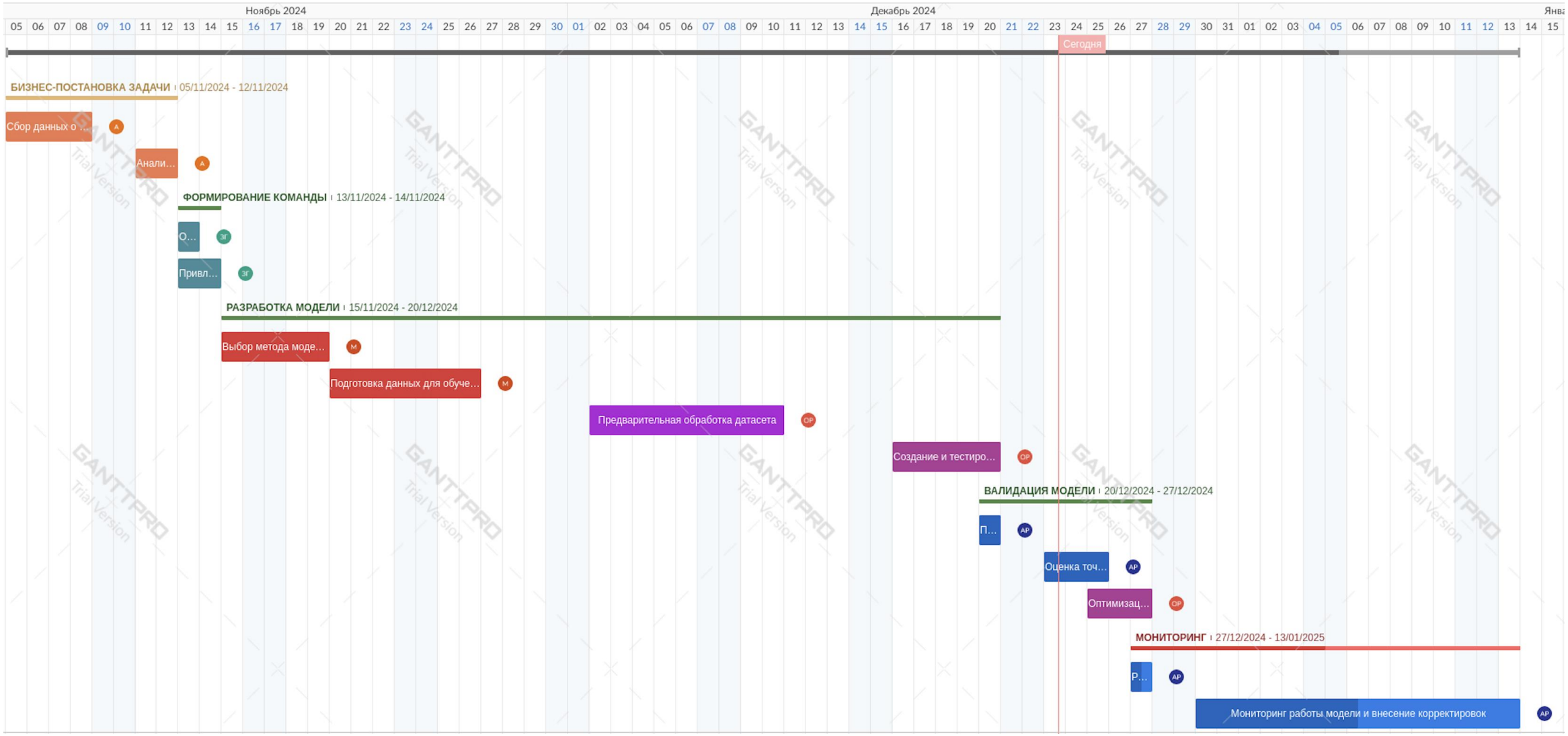
Модель для прогнозирования заболеваемости туберкулезом

Диаграмма Ганта



Анализ альтернатив

A. S. Jaeger алгоритм(программа CAD), который пошагово анализировал рентгенограммы грудной клетки для исключения различных форм туберкулеза
B. Ручная проверка, вариант оставить все «как есть»

Функциональные требования

- FR.01 Система должна иметь возможность принимать снимки флюорографии
- FR.02 Система должна сравнивать с критериями
- FR.03 Система должна использовать сохраненные в систему критерии
- FR.04 Система должна обрабатывать снимки
- FR.05 Система должна передавать статистику по снимкам
- FR.06 Система должна отображать информацию о результатах
- FR.07 Система должна выносить решение о туберкулезе на снимках

Описание датасета

Датасет состоит из 704 рентгеновских снимков грудной клетки, которые были получены из двух источников: базы данных рентгенографии грудной клетки округа Монтгомери (США) и базы данных рентгенографии грудной клетки в Шэньчжэне (Китай).

Набор данных содержит как рентгенограммы с положительным результатом туберкулеза, так и обычные рентгенограммы грудной клетки. Изображения сопровождаются масками для сегментации легких и клиническими метаданными.

Риски связанные с датасетом

- **Некачественные данные:** снимки сделаны разными аппаратами, что приводит к различиям в разрешении, контрасте и других характеристиках изображения.
- **Недостаток разнообразия данных:** все снимки были сделаны в одинаковых условиях, поэтому модель может не уметь адаптироваться к другим условиям.
- **Неправильная разметка данных:** Разметчики могут ошибаться при маркировке изображений как «туберкулёз» или «здоровый», особенно если признаки заболевания трудноразличимы.

```
import torch
import numpy as np
import pandas as pd
from tqdm import tqdm
import seaborn as sns
from pylab import rcParams
import matplotlib.pyplot as plt
from matplotlib import rc
from sklearn.preprocessing import MinMaxScaler
from pandas.plotting import register_matplotlib_converters
from torch import nn, optim
from sklearn.metrics import classification_report
import torchvision
import math
import copy
from PIL import Image
from sklearn.model_selection import train_test_split
import torchvision.transforms as transforms
from torch.utils.data import DataLoader, TensorDataset
```

```
[ ] image_x, image_y = 512, 512
```

```
metadata=pd.read_csv('D:/dataset/chest_X-ray_Dataset_tuberculosis/MetaData.csv')
data_y=metadata["tuberculosis is positive or not"] # где у это название колонки 0 и 1
metadata=metadata.drop("tuberculosis is positive or not",axis = 1)


df=[]
for i in range(int(len(metadata)/2)):
    image_path = f"D:/dataset/chest_X-ray_Dataset_tuberculosis/Chest-X-Ray/Chest-X-Ray/image/{metadata['id'][i]}.png"
    mask_path = f"D:/dataset/chest_X-ray_Dataset_tuberculosis/Chest-X-Ray/Chest-X-Ray/mask/{metadata['id'][i]}.png"

    with Image.open(image_path).convert("L") as image_image, Image.open(mask_path) as image_mask:
        image_image=image_image.resize((image_x, image_y), Image.Resampling.LANCZOS)
        image_mask=image_mask.resize((image_x, image_y), Image.Resampling.LANCZOS)

        image_image_array = np.array(image_image)
        image_mask_array = np.array(image_mask)

        try:
            array = image_image_array * image_mask_array
            flattened_array = array.flatten().tolist()
            # Запись результатов в файл для каждого изображения
            df.append(flattened_array)
        except Exception as e:
            print(f"Error processing {metadata['id'][i]}: {e}")
```

Так как изображения имели разный размер требовалось привести их в единый для модели формат

```
dataframe = pd.DataFrame({"x":df,"y":data_y[0:352]})
```

```
dataframe.to_csv("output.csv",encoding="utf-8",sep="\n")
```

```
img_ar = np.array(df[26]).reshape(512,512)
```

```
img_ar.shape
```

```
(512, 512)
```

```
img = Image.fromarray(img_ar.astype(np.uint8))
img
```

```
model_Resnet50
```

```
ResNet(
  (conv1): Conv2d(1, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  (layer1): Sequential(
    (0): Bottleneck(
```

```
    (fc): Sequential(
      (0): Linear(in_features=2048, out_features=1, bias=True)
      (1): Sigmoid()
    )
  )
```

```
USE_GPU = False
if torch.cuda.is_available():
  USE_GPU=True
EPOCHS = 5
```

```
random_index = 512
```

```
X, Y = df,data_y
```

```
train_data, test_data = train_test_split(dataframe, test_size=0.25,random_state=random_index )
```

```
X_train = torch.tensor(np.vstack(train_data['x'].values).astype(int))
y_train = torch.tensor(np.vstack(train_data['y'].values).astype(int))

X_test = torch.tensor(np.vstack(test_data['x'].values).astype(int))
y_test = torch.tensor(np.vstack(test_data['y'].values).astype(int))
```

Код модели и длительность обучения

```python
model = model_Resnet50
if USE_GPU:
    model = model.cuda()
optimizer = torch.optim.Adam(model.parameters())
criterion = torch.nn.BCELoss()
best_model_wts = copy.deepcopy(model.state_dict())
best_acc = 0.0
batch_size = 1
# Создание датасетов
train_dataset = TensorDataset(X_train, y_train)
test_dataset = TensorDataset(X_test, y_test)
# Создание DataLoader'ов
train_loader = DataLoader(dataset=train_dataset, batch_size=batch_size, shuffle=True)
test_loader = DataLoader(dataset=test_dataset, batch_size=batch_size, shuffle=False)

best_acc = 0.0
train_hist = []
test_hist = []
for epoch in range(EPOCHS):
    for phase in ["train", "test"]:
        if phase == "train":
            model.train()
            data_loader = train_loader
        else:
            model.eval()
            data_loader = test_loader

        samples = 0
        loss_sum = 0
        correct_sum = 0
        for inputs, labels in data_loader:
            optimizer.zero_grad()

            with torch.set_grad_enabled(phase == 'train'):
                outputs = model(inputs.resize(batch_size,1,image_x,image_y).float())
                loss = criterion(outputs.float(), labels.float())

                if phase == 'train':
                    loss.backward()
                    optimizer.step()

                loss_sum += loss.item() * inputs.size(0)  # Обновляем общий loss
                preds = torch.round(outputs)  # Предсказание: 0 или 1
                correct_sum += torch.sum(preds == labels.data)  # Считаем правильные предсказания
            samples += inputs.size(0)
        epoch_loss = loss_sum / samples
        epoch_acc = correct_sum.double() / samples

        print(f"Epoch {epoch+1}/{EPOCHS} - {phase} Loss: {epoch_loss:.4f} Acc: {epoch_acc:.4f}")
        if phase == "train":
            train_hist.append(epoch_loss)
        else:
            test_hist.append(epoch_loss)
        # Копирование лучшей модели
        if phase == 'test' and epoch_acc > best_acc:
            best_acc = epoch_acc
            best_model_wts = copy.deepcopy(model.state_dict())
            torch.save(best_model_wts, "best_model.pth")
# Загрузка лучших весов модели
model.load_state_dict(best_model_wts)
```

```
Epoch 1/5 - train Loss: 0.7053 Acc: 0.5568
Epoch 1/5 - test Loss: 0.6727 Acc: 0.5341
Epoch 2/5 - train Loss: 0.6887 Acc: 0.5795
Epoch 2/5 - test Loss: 1.6018 Acc: 0.6250
Epoch 3/5 - train Loss: 0.6578 Acc: 0.6250
Epoch 3/5 - test Loss: 0.7869 Acc: 0.4659
Epoch 4/5 - train Loss: 0.6571 Acc: 0.6250
Epoch 4/5 - test Loss: 0.5515 Acc: 0.8068
Epoch 5/5 - train Loss: 0.6583 Acc: 0.5985
Epoch 5/5 - test Loss: 0.5592 Acc: 0.7841
<All keys matched successfully>
```

Результат модели

```
predictionlist=[]
for inputs, labels in data_loader:
    optimizer.zero_grad()
    with torch.set_grad_enabled(phase == 'train'):
        outputs = model(inputs.resize(batch_size,1,image_x,image_y).float())
        loss = criterion(outputs.float(), labels.float())
        predictionlist.extend(torch.round(outputs.flatten()).tolist())
```

```
C:\Users\макс\AppData\Roaming\Python\Python310\site-packages\torch\_tensor.py:775: UserWarning: non-inplace resize is deprecated
  warnings.warn("non-inplace resize is deprecated")
```

```
predictions = [int(item) for item in predictionlist]
```

```
print(classification_report(predictions,y_test))
```

```
              precision    recall  f1-score   support

           0       0.74      0.88      0.80        40
           1       0.88      0.75      0.81        48

    accuracy                           0.81        88
   macro avg       0.81      0.81      0.81        88
weighted avg       0.82      0.81      0.81        88
```

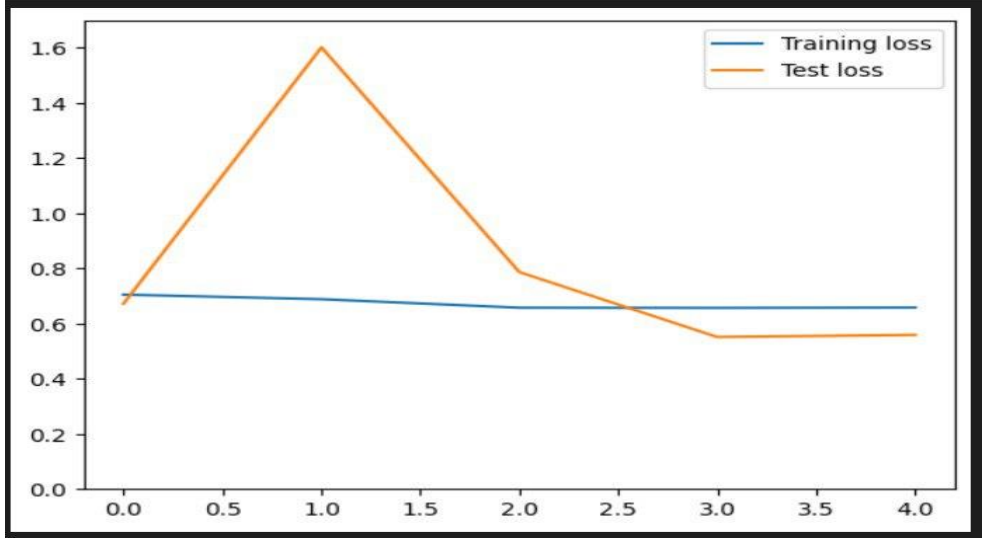
```
p
```

```
plt.plot(train_hist, label="Training loss")
plt.plot(test_hist, label="Test loss")
plt.ylim((0, 5))
plt.legend();
```

```
```

```
plt.plot(train_hist, label="Training loss")
plt.plot(test_hist, label="Test loss")
plt.ylim((0, 1.7))
plt.legend();
```

0.1s




```
plt.plot(train_acc, label="Training acc")
plt.plot(test_acc, label="Test acc")
plt.ylim((0, 1))
plt.legend();
```

0.1s