УДК 004.8

# Применение условных генеративно-состязательных сетей на примере изображений пород собак

*Черкашин Александр Михайлович*
*Приамурский государственный университет имени Шолом-Алейхема*
*Студент*

**Аннотация**
В данной статье описан процесс использования модели условных генеративно-состязательных сетей (CGAN) для генерации изображения пород собак. В работе использовалась библиотека Torch, и модель условные генеративно-состязательные сети (CGAN) для пород собак. В наборе данных представлены разные породы собаки. В результате работы, модель CGAN способен генерировать породы собаки, а также была оценена модель для генерации изображения, в результате работы модель возможно получить новую комбинацию изображения генерированный порода собак.
**Ключевые слова:** GAN, CGAN, Сверточные нейронные сети, Torch.

# Application of conditional generative adversarial networks using images of dog breeds as an example

*Cherkashin Alexander Mihailovich*
*Sholom-Aleichem Priamursky State University*
*student*

**Abstract**
This article describes the process of using a conditional generative adversarial network (CGAN) model to generate images of dog breeds. The work used the Torch library and a conditional generative adversarial network (CGAN) model for dog breeds. The dataset contains different dog breeds. As a result of the work, the CGAN model is able to generate dog breeds, and the model for image generation was also evaluated, as a result of the work of the model it is possible to obtain a new image combination of the generated dog breed.
**Keywords:** GAN, CGAN, Convolutional Neural Network, Torch.

## 1 Введение

### *1.1. Актуальность исследования*

Актуальность исследования заключается в направлении в области машинного обучения и компьютерного зрения. Одним из примеров их применения является генерация изображений пород собак. Эта технология позволяет создавать высококачественные и реалистичные изображения, которые могут быть использованы в различных областях, включая развлекательную индустрию, дизайн и обучение моделей на основе данных.

CGAN используют концепцию генеративных сетей, которые обучаются создавать данные, имитирующие распределение обучающего набора. В контексте генерации изображений собак, CGAN могут принимать условие в виде метки породы, что позволяет точно контролировать процесс генерации. Это дает возможность создавать уникальные изображения различных пород собак с учетом заданных характеристик.

Продвижение в области CGAN открывает новые перспективы в сфере синтеза изображений, привнося инновации в области искусственного интеллекта. Генерация реалистичных изображений пород собак становится более точной и многогранной благодаря использованию CGAN, что акцентирует их значимость в научных и практических приложениях.

*1.2. Цель исследования*

Целью работы создания и обучение модели для подделки изображения породы собак.

*1.3. Обзор исследований*

Исследование, проведенное Г.Г. Крисосом, Дж. Коссаифи и С. Зафейрио, изучает надежные состязательные сети генерации условий. Исследование углубляется в разработку и применение этих сетей с целью достижения надежности при генерации условий. Исследование, вероятно, исследует эффективность предлагаемой состязательной сети в создании условий, устойчивых к возмущениям и состязательным атакам. Подробное понимание методологии, экспериментальной установки и результатов будет необходимо для формирования всестороннего понимания результатов исследования [1].

Исследование, проведенное Х. Чжаном, В. Синдаги и В. М. Пателем, сосредоточено на удалении изображения с использованием состязательной сети генерации условий (CGAN). Исследование углубляется в область обработки изображений, в частности, решая задачу удаления нежелательных артефактов или «колец» с изображений. Предлагаемый подход использует CGAN, который представляет собой тип генеративно-состязательной сети, предназначенной для генерации условных изображений. Исследование, вероятно, изучает эффективность этого метода в уменьшении артефактов звона на изображениях и может дать представление о достижениях в методах восстановления изображений [2].

Х. Дина и др. исследует применение состязательных сетей непрерывной генерации условий (CCGAN) в контексте генерации изображений. Исследователи стремятся повысить качество генерируемых изображений за счет внедрения подхода непрерывной генерации условий. Исследование, вероятно, углубляется в технические аспекты CCGAN и его влияние на повышение реализма и разнообразия генерируемых изображений. Для более глубокого понимания рекомендуется просмотреть полное исследование для получения подробной информации о методологии, результатах и последствиях [3].

Исследование О.М. Эземе, К.Х. Махмуда и А. Азима фокусируется на проектировании и разработке Ad-CGAN, что означает условное создание

состязательных сетей, для обнаружения аномалий. Исследование углубляется в использование генеративно-состязательных сетей (GAN) в условных условиях для выявления и обнаружения аномалий. Авторы исследуют применение Ad-CGAN в контексте обнаружения аномалий, демонстрируя его потенциал в создании соответствующих условных выборок для этой цели. Исследование вносит вклад в область обнаружения аномалий посредством разработки и анализа модели Ad-CGAN [4].

Исследование, проведенное Г. Рампони представляет T-Cgan, условно-генеративно-состязательную сеть, предназначенную для увеличения данных в зашумленных временных рядах с нерегулярной выборкой. Авторы стремятся решить проблемы, возникающие из-за нерегулярной выборки данных временных рядов, используя возможности генеративно-состязательных сетей (GAN). T-Cgan предлагается в качестве решения для повышения качества дополнения данных, особенно в контексте зашумленных данных временных рядов с нерегулярными шаблонами выборки. Исследование, вероятно, изучает эффективность T-Cgan в повышении надежности и обобщении моделей, обученных на таких данных, что способствует более широкой области анализа временных рядов и методов увеличения данных [5].

Исследование, проведенное Щ. Ху представляет собой прогнозирование действий человека с помощью состязательных сетей условного поколения» исследует использование состязательных сетей условного поколения (CGAN) для прогнозирования действий человека. Исследователи сосредоточены на использовании CGAN, типа генеративной модели, для повышения точности прогнозирования действий человека. Исследование, вероятно, углубляется в методологию использования CGAN в этом контексте, стремясь внести вклад в прогресс в области прогнозирования действий. Результаты и последствия исследования могут дать представление о повышении эффективности моделей прогнозирования действий человека [6].

## 2. Рабочий процесс

### *2.1. Набор данных*

Исходные данные Dogs. В исходные данные представлено серия изображения разного размера изображающий разных пород собак, количество 20 580, размер 778.1 МБ. В наборе данных содержится размеченная область изображений, указывающий породы собак, в разметке содержится информация название породы, поза и нахождение область собак. В наборе данных содержится 120 классов пород собак (рис 1) [7].

Этапы выполнение загрузки и обработки изображений и разметки (листинг 2.1):

1.Вырезаем изображений по заданным разметка область нахождение собак.

2. Уменьшаем разрешение изображений до 64x64 пикселей.

3. Преобразуем в тензор

Рисунок 1. Набор данных изображений породы собак

Листинг 2.1. Набор действия для обработки набор данных.

```
class DogDataset(Dataset):
    def __init__(self, path_data, transform = None):
        p_images = os.listdir(path_data + '/images/Images/')
        p_breeds = os.listdir(path_data + '/annotation/Annotation/')
        self.transform = transform
        data_image = []
        data_name = []
        # CROP WITH BOUNDING BOXES TO GET DOGS ONLY
        # https://www.kaggle.com/paulorzp/show-annotations-and-breeds
        for breed in p_breeds:
            for dog in os.listdir(path_data + '/annotation/Annotation/' + breed):
                try:
                    img = Image.open(path_data + "/images/Images/" + breed + "/" + dog + '.jpg')
                except:
                    continue
                tree = ET.parse(path_data + '/annotation/Annotation/' + breed + '/' + dog)
                root = tree.getroot()
                objects = root.findall('object')
                for o in objects:
                    bndbox = o.find('bndbox')
                    xmin = int(bndbox.find('xmin').text)
                    ymin = int(bndbox.find('ymin').text)
                    xmax = int(bndbox.find('xmax').text)
                    ymax = int(bndbox.find('ymax').text)
                    w = np.min((xmax - xmin, ymax - ymin))
                    img2 = img.crop((xmin, ymin, xmin+w, ymin+w))
                    img2 = img2.resize((64,64))
                    data_image.append(img2)
                    data_name.append(breed)
        self.data_image = data_image[:20_000]
        self.data_name = data_name[:20_000]
        self.label_target = torch.eye(20_000)
        self.label_zero = torch.zeros([20_000, 64 * 64 * 3])
    def __getitem__(self, index):
        label = self.data_name[index]
        image = self.data_image[index]
        label_target = self.label_target[index]
        label_zero = self.label_zero[index]
        if self.transform is not None:
            image = self.transform(image)
        return image[0:3], label_target, label_zero, label
    def __len__(self):
        return len(self.data_image)
```

Строка 3 — 9. Инициализация переменных и пути набор данных.

Строка 10 — 15. Загрузка изображений и область разметки данных.

Строка 16 — 29. Обработка и вырезание область изображений и изменение размера до 64x64 изображений.

Строка 30 — 33. Создание единичная матрица для метки истины, а ложная метка в нулевой значение, и обрезание размер массива данных до 20 000 признаков.

Строка 34 — 41. Выведение выделенных набор данных.

Строка 42 — 43. Функция получение размер массива данных.

Листинг 2.2. Действия обработки данных.

```
1  self.transform = transforms.Compose([
2          transforms.Resize(64),
3          transforms.ToTensor()
4  ])
```

Строка 2. Уменьшаем изображений до 64x64.

Строка 3. Преобразуем изображений в тензор.

Исходный код программы для обучения модели и предсказание взято с сайта [8].

*2.1. Модель*

CGAN (Conditional Generative Adversarial Network) - это тип генеративно-состязательных сетей (GAN), который имеет дополнительную особенность: он способен генерировать данные с учетом какого-то условия. В основе работы CGAN лежит взаимодействие двух нейронных сетей - генератора и дискриминатора.

Генератор отвечает за создание новых данных на основе случайного шума. Однако, в отличие от обычных GAN, CGAN принимает дополнительное условие в виде входных данных, которые указывают, какие характеристики или параметры должны присутствовать в создаваемых данных.

Дискриминатор, с другой стороны, обучается отличать между реальными данными и данными, созданными генератором. Здесь также используется условие, чтобы учитывать заданное условие при оценке данных.

Процесс обучения в CGAN включает в себя поочередное обучение генератора и дискриминатора. Генератор старается создавать данные, которые могли бы обмануть дискриминатор, в то время как дискриминатор стремится становиться все более точным в различении реальных данных от сгенерированных. Условие помогает генератору лучше соответствовать требованиям, заданным входными данными.

Этот процесс продолжается до тех пор, пока генератор не достигнет уровня создания данных, которые трудно отличить от реальных с точки зрения дискриминатора. CGAN может использоваться в различных задачах, таких как генерация изображений с определенными характеристиками или условиями [9].

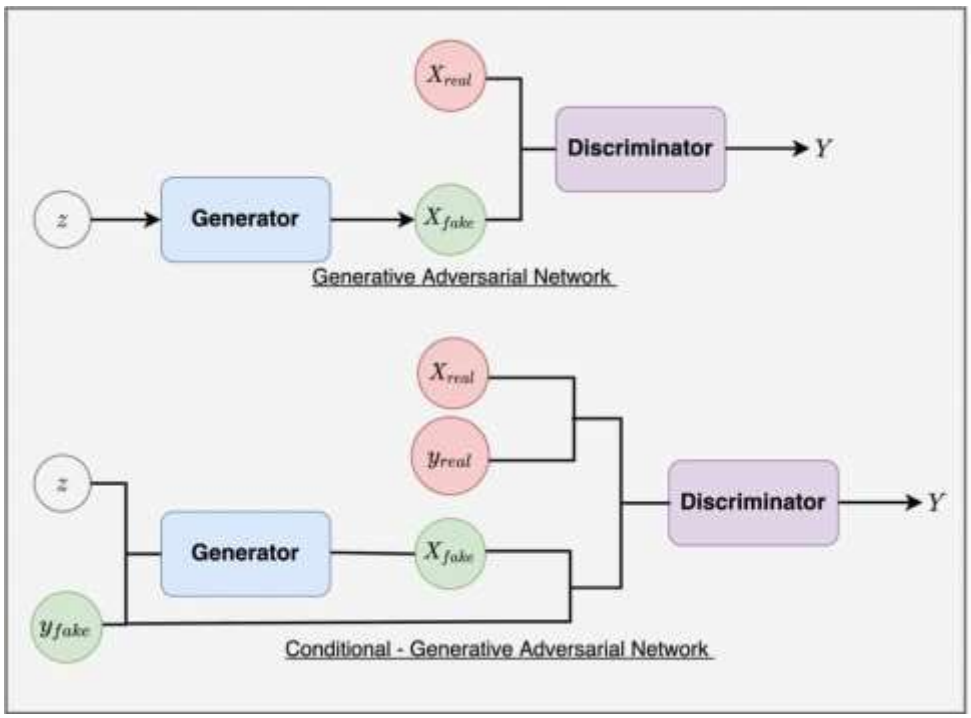


Рисунок 2. Отличие модели CGAN и GAN [10]

Модель дискриминант.

```
Discriminator(
  (fc1): Linear(in_features=20000, out_features=12288, bias=True)
  (conv1): Conv2d(1, 1, kernel_size=(2, 1), stride=(1, 1), bias=False)
)

----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1            [0, 1, 12288]     245,772,288
            Conv2d-2         [0, 1, 1, 12288]               2
================================================================
Total params: 245,772,290
Trainable params: 245,772,288
Non-trainable params: 2
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.00
Params size (MB): 937.55
Estimated Total Size (MB): 937.55
----------------------------------------------------------------
```

Модель генератор.

```
Generator(
  (fc1): Linear(in_features=20000, out_features=12288, bias=True)
)

----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1            [0, 1, 12288]     245,772,288
================================================================
Total params: 245,772,288
Trainable params: 245,772,288
```

```
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.00
Params size (MB): 937.55
Estimated Total Size (MB): 937.55
----------------------------------------------------------------
```

Листинг 2.3. Параметры для обучения модели

```
1   self.train_model = 0
2   self.criterion = nn.BCELoss().to(self.device)
3   self.gen_criterion = nn.MSELoss().to(self.device)
4   self.lr = 0.005
5   self.gen_lr = 0.01
6   self.model = {
7               "Gan_dis": Discriminator().to(self.device),
8               "Gan_gen": Generator().to(self.device)
9   }
10  self.optimizer = {
11              "Gan_dis": torch.optim.Adam(self.model["Gan_dis"].parameters(), lr=self.lr),
12              "Gan_gen": torch.optim.Adam(self.model["Gan_gen"].parameters(), lr=self.gen_lr)
13  }
14  self.model["Gan_dis"].conv1.weight = nn.Parameter(torch.Tensor([[[[ -1.0], [1.0]]]]).to(self.device))
15  for param in self.model["Gan_dis"].conv1.parameters():
16              param.requires_grad = False
```

Строка 1. Переключатель обучение 0 — обучение дискриминанта, 1 — обучение генератора.

Строка 2. Binary Cross Entropy between метрика для дискриминатора модели [11].

Строка 3. Метрика измерение среднеквадратическую отклонения MSELoss для генератора модели [12].

Строка 4. Скорость обучение для дискриминатора.

Строка 5. Скорость обучение для генератора.

Строка 6 — 9. Создаем экземпляр класса модели и загружаем в GPU.

Строка 10 — 13. Оптимизатор Adam.

Строка 14. Устанавливаем значение весов свертки модели генератора.

Строка 15 — 16. Отключаем изменение весов для обучение для сверточный сети модели генератора.

### *2.2. Обучение*

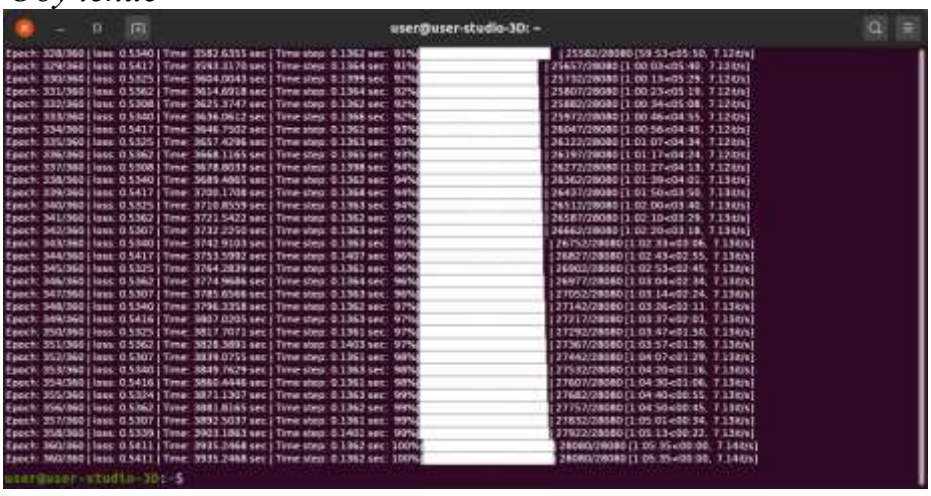


Рисунок 3. Обучение модели дискриминанта

Программа написано на языке Python использовалась библиотека Torch. Программа выполнялась с использованием графического ускорителя GA106 (RTX 3060).

Программа выполняет обучение модели дискримината (рис 3), задано эпохи 360, скорость обучение 0.005. Метрика BCELoss.

Рисунок 4. Обучение модели генератора

Программа выполняет обучение модели генератора (рис 4), задано эпохи 50, скорость обучение 0.01. Метрика MSELoss.

Листинг 2.4. Функция для обучения модели

```
1  def step_train(self, sel: any, index :int) -> None:
2          super().step_train(sel, index)
3          loss = None
4          images, label_target, label_zero, names = sel
5          images = images.reshape((-1, 64*64*3)).to(self.device)
6          label_target = label_target.to(self.device)
7          label_zero = label_zero.to(self.device)
8          if self.train_model == 0: # Train Discriminator
9                  self.model["Gan_dis"].zero_grad()
10                 y_pred = self.model["Gan_dis"](label_zero, label_target)
11                 loss = self.criterion(y_pred, images)
12                 loss.backward()
13                 self.optimizer["Gan_dis"].step()
14                 self.metric.loss = loss.item()
15         elif self.train_model == 1: # Train Generator
16                 self.model["Gan_gen"].zero_grad()
17                 fake, seed = self.model["Gan_gen"](label_target)
18                 y_pred = self.model["Gan_dis"](label_zero, seed)
19                 errG = self.gen_criterion(fake, y_pred)
20                 errG.backward()
21                 self.optimizer["Gan_gen"].step()
22                 self.metric.loss = errG.item()
```

Строка 2. Заглушка.

Строка 4. Получение images - изображения, label_target — метка единичный в диагонали, label_zero — метка нулей, names — имя породы.

Строка 5. Преобразование изображений в двухмерный массив (номер признака, линейный массив).

Строка 6 — 7. Загрузка тензоров в память GPU.

Строка 9 — 14. Обучение модели дискриминанта.

Строка 16 — 22. Обучение модели генератора.

*2.3. Оценка модели*

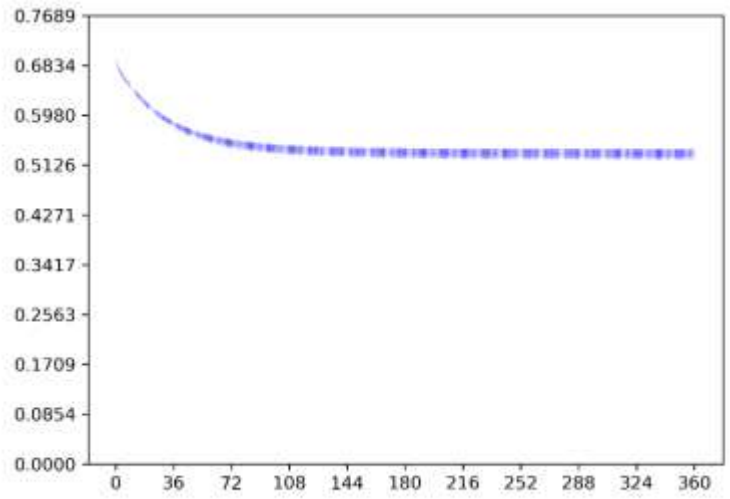


Рисунок 5. Функция потерь для дискриминатора (метрика BCELoss)

На графике представлено функция потерь по оси Y при 360 эпох по оси X, метрика, показатель оценки способность модели дискриминатора оценивать изображения подлинность, устойчивость к подделки.

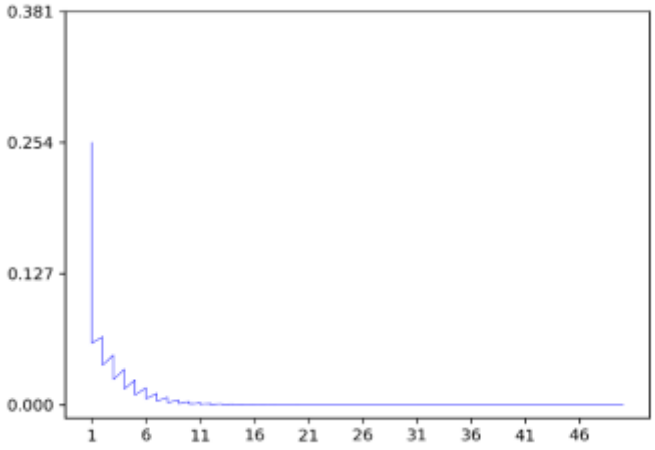


Рисунок 6. Функция потерь для генератора (метрика MSELoss)

Способность модели генератора обманывать модели дискриминатора. Эпох 50 по оси X.

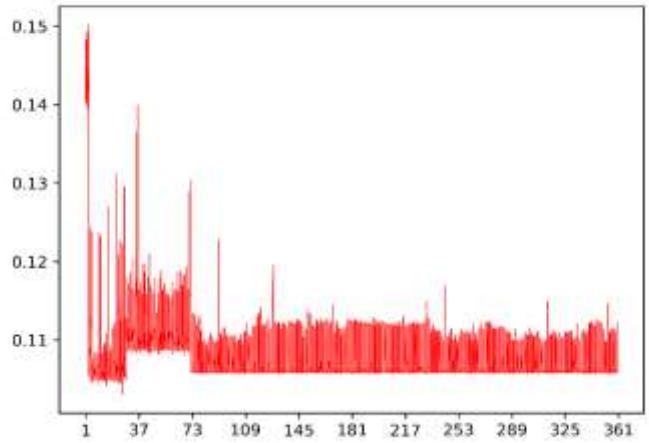


Рисунок 7. Время выполнения обучения дискриминатора в секундах.

На графике представлено время эпох выполнения обучения дискриминатора в секундах.

Время выполнение обучение дискриминанта 50 минуты4 12,2 секунда.

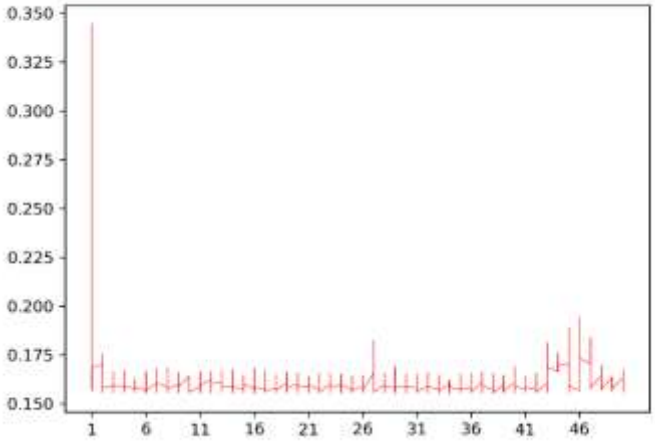


Рисунок 8. Время выполнения обучения генератора в секундах

На графике представлено время шага выполнения обучения генератора в секундах.

Время выполнение обучение генератора 10 минуты 23 секунда.

На графике представлено время шага выполнения обучения в секундах.

Время всего цикла обучение составляет 3635,2 секунда (60 минут, 35,2 секунда).

*2.4. Предсказания модели*

Листинг 2.5. Пример использования модели GAN.

```
1  model_dis = app.model["Gan_dis"]
2  optimizer_dis = app.optimizer["Gan_dis"]
3  model_gen = app.model["Gan_gen"]
4  optimizer_gen = app.optimizer["Gan_gen"]
5
6  def dis_predict():
7          label_zero = torch.zeros([20_000, 64 * 64 * 3])
8          xx = torch.Tensor(np.zeros((20_000))).to(app.device)
9          xx[np.random.randint(20_000)] = 1
10         img = model_dis(torch.Tensor(label_zero[0,:]).to(app.device).reshape((-1,12288)),xx.reshape((-1, 20_000))).reshape((-1,64,64,3))
```

```
11              img = img.detach().cpu().numpy()
12              img = Image.fromarray((255 * img).astype('uint8').reshape((64,64,3)))
13              return img
14  for i in range(10):
15              img_a = dis_predict()
16              app.save_image(app.get_path("view_test", "out_{0}.jpg".format(i)), img_a)
17
18  class DogGenerator():
19              index = 0
20              t = [
21                      transforms.RandomCrop((48, 48), padding=None, pad_if_needed=True, fill=0, padding_mode='symmetric'),
22                      transforms.Resize((64,64))
23              ]
24              tfms = transforms.Compose([
25                      transforms.RandomHorizontalFlip(p=0.5),
26                      transforms.ColorJitter(brightness=(1,1.3), contrast=(1,1.3), saturation=0, hue=0)
27              ])
28              def getDog(self,seed):
29                      xx = torch.Tensor(np.zeros((20_000))).to(app.device)
30                      xx[self.index] = 0.999999
31                      xx[np.random.randint(20_000)] = 0.000001
32                      img = model_gen(xx.reshape((-1,20_000)))
33                      img = img[0].reshape((64, 64, 3)).detach().cpu().numpy() * 255.0
34                      img = img.astype(np.int8)
35                      self.index = (self.index+1)%20_000
36                      return img
37  dgen = DogGenerator()
38  for i in range(10):
39              img_b = dgen.getDog(seed = np.random.normal(0,1,100))
40              app.save_image(app.get_path("view_test", "Gen_out_{0}.jpg".format(i)), img_b)
```

Строка 1 — 4. Использование модели дискриминанта и генератора.

Строка 6 — 13. Функция предсказание модели дискриминанта и пост обработки изображений.

Строка 14 — 16. Выводим изображений полученный дискриминанта.

Строка 18 — 36. Функция предсказание модели генератора и пост обработки изображений

Строка 37 — 40. Выводим изображений полученный генератора.

Рисунок 9. Пример выведено изображений модель дискриминатора

Рисунок 10. Пример выведено изображений модель генератора

**3 Выводы**

В статье была использована модель условных генеративно-состязательных сетей (CGAN) для генерации породы собак, в результате получили оценку работы модели, обучили модель дискриминанта заполнив набор данных и использовали модель дискриминанта качество памяти с отмеченным как подлинными, а также обучали генератора подделывать изображений. В результате работе было представлено изображений и метрики.

## Библиографический список

1. Chrysos G. G., Kossaifi J., Zafeiriou S. Robust conditional generative adversarial networks //arXiv preprint arXiv:1805.08657. – 2018.

2. Zhang H., Sindagi V., Patel V. M. Image de-raining using a conditional generative adversarial network //IEEE transactions on circuits and systems for video technology. 2019. Т. 30. №. 11. С. 3943-3956.
3. Ding X. et al. Ccgan: Continuous conditional generative adversarial networks for image generation //International conference on learning representations. – 2020.
4. Ezeme O. M., Mahmoud Q. H., Azim A. Design and development of AD-CGAN: Conditional generative adversarial networks for anomaly detection //IEEE Access. 2020. Т. 8. С. 177667-177681.
5. Ramponi G. et al. T-cgan: Conditional generative adversarial network for data augmentation in noisy time series with irregular sampling //arXiv preprint arXiv:1811.08295. 2018.
6. Xu W. et al. Prediction-cgan: Human action prediction with conditional generative adversarial networks //Proceedings of the 27th ACM international conference on multimedia. 2019. С. 611-619.
7. Stanford Dogs dataset for Fine-Grained Visual Categorization URL: http://vision.stanford.edu/aditya86/ImageNetDogs/ (дата обращения: 2024-01-15).
8. Memorizer CGAN - pyTorch version | Kaggle // Memorizer CGAN URL: https://www.kaggle.com/code/saneryee/memorizer-cgan-pytorch-version (дата обращения: 2024-01-15).
9. CGANs 101: What is a Conditional Generative Adversarial Network? | TaskUs URL: https://www.taskus.com/insights/cgans-101-what-is-a-conditional-generative-adversarial-network/ (дата обращения: 2024-01-16).
10. Conditional GAN (cGAN) in PyTorch and TensorFlow URL: https://learnopencv.com/conditional-gan-cgan-in-pytorch-and-tensorflow/ (дата обращения: 2024-01-16).
11. BCELoss — PyTorch 2.1 documentation // BCELoss URL: https://pytorch.org/docs/stable/generated/torch.nn.BCELoss.html (дата обращения: 2024-01-17).
12. MSELoss — PyTorch 2.1 documentation // MSELoss URL: https://pytorch.org/docs/stable/generated/torch.nn.MSELoss.html (дата обращения: 2024-01-17).

## 4. Приложения

Листинг 4.1. Исходный код программы

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

import torch
import torch.nn as nn
import torch.optim as optim
import numpy as np
import cv2
from torchsummary import summary
from torchvision import transforms, datasets
from torch.utils.data import Dataset
from torchvision.utils import save_image
import torchvision.transforms.functional as F
from typing import Optional, Callable, TypeVar, Type, Union
from PIL import Image
from Lib.AppMain import *
import Lib.Transforms as Trans
from torch.cuda import amp
import torch.nn.functional as nnf
import matplotlib.pyplot as plt
import os
from collections import deque
import time
import torch.nn.parallel
import torchvision.utils as vutils
from torch.autograd import Variable
import torch.nn.functional as F
import matplotlib.image as mpimg
from tqdm import tqdm_notebook as tqdm
import pandas as pd
import xml.etree.ElementTree as ET
import zipfile

class DogDataset(Dataset):
	def __init__(self, path_data, transform = None):
		p_images = os.listdir(path_data + '/images/Images/')
		p_breeds = os.listdir(path_data + '/annotation/Annotation/')
		self.transform = transform
		data_image = []
		data_name = []
		# CROP WITH BOUNDING BOXES TO GET DOGS ONLY
		# https://www.kaggle.com/paulorzp/show-annotations-and-breeds
		for breed in p_breeds:
			for dog in os.listdir(path_data + '/annotation/Annotation/' + breed):
				try:
					img = Image.open(path_data + "/images/Images/" + breed + "/" + dog + '.jpg')
				except:
					continue
				tree = ET.parse(path_data + '/annotation/Annotation/' + breed + '/' + dog)
				root = tree.getroot()
				objects = root.findall('object')
				for o in objects:
					bndbox = o.find('bndbox')
					xmin = int(bndbox.find('xmin').text)
					ymin = int(bndbox.find('ymin').text)
					xmax = int(bndbox.find('xmax').text)
					ymax = int(bndbox.find('ymax').text)
					w = np.min((xmax - xmin, ymax - ymin))
					img2 = img.crop((xmin, ymin, xmin+w, ymin+w))
					img2 = img2.resize((64,64))
					data_image.append(img2)
					data_name.append(breed)
		self.data_image = data_image[:20_000]
		self.data_name = data_name[:20_000]
		self.label_target = torch.eye(20_000)
		self.label_zero = torch.zeros([20_000, 64 * 64 * 3])

	def __getitem__(self, index):
		label = self.data_name[index]
		image = self.data_image[index]
		label_target = self.label_target[index]
		label_zero = self.label_zero[index]
		if self.transform is not None:
			image = self.transform(image)
		return image[0:3], label_target, label_zero, label
	def __len__(self):
		return len(self.data_image)

class Generator(nn.Module):
	def __init__(self):
		super(Generator, self).__init__()
		self.fc1 = nn.Linear(20_000, 64*64*3)

```

```
84              def forward(self, imgnames):
85                      x = self.fc1(imgnames)
86                      return x, imgnames.view(-1, 20_000)
87
88      class Discriminator(nn.Module):
89              def __init__(self):
90                      super(Discriminator, self).__init__()
91                      self.fc1 = nn.Linear(20_000, 64*64*3)
92                      self.conv1 = nn.Conv2d(1, 1, (2, 1), bias=False)
93
94              def forward(self, imgs, imgnames):
95                      x = self.fc1(imgnames)
96                      x = torch.sigmoid(x)
97                      x = torch.cat((imgs, x), dim=1).view(-1, 1, 2, 64*64*3)
98                      x = self.conv1(x)
99                      return x.view(-1, 64*64*3)
100
101     class App(AppMain):
102             def main(self):
103                     self.dir_prefix = "./Data"
104                     self.dirs = {
105                             "dataset": "Datasets",
106                             "fig": "Fig",
107                             "view": "View",
108                             "view_test": "View_test",
109                             "view_raw": "View_raw"
110                     }
111                     self.profile_name = "default"
112                     self.datasets = deque()
113                     self.model = {}
114                     self.optimizer = {}
115                     self.auto_save_exit = False
116                     self.disp_metric = ["loss"]
117                     self.metric.loss = None
118
119                     self.train_model = 0
120
121                     self.device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
122                     self.init_dirs()
123                     self.criterion = nn.BCELoss()
124                     self.gen_criterion = nn.MSELoss()
125                     self.lr = 0.005
126
127                     self.gen_lr = 0.01
128                     self.model = {
129                             "Gan_dis": Discriminator().to(self.device),
130                             "Gan_gen": Generator().to(self.device)
131                     }
132                     self.optimizer = {
133                             "Gan_dis": torch.optim.Adam(self.model["Gan_dis"].parameters(), lr=self.lr),
134                             "Gan_gen": torch.optim.Adam(self.model["Gan_gen"].parameters(), lr=self.gen_lr)
135                     }
136
137                     self.model["Gan_dis"].conv1.weight = nn.Parameter(torch.Tensor([[[[ -1.0], [1.0]]]]).to(self.device))
138
139                     for param in self.model["Gan_dis"].conv1.parameters():
140                             param.requires_grad = False
141
142                     self.transform = transforms.Compose([
143                             transforms.ToTensor()
144                     ])
145             def init_data(self):
146                     self.datasets.append(Datasets_batch(DogDataset(self.get_path("dataset"), self.transform), batch_size=256))
147                     self.init_datasets(num_workers=0)
148                     self.cache_data = True
149             def start_train(self) -> None:
150                     self.model["Gan_dis"].train()
151             def step_train(self, sel: any, index :int) -> None:
152                     super().step_train(sel, index)
153                     loss = None
154                     images, label_target, label_zero, names = sel
155                     images = images.reshape((-1, 64*64*3)).to(self.device)
156                     label_target = label_target.to(self.device)
157                     label_zero = label_zero.to(self.device)
158
159                     if self.train_model == 0: # Train Discriminator
160                             self.model["Gan_dis"].zero_grad()
161                             y_pred = self.model["Gan_dis"](label_zero, label_target)
162                             loss = self.criterion(y_pred, images)
163                             loss.backward()
164                             self.optimizer["Gan_dis"].step()
165                             self.metric.loss = loss.item()
166
167                     elif self.train_model == 1: # Train Generator
168                             self.model["Gan_gen"].zero_grad()
169                             fake, seed = self.model["Gan_gen"](label_target)
170                             y_pred = self.model["Gan_dis"](label_zero, seed)
171                             errG = self.gen_criterion(fake, y_pred)
172                             errG.backward()
```

```
                                        self.optimizer["Gan_gen"].step()
                                        self.metric.loss = errG.item()

        def load_image(self, path :str) -> Image.Image:
                with open(path, "rb") as f:
                        img = Image.open(f)
                        return img.convert("RGB")
        def save_image(self, path :str, img :Union[np.ndarray, Image.Image]) -> None:
                pic = None
                with open(path, "wb") as f:
                        if type(img) != Image.Image:
                                pic = Image.fromarray(img, "RGB")
                        else:
                                pic = img
                        pic.save(f)

app = App()
app.main()
app.init_data()

summary(app.model["Gan_dis"].to("cpu"), input_size=[(20_000, 64 * 64 * 3), (1, 20_000)], batch_size=0, device="cpu")
summary(app.model["Gan_gen"].to("cpu"), input_size=(1, 20_000), batch_size=0, device="cpu")

app.train_model = 0
app.fit(360)
app.save_model("Gan")

app.train_model = 1
app.fit(50)
app.save_model("Gan")

app.load_model("Gan")

model_dis = app.model["Gan_dis"]
optimizer_dis = app.optimizer["Gan_dis"]

model_gen = app.model["Gan_gen"]
optimizer_gen = app.optimizer["Gan_gen"]


def dis_predict():
        label_zero = torch.zeros([20_000, 64 * 64 * 3])
        xx = torch.Tensor(np.zeros((20_000))).to(app.device)
        xx[np.random.randint(20_000)] = 1
        img = model_dis(torch.Tensor(label_zero[0,:]).to(app.device).reshape((-1,12288)),xx.reshape((-1, 20_000))).reshape((-1,64,64,3))
        img = img.detach().cpu().numpy()
        img = Image.fromarray((255 * img).astype('uint8').reshape((64,64,3)))
        return img

for i in range(10):
        img_a = dis_predict()
        app.save_image(app.get_path("view_test", "out_{0}.jpg".format(i)), img_a)

class DogGenerator():
        index = 0
        t = [
                transforms.RandomCrop((48, 48), padding=None, pad_if_needed=True, fill=0, padding_mode='symmetric'),
                transforms.Resize((64,64))
        ]
        tfms = transforms.Compose([
                transforms.RandomHorizontalFlip(p=0.5),
                transforms.ColorJitter(brightness=(1,1.3), contrast=(1,1.3), saturation=0, hue=0)
                ])
        def getDog(self,seed):
                xx = torch.Tensor(np.zeros((20_000))).to(app.device)
                xx[self.index] = 0.999999
                xx[np.random.randint(20_000)] = 0.000001
                img = model_gen(xx.reshape((-1,20_000)))
                img = img[0].reshape((64, 64, 3)).detach().cpu().numpy() * 255.0
                img = img.astype(np.int8)
                self.index = (self.index+1)%20_000
                return img

dgen = DogGenerator()
for i in range(10):
        img_b = dgen.getDog(seed = np.random.normal(0,1,100))
        app.save_image(app.get_path("view_test", "Gen_out_{0}.jpg".format(i)), img_b)

#-----------------------------------

with app.file_begin(app.get_path("metric", "{0}.json".format("default_Gan")), "rb") as f:
        app.metric.from_json(f.read())

df = pd.DataFrame(app.metric._data)

df_dis = df[(df["Epoch_max"] == 360)]
df_gen = df[(df["Epoch_max"] == 410)]

df_gen["Epoch"] = df_gen["Epoch"]-df_dis["Epoch"].max()
```

```
262
263 df_dis_time = df_dis[df_dis["Time"] < 0.5]
264 df_gen_time = df_gen[df_gen["Time"] < 0.5]
265
266 # Discriminator
267 x_step = 36
268
269 fig, ax = plt.subplots()
270 ax.plot(df_dis_time["Epoch"], df_dis_time["Time"], label ='Time', color="red", linewidth=0.5)
271 ax.set_xticks(np.arange(min(df_dis_time["Epoch"]), max(df_dis_time["Epoch"])+2, x_step))
272 fig.savefig(app.get_path("fig", "{0}.png".format("dis_Time")), dpi = 300)
273
274 #loss
275 x_step = 36
276
277 y_step = (df_dis["loss"].max() - df_dis["loss"].min()) / 2.0
278 fig, ax = plt.subplots()
279 ax.plot(df_dis["Epoch"], df_dis["loss"], label ='loss', color="blue", linewidth=0.5)
280 ax.set_xticks(np.arange(0, max(df_dis["Epoch"])+1, x_step))
281 ax.set_yticks(np.arange(0, max(df_dis["loss"]) + y_step * 1.0, y_step))
282 fig.savefig(app.get_path("fig", "{0}.png".format("dis_loss")), dpi = 300)
283
284 df_dis.iloc[df_dis["loss"].idxmin()]
285
286 # Generator
287 x_step = 5
288
289 fig, ax = plt.subplots()
290 ax.plot(df_gen_time["Epoch"], df_gen_time["Time"], label ='Time', color="red", linewidth=0.5)
291 ax.set_xticks(np.arange(min(df_gen_time["Epoch"]), max(df_gen_time["Epoch"])+1, x_step))
292 fig.savefig(app.get_path("fig", "{0}.png".format("gen_Time")), dpi = 300)
293
294 #loss
295 y_step = (df_gen["loss"].max() - df_gen["loss"].min()) / 2.0
296 fig, ax = plt.subplots()
297 ax.plot(df_gen["Epoch"], df_gen["loss"], label ='loss', color="blue", linewidth=0.5)
298 ax.set_xticks(np.arange(min(df_gen["Epoch"]), max(df_gen["Epoch"])+1, x_step))
299 ax.set_yticks(np.arange(0, max(df_gen["loss"]) + y_step * 1.0, y_step))
300 fig.savefig(app.get_path("fig", "{0}.png".format("gen_loss")), dpi = 300)
301
302 df_gen.iloc[list(df_gen.index).index(df_gen["loss"].idxmin())]
303
304 def get_time_format(data_frame, text):
305         time = data_frame["Time"].sum()/60
306         time_min = np.floor(time)
307         time_sec = (time - time_min) * 60
308         print("{0}: {1}:{2}".format(text, int(time_min), time_sec))
309
310 get_time_format(df_dis, "time train discriminator")
311 get_time_format(df_gen, "time train generator")
312 get_time_format(df, "time train All")
```