Report of semester project: Training of MLP, CNN and GAN models using modern ML techniques.

Christian Mollière
Departamento de Sistemas Informáticos y Computación
Professor: Francisco Casacuberta Nolla, Subject: Redes Neuronales Artificiales

February 7, 2020

Abstract

The following report will discuss the results of the semester project training a MLP on MNIST, as well as a CNN on CIFAR-10. Different model architectures and training methods are discussed and the best result is presented showing its training history and final performance. Furthermore, as an additional project I tried to synthesize dog images using a collection of GAN models trained on a self compiled dataset.

Contents

1 Training MLP on MNIST **3**
1.1 Model Architecture . . . 3
1.2 Hyperparameter Grid . . . 3
1.3 Results . . . 3

2 Training CNN on CIFAR-10 **4**
2.1 Model Architecture . . . 4
2.2 Finding the right model . . . 5
2.3 Results . . . 5

3 Training of Generative Adversarial Networks **7**
3.1 Generating a Dataset . . . 7
3.2 Training of Deep Convolutional GANs . . . 7
3.3 Results of DCGAN Training . . . 8
3.4 Training of Wasserstein GANs . . . 9
3.5 Results of WGAN Training . . . 10

4 Conclusion **16**

A Datasets **17**

B Model Summaries **20**
B.1 MLP MNIST classifier model summary . . . 20
B.2 CNN CIFAR10 classifier model summary . . . 21
B.3 DCGAN model summary . . . 23
B.4 WGAN model summary . . . 25

C List of Dog Breeds **27**

1 Training MLP on MNIST

Training a MLP on MNIST is a classical exercise when getting into the field of machine learning. The data set contains 60000 28x28 grayscale images representing handwritten digits from 0 to 9. In the following the used techniques and parameters are discussed leading to a final result of 6.28% test error. Therefore, only 63 of the given 10000 test data samples are misinterpreted.

1.1 Model Architecture

The used MLP network uses a sequence of fully connected layers employing a ReLU activation function. Furthermore, a softmax activation is applied to the final classification layer, as usual when classifying data samples. Finally, techniques such as batch normalization, noise regularization, dropout and data augmentation are used to further improve the networks performance.

Layer Sizes A simple stack of three hidden layers (1024,1024,1024) and an output layer (10) immediately achieved the wanted performance. The complete model summary can be seen in Section B.1.

Regularization To prevent the model from overfitting the training data gaussian noise, dropout and L2 regularization were applied. Batch Normalization was used to decrease the feature co-variance shift and speed up training.

Learning Rate Schedule A stepped learning rate function (See Figure 1) provided the learning rate at the current epoch, decreasing the learning speed from 0.1 to 0.001 as the training proceeds. After epoch 100 the learning rate is raised again to potentially help the optimizer escape local minima.

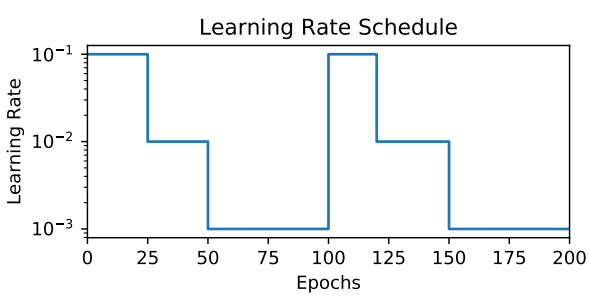

Figure 1: Schedule used for training of MLP model.

Data Augmentation Image preprocessing was used to extend the training data by introducing a positional shift of up to 10% and a rotational shift of up to 5° to the dataset. Since the classification of digits is sensitive to their orientation, no horizontal or vertical flipping was employed.

1.2 Hyperparameter Grid

A grid search on the following hyperparameters was carried out, leading to the achieved result.

- Gaussian Noise $\sigma \in \{0.1, 0.2, 0.3, 0.4\}$
- Dropout Rate $d \in \{0.3, 0.5\}$
- Batch Size = 128
- Activation ReLU & SeLU
- L2 Regularization $\lambda \in \{0, 0.01\}$

The models were trained for 250 epochs without early stopping using stochastic gradient descent and a categorical cross-entropy loss.

1.3 Results

The best model yielded from the grid search achieved the following metrics as shown in Table 1. It was trained using a dropout rate of 0.3, no L2 regularization, ReLU activation and gaussian noise layers with $\sigma = 0.1$.

Train Loss	**Train Acc**	**Test Loss**	**Test Acc**
0.7619	0.9862	0.1490	**0.9932**

Table 1: Metrics achieved by the best MNIST classifier MLP found.

Looking at the model accuracy and loss during training (see Figure 2), one can clearly see the learning rate reboost at epoch 100. Leading the model to temporarily drop in accuracy before increasing performance a little further. Also, the lower training accuracy and higher training loss (compared to the test metrics) indicate that little to no overfitting is taking place and the model is generalizing well. This is probably due to the used regularization techniques as discussed in Section 1.

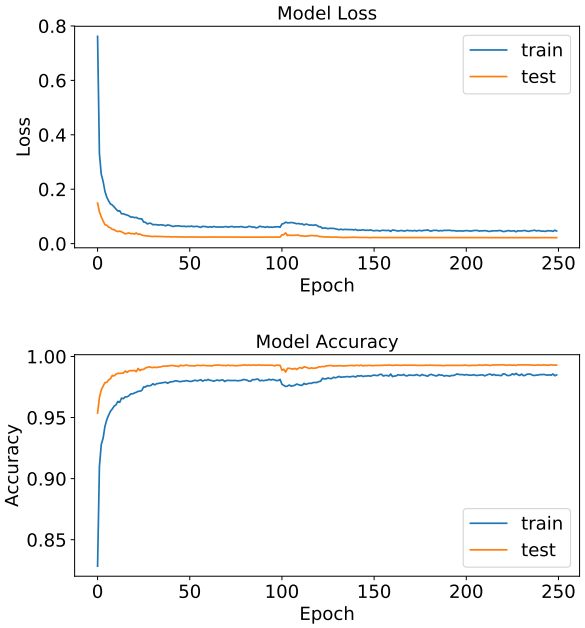

Figure 2: Training history of the MNIST classifier.

2 Training CNN on CIFAR-10

As a second task, a convolutional neural network (CNN) was trained to classify images from the CIFAR-10 image data set, containing 60000 samples of different objects as 32x32 RGB color images. The best model found achieved a test error of 9.71%.

2.1 Model Architecture

The employed model architecture is based on the VGG Model A introduced in [1]. It uses a set of convolutional and pooling layers followed by a block of fully connected layers. Since this model only needs to distinguish between 10 classes at not 100 as in the case of VGG, the size of the fully connected layers is shrinked from 4096 to 256. The complete model summary can be seen in Section B.2 as well as in the last row of Table 3.

Regularization This model uses the same regularization techniques as the MLP model in Section 1. L2 regularization and dropout were completely omitted in this model, since it did not increase performance in various prototypes. Yet recent research suggests that dropout can in fact be successfully combined with batch normalization [2].

Learning Rate Schedule To achieve a high model accuracy above 90% a cyclic learning schedule was combined with stochastic gradient descent as proposed in [3]. The schedule operates between learning rates of 0.001 to 0.3 and is shown in Figure 3.

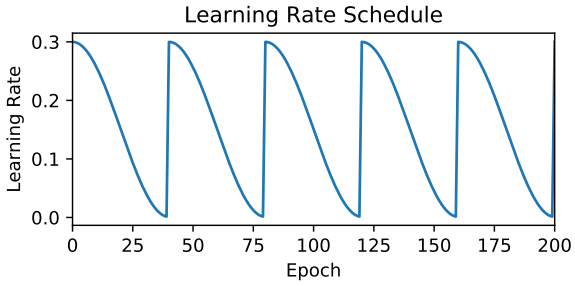

Figure 3: Schedule used for training of CNN model.

Data Augmentation Additional training data was generated using the mixup technique as introduced in [4]. Using mixup the training data set was doubled in size, yielding 100000 individual samples to train on. Futhermore, image preprocessing was employed applying scaling (up to 20%), rotation (up to 20°), positional shift (up to 20%), horizontal flipping and feature-wise data normalization.

2.2 Finding the right model

Since the training of a CNN is much more time expensive than a regular MLP, no grid search was used to determine the best set of model hyperparameters. Instead, a sparse set of manually selected models was trained to achieve the final result. The different trained models can be seen in Table 3 showing the best result in the last row. The main difference between the models is in the variation of dense layer sizes and the learning schedules.

2.3 Results

The best model achieved the following metrics as shown in Table 2.

Train Loss	Train Acc	Test Loss	Test Acc
0.5292	0.9550	0.4274	**0.9005**

Table 2: Metrics achieved by the best CIFAR-10 classifier CNN found.

Looking at the training history of the final model (see Figure 4) the cyclic learning schedule is clearly visible in the loss and accuracy metrics. By reboosting the learning rate every 40 epochs plateauing of the training is inhibited and the model can escape local minima. It seems that the cycle period is a little short because the model does not always plateau before each reboost. The biggest effect of the cyclic learning schedule is seen on the training data set containing mixup samples. The accuracy figure shows slight overfitting of the model towards the end of the training. Yet, these last epochs are necessary to achieve a test error below 10%.

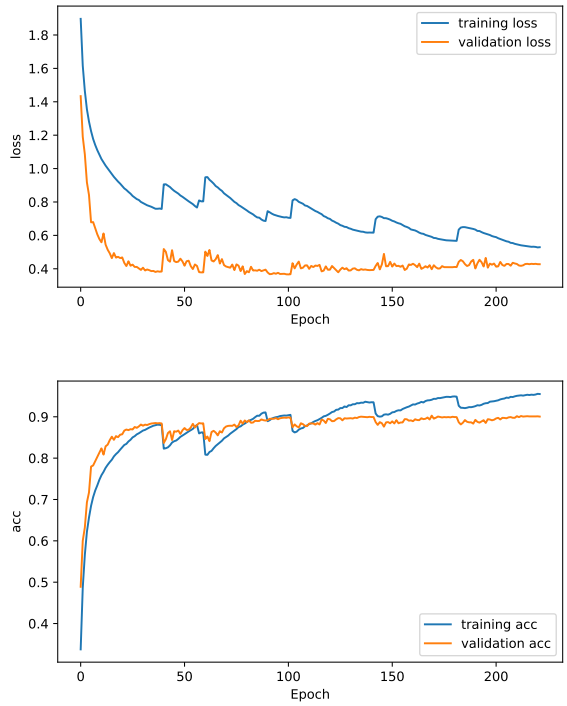

Figure 4: Training history of the CIFAR-10 classifier.

Model Description	LR Schedule	Mixup	Batch Size	Validation Accuracy
Conv2D 3x3@64 + MaxPool, Conv2D 3x3@128 + MaxPool, 2x Conv2D 3x3@256 + MaxPool, 2x Conv2D 3x3@512 + MaxPool, 2x Dense 512	stepped linear 0.1 to 0.001	+50%	64	87.9%
Conv2D 3x3@64 + MaxPool, Conv2D 3x3@128 + MaxPool, 2x Conv2D 3x3@256 + MaxPool, 2x Conv2D 3x3@512 + MaxPool, 2x Dense 512	single reboost 0.25 to 0.001	+50%	32	85.85%
Conv2D 3x3@64 + MaxPool, Conv2D 3x3@128 + MaxPool, 2x Conv2D 3x3@256 + MaxPool, 2x Conv2D 3x3@512 + MaxPool, 2x Dense 512	single reboost 0.25 to 0.001	+100%	64	89.04%
Conv2D 3x3@64 + MaxPool, Conv2D 3x3@128 + MaxPool, 2x Conv2D 3x3@256 + MaxPool, 2x Conv2D 3x3@512 + MaxPool, 2x Dense 512, Dense 256	cyclic 0.3 to 0.001 period 40 epochs	+100%	64	89.72%
Conv2D 3x3@64 + MaxPool, Conv2D 3x3@128 + MaxPool, 2x Conv2D 3x3@256 + MaxPool, 2x Conv2D 3x3@512 + MaxPool, 2x Dense 1024, Dense 512	cyclic 0.3 to 0.001 period 40 epochs	+100%	64	88.65%
Conv2D 3x3@64 + MaxPool, Conv2D 3x3@128 + MaxPool, 2x Conv2D 3x3@256 + MaxPool, 2x Conv2D 3x3@512 + MaxPool, 2x Dense 256, Dense 128	cyclic 0.3 to 0.001 period 40 epochs	+100%	64	**90.29%**

Table 3: Different models trained on CIFAR-10.

3 Training of Generative Adversarial Networks

To gain deeper insight in generative models I chose to train different GAN architectures to synthesize images from a self compiled dataset. The results are discussed in the following.

3.1 Generating a Dataset

Most of machine-learning education revolves around carefully compiled academic datasets to train and compare different algorithms on. Since the objective performance of GAN's remains a difficult measure [5], I decided to compile my own data set of image data, since I would not be able to compare the performance directly to existing models anyway.

Initial dataset As a first step a subject was chosen, in this case the goal was to generate images of dogs[1]. Therefore, a comprehensive list of 531 dog breeds was obtained from [6] which can be seen in Section C. In a next step a web scraper was built to systematically download the first 100 results of a Google Image search for each dog breed name of the obtained list (see Section C). All images were cropped to 64x64 pixels, too small images were dropped from the dataset. The source code of the built web scraper is available via [7]. This yielded an initial dataset D_0 containing 38148 samples of 64x64 RGB images presumably showing dogs. Samples from D_0 can be seen in Figure 16 in Section A.

Filtered datasets As visible in Figure 16, the initial dataset contains a certain amount of samples that are of comic-style or contain a mostly white background. Since the target was to synthesize images that look as real as possible, the dataset was filtered to exclude all images that contain more than 10% of color-less pixels that are brighter than 80%. This yielded a filtered dataset D_1 containing 30094 samples. Samples from D_1 can be seen in Figure 17 in Section A. Finally, the in Section 2 trained an image classifier was used to obtain a third dataset D_2 including all samples from D_1 that were successfully classified as dog images. This yielded a considerably reduced dataset D_2 of only 4169 samples. The strong reduction of samples might be due to the great variety of different perspectives, backgrounds and dog breeds, which are might not similar enough to the samples contained in the CIFAR-10 dataset. Samples from D_2 can be seen in Figure 18 in Section A.

Preprocessing during training The image dataset is rescaled from $[0 - 255]$ to $[-1, 1]$ for the training of all GAN models.

3.2 Training of Deep Convolutional GANs

In general the idea of a GAN is to train two adversarial models that try to outperform each other. The first model is called the generator G, which samples images from a latent space z and generates a fake image. The second model, called discriminator D will then ingress real images from the dataset and fake images from the generator and learn to distinguish between them. If the two models find a balance during training the generator gradually converges to sample from the approximated real distribution of the dataset [8]. This concept is summarized in Figure 5.

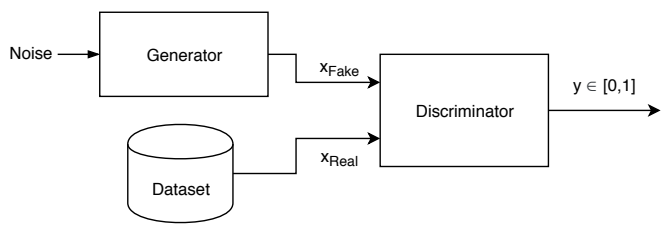

Figure 5: Overview of architecture of a GAN.

Model Architecture The implementation of both generator and discriminator are based on the proposed DCGAN by [9]. It main features in comparison to the traditional CNN GAN's are:

- No Dense layers in the deeper architecture
- ReLU activation and Batch-Norm throughout the generator model
- Leaky ReLU activation in the discriminator model
- Pooling layers are replaced by strided convolutions

[1] The internet has seen enough cat pictures already.

The generator uses transposed convolutional layers with a stride of 2 to upsample the reshaped noise input maps to the final size of 64x64x3. Furthermore, a dropout of 0.2 is applied in the first two transposed convolutions. The final layer of the generator uses a Tanh activation to only allow a model output of $[-1, 1]$, which is coherent to the rescaled dataset. Its model architecture is shown in Figure 6.

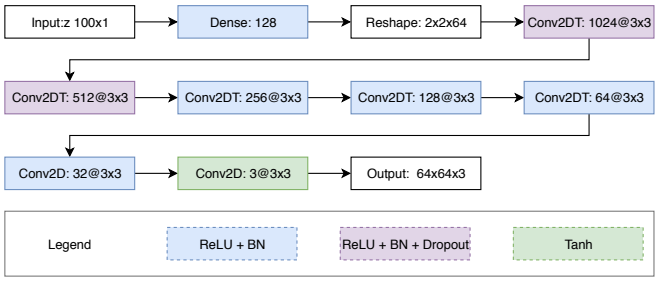

Figure 6: Model architecture of the generator.

The discriminator uses strided convolutions with a step size of 2 to avoid pooling layers. Its input image is therefore reduced from 64x64x3 to 4x4x512 and then passed into a final dense layer employing Sigmoid activation for classification. Its model architecture is shown in Figure 7.

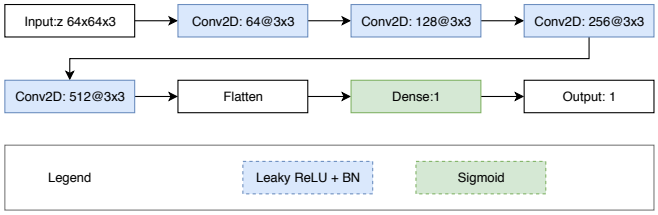

Figure 7: Model architecture of the discriminator.

Both the combined model and the discriminator are trained using a binary cross-entropy loss function. During the training of the combined model the discriminator weights are frozen to ensure that only the generator gets trained during this step. Full model summaries of both discriminator and generator can be seen in Section B.3.

Optimizer As proposed in [9] an Adam optimizer was used with a initial learning rate of $\rho = 0.0002$ and a momentum of $\beta = 0.5$.

Label Smoothing Additional training performance could be achieved employing label smoothing for both the real and the fake labels as discussed in [10] leading to real labels of $y_{\text{real}} \in [0.7, 1]$ and fake labels of $y_{\text{fake}} \in [0, 0.3]$. This prevents the discriminator to become overly confident and shall ensure stable gradients for longer training periods.

3.3 Results of DCGAN Training

The following results where obtained after training the in Section 3.2 defined DCGAN model using all three datasets D_0, D_1 and D_2 for at 200 epochs. The number of training iterations per epoch differ since every dataset has a different amount of samples and the model was trained using a fixed batch size of 128.

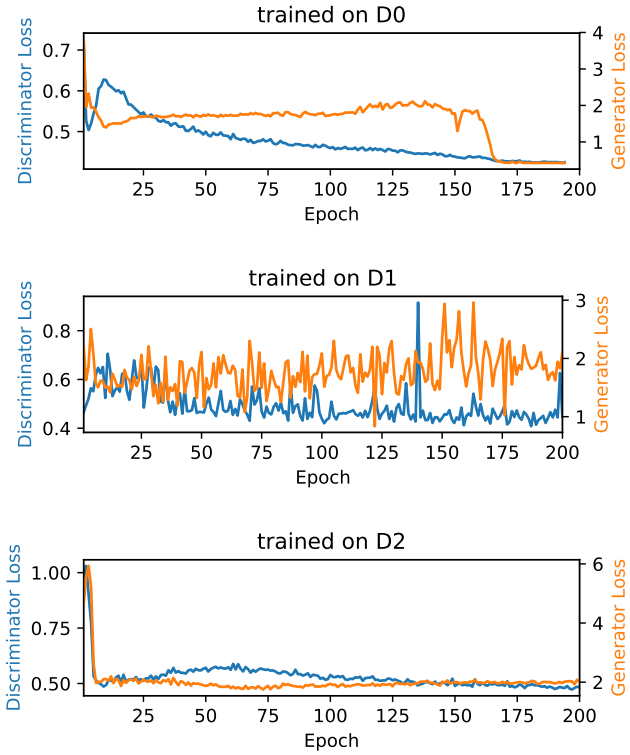

Figure 8: (Top) Loss on D_0
(Middle) Loss on D_1
(Bottom) Loss on D_2

Figure 8 shows the model performance on the datasets D_0 to D_2 respectively. Since the two models are adversaries trying to increase each others loss, the loss metric can in general not be used to infer the model's convergence [5]. Yet, training on all three datasets seems to be stable, since none of model's losses drop to zero, which would indicate a common failure mode of GAN training. To determine the convergence of a model it is useful to visualize the generators output by sampling a number of samples at fixed intervals of iterations.

Figure 12 shows samples of the DCGAN model trained on dataset D_0. In the beginning it learns rather slowly, achieving dog shaped silhouettes from 2.5k iterations. It seems to have problems training simultaneously on the outdoor images and the studio images, which feature a white backdrop, showing some noise white samples in the output.

Figure 13 shows samples of the DCGAN model trained on dataset D_1. Removing a lot of the studio images and comics from D_0 seems to help the model train faster and achieve more complex shapes. Since most of the samples in the input dataset feature a grassy green background, this is also strongly represented in the learned data distribution.

Figure 14 shows samples of the DCGAN model trained on dataset D_2. This dataset seems to work the best for training the DCGAN, learning faster than the previous two datasets and achieving more detailed representation of different dogs and poses. Still, even this best found DCGAN model won't generate images that would trick a human observer. Parts of the images are clearly related to dogs but the produced images are to noisy and often times distorted.

Figure 9: Mode collapse of DCGAN training on D_1 after 25k iterations.

Mode Collapse Training these models longer always resulted in mode collapse (see Figure 9), which means that the learned data distribution collapses to a small space. Therefore, the model is only sampling similar images with low variety. Even further training of 35k+ iterations made the models unstable letting them return to generating only noisy outputs. Smaller batch sizes lead in general to faster mode collapse.

3.4 Training of Wasserstein GANs

The concept of the Wasserstein GAN was initially proposed in 2017 [11]. Instead of a binary cross-entropy it maximizes an approximation of the Earth-Mover distance function, which is called Wasserstein distance d_W (see Equation 2) by the authors. The discriminator is modified to be rather a critic rating the input images to their similarity to the dataset. The authors claim that this implementation of GAN is less sensible to the model architecture and shows a considerably reduced amount of mode collapse. Furthermore, the Wasserstein distance can be used to track convergence of the model, in contrast to the oscillating loss metrics of the traditional GAN implementation. The corresponding loss function C is described in Equation 1 averaging the label $y \in [-1, 1]$ times the, by the critic predicted, score $\hat{y} \in \mathbb{R}$.

$$C = \frac{1}{n} \sum_{i=1}^{n} \hat{y}_i \cdot y_i \tag{1}$$

During training of the critic model real images are labeled $y_r = -1$ and generated images are labeled $y_f = 1$. Therefore, minimizing the loss function C will result in real images achieving a high non-negative score, while generated images should be scored with a high negative score. This will result in a diverging distribution of scores. The Wasserstein distance (see Equation 2) can then be calculated by using the critic model D, the generator model G, samples from the dataset $\mathbf{x}$ and sampled noise inputs for the generator $\mathbf{z}$. The optimizer will try to maximize d_W.

$$d_W = -\left[C(D(\mathbf{x}), \mathbf{y}_r) + C(D(G(\mathbf{z})), \mathbf{y}_f)\right] \tag{2}$$

Training of the generator is then employed using a flipped label $y_f = -1$ for the generated images to target scores within the real image area of the distribution. The optimizer of the generator minimizes C_{gen} (see Equation 3).

$$C_{\text{gen}} = C(D(G(\mathbf{z})), \mathbf{y_f}) \tag{3}$$

Model Architecture The model architecture is essentially the same as described in Section 3.2. Only a few modifications are made to transform the discriminator into the critic model. These modifications are:

- Changing the final discriminator activation from Sigmoid to Linear
- Implementing the Wasserstein loss function (see Equation 1)
- Constraining the critic weights to $\pm c$
- Training the critic n_c steps more often than the generator per generator training step

A full summary of the critic and generator model can be seen in Section B.4.

Optimizer Instead of the Adam optimizer a RMSprop optimizer with a initial learning rate of $\rho = 5\text{e}{-5}$ is used as proposed in [11], because the Adam optimizer causes the training process of the WGAN model to become unstable.

3.5 Results of WGAN Training

Training of the WGAN model was executed using the recommended hyperparameter proposed in [11]. These are:

- Initial learning rate $\rho = 5\text{e}{-5}$
- Weight clipping of the critic to $c = \pm 0.01$
- Critic training per generator training $n_c = 5$
- Batch size = 64

The WGAN model was only trained on dataset D_2, since the results of DCGAN training showed that the models perform best on this dataset (see Section 3.3).

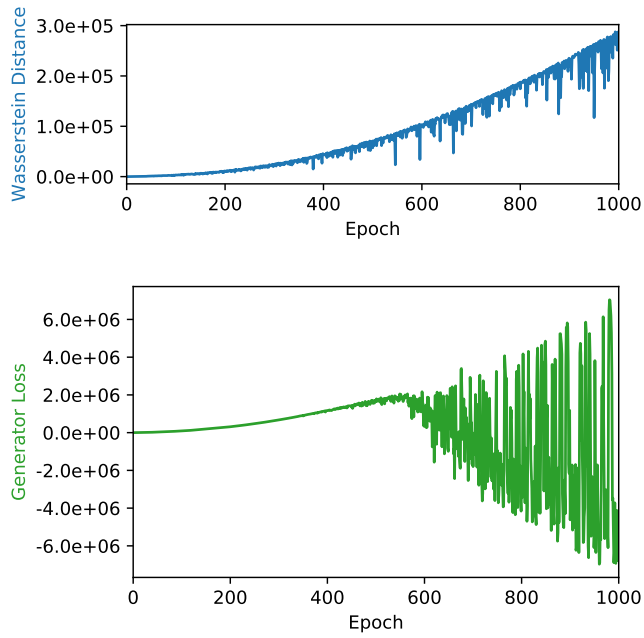

Figure 10: (Top) d_W of WGAN on D_2
(Bottom) C_{gen} of WGAN on D_2

Figure 10 shows the training history of the WGAN model on dataset D_2. The model was trained for 1000 epochs, achieving almost 65k generator training iterations. The Wasserstein distance d_W is gradually increasing during training. This indicates that the critic is progressively able to distinguish between real and generated samples. Ideally, a very good generator should produce a negative loss value since it's objective to its minimization. However, a growing generator loss, like in this case, does not necessarily mean that the generator is not learning, but rather that the critic model is far superior in detecting the generated samples. Unfortunately none of the tested models Wasserstein distance converged, signaling a optimal training state. Instead the training started to get unstable right at the moment when the generator is starting to deceive the critic model at epoch 550 and onwards.

In general the generated samples of the WGAN generator (see Figure 15) seems to contain less noise and more detailed features compared to the results of the DCGANs. The WGAN model seems to focus more on details, while failing to learn the general shape of a dog. Therefore, we see different fur patterns and singular features such as legs, heads or tails from epoch 200 and forward. As seen in epoch 300, the model seems to be distracted by the

image's backgrounds, learning to generate accurate representations of grass and meadows. Finally, from epoch 450 the model is able to generate dog faces but is not able to find a complete representation of a dog shaped object. From there on the model starts getting increasingly unstable, deteriorating in image quality and learned information. This can be seen in Figure 11 showing generated samples from epoch 650.

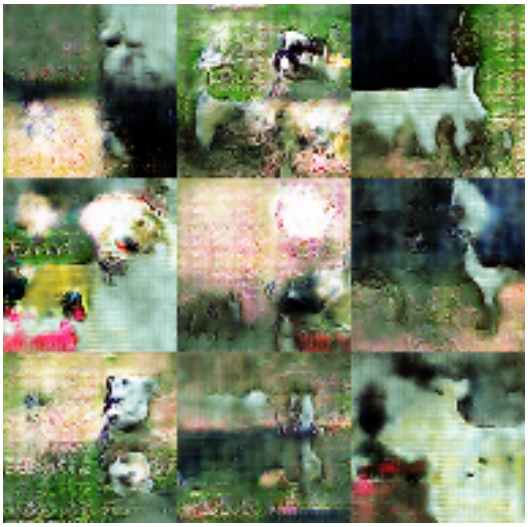

Figure 11: Output of WGAN model at epoch 650.

Figure 12: DCGAN trained on dataset D_0. (Top) 1.25k iterations, (Upper Middle) 2.5k iterations, (Lower Middle) 5k iterations, (Bottom) 6.5k iterations.

Figure 13: DCGAN trained on dataset D_1. (Top) 1.25k iterations, (Upper Middle) 2.5k iterations, (Lower Middle) 5k iterations, (Bottom) 6.5k iterations.

Figure 14: DCGAN trained on dataset D_2. (Top) 1.25k iterations, (Upper Middle) 2.5k iterations, (Lower Middle) 5k iterations, (Bottom) 6.5k iterations.

Figure 15: WGAN trained on dataset D_2. (Top) 6.5k iterations, epoch 100; (Upper Middle) 13k iterations, epoch 200; (Lower Middle) 20k iterations, epoch 300; (Bottom) 30k iterations, epoch 450.

4 Conclusion

The trained models in Section 1 and Section 2 achieved the required performance benchmarks of a MNIST classification better than 99.2% and a CIFAR-10 classification better than 10% using techniques such as learning rate annealing, learning rate restart and data augmentation.

Gathering and compiling a custom dataset (see Section 3.1) can be a tedious process and requires a lot of care and caution. Simply using a lot of data did not perform as well as using less better chosen data. Therefore, building the right dataset for a machine learning project is one of the most important steps in the process.

Training of the DCGAN (see Section 3.2) and WGAN models (see Section 3.4) did not yield a generative model that is able to produce photo-realistic dog images, but the models are able to infer certain traits of the targeted species. Characteristics like silhouette, parts of faces, legs and different furs are partly learned and can be generated. A promising next step to further improve the model performance could be the use of Spectral Weight Normalization (see [12]) to increase stability over longer periods of training or the improved WGAN architecture as described in [13], that avoids clipping of the critic's weights. Since the training was in general very computational expensive, even using the free GPU of Google Colab, the investigation in these models was a rather slow process, limiting my ability to fully search the hyperparameter and model space.

Finally, it seems that training ML algorithms on real world data is much more challenging than simply applying them to a well known dataset. Even though I could not fully achieve my goal of generating photo-realistic images of dogs, this is a useful result to get a grasp of the actual applicability of these algorithms on problems outside the academic realm.

The source code of all model implementations can be found in [14].

A Datasets

In the following a set of samples from every dataset is displayed.

Figure 16: Samples of dataset D_0

Figure 17: Samples of dataset D_1

Figure 18: Samples of dataset D_2

B Model Summaries

B.1 MLP MNIST classifier model summary

```
Model: "MNIST MLP model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
reshape_1 (Reshape)          (None, 784)               0
_________________________________________________________________
gaussian_noise_1 (GaussianNo (None, 784)               0
_________________________________________________________________
dense_1 (Dense)              (None, 1024)              803840
_________________________________________________________________
gaussian_noise_2 (GaussianNo (None, 1024)              0
_________________________________________________________________
activation_1 (Activation)    (None, 1024)              0
_________________________________________________________________
batch_normalization_1 (Batch (None, 1024)              4096
_________________________________________________________________
dropout_1 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_2 (Dense)              (None, 1024)              1049600
_________________________________________________________________
gaussian_noise_3 (GaussianNo (None, 1024)              0
_________________________________________________________________
activation_2 (Activation)    (None, 1024)              0
_________________________________________________________________
batch_normalization_2 (Batch (None, 1024)              4096
_________________________________________________________________
dropout_2 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_3 (Dense)              (None, 1024)              1049600
_________________________________________________________________
gaussian_noise_4 (GaussianNo (None, 1024)              0
_________________________________________________________________
activation_3 (Activation)    (None, 1024)              0
_________________________________________________________________
batch_normalization_3 (Batch (None, 1024)              4096
_________________________________________________________________
dropout_3 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_4 (Dense)              (None, 10)                10250
_________________________________________________________________
activation_4 (Activation)    (None, 10)                0
=================================================================
Total params: 2,925,578
Trainable params: 2,919,434
Non-trainable params: 6,144
_________________________________________________________________
```

B.2 CNN CIFAR10 classifier model summary

Model: "CIFAR−10 CNN model"

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 32, 32, 64)	1792
gaussian_noise_1 (GaussianNo	(None, 32, 32, 64)	0
activation_1 (Activation)	(None, 32, 32, 64)	0
batch_normalization_1 (Batch	(None, 32, 32, 64)	256
max_pooling2d_1 (MaxPooling2	(None, 16, 16, 64)	0
conv2d_2 (Conv2D)	(None, 16, 16, 128)	73856
gaussian_noise_2 (GaussianNo	(None, 16, 16, 128)	0
activation_2 (Activation)	(None, 16, 16, 128)	0
batch_normalization_2 (Batch	(None, 16, 16, 128)	512
max_pooling2d_2 (MaxPooling2	(None, 8, 8, 128)	0
conv2d_3 (Conv2D)	(None, 8, 8, 256)	295168
gaussian_noise_3 (GaussianNo	(None, 8, 8, 256)	0
activation_3 (Activation)	(None, 8, 8, 256)	0
batch_normalization_3 (Batch	(None, 8, 8, 256)	1024
conv2d_4 (Conv2D)	(None, 8, 8, 256)	590080
gaussian_noise_4 (GaussianNo	(None, 8, 8, 256)	0
activation_4 (Activation)	(None, 8, 8, 256)	0
batch_normalization_4 (Batch	(None, 8, 8, 256)	1024
max_pooling2d_3 (MaxPooling2	(None, 4, 4, 256)	0
conv2d_5 (Conv2D)	(None, 4, 4, 512)	1180160
gaussian_noise_5 (GaussianNo	(None, 4, 4, 512)	0
activation_5 (Activation)	(None, 4, 4, 512)	0

```
batch_normalization_5 (Batch (None, 4, 4, 512)         2048
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 4, 4, 512)         2359808
_________________________________________________________________
gaussian_noise_6 (GaussianNo (None, 4, 4, 512)         0
_________________________________________________________________
activation_6 (Activation)    (None, 4, 4, 512)         0
_________________________________________________________________
batch_normalization_6 (Batch (None, 4, 4, 512)         2048
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 2, 2, 512)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 2048)              0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               524544
_________________________________________________________________
gaussian_noise_7 (GaussianNo (None, 256)               0
_________________________________________________________________
activation_7 (Activation)    (None, 256)               0
_________________________________________________________________
batch_normalization_7 (Batch (None, 256)               1024
_________________________________________________________________
dense_2 (Dense)              (None, 256)               65792
_________________________________________________________________
gaussian_noise_8 (GaussianNo (None, 256)               0
_________________________________________________________________
activation_8 (Activation)    (None, 256)               0
_________________________________________________________________
batch_normalization_8 (Batch (None, 256)               1024
_________________________________________________________________
dense_3 (Dense)              (None, 128)               32896
_________________________________________________________________
gaussian_noise_9 (GaussianNo (None, 128)               0
_________________________________________________________________
activation_9 (Activation)    (None, 128)               0
_________________________________________________________________
batch_normalization_9 (Batch (None, 128)               512
_________________________________________________________________
dense_4 (Dense)              (None, 10)                1290
_________________________________________________________________
activation_10 (Activation)   (None, 10)                0
=================================================================
Total params: 5,134,858
Trainable params: 5,130,122
Non-trainable params: 4,736
_________________________________________________________________
```

B.3 DCGAN model summary

```
Model: "DCGAN Discriminator"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 32, 32, 64)        1792
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 32, 32, 64)        0
_________________________________________________________________
batch_normalization_8 (Batch (None, 32, 32, 64)        256
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 16, 16, 128)       73856
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 16, 16, 128)       0
_________________________________________________________________
batch_normalization_9 (Batch (None, 16, 16, 128)       512
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 8, 8, 256)         295168
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 8, 8, 256)         0
_________________________________________________________________
batch_normalization_10 (Batc (None, 8, 8, 256)         1024
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 4, 4, 512)         1180160
_________________________________________________________________
leaky_re_lu_4 (LeakyReLU)    (None, 4, 4, 512)         0
_________________________________________________________________
batch_normalization_11 (Batc (None, 4, 4, 512)         2048
_________________________________________________________________
flatten_1 (Flatten)          (None, 8192)              0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 8193
=================================================================
Total params: 3,124,098
Trainable params: 1,561,089
Non-trainable params: 1,563,009
_________________________________________________________________

Model: "DCGAN Generator"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 256)               25856
_________________________________________________________________
reshape_1 (Reshape)          (None, 2, 2, 64)          0
_________________________________________________________________
batch_normalization_1 (Batch (None, 2, 2, 64)          256
_________________________________________________________________
conv2d_transpose_1 (Conv2DTr (None, 4, 4, 1024)        590848
_________________________________________________________________
```

```
activation_1 (Activation)    (None, 4, 4, 1024)        0
_________________________________________________________________
batch_normalization_2 (Batch (None, 4, 4, 1024)        4096
_________________________________________________________________
dropout_1 (Dropout)          (None, 4, 4, 1024)        0
_________________________________________________________________
conv2d_transpose_2 (Conv2DTr (None, 8, 8, 512)         4719104
_________________________________________________________________
activation_2 (Activation)    (None, 8, 8, 512)         0
_________________________________________________________________
batch_normalization_3 (Batch (None, 8, 8, 512)         2048
_________________________________________________________________
dropout_2 (Dropout)          (None, 8, 8, 512)         0
_________________________________________________________________
conv2d_transpose_3 (Conv2DTr (None, 16, 16, 256)       1179904
_________________________________________________________________
activation_3 (Activation)    (None, 16, 16, 256)       0
_________________________________________________________________
batch_normalization_4 (Batch (None, 16, 16, 256)       1024
_________________________________________________________________
conv2d_transpose_4 (Conv2DTr (None, 32, 32, 128)       295040
_________________________________________________________________
activation_4 (Activation)    (None, 32, 32, 128)       0
_________________________________________________________________
batch_normalization_5 (Batch (None, 32, 32, 128)       512
_________________________________________________________________
conv2d_transpose_5 (Conv2DTr (None, 64, 64, 64)        73792
_________________________________________________________________
activation_5 (Activation)    (None, 64, 64, 64)        0
_________________________________________________________________
batch_normalization_6 (Batch (None, 64, 64, 64)        256
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 64, 64, 32)        18464
_________________________________________________________________
activation_6 (Activation)    (None, 64, 64, 32)        0
_________________________________________________________________
batch_normalization_7 (Batch (None, 64, 64, 32)        128
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 64, 64, 3)         867
_________________________________________________________________
activation_7 (Activation)    (None, 64, 64, 3)         0
=================================================================
Total params: 6,912,195
Trainable params: 6,908,035
Non-trainable params: 4,160
_________________________________________________________________
```

B.4 WGAN model summary

```
Model: "WGAN Critic Model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 32, 32, 64)        1792
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 32, 32, 64)        0
_________________________________________________________________
batch_normalization_8 (Batch (None, 32, 32, 64)        256
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 16, 16, 128)       73856
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 16, 16, 128)       0
_________________________________________________________________
batch_normalization_9 (Batch (None, 16, 16, 128)       512
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 8, 8, 256)         295168
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 8, 8, 256)         0
_________________________________________________________________
batch_normalization_10 (Batc (None, 8, 8, 256)         1024
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 4, 4, 512)         1180160
_________________________________________________________________
leaky_re_lu_4 (LeakyReLU)    (None, 4, 4, 512)         0
_________________________________________________________________
batch_normalization_11 (Batc (None, 4, 4, 512)         2048
_________________________________________________________________
flatten_1 (Flatten)          (None, 8192)              0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 8193
=================================================================
Total params: 3,124,098
Trainable params: 1,561,089
Non-trainable params: 1,563,009
_________________________________________________________________

Model: "WGAN Generator Model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 256)               25856
_________________________________________________________________
activation_1 (Activation)    (None, 256)               0
_________________________________________________________________
reshape_1 (Reshape)          (None, 2, 2, 64)          0
_________________________________________________________________
batch_normalization_1 (Batch (None, 2, 2, 64)          256
_________________________________________________________________
```

```
conv2d_transpose_1 (Conv2DTr (None, 4, 4, 1024)        590848
_________________________________________________________________
activation_2 (Activation)    (None, 4, 4, 1024)        0
_________________________________________________________________
batch_normalization_2 (Batch (None, 4, 4, 1024)        4096
_________________________________________________________________
dropout_1 (Dropout)          (None, 4, 4, 1024)        0
_________________________________________________________________
conv2d_transpose_2 (Conv2DTr (None, 8, 8, 512)         4719104
_________________________________________________________________
activation_3 (Activation)    (None, 8, 8, 512)         0
_________________________________________________________________
batch_normalization_3 (Batch (None, 8, 8, 512)         2048
_________________________________________________________________
dropout_2 (Dropout)          (None, 8, 8, 512)         0
_________________________________________________________________
conv2d_transpose_3 (Conv2DTr (None, 16, 16, 256)       1179904
_________________________________________________________________
activation_4 (Activation)    (None, 16, 16, 256)       0
_________________________________________________________________
batch_normalization_4 (Batch (None, 16, 16, 256)       1024
_________________________________________________________________
conv2d_transpose_4 (Conv2DTr (None, 32, 32, 128)       295040
_________________________________________________________________
activation_5 (Activation)    (None, 32, 32, 128)       0
_________________________________________________________________
batch_normalization_5 (Batch (None, 32, 32, 128)       512
_________________________________________________________________
conv2d_transpose_5 (Conv2DTr (None, 64, 64, 64)        73792
_________________________________________________________________
activation_6 (Activation)    (None, 64, 64, 64)        0
_________________________________________________________________
batch_normalization_6 (Batch (None, 64, 64, 64)        256
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 64, 64, 32)        18464
_________________________________________________________________
activation_7 (Activation)    (None, 64, 64, 32)        0
_________________________________________________________________
batch_normalization_7 (Batch (None, 64, 64, 32)        128
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 64, 64, 3)         867
_________________________________________________________________
activation_8 (Activation)    (None, 64, 64, 3)         0
=================================================================
Total params: 6,912,195
Trainable params: 6,908,035
Non-trainable params: 4,160
_________________________________________________________________
```

C List of Dog Breeds

Affenpinscher dog
Afghan Hound dog
Afghan Shepherd dog
Aidi dog
Airedale Terrier dog
Akbash dog
Akita dog
Alano Espaol dog
Alaskan husky dog
Alaskan Klee Kai dog
Alaskan Malamute dog
Alaunt dog
Alopekis dog
Alpine Dachsbracke dog
Alpine Mastiff dog
Alpine Spaniel dog
American Akita dog
American Bulldog
American Bully dog
American Cocker Spaniel dog
American English Coonhound dog
American Eskimo Dog
American Foxhound dog
American Hairless Terrier dog
American Pit Bull Terrier dog
American Staffordshire Terrier dog
American Water Spaniel dog
Anatolian Shepherd Dog
Andalusian Hound dog
Anglo–Franais de Petite Vnerie dog
Appenzeller Sennenhund dog
Argentine Polar Dog
Ariegeois dog
Armant dog
Armenian Gampr dog
Artois Hound dog
Australian Cattle Dog
Australian Kelpie dog
Australian Shepherd dog
Australian Stumpy Tail Cattle Dog[10]
Australian Terrier dog
Austrian Black and Tan Hound dog
Austrian Pinscher dog
Azawakh dog
Bakharwal dog
Barbado da Terceira dog
Barbet dog
Basenji dog
Basque Shepherd Dog
Basset Artsien Normand dog
Basset Bleu de Gascogne dog
Basset Fauve de Bretagne dog
Basset Hound dog
Bavarian Mountain Hound dog
Beagle dog
Beagle–Harrier dog
Bearded Collie dog
Beauceron dog
Bedlington Terrier dog
Belgian Shepherd Dog (Groenendael)
Belgian Shepherd Dog (Laekenois)
Belgian Shepherd Dog (Malinois)
Belgian Shepherd Dog (Tervuren)
Bergamasco Shepherd dog
Berger Blanc Suisse dog
Berger Picard dog
Bernese Mountain Dog
Bichon Fris dog
Billy dog
Black and Tan Coonhound dog
Black and Tan Virginia Foxhound dog
Black Norwegian Elkhound dog
Black Russian Terrier dog
Black Mouth Cur dog
Bloodhound dog
Blue Heeler dog
Blue Lacy dog
Blue Paul Terrier dog
Blue Picardy Spaniel dog
Bluetick Coonhound dog
Boerboel dog
Bohemian Shepherd dog
Bolognese dog
Border Collie dog
Border Terrier dog
Borzoi dog
Bosnian Coarse–haired Hound dog
Boston Terrier dog
Bouvier des Ardennes dog
Bouvier des Flandres dog
Boxer dog
Boykin Spaniel dog
Bracco Italiano dog
Braque d'Auvergne dog
Braque de l'Ariege dog
Braque du Bourbonnais dog

Braque du Puy dog
Braque Francais dog
Braque Saint−Germain dog
Brazilian Dogo
Brazilian Terrier dog
Briard dog
Briquet Griffon Venden dog
Brittany dog
Broholmer dog
Bruno Jura Hound dog
Brussels Griffon dog
Bucovina Shepherd Dog
Bull and Terrier dog
Bull Terrier dog
Bulldog
Bullenbeisser dog
Bullmastiff dog
Bully Kutta dog
Burgos Pointer dog
Cairn Terrier dog
Canaan Dog
Canadian Eskimo Dog
Cane Corso dog
Cantabrian Water Dog
Co da Serra de Aires dog
Co de Castro Laboreiro dog
Co de Gado Transmontano dog
Co Fila de So Miguel dog
Carolina Dog
Carpathian Shepherd Dog
Catalan Sheepdog
Caucasian Shepherd Dog
Cavalier King Charles Spaniel dog
Central Asian Shepherd Dog
Cesky Fousek dog
Cesky Terrier dog
Chesapeake Bay Retriever dog
Chien Franais Blanc et Noir dog
Chien Franais Blanc et Orange dog
Chien Franais Tricolore dog
Chien−gris dog
Chihuahua dog
Chilean Terrier dog
Chinese Chongqing Dog
Chinese Crested Dog
Chinook dog
Chippiparai dog
Chiribaya Dog
Chow Chow dog
Cierny Sery dog
Cirneco dell 'Etna dog
Clumber Spaniel dog
Collie , Rough dog
Collie , Smooth dog
Combai dog
Cordoba Fighting Dog
Coton de Tulear dog
Cretan Hound dog
Croatian Sheepdog
Cumberland Sheepdog
Curly−Coated Retriever dog
Cursinu dog
Czechoslovakian Wolfdog
Dachshund dog
Dalbo dog
Dalmatian dog
Dandie Dinmont Terrier dog
Danish−Swedish Farmdog
Deutsche Bracke dog
Doberman Pinscher dog
Dogo Argentino
Dogo Cubano
Dogue de Bordeaux
Drentse Patrijshond dog
Drever dog
Dunker dog
Dutch Shepherd dog
Dutch Smoushond dog
East Siberian Laika dog
East European Shepherd dog
Elo dog
English Cocker Spaniel dog
English Foxhound dog
English Mastiff dog
English Pointer dog
English Setter dog
English Shepherd dog
English Springer Spaniel dog
English Toy Terrier (Black & Tan) dog
English Water Spaniel dog
English White Terrier dog
Entlebucher Mountain Dog
Estonian Hound dog
Estrela Mountain Dog
Eurasier dog
Field Spaniel dog
Fila Brasileiro dog
Finnish Hound dog
Finnish Lapphund dog
Finnish Spitz dog

Flat−Coated Retriever dog
Fox Terrier , Smooth dog
Fox Terrier , Wire dog
French Brittany dog
French Bulldog
French Spaniel dog
Gaddi Kutta dog
Galgo Espaol dog
Galician Shepherd Dog
Garafian Shepherd dog
Gascon Saintongeois dog
Georgian Shepherd dog
German Longhaired Pointer dog
German Pinscher dog
German Roughhaired Pointer dog
German Shepherd Dog
German Shorthaired Pointer dog
German Spaniel dog
German Spitz dog
German Wirehaired Pointer dog
Giant Schnauzer dog
Glen of Imaal Terrier dog
Golden Retriever dog
Gordon Setter dog
Gran Mastn de Bornquen dog
Grand Anglo−Franais Blanc et Noir dog
Grand Anglo−Franais Blanc et Orange dog
Grand Anglo−Franais Tricolore dog
Grand Basset Griffon Venden dog
Grand Bleu de Gascogne dog
Grand Griffon Venden dog
Great Dane dog
Great Pyrenees dog
Greater Swiss Mountain Dog
Greek Harehound dog
Greek Shepherd dog
Greenland Dog
Greyhound dog
Griffon Bleu de Gascogne dog
Griffon Fauve de Bretagne dog
Griffon Nivernais dog
Guatemalan Dogo
Gull Terrier dog
Hamiltonstvare dog
Hanover Hound dog
Hare Indian Dog
Harrier dog
Havanese dog
Hawaiian Poi Dog
Himalayan Sheepdog
Hokkaido dog
Hortaya borzaya dog
Hovawart dog
Huntaway dog
Hygen Hound dog
Ibizan Hound dog
Icelandic Sheepdog
Indian pariah dog
Indian Spitz dog
Irish Red and White Setter dog
Irish Setter dog
Irish Terrier dog
Irish Water Spaniel dog
Irish Wolfhound dog
Istrian Coarse−haired Hound dog
Istrian Short−haired Hound dog
Italian Greyhound dog
Jack Russell Terrier dog
Jagdterrier dog
Swedish Elkhound dog
Japanese Chin dog
Japanese Spitz dog
Japanese Terrier dog
Jindo dog
Jonangi dog
Kaikadi dog
Kai Ken dog
Kangal Shepherd Dog
Kanni dog
Karakachan dog
Karelian Bear Dog
Karst Shepherd dog
Keeshond dog
Kerry Beagle dog
Kerry Blue Terrier dog
King Charles Spaniel dog
King Shepherd dog
Kintamani dog
Kishu Ken dog
Komondor dog
Koolie dog
Koyun dog
Kromfohrlnder dog
Kumaon Mastiff dog
Kunming Wolfdog
Kur dog
Kuvasz dog
Kyi−Leo dog
Labrador Husky dog
Labrador Retriever dog

Lagotto Romagnolo dog
Lakeland Terrier dog
Lancashire Heeler dog
Landseer dog
Lapponian Herder dog
Lapponian Shepherd dog
Leonberger dog
Lhasa Apso dog
Lithuanian Hound dog
Louisiana Catahoula Leopard Dog
Lwchen dog
Mackenzie River husky dog
Magyar agr dog
Mahratta Greyhound dog
Maltese dog
Manchester Terrier dog
Maremma Sheepdog
Marquesan Dog
McNab dog
Miniature American Shepherd dog
Miniature Bull Terrier dog
Miniature Fox Terrier dog
Miniature Pinscher dog
Miniature Schnauzer dog
Miniature Shar Pei dog
Molossus dog
Molossus of Epirus dog
Montenegrin Mountain Hound dog
Moscow Watchdog
Moscow Water Dog
Mountain Cur dog
Mucuchies dog
Mudhol Hound dog
Mudi dog
Mnsterlnder , Large dog
Mnsterlnder , Small dog
Neapolitan Mastiff dog
Nederlandse Kooikerhondje dog
Newfoundland dog
New Zealand Heading Dog
Norfolk Spaniel dog
Norfolk Terrier dog
Norrbottenspets dog
North Country Beagle dog
Northern Inuit Dog
Norwegian Buhund dog
Norwegian Elkhound dog
Norwegian Lundehund dog
Norwich Terrier dog
Nova Scotia Duck Tolling Retriever dog
Old Croatian Sighthound dog
Old Danish Pointer dog
Old English Bulldog
Old English Sheepdog
Old English Terrier dog
Old German Shepherd Dog
Old Spanish Pointer dog
Old Time Farm Shepherd dog
Olde English Bulldogge
Otterhound dog
Pachon Navarro dog
Pandikona dog
Paisley Terrier dog
Papillon dog
Parson Russell Terrier dog
Pastore della Lessinia e del Lagorai dog
Patterdale Terrier dog
Pekingese dog
Perro de Pastor Mallorquin dog
Perro de Presa Canario dog
Perro de Presa Mallorquin dog
Peruvian Inca Orchid dog
Petit Basset Griffon Venden dog
Petit Bleu de Gascogne dog
Phalne dog
Pharaoh Hound dog
Phu Quoc Ridgeback dog
Picardy Spaniel dog
Plummer Terrier dog
Plott Hound dog
Podenco Canario dog
Poitevin dog
Polish Greyhound dog
Polish Hound dog
Polish Hunting Dog
Polish Lowland Sheepdog
Polish Tatra Sheepdog
Pomeranian dog
Pont–Audemer Spaniel dog
Poodle dog
Porcelaine dog
Portuguese Podengo dog
Portuguese Pointer dog
Portuguese Water Dog
Posavac Hound dog
Potsdam Greyhound dog
Prask Krysak dog
Pudelpointer dog
Pug dog
Puli dog

Pumi dog
Pungsan dog
Pyrenean Mastiff dog
Pyrenean Shepherd dog
Rafeiro do Alentejo dog
Rajapalayam dog
Rampur Greyhound dog
Rastreador Brasileiro dog
Rat Terrier dog
Ratonero Bodeguero Andaluz dog
Ratonero Mallorquin dog
Ratonero Murciano de Huerta dog
Ratonero Valenciano dog
Redbone Coonhound dog
Rhodesian Ridgeback dog
Romanian Mioritic Shepherd Dog
Romanian Raven Shepherd Dog
Rottweiler dog
Russian Spaniel dog
Russian Toy dog
Russian Tracker dog
Russo–European Laika dog
Russell Terrier dog
Saarloos Wolfdog
Sabueso Espaol dog
Sabueso fino Colombiano dog
Saint Bernard dog
Saint John's water dog
Saint–Usuge Spaniel dog
Sakhalin Husky dog
Salish Wool Dog
Saluki dog
Samoyed dog
Sapsali dog
arplaninac dog
Schapendoes dog
Schillerstvare dog
Schipperke dog
Schweizer Laufhund dog
Schweizerischer Niederlaufhund dog
Scotch Collie dog
Scottish Deerhound dog
Scottish Terrier dog
Sealyham Terrier dog
Segugio Italiano dog
Seppala Siberian Sleddog
Serbian Hound dog
Serbian Tricolour Hound dog
Seskar Seal Dog
Shar Pei dog
Shetland Sheepdog
Shiba Inu dog
Shih Tzu dog
Shikoku dog
Shiloh Shepherd dog
Siberian Husky dog
Silken Windhound dog
Silky Terrier dog
Sinhala Hound dog
Skye Terrier dog
Sloughi dog
Slovak Cuvac dog
Slovakian Wirehaired Pointer dog
Slovensk kopov dog
Smlandsstvare dog
Small Greek Domestic Dog
Soft–Coated Wheaten Terrier dog
South Russian Ovcharka dog
Southern Hound dog
Spanish Mastiff dog
Spanish Water Dog
Spinone Italiano dog
Sporting Lucas Terrier dog
Stabyhoun dog
Staffordshire Bull Terrier dog
Standard Schnauzer dog
Stephens Cur dog
Styrian Coarse–haired Hound dog
Sussex Spaniel dog
Swedish Lapphund dog
Swedish Vallhund dog
Tahitian Dog
Tahltan Bear Dog
Taigan dog
Taiwan Dog
Talbot Hound dog
Tamaskan Dog
Teddy Roosevelt Terrier dog
Telomian dog
Tenterfield Terrier dog
Terceira Mastiff dog
Thai Bangkaew Dog
Thai Ridgeback dog
Tibetan Mastiff dog
Tibetan Spaniel dog
Tibetan Terrier dog
Tornjak dog
Tosa dog
Toy Bulldog
Toy Fox Terrier dog

Toy Manchester Terrier dog
Toy Trawler Spaniel dog
Transylvanian Hound dog
Treeing Cur dog
Treeing Tennessee Brindle dog
Treeing Walker Coonhound dog
Trigg Hound dog
Tweed Water Spaniel dog
Tyrolean Hound dog
Cimarrn Uruguayo dog
Vanjari Hound dog
Villano de Las Encartaciones dog
Villanuco de Las Encartaciones dog
Vizsla dog
Volpino Italiano dog
Weimaraner dog
Welsh Corgi, Cardigan dog
Welsh Corgi, Pembroke dog
Welsh Sheepdog
Welsh Springer Spaniel dog
Welsh Terrier dog
West Highland White Terrier dog
West Siberian Laika dog
Westphalian Dachsbracke dog
Wetterhoun dog
Whippet dog
White Shepherd dog
Wirehaired Pointing Griffon dog
Wirehaired Vizsla dog
Xiasi Dog
Xoloitzcuintli dog
Yakutian Laika dog
Yorkshire Terrier dog

References

[1] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014.

[2] G. Chen, P. Chen, Y. Shi, C.-Y. Hsieh, B. Liao, and S. Zhang, "Rethinking the usage of batch normalization and dropout in the training of deep neural networks," 2019.

[3] L. N. Smith, "Cyclical learning rates for training neural networks," 2015.

[4] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, "mixup: Beyond empirical risk minimization," 2017.

[5] A. Borji, "Pros and cons of GAN evaluation measures," *CoRR*, vol. abs/1802.03446, 2018.

[6] "Wikipedia: List of Dog Breeds." `https://en.wikipedia.org/wiki/List_of_dog_breeds`. Accessed: 2020-01-04.

[7] "Source Code: ImageDatasetCompiler." `https://github.com/chrismolli/ImageDatasetCompiler`. Accessed: 2020-01-04.

[8] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," 2014.

[9] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," 2015.

[10] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training gans," 2016.

[11] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein gan," 2017.

[12] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida, "Spectral normalization for generative adversarial networks," *CoRR*, vol. abs/1802.05957, 2018.

[13] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, "Improved training of wasserstein gans," *CoRR*, vol. abs/1704.00028, 2017.

[14] "Source Code: MLP, CNN and GAN models for MNIST, CIFAR-10 and custom datasets.." `https://github.com/chrismolli/redes_neuronales_semester_project`. Accessed: 2020-01-31.