets different performance graphs to easier data understanding. This new module works offline, once the training process is over.
- Changing the training strategies provided in the baseline. Multiple fine-tuning changes have been carried out according to the following ideas:
  - Change the general rule followed by Pythia in the training scheduler.
  - Put into practice long training with and without regularization.
  - To combine scheduler and regularization.
  - To use pre-trained models for new experiments.

3 TRAINING AND FINE-TUNING A VQA MODEL

In this section we describe the training process for fine-tuning VQA models. Firstly, we explain the different training and fine-tuning strategies we have followed. After that, we present the hyperparameters values we have assigned for an optimum performance. We finish explaining the used training loss functions.

3.1 Training and Fine-Tuning Strategies

To train the Pythia models, we used the configuration file that the framework offers. In this file we found all the needed hyperparameters for the training: optimizer parameters, size of the layers, path of the dataset that the model will use to train, etc. All these parameters can be classified into three sets: optimizer, training and model; which will be explained in the next section. Our training strategy is made up of 3 phases. In each of them we will set hyperparameters to get information about how the model performs to different configurations, focusing on different aspects of the training.

- **Using a scheduler in the training**. We did a lot of training processes with different scheduler steps, iterations and learning rates. Our first thought was that the change of learning rate should be later as the learning rate decreases, but following the results obtained from the training, this idea is not fulfilled. In general, accuracy improvements with this strategy were less than expected.
- **Using regularization in the training**. Weight decay and dropout were tested. The criteria to get the optimum value was mainly that penalization to the training accuracy was not too huge. After doing several tests we concluded that any value for Weight Decay less than 10^{-6} will drastically drop the accuracy of the model. In the case of Dropout we noticed that the accuracy decreases if we deactivate more than the 40 percent of the features. In general, accuracy with this strategy is worse than for the first approach.
- **Using pre-trained models in the training**. This strategy turned out to be the best. The pre-trained model used in this phase was the best one that uses a scheduler without regularization. This is because those models had better performance. In the post training the scheduler is no longer used and we add both regularization methods. With this setup the model was trained for 100,000 iterations reaching to increase the accuracy obtained from the previous strategies.

3.2 Hyperparameters

In this section we explain the values of the hyperparameters that have been used for fine-tuning with the aim of easily understanding the training processes. These hyperparameters are depicted in Table 3 and they can be grouped into three sets: optimizer, training and model.

Table 3: Hyperparameters.

Set	Hyperparameter	Values
Optimizer	Learning Rate	0.0045 to 0.03
Optimizer	Weight Decay	10^{-10} to 10^{-05}
Training	Iterations	20000 to 100000
Training	Batch Size	128
Training	Learning Rate Scheduler	True or False
Training	Learning Rate Steps	10000 to 30000
Training	Learning Rate Ratio	0.05 to 0.1
Training	Device	cpu or cuda
Model	Dropout	0 to 0.4

3.3 Training Loss

The problem that VQA tackles can be explained like a multi-label classification, because the model must choose between different labels, in this case answers, to solve the problem that involves the VQA problem. This means that the model must provide probabilities for a one-hot vector where each index represents an answer. For this kind of task a training loss based on Binary Cross Entropy with logits (BCELogit) is the perfect solution.

This loss function uses a sigmoid activation plus a Cross-Entropy loss. For each output of the model, the sigmoid layer is applied. This function returns the probabilities of each output of the model, where the output is the range 0 to 1. The mathematical expression of the sigmoid function is shown in Eq. 1.

$$Sigmoid(x) = \sigma(x) = \frac{1}{1 + exp(-x)} \tag{1}$$

The sigmoid function is presented in the equation of BCELogit depicted in Eq. 2 and Eq. 3. There we can see the output's model represented as x and the target represented as y. As this is a multi-label classification problem, BCE is applied to each of the labels. The c is the class/answer number and n is the number of the sample in the batch. After the calculations are done, the loss function obtains a list of values for each sample. It is in this moment when Eq. 4 is calculated. PyTorch offers two options for the final stage of the BCELogit: *sum* or *mean*. *Sum* just adds all the results obtained before to get the final error, meanwhile *mean* adds all the components and divide them by the number of samples. In PyTorch, the election between *mean* and *sum* is called reduction. For Pythia's implementation *mean* is used for the training loss as it is more representative than the other option for a loss function.

$$L_c = \{l_{1,c}, ..., l_{N,c}\}^T, l_{n,c} = -w_{n,c}[p_c y_{n,c} log\sigma x_{n,c} + (1 - y_{n,c}) log(1 - \sigma x_{n,c})] \quad (2)$$

$$l_c(x,y) = L_c = \{l_{1,c}, ..., l_{N,c}\} \quad (3)$$

$$l(x,y) = \begin{cases} \text{mean(L)} & \text{if reduction='mean'} \\ \text{sum(L)} & \text{if reduction='sum'} \end{cases} \quad (4)$$

4 EXPERIMENTS AND RESULTS

In this section we describe the experimental results we carried out to validate our VQA proposal. Firstly, we explain the used VQA dataset. After that, we study the used evaluation metrics and the experimentation we have followed. Finally, quantitative and qualitative results are presented.

4.1 Public VQA Dataset

Pythia's baseline model was trained in the VQA-v2.0 dataset. This dataset was used in the challenge that the VQA group runs anually [17]. In this challenge multiple models are presented to try to get the best accuracy, showing the best architectures that the participants could think of. Pythia was one of these models presented and the winner of 2018, with an accuracy of 72.18%. The main components of the VQA-v2 dataset, presented by [17] for the challenge, are the following:

- **Images**: there are 265,016 images in the dataset. These images are the combination of MS COCO dataset, which contains 204,721 images, and the rest are abstract images done by [17] with the purpose of enriching the information and trying to cover all the possible scenarios of the real world.
- **Questions**: Each of these images has at least 3 questions related to the information that contain the picture. This makes more than 1 million of questions for the whole dataset.
- **Answers**: Answers are provided for the questions. These answers were obtained from a process done with its workers, this ensures that the pool of answers is diverse. Each question has 10 answers, this makes a total of more than 11 million of answers.

All the information related to the process of the dataset creation can be read in [1].

The data is stored in two main files. The format of these files are JSON, that means that we can add all the needed information in key-value pairs. This allows us to manage the information in a very efficient way. The main files are:

- **InputQuestionFormat** is the file that contains relevant information about the images and the questions. In this file we can find the link between the two components mentioned before. The fields that this JSON file contains are:

  - **Info**: this is an object that contains information like year, version, description or a timestamp.
  - **Data type**: string type value that contains the name of the dataset used for the images.
  - **Data subtype**: string type value that specifies if the data is for train, test or validation.
  - **License**: contains the url and the license name.
  - **Questions**: it is a list formed by objects that contain the information of the images and the questions. Images and questions are represented as numbers, which help to map them when the data is loaded, increasing the searching speed.
- **AnnotationFormat** is the file that contains the link between questions and answers. This file has the same fields that the previous one, except for "Questions" and "Tasks" type. Instead, this JSON has a new key named "Annotations" that stores new information related to images, questions and answers The field annotations is a list of objects that store the main information of this file. Each annotation includes the following fields:
  - **Question type**: VQA questions are clustered by the first words of each question. This field gives the cluster this question belongs.
  - **Multiple choice answer**: this field is related to the most popular answers given.
  - **Image id**: this field contains the image identificator related to the question and the answers.
  - **Answer type**: Determines the nature of the answer.
  - **Question id**: with this field we obtain the correct question from the previous file
  - **Answers**: it is a list that stores 10 answers for each question. The answers have their own properties, which hereafter will be briefly explained:
    * **Answer**: this is a literal string that contains the response to the question.
    * **Answer confidence**: this field measures the accuracy of the answers.
    * **Answer id**: it gives a number that identifies the answers.

4.2 Evaluation metrics

To evaluate the performance of the Pythia model, the metrics that the framework uses are based on Accuracy and Loss.

Accuracy is a metric used to evaluate classifications problems. This metric measures the ratio between the number of correct predictions and the total number of input samples. The process to know if the prediction is correct or not is really simple. The expression that calculates this metric is shown in Eq. 5 The target is a one-hot vector that represents all the possible responses that the model can give, where the values of this vector are zero except for the index with the correct answer. Then, it is necessary to compare this vector with the output of the model. The output of the Pythia model is a vector where the correct answer is the index with the biggest value, this index must be the same as the target. To get this information it is necessary to apply a Softmax layer to the output. This layer will return a vector with real values where the total sum

is one. With this information we have a better understanding of the classification that the model has done.

$$L_c = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} \tag{5}$$

Pythia obtains two accuracy metrics: median and global avg.

- Median accuracy: apply the median operation over the accuracy returned from the batches
- Global avg accuracy: calculate the average from all the accuracy values returned for the samples.

Loss metric is just the value returned by the loss function. This function calculates how much the output's model fits the target. Interpreting these metrics we can know if the model generalizes correctly in the test data.

4.3 Experimentation

In addition to the Pythia implementation, [15] also offers the VQA-v2 dataset ready to load directly into the model. We have used this implementation for our experimentation. It is composed of three Python packages named training, validation and testing. These packages have the same format:

- *image_name*: This field is the same that was presented in Sec. 4.1. It stores the name of the image with the following format: $COCO\_[type]\_id$, where "type" refers to train, val or test. ID is number.
- *image_id*: This is the number that represents the image. This number is used in the *image_name*.
- *question_id*: Number that refers to a question..
- $feature\_path$: It is a path to the features of the image.
- *question_str*: Field that contains the question.
- *question_tokens*: It is a list where its elements are each word of the question.
- *answers*: List of size 10 that contains the answers.

The sum of the three packages is 1,105,904, which is the total number of questions that VQA-v2 dataset offers. The whole data are divided in three different sets to carry out a correct training process, according to the numbers shown in Table 4. As we can see we will use 40% of the samples for the training, about 20% for the validation of the model and the rest 40% for the testing of the proposals.

Table 4: Dataset division.

Data Type	Number of samples	Percentage
Training	443,757	40.12
Validation	214,354	19.38
Test	447,794	40.5

4.4 Quantitative Results

In the Loss graph we can see two curves that represent the loss of the training and the testing set. Taking a quick look to this graph we can observe if the model is overfitting or underfitting.

From the Accuracy graph we can get the same information as the loss graph. We can know when the model is overfitting when the two curves diverge and they are stuck in different accuracy percentages. Scheduler impact can be seen in this graph too, therefore, we can get similar conclusions.

As it was mentioned in section 3.1, experiments are focused on different parameters depending on what training strategy phase we are. However, there are some common parameters that don't change as:

- Batch Size: 128, due to hardware restrictions.
- Optimizer: Adamax, typical optimizer, which is explained in [18] the total number of experiments done is 44.

From all these experiments we choose the most representative for each training phase, as we depict in Table 5, where we add the accuracy for an easy comparison.

Table 5: Experiment grouped by training phases.

Training strategy phase	Experiments	Accuracy
Scheduler in training	Experiment 1	65.89
Regularization in training	Experiment 4	63.61
Pre-trained model in training	Experiment 11	66.22

In *Experiment*1 we tried to replicate the hyperparameters used in Pythia in their training where they got 66.7 % of accuracy in [15] and 69.81 % in [9], excluding ensambles because we can't reproduce it. The reason for this experiment is to establish a baseline model to compare with.

The training does not use regularization methods, thus, these hyperparameters are set to zero. The scheduler follows the same steps that mentioned in [9], with the same learning rate and the same numbers of iterations. The accuracy obtained in this training is less than reported in the above papers as we can see in the first experiment of 5 and 8.

We probe different changes in the training scheduler reaching only some slight improvements. After that, we checked some regularization methods but we did not obtain any improvement. However, we confirmed the fact that regularization over a pre-trained model increases the accuracy of the model. To confirm this improvement hypothesis, for this experiment we add 70k iterations more to the *Experiment*2 training which was the fine-tuned model of the first phase with better results. The final accuracy increased to 66.22%, reaching this model its best performance with this configuration.

4.5 Qualitative Results

In this section we test our best model with images and questions to check if the answers provided by the model are the correct ones, obtaining in this way some qualitative

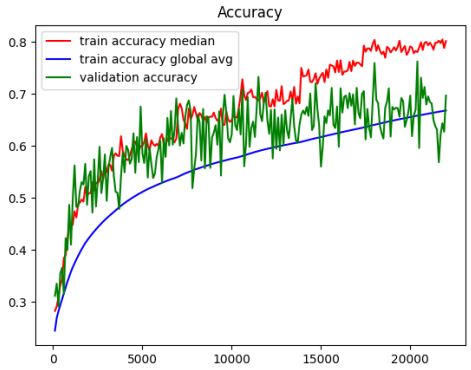

Fig. 8: Accuracy Experiment 1.

results about the working of our proposal. The first step is to load the pre-trained model, in this case corresponding to *Experiment* 11. The second step is to select the images and questions that the model will try to predict their answers for. The first question for the image is "is there some food?". As we can see in the Fig.9, the model's answer is positive with 99%. This shows that the model's performance with general questions is great.

For the second question we decided to focus more on one of the products presented in the image (fruits, vegetable, product of animals, dairy product). Fig. 10 shows the use case. The answers are still positive, but its accuracy decreased a little in comparison with the previous question. In the case of the product of animals and dairy products, the model's performance drops to 74% but still keeps the correct answer. This shows that the model can still recognize different products but it suffers if these products come from other sources.

Fig. 9: Qualitative results for a generic question.

5 CONCLUSIONS AND FUTURE WORKS

In this document we have reviewed the state of the art of the VQA technology and the different modules of the Pythia architecture. We confirmed that the framework that [15] offers is the same that the described in[9]. After that, we successfully tested the Pythia implementation following the [15] framework.

Fig. 10: Qualitative results for specific questions.

In the experiments we could see that the accuracy that we first obtained for our baseline (FAIR framework) was close to the reported results [FAIR, 2020]. Therefore, we began to fine-tune the hyperparameters and the training strategy of the model with the intention to obtain better results regarding our baseline. We managed to slightly improve our first numbers, but the most important was to obtain the necessary knowledge to face the next stage of the described project.

6 Acknowledgment

Authors want to thank to Nielsen for its funding in the development of this project. This work has been also funded in part from the Spanish MICINN/FEDER through the

Techs4AgeCar project (RTI2018-099263-B-C21) and from the RoboCity2030-DIH-CM project (P2018/NMT- 4331), funded by Programas de actividades I+D (CAM) and cofunded by EU Structural Funds.

References

1. S. Antol, A. Agrawal, J. Lu, M. Mitchell, D. Batra, C. Lawrence Zitnick, and D. Parikh, "Vqa: Visual question answering," in *Proceedings of the IEEE international conference on computer vision*, pp. 2425–2433, 2015.
2. Z. Yang, X. He, J. Gao, L. Deng, and A. Smola, "Stacked attention networks for image question answering," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 21–29, 2016.
3. M. Malinowski and M. Fritz, "A multi-world approach to question answering about real-world scenes based on uncertain input," in *Advances in neural information processing systems*, pp. 1682–1690, 2014.
4. D. Teney, P. Anderson, X. He, and A. Van Den Hengel, "Tips and tricks for visual question answering: Learnings from the 2017 challenge," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4223–4232, 2018.
5. K. Yi, J. Wu, C. Gan, A. Torralba, P. Kohli, and J. Tenenbaum, "Neural-symbolic vqa: Disentangling reasoning from vision and language understanding," in *Advances in neural information processing systems*, pp. 1031–1042, 2018.
6. J. Johnson, B. Hariharan, L. van der Maaten, L. Fei-Fei, C. Lawrence Zitnick, and R. Girshick, "Clevr: A diagnostic dataset for compositional language and elementary visual reasoning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2901–2910, 2017.
7. J. Liang, L. Jiang, L. Cao, L.-J. Li, and A. G. Hauptmann, "Focal visual-text attention for visual question answering," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 6135–6143, 2018.
8. P. Anderson, X. He, C. Buehler, D. Teney, M. Johnson, S. Gould, and L. Zhang, "Bottom-up and top-down attention for image captioning and visual question answering," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 6077–6086, 2018.
9. Y. Jiang, V. Natarajan, X. Chen, M. Rohrbach, D. Batra, and D. Parikh, "Pythia v0. 1: the winning entry to the vqa challenge 2018," *arXiv preprint arXiv:1807.09956*, 2018.
10. C. Wu, J. Liu, X. Wang, and R. Li, "Differential networks for visual question answering," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, pp. 8997–9004, 2019.
11. A. Singh, V. Natarajan, M. Shah, Y. Jiang, X. Chen, D. Batra, D. Parikh, and M. Rohrbach, "Towards vqa models that can read," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 8317–8326, 2019.
12. T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context. arxiv 2014," *arXiv preprint arXiv:1405.0312*, 2014.
13. "Vqa v2.0 download," 2020. https://visualqa.org/download.html.
14. "Pythorch," 2020. https://pytorch.org/.
15. "Fair," 2020. https://github.com/facebookresearch/mmf.
16. "Detectron," 2020. https://github.com/facebookresearch/Detectron.
17. "Vqa v2.0," 2020. https://visualqa.org/people.html.
18. D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.