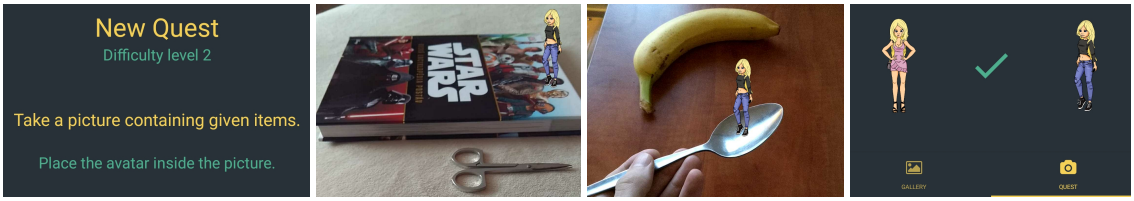
BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2018
http://excel.fit.vutbr.cz

# StageUp - A Photo-Challenge Mobile Application Using Deep Convolutional Neural Networks

Bc. Sebastián Poliak




**Abstract**

This paper introduces an idea for a mobile application, that challenges users to take a picture containing given items. The application is designed to have a functionality, which automatically and instantly evaluates the taken pictures. This is addressed as an image recognition problem, and is solved using multi-label classification. The other option of using object detection is also being discussed and compared. Additional features such as animated avatars, time constrains or sharing the image on social media are added to the application, in order to promote playfulness and user interest. The application as a whole is implemented using a client-server model.

The application is able to classify 29 classes of objects in the image , in a multi-label setting. The model has been evaluated using Precision-Recall and other custom metrics. The evaluated average precision equals to 0.68. At the confidence threshold, which has been set to 0.25, in 54% of cases there are not any false positives, and in 58% of cases there are not any false negatives.

The application is currently being user tested, and the possible issues are being addressed. After the end of the testing phase, this project will provide a unique mobile application, which will be publicly available. Its usage may also create a multi-label image dataset, which with the approval of the users, can be published for the community.

**Keywords:** photo-challenge mobile application — multi-label classification — client-server model

**Supplementary Material:** Demonstration Video

xpolia01@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

## 1. Introduction

We live in an era, where visual content is the main source of information exchange. Since a camera has become an inseparable part of a mobile phone and therefore always accessible, its usage for capturing everyday moments, usually followed by a share on a social media, has grown up enormously. From the observation of such pictures, it is possible to see that their content often includes quite ordinary things, for example a lunch or a dog.

This is where the mobile application developed in this project comes in. It aims to create a playful scenario, where the users are intentionally asked to take a picture containing certain items. Progressively, the needed items get more difficult to find, which requires some degree of interaction in the real life.

The reasoning behind why such an application could potentially be successful and could find its place on the mobile application market, can be seen by comparing it to the other state of the art applications. One of them is Pokémon Go [14], which might have some similarities in gameplay. Mainly the usage of camera

and the real life scene, where users are meant to catch the pokémon, instead of finding items in this case. In 2016, Pokémon Go received The Game Award for Best Mobile/Handheld Game.

As mentioned, the goal is to create a mobile application, which challenges users to take a picture containing certain items. In order to achieve that, there needs to be a functionality which evaluates the taken picture - determines whether it contains all the given items. Technically, it can be formulated as a classification or object detection problem. A proper solution should ideally have an accuracy close to human-like level. It should be computationally feasible, and work under different constrains such as time or operation cost, so it can be integrated with the rest of the application.

The application needs to also implement other features, that further promote playfulness and motivate users to take the pictures. Additionally, it needs to be wrapped in an intuitive and user-friendly interface. These parts might be considered as a bonus part, and can get evaluated by user testing and user feedback.

Currently, it seems that there is not any known mobile application on Google Play [12] nor The App Store [13], that would have the same format. However, there are applications that use camera and image recognition, or could be considered similar in some other ways. An example is the application Not Hotdog [15], which is able to classify a one object in the picture. There exist some photo-challenge applications, for example lifeshot [16], which challenges users to take a picture at the same time and share it, however, none of them use image recognition of any kind. Some similarities can be found with the application Pokémon Go, which has been already mentioned earlier.

Considering the state of the art of image recognition, the most common approach for the stated problem are convolutional neural networks. This part is closely described in Section 3 Image Recognition Background.

The approach that has been used to implement the mobile application as a whole, is using a client-server model, where the single roles are strictly divided between client and server. The image recognition part uses convolutional neural networks, and addresses both possible problem formulations - classification, as well as, object detection. The relevant methods for both formulations were implemented, and the more suitable one has been chosen, which according to different requirements ends up being multi-label classification, trained on COCO dataset.

The final result will create a playful and hopefully an interesting mobile application, which will be publicly available. Over time, the performance of quest evaluation - image recognition can improve, by using the taken pictures as an input to further train the model. With the approval of users, these images can be provided as an open source multi-label image dataset.

## 2. Building The Mobile Application

This section describes the design and all the parts, that are necessary in order to build the desired application. Overall, it can be divided into 4 basic functional roles, which are quest assignment, taking picture, image recognition and storage.

- Quest assignment is responsible for generating and assigning users a list of items, at certain moments in the game. The items are divided into difficulty categories, as well as context categories, needed to have a more diverse quests.
- The application needs to use a camera, which is the only way to complete a quest. The picture has to be taken directly from the application - it cannot be uploaded externally from a gallery. The reasoning behind this is to have a full control over user experience, as well as making sure that the picture is taken by the actual user.
- The image recognition part decides whether the assigned objects are in the picture. The solution for this part is further discussed in Section 4. Suitable Image Classifier.
- Storage is needed in order to store data such as user information, pictures, actual progress or user settings. While some of them might be stored on the device itself, using a database will have several advantages, for example, a user can log in and access the same state from any device.

The additional features that have been added to the application are time constrains on completing quests, placing user's animated avatar into the picture or the ability to share the pictures on other social media. Time constrains were introduced in order to add dynamics to the game. When the time runs out, a new set of items at the same difficulty level is generated. This behaviour might also be desirable, in case the user is not able to find the objects. The usage of animated avatars and their placement into the picture is meant to promote personalization and customization, and therefore improve the user experience. After the quest is completed, the users might be interested in sharing their final picture containing the placed avatar.

The user interface is designed in a way to quickly understand the idea and functionality of the application, without need to include any additional tutorials. The main part of the application is composed of 2 basic

tab views, which are Gallery and Quest. Gallery view contains the pictures of all completed quests, as well as the basic information about the current progress. On click, the pictures can be viewed in detail and shared. Quest view guides the user through the whole process of completing quests. After it's acceptance, a camera and the list of items appears. Whenever a picture is taken, the items get evaluated, showing a check or cross sign for every item in the list. If all the items checks the quest is completed, and an avatar can be chosen, moved around and placed into the picture, which is the last part of the view. On top of both views, there is a drawer navigation, where sections such as About or Settings could be placed.

### 2.1 Client-Server Model

As mentioned in the Introduction, the approach that has been used to implement the application as a whole, is based on a client-server model.
The client, in this case is the application running on a mobile device, is responsible mainly for taking a picture, communication with the server, and user interaction through the user interface, together with presentation of received data. The image recognition, as well as the functionality that generates and assigns quests to the users, are handled by the server. Another part of the server is the database.
There would be an option to have all the functionality contained directly in the mobile application, which is technically feasible. However, the choice of the client-server model has been made based on the advantages it brings, which are

- Flexibility - the game logic can be easily modified and deployed during production. The changes on the server are instantly distributed to the clients, without any need to update the application. This offers an opportunity to quickly create new quests, for example during seasonal events such as Christmas or Easter. Another advantage is that the object detection can gradually improve and change without user's notice.
- Performance - all the computation is done on the server, and therefore, the client's device needs to use very little computational power. This results in longer battery life, because, mainly the image recognition part could drain a lot of battery. The system requirements for the device are also lower.
- Feedback - the information about the actions and state of the users is available, and can be used to get an idea about their behavior in the game. The images sent to the server can be analyzed and used as a new input to further improve the image recognition model.
- Platform independence - the same server can be used for all the different mobile platforms.

These disadvantages have been also considered

- Network dependence - the internet connection is needed in order to communicate with the server. This could be problematic if the quest items are set somewhere outside, and it would be solved if the game logic was present in the device itself and the application could run offline. Nowadays, however, most of the users have a data connection from their telecommunication provider. Another possibility would be to allow users taking a picture offline, and submitting it for the evaluation later on, when the connection is available.
- Operation cost - running the computations on the server can get costly, especially if a GPU is used, which would be an ideal scenario for the inference of image recognition.
- Maintenance and scaling - the game logic of all the clients is centralized. The number of clients can grow over time, and the server has to be able to handle multiple incoming requests at once, be safe, reliable and always accessible.

## 3. Image Recognition Background

This section is devoted to the core part of the application - Image Recognition, with the main focus on convolutional neural networks, whose state of the art is further discussed.
A convolutional neural network (CNN, or ConvNet) is a class of deep, feed-forward artificial neural networks that has successfully been applied to analysing visual imagery. The popular tasks it has been applied to include identifying faces, objects or traffic signs, as well as powering vision in robots and self-driving cars. [8]
In general, the architecture of a CNN might be divided into two parts - feature extraction from image and classification. The feature extraction part usually consists of convolutions, rectified linear units (ReLU) and pooling layers. The classification part consists of fully connected layers, corresponding to the architecture of the regular neural networks.
The tasks solved by CNNs are most often formulated as a classification problem, but it is also capable of solving prediction, regression, or most recently image generation problems. In our case, the main focus is put on image classification.
An image classifier takes an image and outputs a set of scores, from which according to the representation, a

given class can be determined. There could be for example a dog-cat classifier, that takes an image and classifies it as a dog or a cat. However, in a scenario where both dog and cat are present in the image, the common approach fails. There needs to be a multi-label classifier, which has a different architecture. In case of CNNs, the difference is mainly in the final layer, where the activation function needs to be changed, usually from softmax [18] to sigmoid [17] function.

Along with the class of the object, there are ways of finding its location in the image. This is known as an object detection, or multi-class object detection, in case multiple kinds of objects are detected in a single image. The location of the object is represented as a rectangle called bounding box. Object detection can be formulated as a classification problem where the windows of fixed sizes are taken from an input image at all possible locations and proposed to an image classifier. Although it sounds like a simple concept, it gets complicated by the fact that the objects can have different sizes and aspect ratios, and therefore additional methods have to be used. There are several architectures for object detection, which are based on different principles, and vary in their speed and accuracy. The ones that have been considered in this project are Faster R-CNN [1], which is characterized by a high accuracy, and Single Shot Detector (SSD) [2], which is known by its trade-off between speed and accuracy.

To train an image classifier, there needs to be a set of input images with their corresponding outputs. The output contains the classes found in the image, and can also include the given bounding boxes. These sets are called image datasets. Many of them are publicly available, and may vary in the context of the images. In our case, the context are the basic objects, that can be found in the real life. The possible datasets that would fit this description and could be used are The Open Images dataset [5], ImageNet [6] and COCO: Common Objects in Context [4].

## 4. Suitable Image Classifier

This section discusses the process of building and choosing a suitable image classifier, which fits the needs of this project. As discussed in Section 3 Image Recognition Background, there are several possibilities that could achieve the same goal, under different circumstances. The conceptual difference is mainly between multi-label classification and object detection, while both can use different methods and be trained on different datasets.

Multi-label image classifier using Inception V3 model

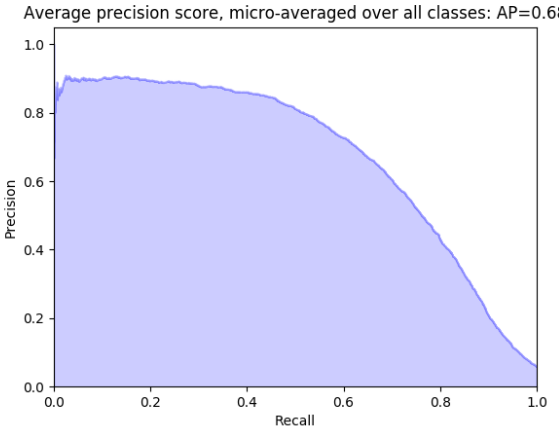



**Figure 1.** The precision-recall curve of Inception V3 model, trained as a multi-label image classifier on 29 classes from COCO dataset, reaching the average precision score of 0.68 over all classes, on the validation set.

[3], had been initially trained on The Open Images dataset, from which a subset of 20 classes, according to the desired gameplay of the application was selected. Although its accuracy on validation part of the dataset reached more than 0.9, it failed to properly classify all the objects in the multi-label setting. This was caused by the fact, that the dataset rarely contained images containing multiple objects from the selected subset at once, and therefore, the model was only able to classify a single object in the image. Considering this issue, a different dataset had to be used. COCO dataset was more promising in this case, because its images often contain 2-5 of the selected objects at once. A subset of 29 objects from this dataset was selected in a similar way.

The trained model was evaluated on the validation part of the dataset, using Precision-Recall metric [7] and reached the average precision of 0.68 (mean average precision (mAP) of 0.69) over all the selected classes (see Figure 1). The precision for each class varied from the highest 0.98 for tennis racket, to the lowest 0.39 for spoon (see Table 1). This metric could help in setting the threshold for recognizing the objects in the game, however, another custom metric was created, which is more suited for this purpose (see Figure 2). It is based on the number of false positives and false negatives per image in a multi-label setting. Considering the gameplay of the application, false positive represents the case when the application recognizes the object that is not in the picture - user is able to fool the application, while false negative represents the case when the object is in the picture and does not get recognized. By increasing the threshold, the number of false pos-


**Figure 2.** A graph showing the percentage of cases given the number of false positives and false negatives, at threshold of 0.25. The metric has been evaluated using the multi-label classifier on the validation part of COCO dataset, taking into consideration only the 29 trained classes.

itives decreases, while the number of false negatives increases. The threshold has been experimentally set to 0.25, where both false positives and false negatives are in an acceptable range. At this threshold, in 54% of cases there are not any false positives and in 58% of cases there are not any false negatives. The rest mostly contains one false positive or false negative, both at 30% of cases. Additional evaluation would be needed on the images directly from the game, which might have some characteristics compared to COCO images. However, at this moment, there is not enough of such images for the evaluation to be objective (see Section 5 User Testing).

Another approach that was used is to train Faster R-CNN and Single Shot Detector (SSD) object detectors, using Object Detection API [11]. Although knowing the location of the objects is not necessary for the designed functionality, the usage of a different method for classifying the objects in the image may differ in performance from the multi-label classification. Since we deal with object detection, the training images need to include the classes, as well as the bounding boxes of the objects in the image. COCO dataset has the bounding boxes available, and has been used for this purpose. Faster R-CNN reached mean average precision (mAP) of 0.54@0.5IOU, and Single Shot Detector reached mAP of 0.45@0.5IOU, where IOU stands for Intersection over Union [9]. The Intersection over Union decides, when the given object is detected, based on the ratio between area of overlap of ground-truth bounding box with predicted bouding box, and their union. By decreasing the value of IOU, there is a possibly to increase the mean average precision of classifying objects in the image, but it has not been evaluated in this scope.

Considering the discussed options, multi-label classification has been chosen for the purpose of this application. The object detection has been disregarded mainly because of its high computational complexity - on GPU, the speed of Faster R-CNN reaches 5 fps, and the speed of SSD reaches 59 fps. SSD could potentially be considered and used, however, the operation costs of using a GPU on the server are high, and running the inference on CPU is unacceptable. Another advantage of multi-label classification is that its input images for the training do not need the bounding boxes around the objects, and therefore, the model can be easily scaled. Comparing the mean average precision of the methods, the multi-label classification model reached mAP of 0.69, while the object detection models reached mAP of 0.54@0.5IOU and 0.45@0.5IOU.

## 5. User Testing

So far, the application has been given to 10 users, in order to test the concept and get a feedback about the user experience. The information about their progress in the game, as well as the taken pictures were gathered in the database. Additionally, the users were personally asked about their overall experience, possible issues or the ideas.

The main part that has been taken into consideration and already addressed is having quests as a list of 3 items, which caused several complications. It was difficult to put all of them into one picture, since they may vary in size, and the camera frame in the application has a square shape (this was used in order to not have to resize the image before CNN inference, so it can hold its proportions). Additionally, even when the users managed to fit all the items into the picture, the recognition part did not work as planned, and usually only correctly classified 2 out of 3 items. This was mostly caused by the confidence threshold for approving items in the image. The confidence for the items was often spread out, and the third item was usually below the threshold. Lowering the threshold, which was already quite low, would cause having more false positives, which is undesirable. Therefore, the decision to have only quests of 2 items was made. An advantage it brings is having more quests along the way.

The user interface has been tested as well, mainly measuring the time from the first log in into the application, to the time a user gets to the Quest view and generates his first quest. This was on average 13 seconds.

Considering the fact that the users have not seen the application before, together with the time it takes to read the quest instructions, this result seems to be satisfactory.
The log in has been made using Facebook authentication [10], which had some negative comments from the users, however, it is still more convenient than having a manual registration. The authentication cannot be removed, because it is needed in order to have a unique id in the database, and to be able to access the profile from the other devices. Other possibilities of log in could be provided.
As mentioned, all the pictures taken by the users were saved in the database, with the intention to further evaluate and improve the model. Most of them, however, contain the objects used in the first few levels the users played through, and therefore, it is not appropriate to use them, yet. Currently, the testing is still going on.

## 6. Conclusions

This paper discussed the design, the background, and the process of building a photo-challenge mobile application, while addressing the reasons for most of the decisions that have been made. It focused on the core part, image recognition, as well as on the mobile application itself.
The core part of image recognition has been solved by training a multi-label classifier on a subset of 29 classes of COCO dataset, and evaluated on its validation part using Precision-Recall and other custom metrics. The evaluated average precision equals to 0.68. At the confidence threshold, which has been set to 0.25, in 54% of cases there are not any false positives, and in 58% of cases there are not any false negatives.
The mobile application can be considered unique in its way, and will be publicly available. Other than hopefully being interesting for the users, its usage may create a multi-label image dataset, which with the approval of the users, can be published for the community.
The actual state of the application can be considered as a minimum viable product. The main goal right now is to deploy the server to the production environment, and put the application on Google Play. By having a larger feedback, additional features, objects and other changes could be introduced. If the application gets popular, there is a possibility of monetizing and building it further on.

## Acknowledgements

I would like to thank to my supervisor Ing. Jakub Sochor, for the guidance in this project, mainly in its core part - image recognition.
Additionally, I would like to thank to Ing. Rastislav Mrazik, for the creation of animated avatars, used to enhance the gameplay, and to Bc. Zdeno Olšovský, for new ideas and discussions about the mobile part of the application.
Last but not least, I would like to thank to my family and friends, for the overall support during this process.

## References

[1] REN, Shaoqing, Kaiming HE, Ross GIRSHICK and Jian SUN. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks.* 2016.

[2] LIU, Wei, Dragomir ANGUELOV, Dumitru ERHAN, Christian SZEGEDY, Scott REED, Cheng-Yang FU a Alexander C. BERG. *SSD: Single Shot MultiBox Detector.* 2016.

[3] SZEGEDY, Christian, Vincent VANHOUCKE, Sergey IOFFE, Jonathon SHLENS a Zbigniew WOJNA. *Rethinking the Inception Architecture for Computer Vision.* 2015.

[4] WOJNA, Zbigniew, Michael MAIRE, Serge BELONGIE, et al. *Microsoft COCO: Common Objects in Context.* 2015.

[5] Krasin I., Duerig T., Alldrin N., Ferrari V., Abu-El-Haija S., Kuznetsova A., Rom H., Uijlings J., Popov S., Veit A., Belongie S., Gomes V., Gupta A., Sun C., Chechik G., Cai D., Feng Z., Narayanan D., Murphy K. *OpenImages: A public dataset for large-scale multi-label and multi-class image classification, 2017.* Available at https://github.com/openimages

[6] DENG, Jia, Wei DONG, Richard SOCHER, Li-Jia LI, Kai LI and Li FEI-FEI. *ImageNet: A Large-Scale Hierarchical Image Database.* 2009.

[7] Precision and recall. *In: Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2018-04-08]. Available at: https://en.wikipedia.org/wiki/Precision_and_recall

[8] Convolutional neural network. *In: Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2018-04-08]. Available at: https://en.wikipedia.org/wiki/Convolutional_neural_network

**Table 1.** The table shows the average precision (AP) per object class, for both methods and used models, discussed in Section 2 Suitable Image Classifier. The items marked with * are missing values, caused by the different selection of objects classes, used to train the models.

| | Multi-label classification | Object detection | |
|---|---|---|---|
| object class | Inception V3 AP | Faster R-CNN AP@0.5IOU | Single Shot Detector AP@0.5IOU |
| bicycle | 0.68 | 0.47 | 0.30 |
| fire hydrant | 0.81 | 0.78 | 0.69 |
| cat | 0.89 | 0.90 | 0.86 |
| dog | 0.87 | 0.82 | 0.72 |
| backpack | 0.57 | * | * |
| tie | 0.76 | 0.43 | 0.32 |
| frisbee | 0.82 | 0.76 | 0.55 |
| skateboard | 0.91 | 0.69 | 0.54 |
| tennis racket | 0.98 | 0.68 | 0.49 |
| cup | 0.61 | * | * |
| fork | 0.66 | 0.36 | 0.24 |
| knife | 0.48 | * | * |
| spoon | 0.39 | * | * |
| banana | 0.72 | 0.33 | 0.29 |
| apple | 0.51 | 0.21 | 0.16 |
| orange | 0.61 | 0.35 | 0.32 |
| broccoli | 0.78 | 0.40 | 0.29 |
| pizza | 0.83 | 0.64 | 0.54 |
| donut | 0.64 | 0.45 | 0.38 |
| cake | 0.70 | 0.38 | 0.39 |
| chair | 0.71 | * | * |
| tv | 0.82 | * | * |
| laptop | 0.77 | * | * |
| cell phone | 0.51 | * | * |
| book | 0.57 | * | * |
| clock | 0.76 | 0.69 | 0.55 |
| scissors | 0.44 | 0.32 | 0.38 |
| teddy bear | 0.69 | 0.60 | 0.56 |
| toothbrush | 0.53 | * | * |


[9] Jaccard index. *In: Wikipedia: the free encyclopedia* [online]. San Fran- cisco (CA): Wikimedia Foundation, 2001- [cit. 2018-04-08]. Available at: https://en.wikipedia.org/wiki/Jaccard_index

[10] Facebook Login. *Facebook for developers* [online]. [cit. 2018-04-08]. Available at: https://developers.facebook.com/docs/facebook-login

[11] Tensorflow Object Detection API. *TensorFlow Models* [online]. [cit. 2018-04-08]. Available at: https://github.com/tensorflow/models/tree/master/research/object_detection

[12] Google Play. *Google Play* [online]. [cit. 2018-04-08]. Available at: https://play.google.com

[13] The App Store. *The App Store* [online]. [cit. 2018-04-08]. Available at: https://www.apple.com/lae/ios/app-store/

[14] Pokémon GO. *Google Play* [online]. [cit. 2018-04-08]. Available at: https://play.google.com/store/apps/details?id=com.nianticlabs.pokemongo&hl=cs

[15] Not Hotdog. *The App Store* [online]. [cit. 2018-04-08]. Available at: https://itunes.apple.com/us/app/not-hotdog/id1212457521?mt=8

[16] lifeshot – Photo Challenge App. *Google Play* [online]. [cit. 2018-04-08]. Available at: https://play.google.com/store/apps/details?id=com.keepinmind.lifeshot

[17] Sigmoid function. *In: Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2018-04-08]. Available at: https://en.wikipedia.org/wiki/Sigmoid_function

[18] Softmax function. *In: Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2018-04-08]. Available at: https://en.wikipedia.org/wiki/Softmax_function