Федеральное государственное бюджетное образовательное учреждение высшего образования

**«ФИНАНСОВЫЙ УНИВЕРСИТЕТ ПРИ ПРАВИТЕЛЬСТВЕ РОССИЙСКОЙ ФЕДЕРАЦИИ»**

**Департамент анализа данных и машинного обучения**

Пояснительная записка к курсовой работе

по дисциплине “Машинное обучение”

на тему:

# «Машинное обучение в задачах идентификации личности по изображению»

Выполнил:

студент группы ПМ22-6 факультета информационных технологий и анализа больших данных

Гусев Яромир Георгиевич

Научный руководитель:

Фурлетов Юрий Михайлович

2024 г

# Гусев Яромир ПМ22-6

# Машинное обучение в задачах идентификации личности по изображению

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import cv2
import os
from skimage.feature import hog, local_binary_pattern
from skimage import exposure
from PIL import Image
from tqdm import tqdm, trange
import seaborn as sns
from copy import deepcopy
import time

from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, f1_score
from sklearn.ensemble import RandomForestClassifier
from catboost import CatBoostClassifier, Pool
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB

from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline

from torch.autograd import Variable
import torchvision.transforms as transforms
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.utils.data import Dataset
import torch
import timm
```

Для данной задчи я выбрал набор данных CelebA_500, который является подмножеством известного набора данных CelebA, который содержит более 200 000 изображений известных знаменитостей из различных областей, таких как кино, спорт, музыка и т.д. CelebA_500 содержит изображения лишь 500 человек, что делает его более компактным и подходящим для экспериментов и тестирования. Решаемой задачей является классификация личности на основе изображения.

In [74]:
```
img_names = os.listdir('./celebA_500/celebA_imgs/')
print(f'Всего {len(img_names)} картинок')
```

```
Всего 23948 картинок
```

In [28]:
```
f = open('celebA_500/celebA_anno.txt')
lines = f.readlines()
dict1 = dict()
for i in range(0,500):
    dict1[i] = []
for i in lines:
    z = i.split()
    dict1[int(z[1])].append(z[0])

a = {1:[2,3]}
names = [dict1[i] for i in dict1]
cnts = [len(dict1[i]) for i in dict1]
df = pd.DataFrame({'images':names,'img_counts':cnts})
df
```

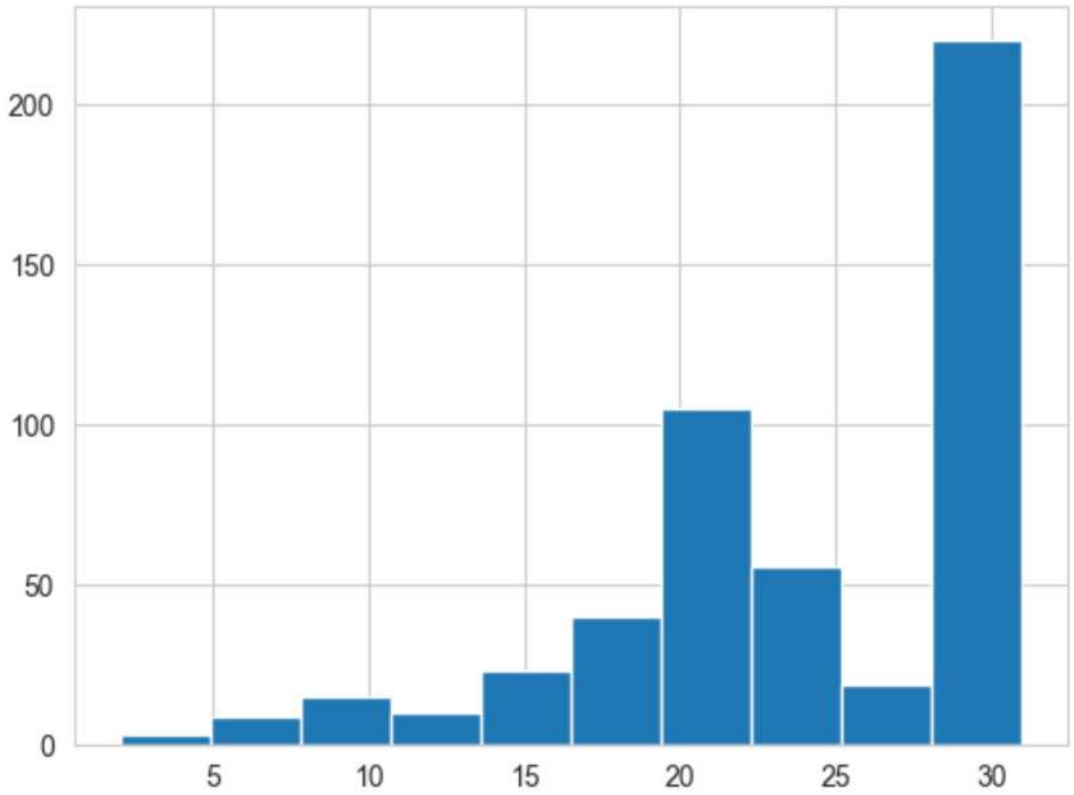
Out[28]:

| | images | img_counts |
|---|---|---|
| **0** | [000001.jpg, 000404.jpg, 003415.jpg, 004390.jp... | 30 |
| **1** | [000002.jpg, 011437.jpg, 016335.jpg, 017121.jp... | 30 |
| **2** | [000003.jpg, 015648.jpg, 033840.jpg, 038887.jp... | 30 |
| **3** | [000004.jpg, 001778.jpg, 010191.jpg, 013676.jp... | 30 |
| **4** | [000005.jpg, 008431.jpg, 014427.jpg, 016680.jp... | 25 |
| **...** | ... | ... |
| **495** | [000515.jpg, 005410.jpg, 020137.jpg, 024989.jp... | 20 |
| **496** | [000516.jpg, 033078.jpg, 058354.jpg, 059067.jp... | 22 |
| **497** | [000517.jpg, 001649.jpg, 029332.jpg, 034151.jp... | 22 |
| **498** | [000518.jpg, 001543.jpg, 003209.jpg, 008966.jp... | 30 |
| **499** | [000519.jpg, 004754.jpg, 005061.jpg, 007359.jp... | 28 |

500 rows × 2 columns

Выведем гистограмму распределения количества картинок для каждого класса

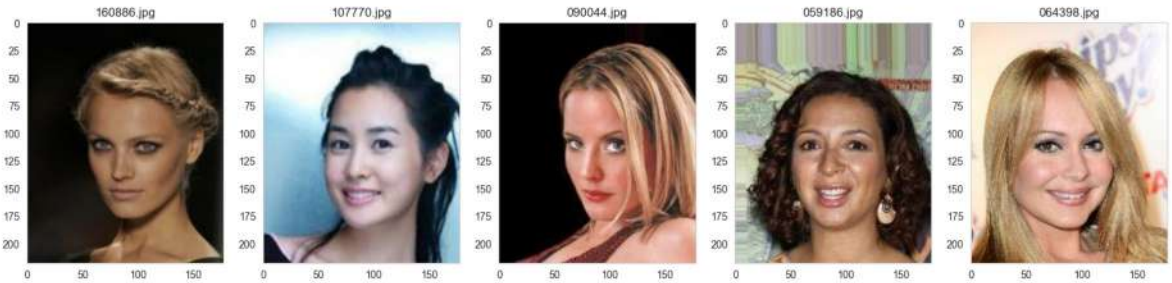
In [76]:
```
df['img_counts'].hist();
```




В целом, нет явного дисбаланса классов, так как классы имеют примерно одинаковое количество картинок в выборке.

Выведем 5 случайных картинок людей

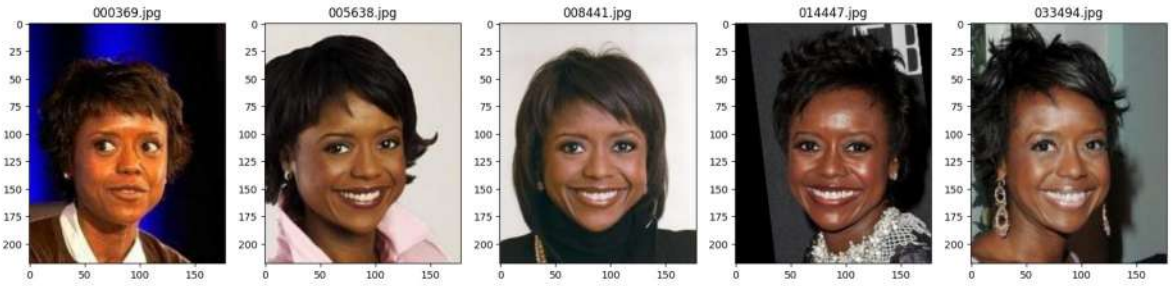
In [78]:
```
a = [img_names[i] for i in [np.random.randint(0,len(img_names)) for i in range(5
imgs = []
for img in a:
    img = Image.open(os.path.join('celebA_500/celebA_imgs/', img))
    imgs.append(np.array(img))

f, axes= plt.subplots(1, len(imgs), figsize=(4*len(imgs),5))
for i, axis in enumerate(axes):
    axes[i].grid(False)
    axes[i].imshow(imgs[i], cmap='gray')
    axes[i].set_title(a[i])
plt.show()
```




Выведем 5 картинок одного человека.

In [ ]:
```
a = df.iloc[np.random.randint(0,len(df))]['images'][:5]
imgs = []
for img in a:
    img = Image.open(os.path.join('celebA_500/celebA_imgs/', img))
```

```
    imgs.append(np.array(img))

f, axes= plt.subplots(1, len(imgs), figsize=(4*len(imgs),5))
for i, axis in enumerate(axes):
    axes[i].grid(False)
    axes[i].imshow(imgs[i], cmap='gray')
    axes[i].set_title(a[i])
plt.show()
```




In [29]:
```
for i in range(len(lines)):
    lines[i] = lines[i].split()[0]
```

In [ ]:
```
def load_images_from_folder(folder, lines):
    images = []
    filenames = []
    for filename in tqdm(lines):
        img = cv2.imread(os.path.join(folder, filename))
        if img is not None:
            images.append(img)
            filenames.append(filename)
    return images, filenames


folder = 'celebA_500/celebA_imgs'
images, filenames = load_images_from_folder(folder, lines)
```

```
100%|████████| 12011/12011 [00:06<00:00, 1787.94it/s]
```

Преобразуем картинки в плоские численные массивы. Я буду получать:

Гистограммы цветов: Для каждого канала цвета (красный, зелёный, синий) вычисляются гистограммы с использованием функции cv2.calcHist(). Гистограммы каждого канала сглаживаются и объединяются в один вектор признаков.

HOG (Histogram of Oriented Gradients) признаки: Эти признаки очень много весят, но при этом не особо влияют на качество классификации, так что их не будем использовать.

LBP (Local Binary Pattern) признаки: Вычисляются признаки LBP с использованием функции local_binary_pattern() из библиотеки scikit-image. Затем эти признаки нормализуются.

Статистические признаки: Вычисляются среднее, стандартное отклонение, максимальное и минимальное значения для каждого канала цвета.

Все эти признаки объединяются в один общий вектор признаков для каждого

изображения.

In [ ]:
```
def extract_features(images):
    all_features = []

    for img in tqdm(images):
        hist_features = []
        for i in range(3):  # Для каждого канала цвета
            hist = cv2.calcHist([img], [i], None, [256], [0, 256])
            hist_features.extend(hist.flatten())

        # HOG features
#         fd, hog_image = hog(img, orientations=8, pixels_per_cell=(16, 16),
#                             cells_per_block=(1, 1), visualize=True, channel_ax

#         LBP features
        lbp = local_binary_pattern(cv2.cvtColor(img, cv2.COLOR_BGR2GRAY), 8, 1,
        lbp_hist, _ = np.histogram(lbp.ravel(), bins=np.arange(0, 27), range=(0,
        lbp_hist = lbp_hist.astype("float")
        lbp_hist /= (lbp_hist.sum() + 1e-6)  # Normalize

#         Статистические признаки
        mean = np.mean(img, axis=(0, 1))
        std = np.std(img, axis=(0, 1))
        max_ = np.max(img, axis=(0, 1))
        min_ = np.min(img, axis=(0, 1))

        # Объединение признаков
        combined_features = np.hstack([
            hist_features,
#             fd,
            lbp_hist,
            mean,
            std,
            max_,
            min_
        ])
        all_features.append(combined_features)

    features_df = pd.DataFrame(all_features)

    return features_df

X = extract_features(images)
```

```
100%|████████| 12011/12011 [02:16<00:00, 88.25it/s]
```

In [ ]:
```
X
```

Out[ ]:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ... | 7 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1780.0 | 184.0 | 208.0 | 211.0 | 209.0 | 238.0 | 228.0 | 256.0 | 232.0 | 219.0 | ... | 181.0451 |
| **1** | 14.0 | 3.0 | 7.0 | 10.0 | 9.0 | 23.0 | 34.0 | 34.0 | 53.0 | 55.0 | ... | 120.9027 |
| **2** | 961.0 | 252.0 | 259.0 | 335.0 | 354.0 | 427.0 | 540.0 | 559.0 | 669.0 | 652.0 | ... | 106.8634 |
| **3** | 743.0 | 142.0 | 141.0 | 159.0 | 176.0 | 185.0 | 177.0 | 234.0 | 268.0 | 285.0 | ... | 115.3907 |
| **4** | 2635.0 | 686.0 | 624.0 | 642.0 | 663.0 | 627.0 | 574.0 | 545.0 | 456.0 | 491.0 | ... | 93.6027 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **12006** | 1305.0 | 431.0 | 600.0 | 788.0 | 783.0 | 740.0 | 672.0 | 608.0 | 526.0 | 488.0 | ... | 114.5991 |
| **12007** | 417.0 | 170.0 | 258.0 | 310.0 | 299.0 | 334.0 | 320.0 | 276.0 | 315.0 | 314.0 | ... | 165.1238 |
| **12008** | 380.0 | 146.0 | 178.0 | 202.0 | 237.0 | 259.0 | 308.0 | 315.0 | 364.0 | 360.0 | ... | 104.9312 |
| **12009** | 2435.0 | 886.0 | 792.0 | 630.0 | 636.0 | 651.0 | 643.0 | 672.0 | 604.0 | 573.0 | ... | 87.1556 |
| **12010** | 1273.0 | 758.0 | 565.0 | 579.0 | 564.0 | 585.0 | 572.0 | 600.0 | 435.0 | 417.0 | ... | 158.3221 |

12011 rows × 806 columns

In [26]:
```
targets = []
f = open('celebA_500/celebA_anno.txt')
lines2 = f.readlines()
for i in range(len(lines2)):
    targets.append(int(lines2[i].split()[1]))
```

In [ ]:
```
data = pd.concat([X, pd.Series(targets, name='target')], axis=1)
data
```

Out[ ]:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ... | 79 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1780.0 | 184.0 | 208.0 | 211.0 | 209.0 | 238.0 | 228.0 | 256.0 | 232.0 | 219.0 | ... | 71.63684 |
| **1** | 14.0 | 3.0 | 7.0 | 10.0 | 9.0 | 23.0 | 34.0 | 34.0 | 53.0 | 55.0 | ... | 53.66243 |
| **2** | 961.0 | 252.0 | 259.0 | 335.0 | 354.0 | 427.0 | 540.0 | 559.0 | 669.0 | 652.0 | ... | 39.66446 |
| **3** | 743.0 | 142.0 | 141.0 | 159.0 | 176.0 | 185.0 | 177.0 | 234.0 | 268.0 | 285.0 | ... | 41.65609 |
| **4** | 2635.0 | 686.0 | 624.0 | 642.0 | 663.0 | 627.0 | 574.0 | 545.0 | 456.0 | 491.0 | ... | 29.57742 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **12006** | 1305.0 | 431.0 | 600.0 | 788.0 | 783.0 | 740.0 | 672.0 | 608.0 | 526.0 | 488.0 | ... | 65.39935 |
| **12007** | 417.0 | 170.0 | 258.0 | 310.0 | 299.0 | 334.0 | 320.0 | 276.0 | 315.0 | 314.0 | ... | 80.02162 |
| **12008** | 380.0 | 146.0 | 178.0 | 202.0 | 237.0 | 259.0 | 308.0 | 315.0 | 364.0 | 360.0 | ... | 85.93983 |
| **12009** | 2435.0 | 886.0 | 792.0 | 630.0 | 636.0 | 651.0 | 643.0 | 672.0 | 604.0 | 573.0 | ... | 62.17703 |
| **12010** | 1273.0 | 758.0 | 565.0 | 579.0 | 564.0 | 585.0 | 572.0 | 600.0 | 435.0 | 417.0 | ... | 66.33800 |

12011 rows × 807 columns

Получился огромный датасет, который я сохраню в csv, чтобы не выполнять feature extraction каждый раз.

In [ ]:
```
data.to_csv('data.csv')
```

In [ ]:
```
data = pd.read_csv('data.csv').drop(['Unnamed: 0'], axis=1)
data
```

Out[ ]:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ... | 79 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1780.0 | 184.0 | 208.0 | 211.0 | 209.0 | 238.0 | 228.0 | 256.0 | 232.0 | 219.0 | ... | 71.63684 |
| **1** | 14.0 | 3.0 | 7.0 | 10.0 | 9.0 | 23.0 | 34.0 | 34.0 | 53.0 | 55.0 | ... | 53.66243 |
| **2** | 961.0 | 252.0 | 259.0 | 335.0 | 354.0 | 427.0 | 540.0 | 559.0 | 669.0 | 652.0 | ... | 39.66446 |
| **3** | 743.0 | 142.0 | 141.0 | 159.0 | 176.0 | 185.0 | 177.0 | 234.0 | 268.0 | 285.0 | ... | 41.65609 |
| **4** | 2635.0 | 686.0 | 624.0 | 642.0 | 663.0 | 627.0 | 574.0 | 545.0 | 456.0 | 491.0 | ... | 29.57742 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | . |
| **12006** | 1305.0 | 431.0 | 600.0 | 788.0 | 783.0 | 740.0 | 672.0 | 608.0 | 526.0 | 488.0 | ... | 65.39935 |
| **12007** | 417.0 | 170.0 | 258.0 | 310.0 | 299.0 | 334.0 | 320.0 | 276.0 | 315.0 | 314.0 | ... | 80.02162 |
| **12008** | 380.0 | 146.0 | 178.0 | 202.0 | 237.0 | 259.0 | 308.0 | 315.0 | 364.0 | 360.0 | ... | 85.93983 |
| **12009** | 2435.0 | 886.0 | 792.0 | 630.0 | 636.0 | 651.0 | 643.0 | 672.0 | 604.0 | 573.0 | ... | 62.17703 |
| **12010** | 1273.0 | 758.0 | 565.0 | 579.0 | 564.0 | 585.0 | 572.0 | 600.0 | 435.0 | 417.0 | ... | 66.33800 |

12011 rows × 807 columns

In [ ]:
```
X = data.drop(["target"], axis=1)
y = data["target"]
```

Выполню разбиение на тренировочную и тестовую выборки. Размер тестовой выборки равен 20% от всего датасета, так как обучающая выборка должна быть достаточно большой для того, чтобы модель могла изучить разнообразные закономерности в данных, а тестовая выборка должна быть достаточно большой для того, чтобы дать надежную оценку обобщающей способности модели. Также будет использован случайны метод разделения данных, чтобы избежать каких-либо предвзятостей или зависимостей между обучающей и тестовой выборками.

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.2,
                                                    random_state=42,
                                                    stratify=y,
                                                    shuffle=True)
```

Так как датасет получился крайне большой, необходимо произвести процедуру понижения размерности (PCA), так как иначе все модели будут учиться очень долго, а также нормализацию, для удобства буду использовать make_pipeline из sklearn.

In [ ]:
```
pipeline = make_pipeline(
```

```
    StandardScaler(),
    PCA(n_components=0.95)
)

X_train = pipeline.fit_transform(X_train)
X_test = pipeline.transform(X_test)

X_train.shape, X_test.shape
```

Out[ ]: ((9608, 275), (2403, 275))

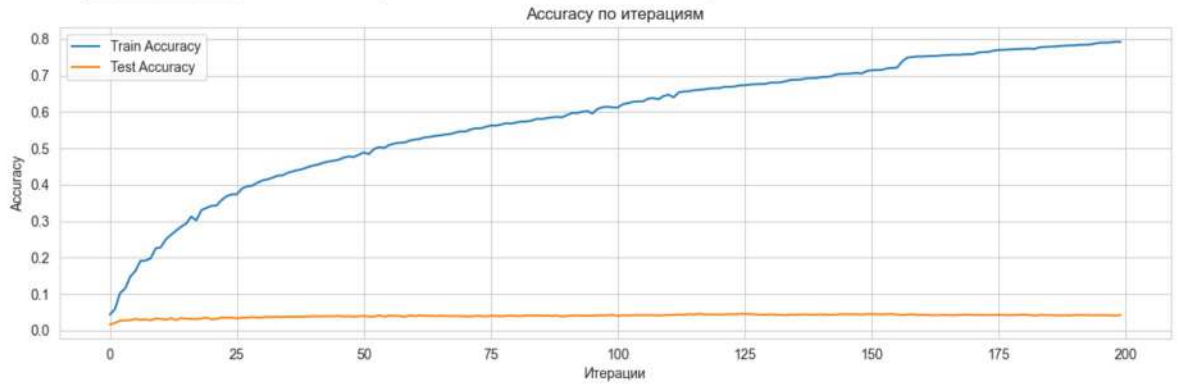
Осталось всего 275 столбцов, которые теперь я буду использовать для обучения моделей.

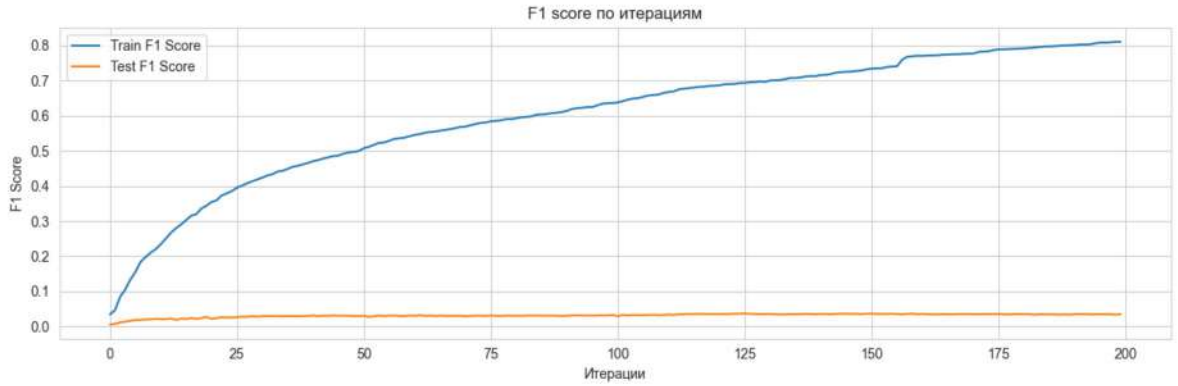
Сначала попробую логистическую регрессию.

In [ ]:
```
train_accuracy = []
test_accuracy = []
train_f1 = []
test_f1 = []
start_time = time.time()

model = LogisticRegression(max_iter=1, warm_start=True)

for i in tqdm(range(1, 201)):
    model.fit(X_train, y_train)

    train_pred = model.predict(X_train)
    test_pred = model.predict(X_test)

    train_acc = accuracy_score(y_train, train_pred)
    test_acc = accuracy_score(y_test, test_pred)
    train_accuracy.append(train_acc)
    test_accuracy.append(test_acc)

    train_f1_score = f1_score(y_train, train_pred, average='macro')
    test_f1_score = f1_score(y_test, test_pred, average='macro')
    train_f1.append(train_f1_score)
    test_f1.append(test_f1_score)
end_time = time.time()
sns.set_style('whitegrid')
plt.figure(figsize=(12, 8))

plt.subplot(2, 1, 1)
plt.plot(train_accuracy, label='Train Accuracy')
plt.plot(test_accuracy, label='Test Accuracy')
plt.title('Accuracy по итерациям')
plt.xlabel('Итерации')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)

plt.subplot(2, 1, 2)
plt.plot(train_f1, label='Train F1 Score')
plt.plot(test_f1, label='Test F1 Score')
plt.title('F1 score по итерациям')
plt.xlabel('Итерации')
plt.ylabel('F1 Score')
plt.legend()
```

```
plt.grid(True)

plt.tight_layout()
plt.show()
print(f'Обучалось {end_time - start_time} секунд')
```

```
100%|██████████| 200/200 [01:39<00:00,  2.01it/s]
```




```
Обучалось 99.62942123413086 секунд
```

In [ ]:
```
y_pred= model.predict(X_test)
accuracy_score(y_pred, y_test)
```

Out[ ]:
```
0.04203079483978361
```

Логистическая регрессия сталкивается с проблемой переобучения. Попробую случайный лес, так как ему оно безразличнее.

In [ ]:
```
start_time = time.time()
model_rf = RandomForestClassifier(n_jobs=-1, n_estimators=25)

model_rf.fit(X_train, y_train)
end_time = time.time()
print(f'Обучалось {end_time - start_time} секунд')
y_pred = model_rf.predict(X_test)

accuracy_score(y_pred, y_test)
```

```
Обучалось 40.71551847457886 секунд
```

Out[ ]:
```
0.03162713275072826
```

In [ ]:
```
y_pred = model_rf.predict(X_train)

accuracy_score(y_pred, y_train)
```

Out[ ]:
```
0.9998959200666112
```

Здесь тоже самое. Попробую еще SVM, KNN, наивный байес и CatBoost.

In [ ]:
```
parameters = {
    'C': [0.1, 1, 10],  # Регуляризация
    'kernel': ['linear', 'rbf'],  # Типы ядер
}
start_time = time.time()
svm_model = SVC()
grid_search = GridSearchCV(svm_model, parameters, cv=3, scoring='accuracy')

grid_search.fit(X_train, y_train)

print("Лучшие параметры:", grid_search.best_params_)
end_time = time.time()
print(f'Обучалось {end_time - start_time} секунд')
best_svm = grid_search.best_estimator_

y_pred = best_svm.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy с лучшими параметрами: {accuracy}")
```

```
Лучшие параметры: {'C': 10, 'kernel': 'rbf'}
Обучалось 540.815099477768 секунд
Accuracy с лучшими параметрами: 0.06658343736995423
```

In [ ]:
```
start_time = time.time()
knn_model = KNeighborsClassifier()
knn_model.fit(X_train, y_train)
end_time = time.time()
print(f'Обучалось {end_time - start_time} секунд')
y_pred = knn_model.predict(X_test)
accuracy_score(y_test, y_pred)
```

```
Обучалось 0.012995243072509766 секунд
```

Out[ ]:
```
0.026217228464419477
```

In [ ]:
```
nb_model = GaussianNB()
start_time = time.time()
nb_model.fit(X_train, y_train)
end_time = time.time()
print(f'Обучалось {end_time - start_time} секунд')
y_pred = nb_model.predict(X_test)
accuracy_score(y_test, y_pred)
```

```
Обучалось 0.10299921035766602 секунд
```

Out[ ]:
```
0.02039117769454848
```

In [ ]:
```
train_pca_pool = Pool(data=X_train, label=y_train)
test_pool = Pool(data=X_test, label=y_test)
```

In [ ]:
```
model_cb = CatBoostClassifier(loss_function='MultiClass',
                              random_seed=42,
                              task_type='GPU',
                              iterations=200,
                              depth=5,
                              learning_rate=0.2)
start_time = time.time()
model_cb.fit(train_pca_pool)
end_time = time.time()
```

```
print(f'Обучалось {end_time - start_time} секунд')
```

```
0:      learn: 6.1479797        total: 8.62s    remaining: 28m 35s
1:      learn: 6.1007559        total: 18.9s    remaining: 31m 14s
2:      learn: 6.0664953        total: 20.5s    remaining: 22m 25s
3:      learn: 6.0179770        total: 22.2s    remaining: 18m 8s
4:      learn: 5.9685910        total: 23.8s    remaining: 15m 29s
5:      learn: 5.9334868        total: 24.7s    remaining: 13m 19s
6:      learn: 5.8789851        total: 25.6s    remaining: 11m 47s
7:      learn: 5.8371848        total: 26.6s    remaining: 10m 37s
8:      learn: 5.7818672        total: 27.5s    remaining: 9m 43s
9:      learn: 5.7455014        total: 28.4s    remaining: 8m 59s
10:     learn: 5.7130524        total: 29.3s    remaining: 8m 22s
11:     learn: 5.6849752        total: 30.1s    remaining: 7m 51s
12:     learn: 5.6435608        total: 31s      remaining: 7m 25s
13:     learn: 5.6149644        total: 31.8s    remaining: 7m 2s
14:     learn: 5.5656273        total: 32.6s    remaining: 6m 41s
15:     learn: 5.5344273        total: 33.3s    remaining: 6m 22s
16:     learn: 5.5031647        total: 34.1s    remaining: 6m 7s
17:     learn: 5.4786095        total: 34.9s    remaining: 5m 52s
18:     learn: 5.4406444        total: 35.7s    remaining: 5m 39s
19:     learn: 5.4195714        total: 36.4s    remaining: 5m 27s
20:     learn: 5.3982659        total: 37.2s    remaining: 5m 16s
21:     learn: 5.3752822        total: 37.9s    remaining: 5m 6s
22:     learn: 5.3475579        total: 38.9s    remaining: 4m 59s
23:     learn: 5.3171039        total: 39.8s    remaining: 4m 51s
24:     learn: 5.3047977        total: 40.6s    remaining: 4m 44s
25:     learn: 5.3042671        total: 40.8s    remaining: 4m 33s
26:     learn: 5.2894707        total: 41.7s    remaining: 4m 26s
27:     learn: 5.2724870        total: 42.5s    remaining: 4m 21s
28:     learn: 5.2357097        total: 43.4s    remaining: 4m 16s
29:     learn: 5.2272061        total: 44.2s    remaining: 4m 10s
30:     learn: 5.2268894        total: 44.4s    remaining: 4m 2s
31:     learn: 5.1967131        total: 45.3s    remaining: 3m 57s
32:     learn: 5.1804075        total: 46.1s    remaining: 3m 53s
33:     learn: 5.1653342        total: 47s      remaining: 3m 49s
34:     learn: 5.1468739        total: 47.9s    remaining: 3m 45s
35:     learn: 5.1181161        total: 48.7s    remaining: 3m 42s
36:     learn: 5.1045983        total: 49.6s    remaining: 3m 38s
37:     learn: 5.0785263        total: 50.5s    remaining: 3m 35s
38:     learn: 5.0590032        total: 51.4s    remaining: 3m 32s
39:     learn: 5.0380388        total: 52.3s    remaining: 3m 29s
40:     learn: 5.0217653        total: 53s      remaining: 3m 25s
41:     learn: 5.0004663        total: 53.8s    remaining: 3m 22s
42:     learn: 4.9680686        total: 54.8s    remaining: 3m 20s
43:     learn: 4.9298213        total: 55.8s    remaining: 3m 17s
44:     learn: 4.9206102        total: 56.6s    remaining: 3m 14s
45:     learn: 4.9077079        total: 57.4s    remaining: 3m 12s
46:     learn: 4.8904839        total: 58.2s    remaining: 3m 9s
47:     learn: 4.8706063        total: 59.1s    remaining: 3m 7s
48:     learn: 4.8444363        total: 59.9s    remaining: 3m 4s
49:     learn: 4.8407671        total: 1m       remaining: 3m 2s
50:     learn: 4.8002275        total: 1m 1s    remaining: 2m 59s
51:     learn: 4.7785602        total: 1m 2s    remaining: 2m 57s
52:     learn: 4.7650680        total: 1m 3s    remaining: 2m 55s
53:     learn: 4.7422078        total: 1m 4s    remaining: 2m 53s
54:     learn: 4.7317332        total: 1m 5s    remaining: 2m 51s
55:     learn: 4.7194513        total: 1m 6s    remaining: 2m 49s
56:     learn: 4.7003474        total: 1m 6s    remaining: 2m 47s
57:     learn: 4.6802656        total: 1m 7s    remaining: 2m 45s
58:     learn: 4.6607474        total: 1m 8s    remaining: 2m 43s
59:     learn: 4.6484355        total: 1m 9s    remaining: 2m 41s
```

```
60:     learn: 4.6240946        total: 1m 10s   remaining: 2m 39s
61:     learn: 4.6110452        total: 1m 10s   remaining: 2m 37s
62:     learn: 4.6005721        total: 1m 11s   remaining: 2m 35s
63:     learn: 4.5829452        total: 1m 12s   remaining: 2m 34s
64:     learn: 4.5672995        total: 1m 13s   remaining: 2m 32s
65:     learn: 4.5325355        total: 1m 14s   remaining: 2m 30s
66:     learn: 4.5064814        total: 1m 14s   remaining: 2m 28s
67:     learn: 4.4937715        total: 1m 15s   remaining: 2m 27s
68:     learn: 4.4667111        total: 1m 16s   remaining: 2m 25s
69:     learn: 4.4503405        total: 1m 17s   remaining: 2m 23s
70:     learn: 4.4363226        total: 1m 18s   remaining: 2m 22s
71:     learn: 4.4069090        total: 1m 19s   remaining: 2m 20s
72:     learn: 4.3871843        total: 1m 20s   remaining: 2m 19s
73:     learn: 4.3584017        total: 1m 21s   remaining: 2m 18s
74:     learn: 4.3250725        total: 1m 22s   remaining: 2m 16s
75:     learn: 4.3066544        total: 1m 22s   remaining: 2m 15s
76:     learn: 4.2813041        total: 1m 23s   remaining: 2m 13s
77:     learn: 4.2747235        total: 1m 24s   remaining: 2m 12s
78:     learn: 4.2532647        total: 1m 25s   remaining: 2m 11s
79:     learn: 4.2268760        total: 1m 26s   remaining: 2m 9s
80:     learn: 4.2038503        total: 1m 27s   remaining: 2m 8s
81:     learn: 4.1911379        total: 1m 28s   remaining: 2m 6s
82:     learn: 4.1717099        total: 1m 29s   remaining: 2m 5s
83:     learn: 4.1414361        total: 1m 30s   remaining: 2m 4s
84:     learn: 4.1057798        total: 1m 31s   remaining: 2m 3s
85:     learn: 4.0861607        total: 1m 31s   remaining: 2m 1s
86:     learn: 4.0526258        total: 1m 32s   remaining: 2m
87:     learn: 4.0327213        total: 1m 33s   remaining: 1m 59s
88:     learn: 4.0064404        total: 1m 34s   remaining: 1m 58s
89:     learn: 3.9705092        total: 1m 35s   remaining: 1m 56s
90:     learn: 3.9388360        total: 1m 36s   remaining: 1m 55s
91:     learn: 3.9099054        total: 1m 37s   remaining: 1m 54s
92:     learn: 3.8729538        total: 1m 38s   remaining: 1m 53s
93:     learn: 3.8294451        total: 1m 39s   remaining: 1m 51s
94:     learn: 3.7968652        total: 1m 40s   remaining: 1m 50s
95:     learn: 3.7769559        total: 1m 40s   remaining: 1m 49s
96:     learn: 3.7597473        total: 1m 41s   remaining: 1m 47s
97:     learn: 3.7280912        total: 1m 42s   remaining: 1m 46s
98:     learn: 3.6913571        total: 1m 43s   remaining: 1m 45s
99:     learn: 3.6816780        total: 1m 44s   remaining: 1m 44s
100:    learn: 3.6561358        total: 1m 45s   remaining: 1m 42s
101:    learn: 3.6242044        total: 1m 45s   remaining: 1m 41s
102:    learn: 3.5847674        total: 1m 46s   remaining: 1m 40s
103:    learn: 3.5382063        total: 1m 47s   remaining: 1m 39s
104:    learn: 3.5184970        total: 1m 48s   remaining: 1m 38s
105:    learn: 3.4725343        total: 1m 49s   remaining: 1m 37s
106:    learn: 3.4336706        total: 1m 50s   remaining: 1m 35s
107:    learn: 3.3984617        total: 1m 51s   remaining: 1m 34s
108:    learn: 3.3703455        total: 1m 52s   remaining: 1m 33s
109:    learn: 3.3626033        total: 1m 52s   remaining: 1m 32s
110:    learn: 3.3427812        total: 1m 53s   remaining: 1m 31s
111:    learn: 3.3299183        total: 1m 54s   remaining: 1m 29s
112:    learn: 3.2892546        total: 1m 55s   remaining: 1m 28s
113:    learn: 3.2563019        total: 1m 56s   remaining: 1m 27s
114:    learn: 3.2229376        total: 1m 56s   remaining: 1m 26s
115:    learn: 3.1966635        total: 1m 57s   remaining: 1m 25s
116:    learn: 3.1667204        total: 1m 58s   remaining: 1m 24s
117:    learn: 3.1374565        total: 1m 59s   remaining: 1m 23s
118:    learn: 3.0977683        total: 2m       remaining: 1m 21s
119:    learn: 3.0705916        total: 2m 1s    remaining: 1m 20s
```

```
120:    learn: 3.0231059        total: 2m 2s    remaining: 1m 19s
121:    learn: 2.9834901        total: 2m 3s    remaining: 1m 18s
122:    learn: 2.9498851        total: 2m 3s    remaining: 1m 17s
123:    learn: 2.9076236        total: 2m 4s    remaining: 1m 16s
124:    learn: 2.8619300        total: 2m 5s    remaining: 1m 15s
125:    learn: 2.8196287        total: 2m 6s    remaining: 1m 14s
126:    learn: 2.7753975        total: 2m 7s    remaining: 1m 13s
127:    learn: 2.7470536        total: 2m 8s    remaining: 1m 12s
128:    learn: 2.7025676        total: 2m 9s    remaining: 1m 11s
129:    learn: 2.6842526        total: 2m 9s    remaining: 1m 9s
130:    learn: 2.6690083        total: 2m 10s   remaining: 1m 8s
131:    learn: 2.6335077        total: 2m 11s   remaining: 1m 7s
132:    learn: 2.5992815        total: 2m 12s   remaining: 1m 6s
133:    learn: 2.5592812        total: 2m 13s   remaining: 1m 5s
134:    learn: 2.5232395        total: 2m 14s   remaining: 1m 4s
135:    learn: 2.4952166        total: 2m 15s   remaining: 1m 3s
136:    learn: 2.4530575        total: 2m 15s   remaining: 1m 2s
137:    learn: 2.4353401        total: 2m 16s   remaining: 1m 1s
138:    learn: 2.3963921        total: 2m 17s   remaining: 1m
139:    learn: 2.3585056        total: 2m 18s   remaining: 59.3s
140:    learn: 2.3178406        total: 2m 19s   remaining: 58.3s
141:    learn: 2.2796492        total: 2m 20s   remaining: 57.2s
142:    learn: 2.2520842        total: 2m 20s   remaining: 56.2s
143:    learn: 2.2143488        total: 2m 21s   remaining: 55.2s
144:    learn: 2.1768806        total: 2m 22s   remaining: 54.1s
145:    learn: 2.1415709        total: 2m 23s   remaining: 53.1s
146:    learn: 2.1036358        total: 2m 24s   remaining: 52.1s
147:    learn: 2.0660023        total: 2m 25s   remaining: 51.1s
148:    learn: 2.0300848        total: 2m 26s   remaining: 50.1s
149:    learn: 1.9980654        total: 2m 27s   remaining: 49.1s
150:    learn: 1.9619277        total: 2m 28s   remaining: 48.1s
151:    learn: 1.9312802        total: 2m 29s   remaining: 47.1s
152:    learn: 1.8956625        total: 2m 30s   remaining: 46.1s
153:    learn: 1.8633739        total: 2m 31s   remaining: 45.1s
154:    learn: 1.8293077        total: 2m 31s   remaining: 44.1s
155:    learn: 1.7950786        total: 2m 32s   remaining: 43.1s
156:    learn: 1.7634277        total: 2m 33s   remaining: 42.1s
157:    learn: 1.7295385        total: 2m 34s   remaining: 41.1s
158:    learn: 1.6963558        total: 2m 35s   remaining: 40.1s
159:    learn: 1.6640214        total: 2m 36s   remaining: 39.1s
160:    learn: 1.6372389        total: 2m 37s   remaining: 38.2s
161:    learn: 1.6076865        total: 2m 38s   remaining: 37.2s
162:    learn: 1.5778087        total: 2m 39s   remaining: 36.2s
163:    learn: 1.5470953        total: 2m 40s   remaining: 35.2s
164:    learn: 1.5176384        total: 2m 41s   remaining: 34.2s
165:    learn: 1.4881755        total: 2m 42s   remaining: 33.2s
166:    learn: 1.4587065        total: 2m 43s   remaining: 32.2s
167:    learn: 1.4319919        total: 2m 44s   remaining: 31.3s
168:    learn: 1.4064426        total: 2m 45s   remaining: 30.3s
169:    learn: 1.3792220        total: 2m 45s   remaining: 29.3s
170:    learn: 1.3518183        total: 2m 46s   remaining: 28.3s
171:    learn: 1.3254672        total: 2m 47s   remaining: 27.3s
172:    learn: 1.2993597        total: 2m 48s   remaining: 26.3s
173:    learn: 1.2738010        total: 2m 49s   remaining: 25.4s
174:    learn: 1.2486201        total: 2m 50s   remaining: 24.4s
175:    learn: 1.2237788        total: 2m 51s   remaining: 23.4s
176:    learn: 1.1993338        total: 2m 52s   remaining: 22.4s
177:    learn: 1.1757777        total: 2m 53s   remaining: 21.4s
178:    learn: 1.1529748        total: 2m 54s   remaining: 20.5s
179:    learn: 1.1294450        total: 2m 55s   remaining: 19.5s
```

```
180:    learn: 1.1074136        total: 2m 56s   remaining: 18.5s
181:    learn: 1.0864863        total: 2m 57s   remaining: 17.5s
182:    learn: 1.0649336        total: 2m 57s   remaining: 16.5s
183:    learn: 1.0439323        total: 2m 58s   remaining: 15.5s
184:    learn: 1.0240907        total: 2m 59s   remaining: 14.6s
185:    learn: 1.0044046        total: 3m       remaining: 13.6s
186:    learn: 0.9855994        total: 3m 1s    remaining: 12.6s
187:    learn: 0.9674615        total: 3m 2s    remaining: 11.6s
188:    learn: 0.9484226        total: 3m 3s    remaining: 10.7s
189:    learn: 0.9300864        total: 3m 3s    remaining: 9.68s
190:    learn: 0.9119650        total: 3m 4s    remaining: 8.71s
191:    learn: 0.8940322        total: 3m 5s    remaining: 7.73s
192:    learn: 0.8761347        total: 3m 6s    remaining: 6.76s
193:    learn: 0.8623662        total: 3m 7s    remaining: 5.79s
194:    learn: 0.8456495        total: 3m 8s    remaining: 4.82s
195:    learn: 0.8288680        total: 3m 9s    remaining: 3.86s
196:    learn: 0.8124372        total: 3m 9s    remaining: 2.89s
197:    learn: 0.7972461        total: 3m 10s   remaining: 1.93s
198:    learn: 0.7817606        total: 3m 11s   remaining: 963ms
199:    learn: 0.7669052        total: 3m 12s   remaining: 0us
Обучалось 193.3386640548706 секунд
```

In [ ]:
```
y_pred = model_cb.predict(X_test)
accuracy_score(y_test, y_pred)
```

Out[ ]: 0.05076987099459009

В целом, методы классического машинного обучения намного хуже справляются с картинками, чем методы глубокого обучения, так что сейчас попробуем дообучить нейронную сеть

In [2]:
```
def get_paths(dataset_type='train'):
    labels_dict = {
        'train': 0,
        'val': 1,
        'test': 2,
    }

    f = open('celebA_500/celebA_train_split.txt')
    lines = f.readlines()
    f.close()

    lines = [i.strip().split() for i in lines]
    lines = [i[0] for i in lines if int(i[1]) == labels_dict[dataset_type]]

    images_paths = []
    for filename in lines:
        images_paths.append(os.path.join('celebA_500/celebA_imgs/', filename))

    return np.array(images_paths)

class celebA(Dataset):
    def __init__(self, dataset_type, transform):
        self.images = get_paths(dataset_type=dataset_type)

        f = open('celebA_500/celebA_anno.txt', 'r')
        labels = f.readlines()
        f.close()
        labels = [x.strip().split() for x in labels]
```

```
        labels = {x:int(y) for x, y in labels}
        self.labels = [labels[x.split('/')[-1]] for x in self.images]
        self.transform = transform

    def __len__(self):
        return len(self.images)

    def __getitem__(self, idx):
        img_name = self.images[idx]
        label = self.labels[idx]

        image = Image.open(img_name)

        sample = {
            'image': self.transform(image),
            'label': label,
        }

        return (sample['image'], sample['label'])

    def get_photos(self, person_number):
        photos = []
        for i, j in enumerate(self.labels):
            if j == person_number:
                photos.append(self.images[i])
        if len(photos) != 0:
            photos = torch.stack([self.transform(Image.open(x)) for x in photos])
        else:
            photos = torch.Tensor()
        return photos
```

In [3]:
```
transform = transforms.Compose([
      transforms.ToTensor(),
      transforms.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225)),
])

transform2 = transforms.Compose([
      transforms.ToTensor(),
      transforms.RandomHorizontalFlip(),
      transforms.ColorJitter(),
      transforms.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225)),
])


train_data = celebA('train', transform2)
val_data = celebA('val', transform)
test_data = celebA('test', transform)

batch_size = 32
train_loader = torch.utils.data.DataLoader(train_data, batch_size=batch_size, sh
val_loader = torch.utils.data.DataLoader(val_data, batch_size=batch_size, shuffl
test_loader = torch.utils.data.DataLoader(test_data, batch_size=batch_size, shuf
```

In [4]:
```
print(len(train_data))
print(len(val_data))
print(len(test_data))
print(len(train_loader))
print(len(val_loader))
print(len(test_loader))
```

```
8544
1878
1589
267
59
50
```

In [5]:
```
for i in timm.list_models():
    if i.find('inception')!= -1:
        print(i)
```

```
inception_next_base
inception_next_small
inception_next_tiny
inception_resnet_v2
inception_v3
inception_v4
```

In [6]:
```
model = timm.create_model('inception_resnet_v2', pretrained=True, num_classes=50
model.last_linear = nn.Linear(in_features=1792, out_features=512, bias=False)
model.last_bn = nn.BatchNorm1d(512, eps=0.001, momentum=0.1, affine=True, track_
model.logits = nn.Linear(in_features=512, out_features=500, bias=True)
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model.to(device);
```

In [7]:
```
def train(model, train_loader, val_loader, optimizer, epochs=10, device=torch.de
    train_losses, val_losses, val_full_acc, train_full_acc = [], [], [], []
    best_acc = 0.0
    loss_fn = nn.CrossEntropyLoss()
    best_model_weights = deepcopy(model.state_dict())
    start_time = time.time()
    for epoch in range(epochs):
        print("Epoch ", epoch+1)
        model.train()
        current_train_loss = 0
        current_train_correct = 0

        for inputs, labels in tqdm(train_loader, leave=True, desc='Training'):
            X_batch = inputs.to(device)
            Y_batch = labels.to(device)

            optimizer.zero_grad()

            Y_pred = model(X_batch)
            preds = torch.argmax(Y_pred, 1)
            loss = loss_fn(Y_pred, Y_batch)
            loss.backward()
            optimizer.step()

            current_train_loss += loss.item() * X_batch.size(0)
            current_train_correct += torch.sum(preds == Y_batch)

        train_loss = current_train_loss / len(train_loader.dataset)
        train_losses.append(train_loss)
        train_acc = current_train_correct / len(train_loader.dataset)
        train_full_acc.append(train_acc)
        print('train acc = {:.2f}%'.format(train_acc.item()*100))

        model.eval()
        current_val_loss = 0
```

```
        current_val_correct = 0
        with torch.no_grad():
            for inputs, labels in val_loader:
                X_val = inputs.to(device)
                Y_val = labels.to(device)

                outputs = model(X_val)
                val_loss = loss_fn(outputs, Y_val)
                preds = torch.argmax(outputs, 1)
                current_val_correct += torch.sum(preds == Y_val)
                current_val_loss += val_loss.item() * X_val.size(0)

        val_acc = current_val_correct / len(val_loader.dataset)
        val_loss = current_val_loss / len(val_loader.dataset)

        print('val acc = {:.2f}%'.format(val_acc.item() * 100))
        val_losses.append(val_loss)
        val_full_acc.append(val_acc)

        if val_acc > best_acc:
            best_acc = val_acc
            best_model_weights = deepcopy(model.state_dict())
    end_time = time.time()
    print(f'Время обучения: {end_time - start_time} секунд')
    return best_model_weights, train_losses, val_losses, val_full_acc, train_ful
```

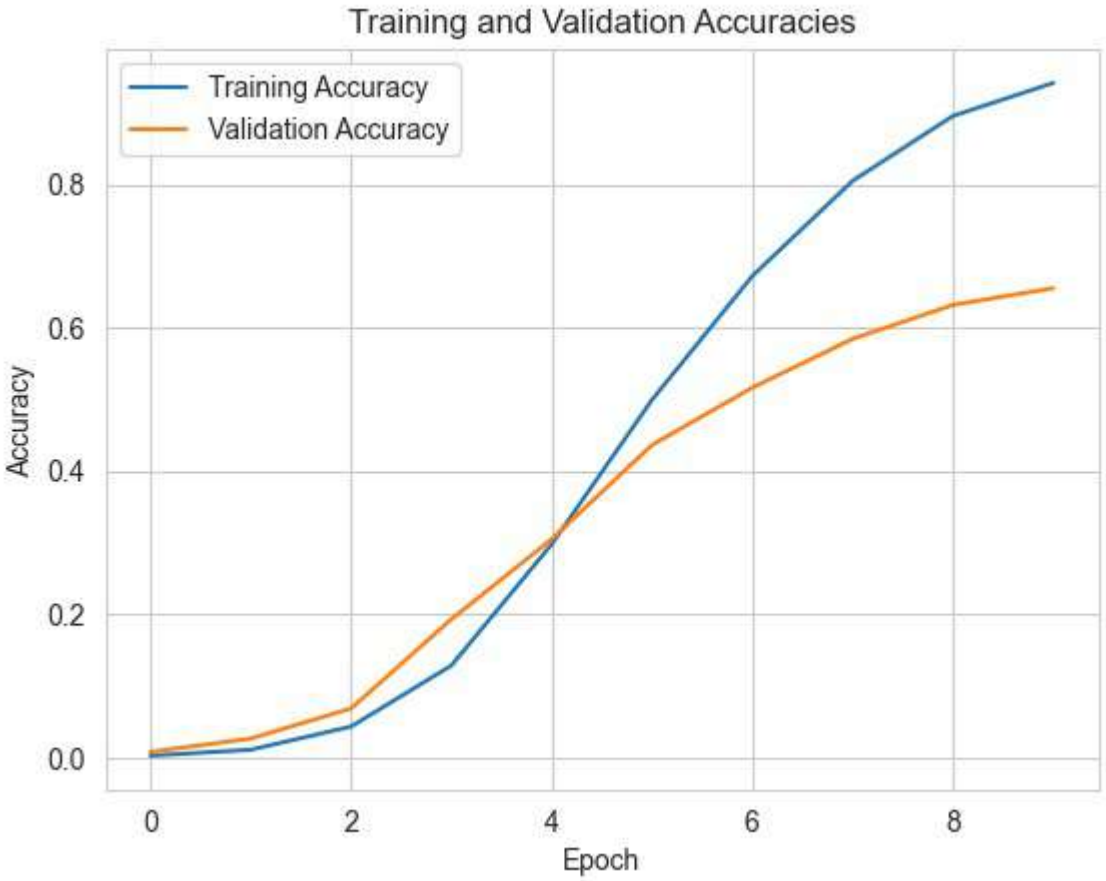
In [8]:
```
opt = optim.AdamW(filter(lambda p: p.requires_grad, model.parameters()))
best_model_weights, train_losses, val_losses, val_full_acc, train_full_acc = tra
```

```
Epoch  1
Training:   0%|          | 0/267 [00:00<?, ?it/s]c:\Users\yarom\AppData\Local\Pro
grams\Python\Python312\Lib\site-packages\torch\autograd\graph.py:744: UserWarning
: Plan failed with a cudnnException: CUDNN_BACKEND_EXECUTION_PLAN_DESCRIPTOR: cud
nnFinalize Descriptor Failed cudnn_status: CUDNN_STATUS_NOT_SUPPORTED (Triggered
internally at ..\aten\src\ATen\native\cudnn\Conv_v8.cpp:919.)
  return Variable._execution_engine.run_backward(  # Calls into the C++ engine to
run the backward pass
Training: 100%|██████████| 267/267 [01:11<00:00,  3.74it/s]
train acc = 0.22%
val acc = 0.75%
Epoch  2
Training: 100%|██████████| 267/267 [01:11<00:00,  3.76it/s]
train acc = 1.05%
val acc = 2.61%
Epoch  3
Training: 100%|██████████| 267/267 [01:08<00:00,  3.92it/s]
train acc = 4.28%
val acc = 6.82%
Epoch  4
Training: 100%|██████████| 267/267 [01:04<00:00,  4.14it/s]
train acc = 12.82%
val acc = 19.28%
Epoch  5
Training: 100%|██████████| 267/267 [01:05<00:00,  4.09it/s]
train acc = 29.74%
val acc = 30.46%
Epoch  6
Training: 100%|██████████| 267/267 [01:07<00:00,  3.95it/s]
```

```
train acc = 49.91%
val acc = 43.61%
Epoch  7
Training: 100%|██████████| 267/267 [01:10<00:00,  3.79it/s]
train acc = 67.18%
val acc = 51.60%
Epoch  8
Training: 100%|██████████| 267/267 [01:09<00:00,  3.85it/s]
train acc = 80.48%
val acc = 58.41%
Epoch  9
Training: 100%|██████████| 267/267 [01:08<00:00,  3.88it/s]
train acc = 89.54%
val acc = 63.15%
Epoch  10
Training: 100%|██████████| 267/267 [01:09<00:00,  3.83it/s]
train acc = 94.15%
val acc = 65.50%
Время обучения: 744.4108598232269 секунд
```

In [9]:
```python
def draw(train_losses, val_losses, val_full_acc, train_full_acc):
    train_full_acc_cpu = []
    for i in train_full_acc:
        train_full_acc_cpu.append(i.cpu().numpy())

    val_full_acc_cpu = []
    for i in val_full_acc:
        val_full_acc_cpu.append(i.cpu().numpy())

    sns.set_style('whitegrid')
    plt.plot(train_losses, label='Training Loss')
    plt.plot(val_losses, label='Validation Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.title('Training and Validation Losses')
    plt.legend()
    plt.grid(True)
    plt.show()

    plt.plot(train_full_acc_cpu, label='Training Accuracy')
    plt.plot(val_full_acc_cpu, label='Validation Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.title('Training and Validation Accuracies')
    plt.legend()
    plt.grid(True)
    plt.show()
```

In [10]:
```python
draw(train_losses, val_losses, val_full_acc, train_full_acc)
```

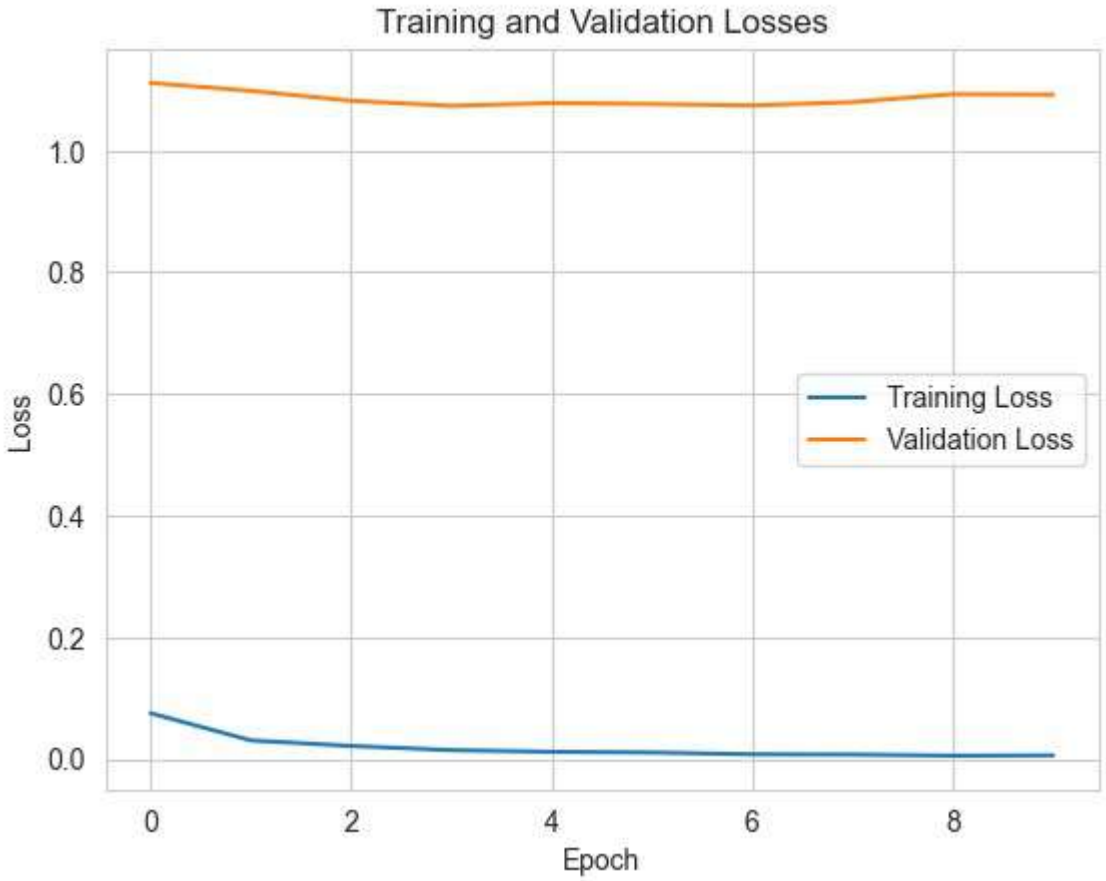



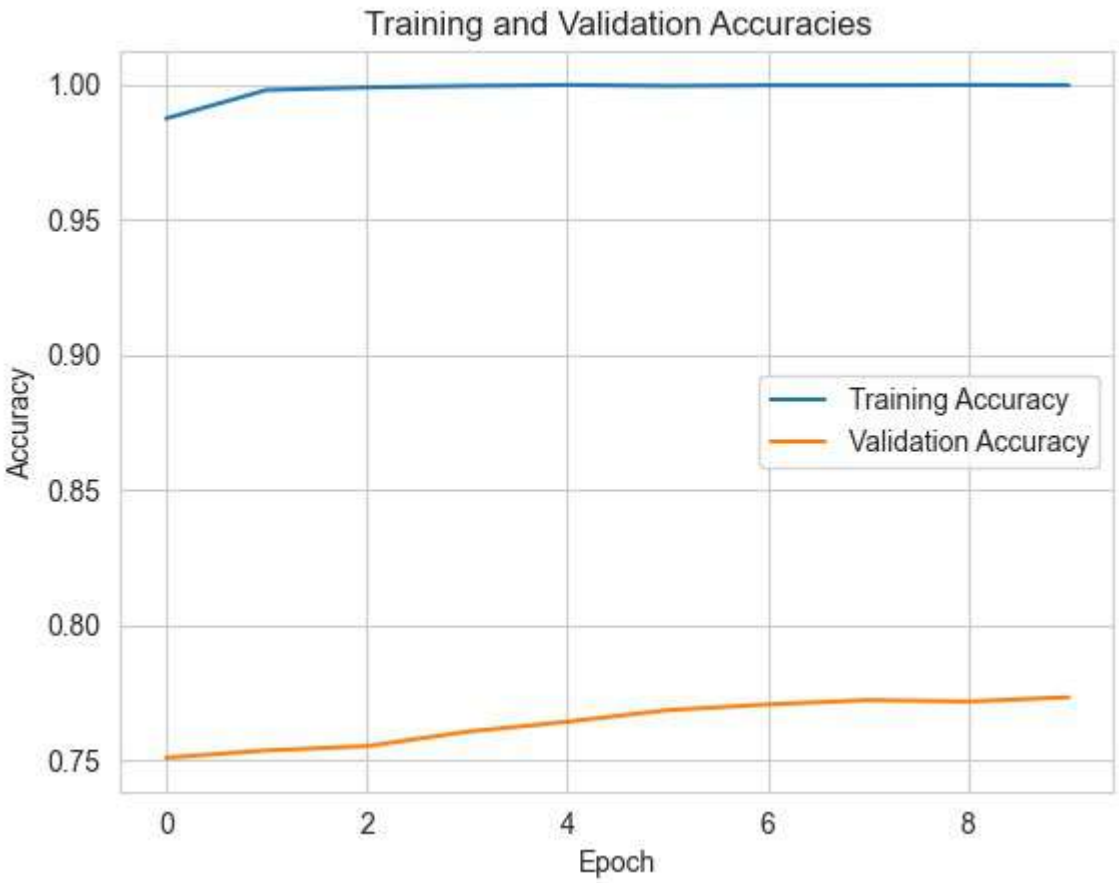

5 эпох действительно мало для обучения нейронной сети, хотя результат уже виден. Попробую дообучить еще 10 эпох, но при этом поменяв learning_rate на 0.0001, так как стандартно в оптимизаторе AdamW стоит 0.001, что является нормальным для первых эпох, но многовато для последующих.

In [11]:
```
opt = optim.AdamW(filter(lambda p: p.requires_grad, model.parameters()), lr=0.00
best_model_weights, train_losses, val_losses, val_full_acc, train_full_acc = tra
```

```
Epoch  1
Training: 100%|██████████| 267/267 [01:09<00:00,  3.82it/s]
train acc = 98.74%
val acc = 75.08%
Epoch  2
Training: 100%|██████████| 267/267 [01:10<00:00,  3.81it/s]
train acc = 99.79%
val acc = 75.35%
Epoch  3
Training: 100%|██████████| 267/267 [01:08<00:00,  3.90it/s]
train acc = 99.88%
val acc = 75.51%
Epoch  4
Training: 100%|██████████| 267/267 [01:09<00:00,  3.83it/s]
train acc = 99.94%
val acc = 76.04%
Epoch  5
Training: 100%|██████████| 267/267 [01:13<00:00,  3.65it/s]
train acc = 99.98%
val acc = 76.41%
Epoch  6
Training: 100%|██████████| 267/267 [01:10<00:00,  3.78it/s]
train acc = 99.94%
val acc = 76.84%
Epoch  7
Training: 100%|██████████| 267/267 [01:08<00:00,  3.88it/s]
train acc = 99.96%
val acc = 77.05%
Epoch  8
Training: 100%|██████████| 267/267 [01:11<00:00,  3.72it/s]
train acc = 99.96%
val acc = 77.21%
Epoch  9
Training: 100%|██████████| 267/267 [01:06<00:00,  4.00it/s]
train acc = 99.98%
val acc = 77.16%
Epoch  10
Training: 100%|██████████| 267/267 [01:07<00:00,  3.95it/s]
train acc = 99.96%
val acc = 77.32%
Время обучения: 756.4568605422974 секунд
```

In [12]:
```
draw(train_losses, val_losses, val_full_acc, train_full_acc)
```




Получилось всего 77% на валидационной выборке, что относительно мало. Попробую другую модель, которая уже предобучена на датасете с фотографиями лиц и дообучу ее на своем датасете.

Возьмем модель Inception-Resnet-V1, так как она сочетает в себе идеи из архитектур Inception и Resnet и является одной из лучших моделей для распознавания лиц.

Основные особенности Inception-ResNet-V1:

1. Inception-модули позволяют сети эффективно масштабироваться и обрабатывать информацию на различных уровнях абстракции и масштабах. Это достигается за счет использования параллельных сверточных операций с различными размерами фильтров (1x1, 3x3, 5x5) и последующим объединением результатов.

2. Вместо того чтобы просто стекать слои, как в традиционных сетях, Inception-ResNet вводит остаточные соединения, которые позволяют градиентам течь непосредственно через архитектуру сети без ослабления на глубоких уровнях. Эти соединения помогают бороться с проблемой исчезающих градиентов при обучении очень глубоких сетей.

3. Так как Inception-ResNet-V1 использует идеи из Inception и Resnet, то гибридизация этих двух архитектур предлагает баланс между глубиной и шириной сети, что позволяет эффективно обрабатывать сложные признаки и обучаться быстрее по сравнению с отдельными архитектурами.

In [13]:
```
from models import inception_resnet_v1
model = inception_resnet_v1.InceptionResnetV1(pretrained='vggface2', classify=Tr
```

In [14]:
```
for param in model.parameters():
    param.requires_grad = False

model.last_linear = nn.Linear(in_features=1792, out_features=512, bias=False)
model.last_bn = nn.BatchNorm1d(512, eps=0.001, momentum=0.1, affine=True, track_
model.logits = nn.Linear(in_features=512, out_features=500, bias=True)
```

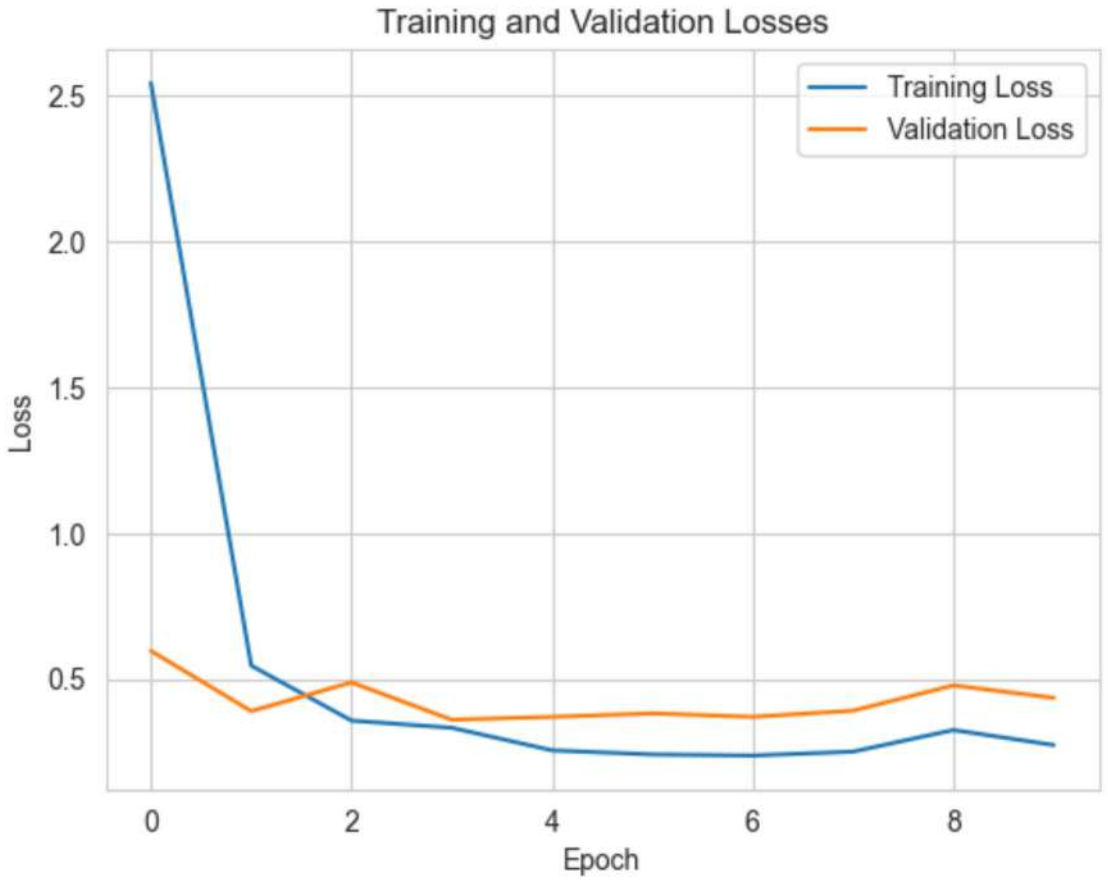
In [15]:
```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model.to(device);
```

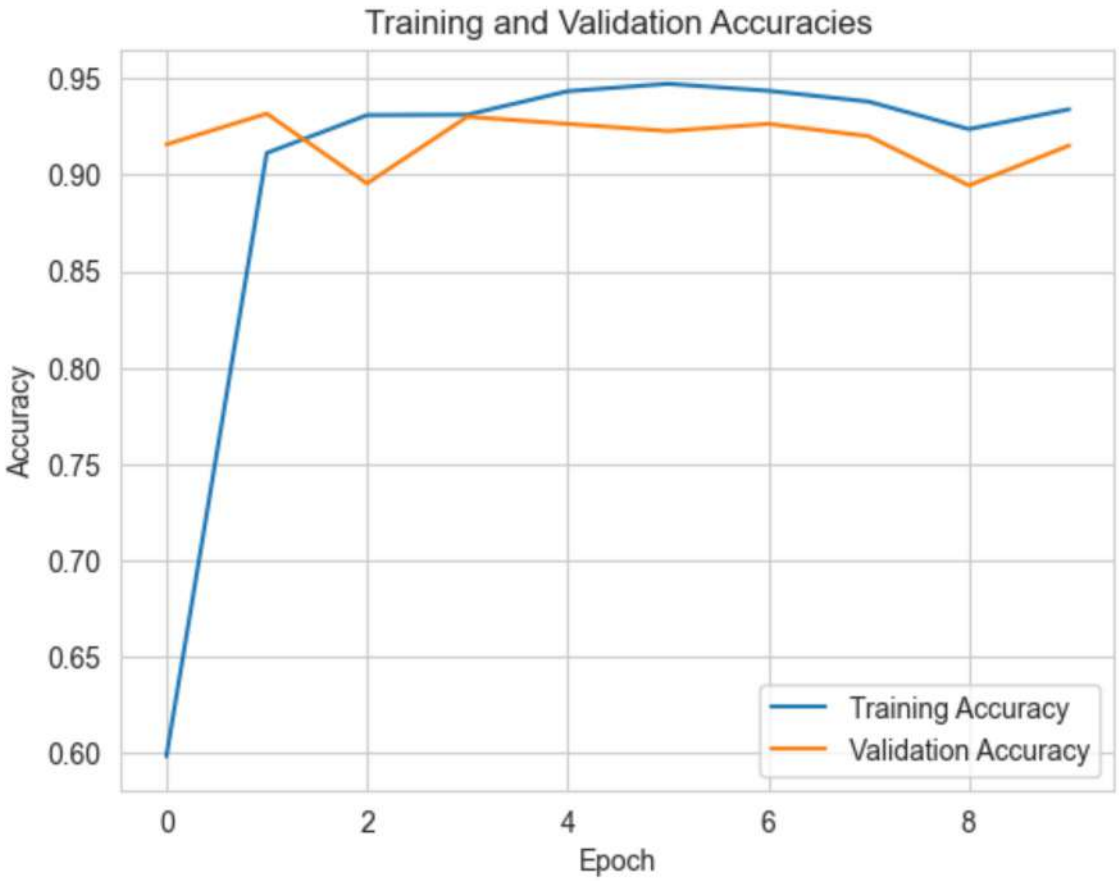
In [16]:
```
opt = optim.AdamW(filter(lambda p: p.requires_grad, model.parameters()))
best_model_weights, train_losses, val_losses, val_full_acc, train_full_acc = tra
```

```
Epoch  1
Training:   0%|          | 0/267 [00:00<?, ?it/s]c:\Users\yarom\AppData\Local\Pro
grams\Python\Python312\Lib\site-packages\torch\nn\modules\conv.py:456: UserWarnin
g: Plan failed with a cudnnException: CUDNN_BACKEND_EXECUTION_PLAN_DESCRIPTOR: cu
dnnFinalize Descriptor Failed cudnn_status: CUDNN_STATUS_NOT_SUPPORTED (Triggered
internally at ..\aten\src\ATen\native\cudnn\Conv_v8.cpp:919.)
  return F.conv2d(input, weight, bias, self.stride,
Training: 100%|██████████| 267/267 [00:20<00:00, 13.34it/s]
train acc = 59.83%
val acc = 91.59%
Epoch  2
Training: 100%|██████████| 267/267 [00:20<00:00, 12.89it/s]
train acc = 91.14%
val acc = 93.18%
Epoch  3
```

```
Training: 100%|████████| 267/267 [00:19<00:00, 13.36it/s]
train acc = 93.11%
val acc = 89.56%
Epoch  4
Training: 100%|████████| 267/267 [00:19<00:00, 13.45it/s]
train acc = 93.13%
val acc = 93.02%
Epoch  5
Training: 100%|████████| 267/267 [00:19<00:00, 13.39it/s]
train acc = 94.35%
val acc = 92.65%
Epoch  6
Training: 100%|████████| 267/267 [00:19<00:00, 13.45it/s]
train acc = 94.73%
val acc = 92.28%
Epoch  7
Training: 100%|████████| 267/267 [00:20<00:00, 12.83it/s]
train acc = 94.37%
val acc = 92.65%
Epoch  8
Training: 100%|████████| 267/267 [00:20<00:00, 12.89it/s]
train acc = 93.81%
val acc = 92.01%
Epoch  9
Training: 100%|████████| 267/267 [00:21<00:00, 12.65it/s]
train acc = 92.38%
val acc = 89.46%
Epoch  10
Training: 100%|████████| 267/267 [00:20<00:00, 13.02it/s]
train acc = 93.41%
val acc = 91.53%
Время обучения: 245.04832458496094 секунд
```

In [17]:
```
draw(train_losses, val_losses, val_full_acc, train_full_acc)
```




Опять же дообучим еще 10 эпох на learning_rate = 0.0001 для более точного схождения модели.

In [18]:
```
model.load_state_dict(best_model_weights)
```

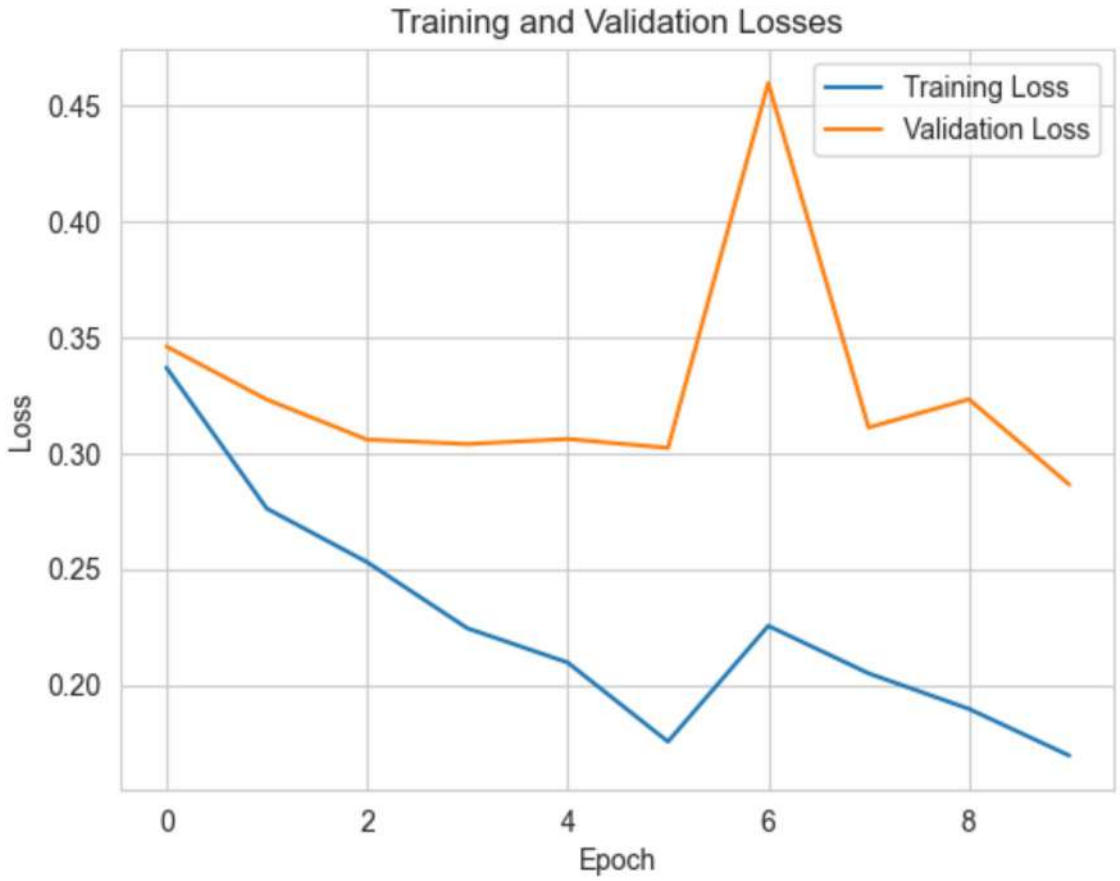
Out[18]: <All keys matched successfully>

In [19]:
```
opt = optim.AdamW(filter(lambda p: p.requires_grad, model.parameters()), lr=0.00
best_model_weights, train_losses, val_losses, val_full_acc, train_full_acc = tra
```

```
Epoch  1
Training: 100%|██████████| 267/267 [00:20<00:00, 12.83it/s]
train acc = 94.41%
val acc = 93.72%
Epoch  2
Training: 100%|██████████| 267/267 [00:21<00:00, 12.65it/s]
train acc = 95.76%
val acc = 94.04%
Epoch  3
Training: 100%|██████████| 267/267 [00:20<00:00, 12.97it/s]
train acc = 95.90%
val acc = 94.25%
Epoch  4
Training: 100%|██████████| 267/267 [00:20<00:00, 13.10it/s]
train acc = 96.55%
val acc = 94.14%
Epoch  5
Training: 100%|██████████| 267/267 [00:20<00:00, 12.85it/s]
train acc = 96.70%
val acc = 93.93%
Epoch  6
Training: 100%|██████████| 267/267 [00:20<00:00, 12.87it/s]
train acc = 97.34%
val acc = 94.14%
Epoch  7
Training: 100%|██████████| 267/267 [00:20<00:00, 13.08it/s]
train acc = 96.00%
val acc = 91.59%
Epoch  8
Training: 100%|██████████| 267/267 [00:20<00:00, 12.84it/s]
train acc = 96.57%
val acc = 94.36%
Epoch  9
Training: 100%|██████████| 267/267 [00:20<00:00, 12.93it/s]
train acc = 96.79%
val acc = 93.77%
Epoch  10
Training: 100%|██████████| 267/267 [00:20<00:00, 13.27it/s]
train acc = 97.03%
val acc = 94.52%
Время обучения: 246.25068593025208 секунд
```

Отлично, получилось целых 93% на валидационной выборке. Выведу графики обучения.

In [20]:
```
draw(train_losses, val_losses, val_full_acc, train_full_acc)
```





Посмотрим accuracy на тестовой выборке.

In [21]:
```
def test_result(model, test_loader):
    model.eval()
    with torch.no_grad():
```

```
        current_test_acc = 0
        for inputs, labels in test_loader:
            X_test = inputs.to(device)
            Y_test = labels.to(device)
            outputs = model(X_test)
            preds = torch.argmax(outputs, 1)
            current_test_acc += torch.sum(Y_test == preds)
    print('Correct answers: {} from {} '\
          .format(current_test_acc, len(test_data)))
    test_acc = current_test_acc / len(test_data)
    print('Test accuracy = {:.2f}%'.format(test_acc*100))
```

In [22]:
```
best_model = inception_resnet_v1.InceptionResnetV1(pretrained='vggface2', classi

best_model.load_state_dict(best_model_weights)

test_result(best_model.to(device), test_loader)
```

```
Correct answers: 1504 from 1589
Test accuracy = 94.65%
c:\Users\yarom\AppData\Local\Programs\Python\Python312\Lib\site-packages\torch\nn
\modules\conv.py:456: UserWarning: Plan failed with a cudnnException: CUDNN_BACKE
ND_EXECUTION_PLAN_DESCRIPTOR: cudnnFinalize Descriptor Failed cudnn_status: CUDNN
_STATUS_NOT_SUPPORTED (Triggered internally at ..\aten\src\ATen\native\cudnn\Conv
_v8.cpp:919.)
  return F.conv2d(input, weight, bias, self.stride,
```

In [23]:
```
torch.save(best_model, 'best_model.pth')
```

In [25]:
```
models_dict = {'logistic regression':['4.2%',99],'random forest':['3.16%',40],'S
models_df = pd.DataFrame.from_dict(models_dict, orient='index', columns=['accura
models_df
```

Out[25]:

| | accuracy | time |
|---|---|---|
| **logistic regression** | 4.2% | 99.000 |
| **random forest** | 3.16% | 40.000 |
| **SVM** | 6.65% | 540.000 |
| **KNN** | 2.62% | 0.012 |
| **GaussianNB** | 2.03% | 0.100 |
| **CatBoost** | 5.07% | 193.000 |
| **inception_resnet_v2** | 77.32% | 1512.000 |
| **inception_resnet_v1** | 94.65% | 492.000 |

# Вывод

Модели, основанные на глубоком обучении (например, Inception-ResNet-V1 и Inception-ResNet-V2), показали значительно более высокую точность по сравнению с традиционными методами машинного обучения, такими как логистическая регрессия, случайный лес и SVM. Inception-ResNet-V1 с точностью в 94.65% является

наиболее успешной моделью в задачах идентификации личности, что подчеркивает преимущества использования сложных архитектур нейронных сетей для обработки и анализа изображений в таких задачах.

Несмотря на высокую точность, нейронные сети требуют значительно больше времени для обучения. Например, Inception-ResNet-V2 требовало 1512 секунд, в то время как более простые модели, такие как KNN, обучались менее чем за секунду. Это является минусом в случаях, когда необходимо быстро обучить модель и получить результаты.

Если сравнить эти результаты с аналогичными решениями в индустрии или академических кругах, можно заметить, что использование нейронных сетей показывает лучшие результапы в любых задачах, связанных с изображениями.

Сама модель имеет большую область применения в задачах безопасности, так как идентификация личности по изображению применяется в разблокировке смартфонов, доступах в какие-либо места по биометрии (например, в метро или офис), а также при входе в банковские приложения и т.д. Эти примеры демонстрируют широкий потенциал применения идентификации личности по изображению, подчеркивая её важность и многообещающие перспективы для дальнейшего развития и интеграции в различные аспекты повседневной жизни и бизнеса.

Рассуждая о перспективах развития модели, можно отметить, что для улучшения производительности и увеличения её устойчивости к изменениям во входных данных, можно использовать различные техники аугментации изображений. Это может включать изменения масштаба, повороты, изменение яркости и контрастности, что позволит модели лучше обобщать и работать с разнообразными условиями освещения и позами лиц. Кроме того, к улучшению точности модели может привести пополнение набора данных изображениями, которые лучше отражают целевую популяцию и различные условия окружающей среды. Кроме того помимо Inception-ResNet можно исследовать другие современные архитектуры, такие как EfficientNet или трансформеры для визуальных задач, которые показывают лучшие результаты. Также, поскольку идентификация личности по изображению часто применяется в задачах безопаности, необходимо принимать во внимание False positive rate (FPR), так как модель не должна давать положительный результат, когда на изображении на самом деле другая личность, поскольку это может привести к угрозе безопасности, что является критически важным в данном направлении.

Ссылка на репозиторий с кодом: https://github.com/yaromirgusev/course_work