Deep Learning Project Report

Multiclass Image Classification of Quick Draw dataset

Daniel Philippi (m20200733@novaims.unl.pt)
Mario Rodríguez Ibáñez (m20200668@novaims.unl.pt)
Doris Macean (m20200609@novaims.unl.pt)

Our team has been asked to explore the implementation of various Deep Learning concepts within the context of some problem. We have defined a multiclass image classification problem for exploration. Our proposal is to develop a Convolutional Neural Network (CNN) with tuned hyperparameters to obtain the best performance on our image dataset. We will simultaneously develop a Generative Adversarial Network (GAN) to develop synthetic images which will also be fed to our network. Our objective is to see if data augmentation by means of a GAN can help to improve the performance of our CNN.

The proposed system will have real and synthetic images as input and output the image class. This structure is simulating the experience of not having sufficient data to train a good model and requiring the generation of artificial images that accurately mimic the image classes.

1 Data Understanding

The purpose of our project is to run several experiments to better understand the functionality of each parameter and compare several different configurations. For this reason, we have selected a simple set of hand drawn images from Google's Quick Draw application [1]. The images are grayscale and as such, have only one channel. The low 28 x 28 resolution of the images allows for faster processing. In addition, the dataset has enough images to be able to evaluate the impact of varying sample sizes.

To develop an interesting and challenging problem, and introduce some complexity to our problem, we chose a selection of 11 classes of images that we considered to be similar. The selected classes are the following large mammals: Camel, Cow, Elephant, Giraffe, Horse, Kangaroo, Lion, Panda, Rhinoceros, Tiger, and Zebra. These classes have similar shapes and require the trained network to find more elaborate patterns to distinguish them.

2 Approach

Our suggested approach is to initialize and finetune the CNN and the GAN independently, using the same data. Once the optimal model architecture and parameters have been chosen, the artificial images generated by the GAN will be fed to the CNN, together with the original set of images, to train the CNN. The GAN will essentially be functioning as a data augmentation tool. We will assess the ability of data augmentation to improve model performance and compare the performance with and without synthetic data.

One challenge that we expect to see is the inability of the model to correctly classify all the classes. As the images are hand drawn, there is an added challenge of interpreting human error. Understanding how to work with images in numpy format, as well as aligning the keras elements to ensure they are fed the appropriate object, is another difficulty that we foresee. In our process, we evaluated the configurations that we felt were most interesting. Our team took a systematic approach to testing the classifier models – one parameter at a time, as well as in combination with one another.

Training the GAN models, we could expect the difficulty of having to change parameters and/or architecture simultaneously in the discriminator and the generator. Alternatively, we could be faced with the challenge of having to identify the training process of the combined model as stable, mode collapse, or convergence failure. These last two behaviours occur when one of the parts (generator

or discriminator) performs much better than the other, leading to a general bad result.

To be able to test several configurations, we developed a custom-built framework that allows for convenient and fast modification of the data being fed to the model, the model architecture as well as the training parameters. This structure provides instant insights and allows us to track experiment results.

The process our team has taken is to iteratively build, run and evaluate experiments, designed to answer certain sub-questions. This has taken the form of posing some question of interest, translating the question into an experiment, and running several instances of the experiment to be able to compare the results. Experiments will be run on smaller amounts of data which will result in lower values of accuracy. Our intention is to use this process to understand what parameters improve performance and thus, are prioritizing efficiency. A final model with the optimal configurations will be selected for further evaluation.

3 Dynamic Framework

Loading and Data Preparation

Data was loaded by downloading and caching the NumPy arrays of the original images. Pixel values were rescaled from a range of [0,255] to [0,1], and images were reshaped to an output shape of 28 x 28 x 1. Class names were mapped to integer values and the labels were one hot encoded to obtain a binary class matrix. A select number of images per class was specified and random samples of that size were extracted. Images were then shuffled to ensure that classes were randomly distributed. The data to be used for training is split into train, validation, and test sets. Data generators for each of the subsets were built and used in training the CNN models. The image data preprocessing takes place dynamically so that changes can be easily implemented to allow for adjustments in the number of images, batch size and the inclusion of data randomization or augmentation. This organization was employed to allow for the inclusion of the GAN's artificial images.

CNN

Model Architecture and Compilation

Several models were defined and named according to their architecture. Our process was created in such a way that the name of the model is entered into the model configurations, along with the loss function (i.e., categorical cross entropy), validation metric (i.e., categorical accuracy) and the optimizer, including its specific parameters (i.e., Adam, with learning rate set to 0.01). As with the previous section on data preparation, the ability to modify these model configurations dynamically has allowed for efficient testing of our models.

Training Parameters

Similarly, the training configurations were set in such a way that they were easily altered and allow for quick and efficient adaptations of the data. Our team has implemented callbacks that automate the learning process, improve efficiency, and reduce overfitting. Early stopping is used to stop the training process when the validation loss does not improve for a predetermined number of epochs (i.e., patience). In addition, the number of epochs can be stipulated; however, as the early stopping is in place, the selection of this parameter becomes less crucial to the training process. The reduce learning rate plateau (lr-plateau) [2] callback also provides some automation by modifying the learning rate when the validation loss has not improved for a set number of epochs. Setting the early stopping patience to a multiple of the lr-plateau patience allows for the possibility of adapting the learning rate to further improve validation loss before reaching the final early stopping criteria.

Model Evaluation

To evaluate the performance of our models, we have chosen a few measures to track throughout the training process. With a multiclass classification problem and binary labels, the recommended loss function is categorical cross entropy. Similarly, the recommend metric is the categorical accuracy [3], which we will use to evaluate the live training process. Both will be monitored and saved on every epoch to analyze the learning capabilities of the model, as well as signs of overfitting.

After the conclusion of training, the model will be evaluated on unseen data. Here, we will look at

precision and recall, both per class and on average over all classes. Precision will let us know how much we can trust the model when it predicts a certain class. While recall gives insights on how well the classes are represented in the prediction.

GAN

Model Architecture and compilation

The chosen architecture for the GAN of this study was the Auxiliar Classifier – GAN (AC-GAN) [4], which essentially is a regular GAN with one slight alteration in the generator, and another in the discriminator. The modification in the generator consists of the addition of the label of the image that we want to generate, which is encoded as a positive integer and then multiplied by the input random noise. Therefore, the latent space becomes dependent on the input label. In the discriminator the label of the image is also used, although not as an input, as in a conditional GAN (C-GAN [5]), but as an output. The mission of the discriminator does not consist uniquely of maximizing the accuracy of classifying the images as synthetic or original, but also of correctly classifying the images in their respective category.

The usage of an AC-GAN instead of a regular GAN served two purposes: first, it allowed us to train the generative model on images of different categories, in such a way that we could obtain a final generator capable of creating synthetic images of the wanted category; second, once the discriminator is trained it could be used as a category classifier disregarding the classification of synthetic or original.

Generator

The generator is a deep convolutional network that takes two inputs: random noise, and a random class label. From that point of the latent space, it outputs an array with dimensions 28x28x1, which is initially used to feed the discriminator, and finally to create a synthetic data set.

The generator's loss function will depend uniquely on the performance of the discriminator classifying the images as synthetic or original. The generator will try to maximize the error of the discriminator predicting the label of the origin of the image.

The general architecture of this network consists of:

1. Two inputs: Random noise and a random class label.
2. A flat embedding layer for the class label.
3. A multiplication of layers to join the random noise input layer with the flat embedding.
4. A complete CNN which varies depending on the experiment.

Discriminator

The discriminator is another deep convolutional network that takes one input: an array with dimensions 28x28x1 which corresponds to an image of said dimensions. The outputs of this network are two predictions: the class label of the image, and another label that indicates if the image was created by the generator or not.
The discriminator's loss function will depend on the entropy of both predicted labels and their actual values. The discriminator will try to minimize the binary cross entropy of the predicted and real source label, and the sparse categorical cross entropy of the predicted and real class label.

The general architecture of this network consists of:

1. One input: The array corresponding to an image (synthetic or original).
2. A complete CNN which varies depending on the experiment.
3. A flattening layer for the output of the CNN.
4. Two output layers: A dense layer with binary output for predicting the source of the image. A dense layer which outputs the prediction of the class of the image.

Compilation and Training Parameters

The compilation of data and modification of training parameters of the GAN model experiments were analogous to the CNN: all of them saved the configuration reports and final model. In addition, a sample of images created by the generator were also stored.

Model Evaluation

The performance of the AC-GAN was evaluated following subjective and objective approaches. Even though reaching decisions subjectively is not the most orthodox practice on a scientific report, it was used for the purpose of this study as it allowed us to work in a more efficient manner. Given that the

initial objective of the GAN is to augment data, the human eye can suffice deciding if the generated images are good enough to be used for next steps, or if that solution should be dropped. For example, if the synthetic images are just homogeneous squares with no edges at all, they can be easily discarded as there is no resemblance with the original pictures.

Once this initial subjective threshold is passed, we could use more precise and objective metrics to evaluate the performance of the GAN. The loss functions of generator and discriminator use two metrics: binary cross entropy, and categorical cross entropy. Keeping track of these measurements and the classification accuracies during the training allowed us to examine some insights as mode collapse and convergence failure, which are critical for tuning the combined GAN model.

Experiment tracker

To gain further insights on the different experiments and compare them to one another, we designed and implemented a custom-built experiment tracker. On the top level we will use an overview table across all experiments that have been run. It compares basic metrics like accuracy and runtime. Furthermore, it serves as a name node to look up details on every experiment. Every time a new model is saved, the run_id will auto increment and a new row will be appended to the overview file.

Additionally, several details of the specific experiment will be stored in the corresponding sub-folder: experiment configuration file, model architecture, plot of model train and validation history, test metrics (i.e., confusion matrix, precision, and recall), as well as the entire model including weights for possible future use. Furthermore, we implemented the plotting of class activation maps (CAM) to understand whether the classifier focuses on plausible patterns and to ensure that it does not learn 'by accident'. Therefore, we compare samples of CAMs that were rightly classified with the ones misclassified for every class. This also gives further insights on the granularity of the learned patterns, which can be traced back to the selected model architecture.

4 Training & Tuning

CNN

We designed a few questions that we wanted to explore in order to understand the functionality of our models. As previously stated, we took a systematic approach, testing each parameter individually, as well as in combination with one another. We started with a baseline model from which we changed certain parameters. The structure of the baseline model can be seen in Figure 4-1.

Summary run_0030
Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_1 (Conv2D)	(None, 11, 11, 64)	18496
max_pooling2d_1 (MaxPooling2	(None, 5, 5, 64)	0
conv2d_2 (Conv2D)	(None, 3, 3, 128)	73856
max_pooling2d_2 (MaxPooling2	(None, 1, 1, 128)	0
flatten (Flatten)	(None, 128)	0
dense (Dense)	(None, 512)	66048
dense_1 (Dense)	(None, 11)	5643

Total params: 164,363
Trainable params: 164,363
Non-trainable params: 0

Figure 4-1: Structure of the baseline classifier model.

Firstly, we wanted to understand the trade-off between batch size and the number of iterations. Ceteris paribus, we ran several models with a batch size of 32, 64, 128 and 256. We found that the larger the batch size, the longer the training duration. It also appears that a larger batch size leads to slightly improved performance and generalizability. We tested a batch size of 1000 in order to see if the pattern continued. In this case, the model results were poorer, and the training time was much higher. Within some range, the performance improves as you increase the batch size; however, there is a point after which performance is deteriorating.

Next, we tested our baseline model using different optimizers. We tested Stochastic Gradient Descent,

RMSprop and adam optimizers. While SGD is able to quickly run through several epochs, it results in a slightly lower level of performance. RMSprop provides an improvement to the adam optimizer, although not significant.

As the performance of a CNN can be quite sensitive to the filter size, we wanted to explore the impact of varying this parameter. We tested filters of size 3x3, 4x4 and 5x5. In reviewing the results, it becomes clear that the larger the size, the better the performance. The filter of 4x4 provides quite a significant improvement on the 3x3 filter, whereas the 5x5 filter is a only a slight enhancement to the 4x4. It would appear that the 3x3 filter is too small to recognize any patterns in our dataset.

Average pooling layers, as opposed to max pooling, also offered an improvement on the performance of each of the train, validation, and test set. Average pooling smooths out the image, resulting in a less sharp image. The fact that the images are hand drawn could perhaps explain the improved performance of average pooling. It is important to note that max pooling selects the brighter pixels and, as our images are inverted (white drawings on a black background), it may still be optimal to proceed with Max pooling. Subsequently, both pooling methods were attempted with a 3x3 pool size, which resulted in improved accuracy on the test and validation set compared to a pool size of 2x2.

The inclusion of Batch Normalization resulted in a very negligible improvement on the baseline model. Of the hyperparameters that we tuned, this model had the highest amount of overfitting, albeit still an improvement on the baseline performance.

Some parameters played a large role in significantly reducing overfitting. The model with dropout layers after each max pooling layer, and before the dense layers, was able to significantly reduce overfitting. The implementation of some data randomization almost reduced overfitting completely. With only a few augmentations implemented – horizontal flipping, vertical flipping and a 45-degree rotation range – there was already a huge improvement. While in both cases overfitting was significantly decreased, the overall accuracy obtained on the test set by implementing the dropout layers was substantially higher. That is, dropout resulted in a score of 0.83 compared to 0.77 obtained by data normalization. Figure 4-2 illustrates the reduction in overfitting between the baseline model compared to the model implemented with data randomization.

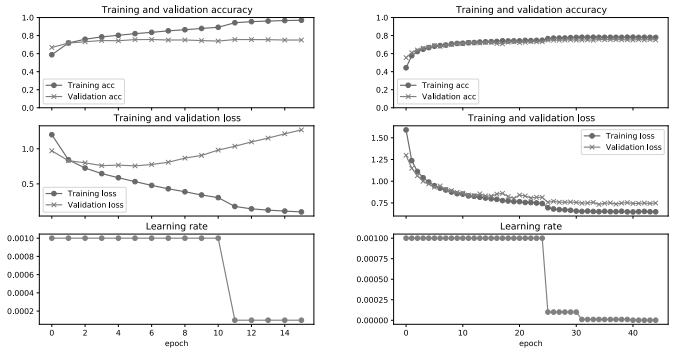

Figure 4-2: Training metrics along epochs for the baseline model (left) in comparison with performance when data randomization is introduced (right).

We were also interested in understanding the impact of adding layers to our network. As such, we tested one model that implemented two additional convolutional layers, as well as one that doubled the convolutional layers before each max pooling layer. The addition of layers led to the model overfitting the training data, with the latter showing a performance of 99% on the training set compared to 80% on the test set.

In exploring various convolutional networks that have gained popularity over the years, we took some inspiration from the well-known AlexNet [6] and tested a couple models with filter size increasing for each convolutional layer. We also implemented an additional dense layer. These models resulted in a higher performance on the test set compared to the training set.

Finally, we explored the impact of combining several of the variations discussed above, to obtain a superior model. Several of these models were either largely overfitting or underfitting. Namely, data randomization, that performed so well when it was tested independently, resulted in underperformance when combined with other parameters.

GAN

For training the GAN we started with baseline models with few convolutional layers.

The challenge of experimenting with the parameters of the discriminator and generator is that these models need to be balanced so that the GAN does not reach mode collapse or convergence failure.

First, we stablished a ground truth of 6000 images per class, a batch size of 100, and a limit of 2000 epochs, which remained constant during almost all experiments. These values were set so we could get some results with the baseline models which were not just noise. We used an adam optimizer with a learning rate of 0.004 and a beta equal to 0.5, following the suggestion described in [7].

Then by looking at the images output by the generator we determined that the convolutional networks were too simple to catch the difference between classes (See Figure 4-3).

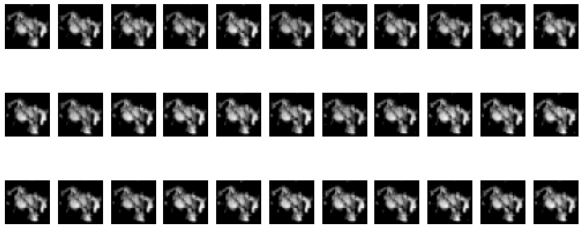

Figure 4-3: All produced images are very similar regardless the class or instance.

The next step was to add more convolutional layers (some of which were depth wise convolutional layers, although having a single channel makes them equal to the conventional ones), which did not significantly improve the performance until we added dropout layers.

Another experiment consisted of adding two dense layers at the end of the CNN of the discriminator (inspired by AlexNet [6]), which led to a better classification of the images class.

Additionally, different combinations of filter sizes for each layer were tested in both the generator and discriminator. Best results were obtained when the first layers had larger filter size (15x15, 10x10…) and final layers a smaller one (3x3).

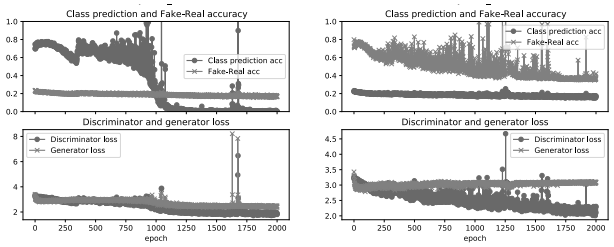

Figure 4-4: Examples of convergence failure when using a more advanced discriminator than generator, and of mode collapse where the generator is more complex than the discriminator.

5 Results

Best result of the CNN

According to the experiments done to tune the CNN, a combination of the configurations were implemented and the model was trained on 6000 images. The architecture of the selected CNN is as follows. The model utilizes 3 convolutional layers with 5x5 filters. Average pooling with a pool size of 3x3 and a dropout of 0.2 was implemented after each convolutional layer. The model uses a batch size of 256, along with the adam optimizer.

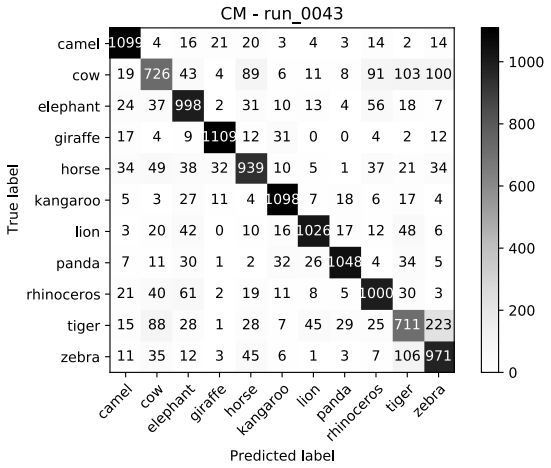

Figure 5-1: Resulting confusion matrix from the best CNN

In the multiclass confusion matrix above, you can see the ability of the selected model to correctly classify each of the categories. As can be expected, the model had a tough time differentiating between a tiger and a zebra. In addition, cow drawings were incorrectly classified on many instances. With the exception of these classes, the model was quite accurate in classifying the remaining categories.

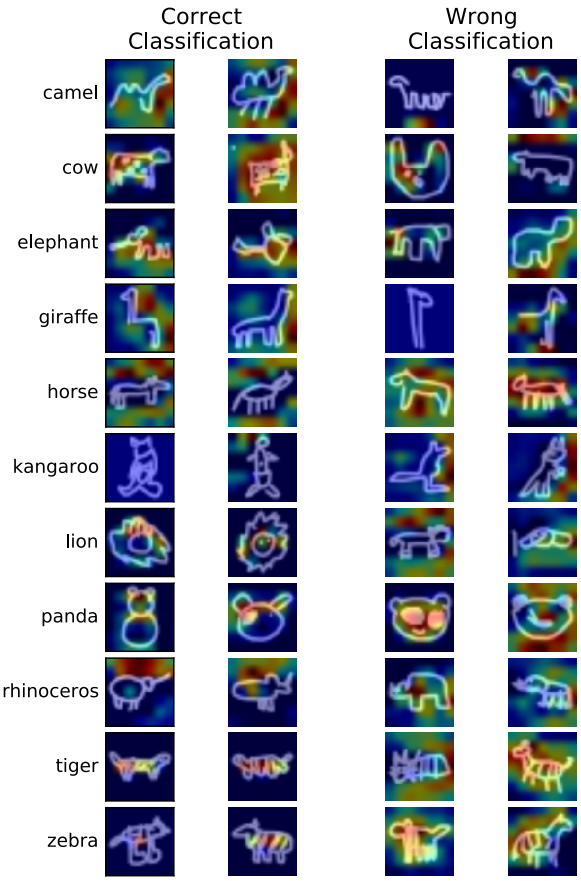

Figure 5-2: Class activation map (CAM) for each category.

The CAMs highlight areas within the images which the classifier focuses on in order to make its predictions. Looking at the examples of correctly classified tigers and zebras, we can see that in both cases the classifier looks at the stripes on their torso. As these shapes apply to both species, this won't be enough to separate them. At the same time clearly distinguishable characteristics like the dots on the cow and the long neck of the giraffe are important for the classifier. Considering how often a cow got misclassified as a tiger or a zebra, we can assume that those images were not showing these characteristics. This leads us to another aspect of training good classifiers – reliable data.

Best result of the GAN

Of all the possible GAN models tested in this study the best results were obtained for the discriminator with 9 convolutional layers, dropout layers (0.25) every two convolutional layers, output units increasing per layer, filter size decreasing per layer, and 2 fully connected layers at the end of the CNN (See dis_4_mod3 function in the discriminator.py file). For the generator we used 6 convolutional layers with filters of 5x5, 1 convolutional layer with a filter of 3x3, dropout layers of 0.25, batch normalization layers, leaky relu activation functions, and 2 final fully connected layers (See gen_5 function in generator.py).

With this configuration the outputs of discriminator and generator are not impressive. The images created by the generator as can be seen in Figure 5-3 are very blurry even for this low resolution. Nevertheless, some images might resemble animal figures.

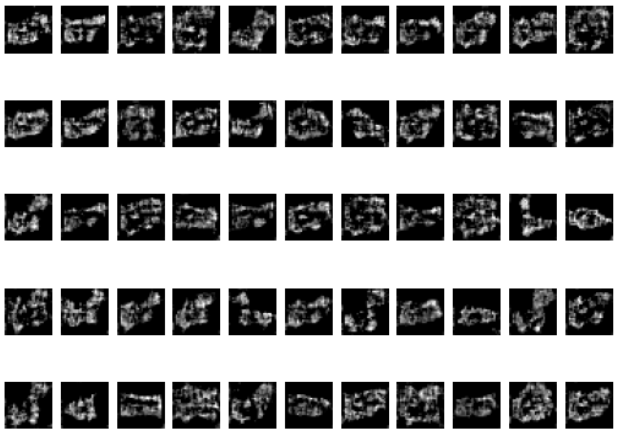

Figure 5-3: Synthetic images created by the best generator after 2000 epochs.

Regarding the performance of the discriminator after 2000 epochs, it is able to predict the source of the images with an accuracy of 42%, and the class label with an accuracy of 62.8%, which given the computational cost is not outstanding.

CNN vs. GAN

As a next step, and based on the best CNN model configuration, we then replaced 30% of the training images with artificial ones, produced by the GAN. We find that the test accuracy remains unchanged at 0.81, while training accuracy dropped dramatically to 0.67. This can be explained by the randomization and noise the GAN images introduce to the data. The randomization can serve as a powerful tool to cope with overfitting. At the same it is not advisable as a replacement to the original images, but rather as an augmentation. According to these findings GAN augmented data does not help to overcome the problem of data scarcity.

Considering the low training score and very slow convergence of using the augmented training data, the model requires further adaptations to the configuration in order to improve performance.

We compared the performance of the best CNN and the discriminator of the best GAN when predicting the classes of images, assuming we only have 6000 images per category. The CNN is required to split this data into three parts, which leads to training on 3600 images, with the remaining data for validation and testing (1200 for each). The discriminator does not specifically need to validate so it has the advantage of using more images for training and testing (4800 and 2200 respectively).

The best CNN classifier obtained a precision of 0.81, and a recall of 0.81, whereas the discriminator of the best GAN got a precision of 0.60, and a recall of 0.58. It is clear that it is preferable to use the CNN classifier as it is computationally cheaper and unquestionably outperforms the GAN.

6 Conclusion

Our solution could be extended to other problems and datasets where the use of data augmentation through deep learning techniques might be helpful; however, a further parameter tuning should be performed.

The performance of the classifier using “traditional” data augmentation techniques and using the data generated by the GAN did not differ significantly. It is possible that improved tuning of the GAN could be preferred to the conventional data augmentation methods; however, given the difficulty and the computational cost of the task, it is not worth it.

Keeping track of experiments is a crucial part on the development of Deep Learning models. In addition, a methodical and gradual modification of parameters when trying to avoid expensive grid searches is vital.

7 References:

[1] J. Jongejan, H. Rowley, T. Kawashima, J. Kim and Fox-Gieg, "The Quick, Draw! - A.I. Experiment," 2021. [Online]. Available: https://quickdraw.withgoogle.com/.

[2] H. Singh, "Is your epochs not performing, try callbacks," Medium, 3 January 2020. [Online]. Available: https://medium.com/black-feathers-labs/is-your-epochs-not-performing-try-callbacks-76519f0368a9. [Accessed 24 March 2021].

[3] "Core - Keras Documentation," , . [Online]. Available: https://keras.io/layers/core/. [Accessed 4 4 2021].

[4] A. Odena, C. Olah and J. Shlens, "Conditional image synthesis with auxiliary classifier gans.," in *International conference on machine learning. PMLR*, 2017.

[5] M. Mirza and S. Osindero, "Conditional generative adversarial nets.," *arXiv preprint arXiv:1411.1784,* 2014.

[6] J. Wei, "AlexNet: The Architecture that Challenged CNNs," Towards Data Science, 3 July 2019. [Online]. Available: https://towardsdatascience.com/alexnet-the-architecture-that-challenged-cnns-e406d5297951. [Accessed 29 March 2020].

[7] A. Radford, L. Metz and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks.," *arXiv preprint arXiv:1511.06434,* 2015.