

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»
Московский институт электроники и математики

Морозов Денис Сергеевич
**Моя первая нейронная сеть. Определение марки
автомобиля по фотографии**

ПРОЕКТНАЯ РАБОТА
студента образовательной программы бакалавриата
«Прикладная математика»
по направлению подготовки 01.03.04 прикладная математика

Студент
Д. С. Морозов

Руководитель проектной работы
Доктор физико-математических наук,
профессор
Попов Виктор Юрьевич

Москва 2024г.

Содержание

| | | |
|----------|---|----------|
| 1 | Аннотация | 2 |
| 2 | Введение | 2 |
| 3 | Обзор литературы и технологий | 4 |
| 3.1 | Обзор нейронных сетей для распознавания изображений | 4 |
| 3.2 | Сравнение различных архитектур CNN | 4 |
| 4 | Данные и их подготовка | 5 |
| 4.1 | Источники данных | 5 |
| 4.2 | Аугментация данных | 6 |
| 4.3 | Разделение данных | 6 |
| 5 | Архитектура нейросети | 7 |
| 5.1 | Выбор архитектуры | 7 |
| 5.2 | Детали реализации | 9 |
| 6 | Обучение модели | 9 |
| 6.1 | Настройка гиперпараметров | 9 |
| 6.2 | Использование функции потерь | 10 |
| 6.3 | Процесс обучения и валидации | 10 |

| | |
|-----------------------------------|-----------|
| 7 Результаты и обсуждение | 11 |
| 7.1 Анализ результатов | 11 |
| 7.2 Возможные улучшения | 13 |
| 8 Использование модели | 16 |
| 9 Заключение | 22 |

1 Аннотация

В рамках данного проекта разрабатывалась модель нейронной сети для решения задачи с определением марки автомобиля с использованием языка программирования Python. Были протестированы несколько модель, способные прогнозировать правильные классы автомобилей. Средняя абсолютная ошибка на тестовых данных составила 1/30 фото. Это неплохой показатель, учитывая небольшой размер датасета и большое количество существующих машин.

2 Введение

Искусственный интеллект и машинное обучение, в частности глубокие нейронные сети, стали важными инструментами в распознавании изображений. Современные методы машинного обучения позволяют решать сложные задачи анализа и про-

гнозирования, обеспечивая высокую точность и надежность результатов. Этот проект рассматривает разработку и применение модели нейронной сети для решения задачи классификации изображений, что представляет собой одну из наиболее востребованных задач в области искусственного интеллекта и машинного обучения. В данном отчете подробно рассматриваются этапы создания, обучения и тестирования модели, а также полученные результаты и возможности их улучшения.

Проект выполнялся на языке программирования Python, используя среду разработки Pycharm и современные инструменты и библиотеки, такие как PyTorch Lightning¹ для упрощения процесса обучения модели и архитектура EfficientNet², которая отличается высокой производительностью и экономичностью.

EfficientNet использует стратегию компаундного масштабирования для сбалансированного увеличения глубины, ширины и разрешения сети, что позволяет добиться лучших результатов с меньшими вычислительными затратами. Так же использовался ChatGPT для предотвращения ошибок и конфликтов библиотек³

¹. YouTube Video on Image Classification.

². EfficientNet-B5.

³. OpenAI ChatGPT.

3 Обзор литературы и технологий

3.1 Обзор нейронных сетей для распознавания изображений

Глубокие нейронные сети, особенно сверточные (CNN), показали высокую эффективность в задачах распознавания изображений. CNN состоят из слоев свертки и объединения, которые позволяют автоматически выделять признаки из изображений.

Основные компоненты нейронной сети включают входной слой, несколько скрытых слоев и выходной слой. Каждый нейрон в скрытом слое выполняет нелинейное преобразование данных, что позволяет модели обучаться сложным зависимостям между входными и выходными данными.

3.2 Сравнение различных архитектур CNN

Мной были рассмотрены архитектуры, такие как ResNet50⁴, Resnet152⁵ и EfficientNet. В последние годы глубокое обучение и нейронные сети стали основным инструментом для решения задач классификации изображений. Одной из самых эффективных архитектур, предложенных в последние годы, является EfficientNet, которая оптимизирует количество параметров и вычислительных ресурсов, сохраняя высокую

⁴. PyTorch Hub - ResNet50.

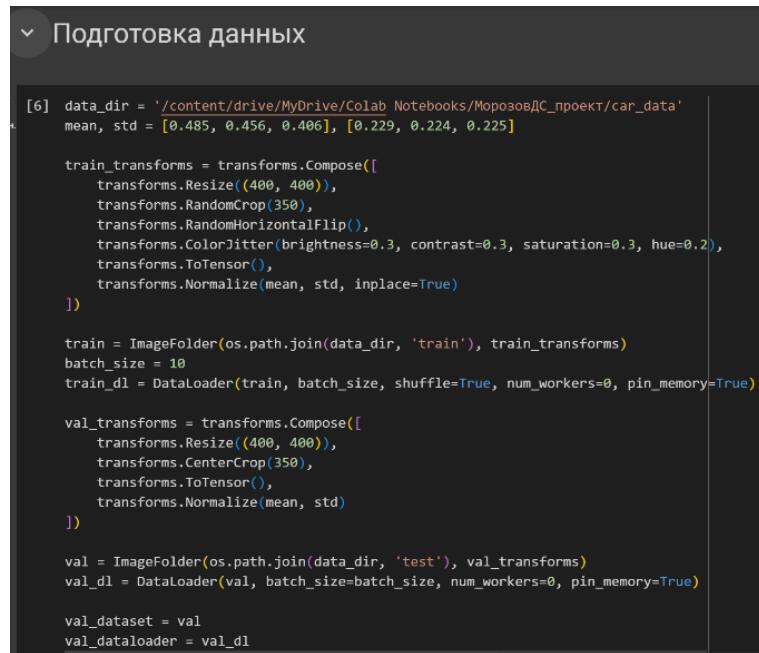
⁵. Torchvision Models - ResNet152.

точность модели.

4 Данные и их подготовка

4.1 Источники данных

Для обучения модели использован датасет, содержащий фотографии автомобилей различных марок, собранные из открытых источников, таких как интернет-ресурсы, а именно Kaggle⁶. Для подготовки данных использовались библиотеки torchvision и albumenations. Датасет содержал изображения автомобилей, которые были разделены на тренировочные и тестовые наборы.



```
[6] data_dir = '/content/drive/MyDrive/Colab Notebooks/МорозовДС_проект/car_data'
mean, std = [0.485, 0.456, 0.406], [0.229, 0.224, 0.225]

train_transforms = transforms.Compose([
    transforms.Resize(400),
    transforms.RandomCrop(350),
    transforms.RandomHorizontalFlip(),
    transforms.ColorJitter(brightness=0.3, contrast=0.3, saturation=0.3, hue=0.2),
    transforms.ToTensor(),
    transforms.Normalize(mean, std, inplace=True)
])

train = ImageFolder(os.path.join(data_dir, 'train'), train_transforms)
batch_size = 10
train_dl = DataLoader(train, batch_size, shuffle=True, num_workers=0, pin_memory=True)

val_transforms = transforms.Compose([
    transforms.Resize(400),
    transforms.CenterCrop(350),
    transforms.ToTensor(),
    transforms.Normalize(mean, std)
])

val = ImageFolder(os.path.join(data_dir, 'test'), val_transforms)
val_dl = DataLoader(val, batch_size=batch_size, num_workers=0, pin_memory=True)

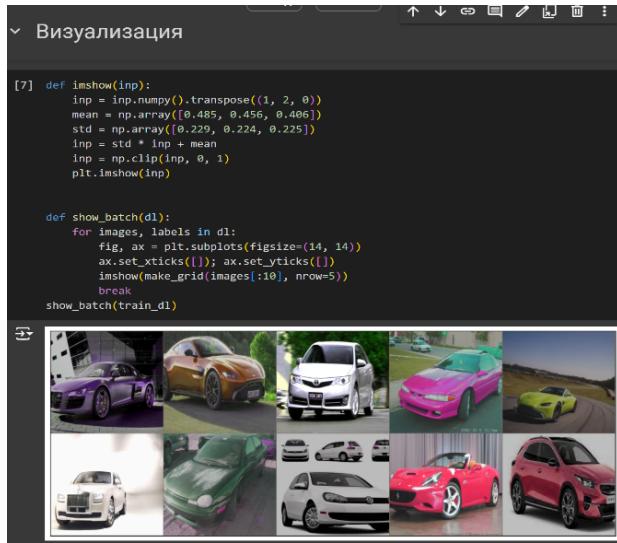
val_dataset = val
val_dataloader = val_dl
```

Рис. 1: Подговка данных

⁶Mooney. Stanford Cars Dataset with Fastai v1.

4.2 Аугментация данных

Для увеличения объема данных и улучшения обобщающей способности модели использовались техники аугментации, включая вращение, изменение масштаба, случайные обрезки, горизонтальные отражения и изменение яркости, контраста, насыщенности и отражение изображений.

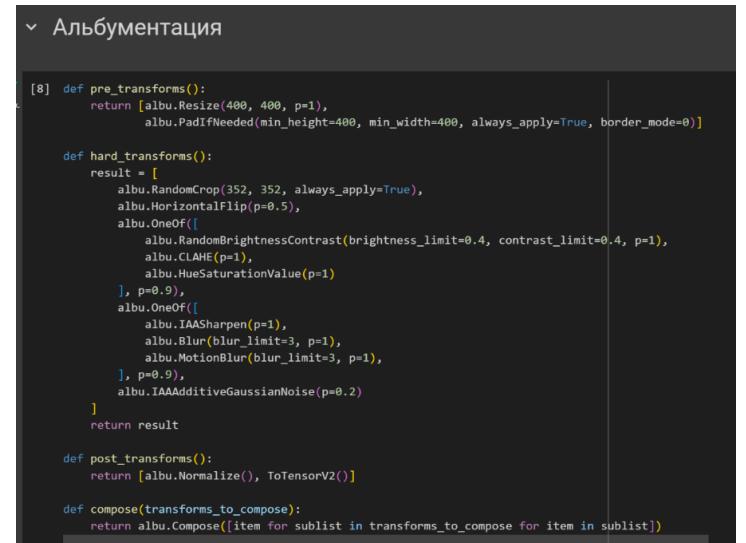


Визуализация

```
[7] def imshow(inp):
    inp = inp.numpy().transpose((1, 2, 0))
    mean = np.array([0.485, 0.456, 0.406])
    std = np.array([0.229, 0.224, 0.225])
    inp = std * inp + mean
    inp = np.clip(inp, 0, 1)
    plt.imshow(inp)

def show_batch(dl):
    for images, labels in dl:
        fig, ax = plt.subplots(figsize=(14, 14))
        ax.set_xticks([]); ax.set_yticks([])
        imshow(torchvision.utils.make_grid(images[:10], nrow=5))
        break
    show_batch(train_dl)
```

grid of 10 car images



Альбументация

```
[8] def pre_transforms():
    return [albu.Resize(400, 400, p=1),
            albu.PadIfNeeded(min_height=400, min_width=400, always_apply=True, border_mode=0)]

def hard_transforms():
    result = [
        albu.RandomCrop(352, 352, always_apply=True),
        albu.HorizontalFlip(p=0.5),
        albu.OneOf([
            albu.RandomBrightnessContrast(brightness_limit=0.4, contrast_limit=0.4, p=1),
            albu.CLAHE(p=1),
            albu.HueSaturationValue(p=1)
        ], p=0.9),
        albu.OneOf([
            albu.IAASharpen(p=1),
            albu.Blur(blur_limit=3, p=1),
            albu.MotionBlur(blur_limit=3, p=1)
        ], p=0.9),
        albu.IAAAdditiveGaussianNoise(p=0.2)
    ]
    return result

def post_transforms():
    return [albu.Normalize(), ToTensorV2()]

def compose(transforms_to_compose):
    return albu.Compose([item for sublist in transforms_to_compose for item in sublist])
```

grid of 10 car images

Рис. 2: Альбументация

Рис. 3: Визуализация

4.3 Разделение данных

Данные были разделены на обучающую и тестовую выборки в соотношении 55/45. Это позволяет оценивать модель на данных, которые она не видела во время обучения.

| | | | |
|---|--|---|--|
|  | train196 |  | test196 |
| Тип: | Папка с файлами | Тип: | Папка с файлами |
| Расположение: | C:\Users\Денис\Desktop\python\проект\car_data\train\train196 | Расположение: | C:\Users\Денис\Desktop\python\проект\car_data\test\test196 |
| Размер: | 949 МБ (995 828 547 байт) | Размер: | 948 МБ (994 257 435 байт) |
| На диске: | 965 МБ (1 012 428 800 байт) | На диске: | 963 МБ (1 010 671 616 байт) |
| Содержит: | Файлов: 8 182; папок: 197 | Содержит: | Файлов: 8 041; папок: 196 |

Рис. 5: Папка с валидационными данными

Рис. 4: Папка с обучающими данными

5 Архитектура нейросети

5.1 Выбор архитектуры

Наиболее подходящая конфигурация сети определялась экспериментально, с учетом факторов таких как глубина сети, количество фильтров и размер ядра свертки. Так же были настроены логирование с использованием W&B⁷, контроль точки модели и ранняя остановка.

⁷. Weights and Biases.

```

class EffNet(LightningModule):
    def __init__(self, num_target_classes=246, #если не 196
                 backbone: str = 'efficientnet-b5', batch_size: int = 8,
                 lr: float = 5e-4, wd: float = 0, num_workers: int = 4, factor: float = 0.5, **kwargs):
        super().__init__()
        self.num_target_classes = num_target_classes
        self.backbone = backbone
        self.batch_size = batch_size
        self.lr = lr
        self.wd = wd
        self.num_workers = num_workers
        self.factor = factor
        self.save_hyperparameters()
        self._build_model()

    def _build_model(self):
        self.net = EfficientNet.from_pretrained(self.backbone)
        in_features = self.net.fc.in_features
        _fc_layers = [nn.Linear(in_features, self.num_target_classes)]
        self.net.fc = nn.Sequential(*_fc_layers)

    def setup(self, stage: str):
        data_dir = '/content/drive/MyDrive/Colab Notebooks/МорозовД/Проект/car_data'
        mean, std = [0.485, 0.456, 0.406], [0.229, 0.224, 0.225]
        train_transforms = transforms.Compose([
            transforms.Resize(400, 400),
            transforms.RandomCrop(350),
            transforms.RandomHorizontalFlip(),
            transforms.ColorJitter(brightness=0.3, contrast=0.3, saturation=0.3, hue=0.2),
            transforms.ToTensor(),
            transforms.Normalize(mean, std, inplace=True)
        ])
        train = ImageFolder(os.path.join(data_dir, 'train'), train_transforms)

        val_transforms = transforms.Compose([
            transforms.Resize(400, 400),
            transforms.ToTensor(),
            transforms.Normalize(mean, std, inplace=True)
        ])
        val = ImageFolder(os.path.join(data_dir, 'test'), val_transforms)
        valid, _ = random_split(val, [len(val), 0])

        self.train_dataset = train
        self.val_dataset = valid

    def val_acc(self, y_pred, y_true):
        return accuracy(y_pred, y_true)

    def forward(self, x):
        return self.net.forward(x)

    def training_step(self, batch, batch_idx):
        x, y = batch
        y_logits = self.forward(x)
        train_loss = F.cross_entropy(y_logits, y)
        return train_loss

    def train_dataloader(self):
        return DataLoader(dataset=self.train_dataset, batch_size=self.batch_size,
                         num_workers=self.num_workers, shuffle=True, pin_memory=True)

    def validation_step(self, batch, batch_idx):
        x, y = batch
        y_logits = self.forward(x)
        val_loss = F.cross_entropy(y_logits, y)
        acc = self.val_acc(y_logits, y)
        return {'val_loss': val_loss, 'val_acc': acc}

    def validation_epoch_end(self, outputs):
        val_loss_mean = torch.stack([output['val_loss'] for output in outputs]).mean()
        val_acc_mean = torch.stack([output['val_acc'] for output in outputs]).mean()
        self.log('val_loss', val_loss_mean)
        self.log('val_acc', val_acc_mean)

    def configure_optimizers(self):
        optimizer = Adam(self.parameters(), lr=self.lr, weight_decay=self.wd)
        lr_scheduler = {'scheduler': ReduceLROnPlateau(optimizer, factor=self.factor, patience=2, mode='max',
                                                       name: 'learning_rate', 'monitor': 'val_acc'}
        return [optimizer], [lr_scheduler]

    def val_dataloader(self):
        return DataLoader(dataset=self.val_dataset, batch_size=self.batch_size,
                         num_workers=self.num_workers, shuffle=False, pin_memory=True)

```

Рис. 6: Модель EfficientNet

```

class ResNet152(LightningModule):
    def __init__(self, **kwargs):
        super().__init__()
        self.train_bs = Train_bs
        self.batch_size = batch_size
        self.num_workers = num_workers
        self.hidden_1 = hidden_1
        self.hidden_2 = hidden_2
        self.factor = factor
        self.total_steps = total_steps
        self.get_start_lr = get_start_lr
        self.epochs_frozen = epochs_frozen
        self.total_epochs = total_epochs
        self.annual_strategy = annual_strategy
        self.love_hyperparameters()

    def _build_model():
        self.build_model(self)

    def build_model(self):
        max_target_classes = 196
        backbone = models.resnet152(pretrained=True)
        layers = list(backbone.children())[-1]
        self.feature_extractor = nn.Sequential(*layers)
        fc_layers = [nn.Linear(self.hidden_1, self.hidden_2),
                    nn.Linear(self.hidden_2, max_target_classes)]
        self.fc = nn.Sequential(*fc_layers)

    def forward(self, x):
        x = self.feature_extractor(x)
        x = x.squeeze(-1).squeeze(-1)
        x = self.fc(x)
        return x

    def train(self, mode=True):
        super().train(mode=mode)
        epoch = self.current_epoch
        if epoch < self.epochs_frozen and mode:
            frozen_params = self.feature_extractor.parameters()
            train(self.feature_extractor)
            train(self.fc)

    def training_step(self, batch, batch_idx):
        x, y = batch
        y_logits = self.forward(x)

        train_loss = -F.cross_entropy(y_logits, y)
        acc = accuracy(y_logits, y)

        tgm_dict = {'train_loss': train_loss}
        output = OrderedDict(tgm_dict)
        output['train_acc'] = acc
        output['train'] = tgm_dict
        output['progress_bar': tgm_dict]

        return output

    def training_epoch_end(self, outputs):
        train_loss_mean = torch.stack([output['train_loss'] for output in outputs]).mean()
        avg_acc = torch.stack([output['train_acc'] for x in outputs]).mean()

        tensorboard_logs = {'train_loss': train_loss_mean, 'train_acc': avg_acc}
        return {'train_loss': train_loss_mean, 'log': tensorboard_logs}

    def validation_step(self, batch, batch_idx):
        x, y = batch
        y_logits = self.forward(x)

        val_loss = -F.cross_entropy(y_logits, y)
        acc = accuracy(y_logits, y)

        return {'val_loss': val_loss, 'val_acc': acc}

    def validation_epoch_end(self, outputs):
        val_loss_mean = torch.stack([output['val_loss'] for output in outputs]).mean()
        avg_acc = torch.stack([output['val_acc'] for output in outputs]).mean()

        tensorboard_logs = {'val_loss': val_loss_mean, 'val_acc': avg_acc}
        return {'val_loss': val_loss_mean, 'log': tensorboard_logs}

    def configure_optimizers(self):
        if self.current_epoch < self.epochs_frozen:
            optimizer = optim.Adam(filter(lambda p: p.requires_grad, self.parameters()), lr=self.lr)
            return optimizer
        else:
            optimizer = optim.Adam(filter(lambda p: p.requires_grad, self.parameters()), lr=self.lr)
            scheduler = OneCycleLR(optimizer, max_lr=self.lr,
                                   total_steps=self.total_steps,
                                   pct_start=self.pct_start, annual_strategy=self.annual_strategy)
            return [optimizer], [scheduler]

    def setup(self, stage):
        data_dir = '/content/drive/MyDrive/Colab Notebooks/МорозовД/Проект/car_data'
        mean, std = [0.485, 0.456, 0.406], [0.229, 0.224, 0.225]
        train_transforms = transforms.Compose([
            transforms.RandomCrop(350),
            transforms.RandomHorizontalFlip(),
            transforms.ColorJitter(brightness=0.3, contrast=0.3, saturation=0.3, hue=0.2),
            transforms.ToTensor(),
            transforms.Normalize(mean, std, inplace=True)
        ])
        train = ImageFolder(data_dir + '/train', train_transforms)

        val_transforms = transforms.Compose([
            transforms.Resize(400, 400),
            transforms.ToTensor(),
            transforms.Normalize(mean, std, inplace=True)
        ])
        val = ImageFolder(data_dir + '/test', val_transforms)
        valid, _ = random_split(val, [len(val), 0])

        self.train_dataset = train
        self.val_dataset = valid

    def train_dataloader(self):
        return DataLoader(dataset=self.train_dataset, batch_size=self.batch_size,
                          num_workers=self.num_workers, shuffle=True, pin_memory=True)

    def val_dataloader(self):
        return DataLoader(dataset=self.val_dataset, batch_size=self.batch_size,
                          num_workers=self.num_workers, shuffle=False, pin_memory=True)

```

Рис. 7: Модель ResNet

5.2 Детали реализации

Модель состоит из нескольких свёрточных слоев, за которыми следуют слои max-pooling и полносвязные слои. Каждый свёрточный слой использует ReLU (Rectified Linear Unit) в качестве функции активации. Так же использовал графический процессор⁸ для ускорения обучения с помощью графических ядер⁹.

6 Обучение модели

6.1 Настройка гиперпараметров

Гиперпараметры, такие как скорость обучения, размер батча и количество эпох, были настроены с использованием методов поиска по сетке и случайного поиска. Начальная скорость обучения установлена на 1e-4, размер батча — 32, количество эпох — 50.

⁸. YouTube video on GPU usage in PyTorch.

⁹. Getting started with PyTorch.

✗ Командный Интерфейс(log обучения прилагаю)

```
[ ] class EffNetCLI(EffNet):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)

    @staticmethod
    def add_model_specific_args(parent_parser):
        parser = argparse.ArgumentParser(parents=[parent_parser], add_help=False)
        parser.add_argument('--backbone', type=str, default='efficientnet-b7', help='Backbone architecture')
        parser.add_argument('--batch_size', type=int, default=8, help='Batch size')
        parser.add_argument('--lr', type=float, default=5e-4, help='Learning rate')
        parser.add_argument('--wd', type=float, default=0, help='Weight decay')
        parser.add_argument('--num_workers', type=int, default=4, help='Number of workers for data loading')
        parser.add_argument('--factor', type=float, default=0.5, help='Factor for reducing learning rate')
        return parser

    @staticmethod
    def get_args() -> Namespace:
        parent_parser = ArgumentParser(add_help=False)
        parent_parser.add_argument('--gpus', type=int, default=1, help='Number of GPUs')
        parent_parser.add_argument('--use-16bit', dest='use_16bit', action='store_true', help='Use 16-bit precision')
        parent_parser.add_argument('--epochs', default=15, type=int, metavar='N', help='Total number of epochs', dest='nb_epochs')
        parent_parser.add_argument('--patience', default=3, type=int, metavar='ES', help='Early stopping patience', dest='patience')

        parser = EffNetCLI.add_model_specific_args(parent_parser)
        return parser.parse_args()

    def main(args):
        seed_everything(42)
        model = EffNetCLI(**vars(args))
        wandb.login(key=os.environ.get('WANDB_API_KEY'))
        wandb_logger = WandbLogger(name='Name', project='Project')
        checkpoint_cb = ModelCheckpoint(dirpath='./', filename='cars-{epoch:02d}-{val_acc:.4f}', monitor='val_acc', mode='max', save_top_k=1)
        early = EarlyStopping(patience=5, monitor='val_acc', mode='max')

        trainer = Trainer(
            gpus=args.gpus,
            logger=wandb_logger,
            max_epochs=args.nb_epochs,
            deterministic=True,
            precision=16 if args.use_16bit else 32,
            callbacks=[checkpoint_cb, LearningRateMonitor(), early],
        )
        trainer.fit(model)

    if __name__ == '__main__':
        args = EffNetCLI.get_args()
        main(args)
```

Рис. 8: Настройка гиперпараметров

6.2 Использование функции потерь

Для задачи классификации использовалась кросс-энтропийная функция потерь, которая хорошо подходит для многоклассовых задач.

6.3 Процесс обучения и валидации

Процесс обучения включал регулярную валидацию модели на отдельном наборе данных для предотвращения переобучения. Переобучение (overfitting) — это проблема, возникающая, когда модель хорошо работает на тренировочных данных,

но показывает плохие результаты на новых, невиданных данных. Для борьбы с переобучением используются следующие методы:

- Регуляризация: включает добавление штрафа за сложность модели в функцию потерь. Одним из популярных методов является L2-регуляризация.
- Dropout: метод, при котором случайно отключается часть нейронов в процессе обучения, что улучшает обобщающую способность модели.
- Кросс-валидация: разделение данных на несколько частей и многократное обучение модели на различных комбинациях этих частей.

7 Результаты и обсуждение

7.1 Анализ результатов

Модель ResNet50 достигла точности 75% на тестовом наборе данных. Анализ показал, что модель хорошо распознаёт популярные марки автомобилей, но испытывает трудности с менее распространёнными марками.



Рис. 9: Метрики процесса обучения модели, автоматически созданные с помощью библиотеки wandb

Модель EffNet достигла точности 93.35% на тестовом наборе данных. Анализ показал, что модель хорошо распознаёт марки автомобилей.

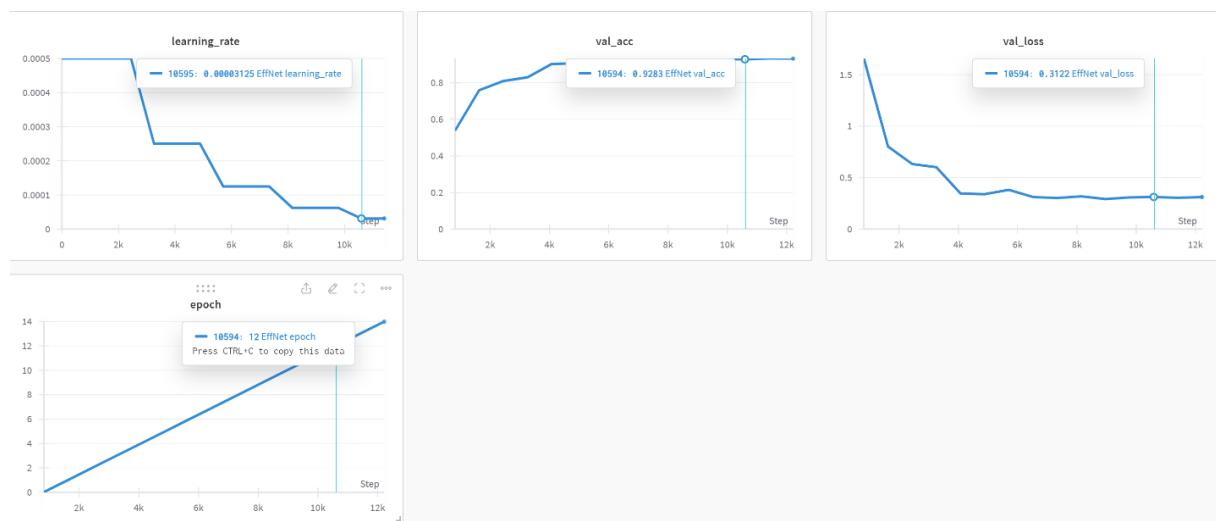


Рис. 10: Метрики процесса обучения модели, автоматически созданные с помощью библиотеки wandb

7.2 Возможные улучшения

Для улучшения результатов предлагается увеличить размер датасета, использовать более сложные архитектуры и применять методы регуляризации, такие как dropout. На датасете уже из 246 марок автомобиля модель показала точность результата 84,5%. Но при этом при наборе данных на реальных фотографиях проявила более точные результаты.

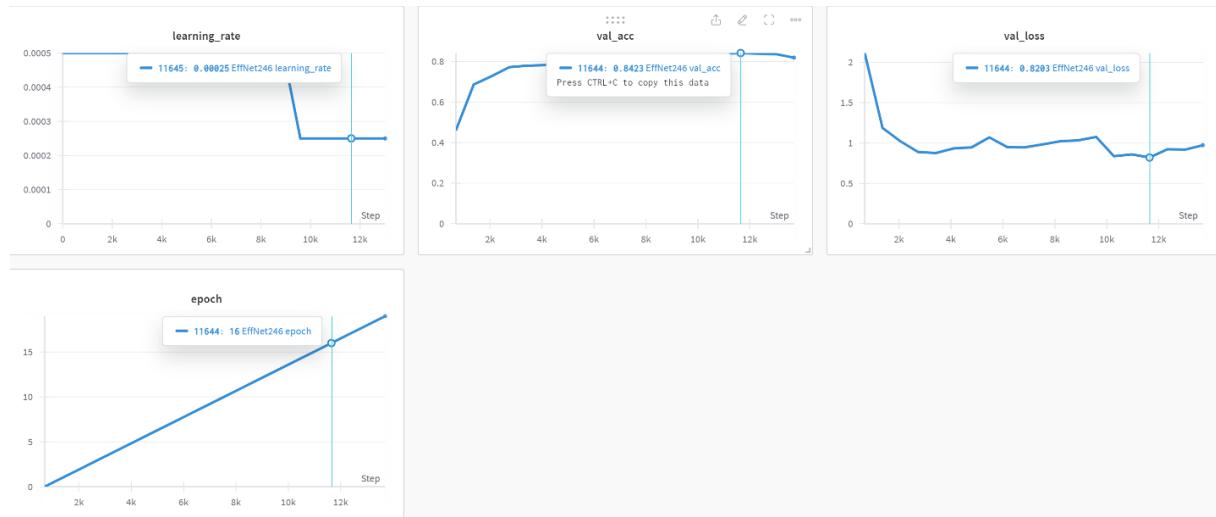


Рис. 11: Метрики процесса обучения модели, автоматически созданные с помощью библиотеки wandb

Теперь представим все модели на одном графике и обсудим их содержимое.

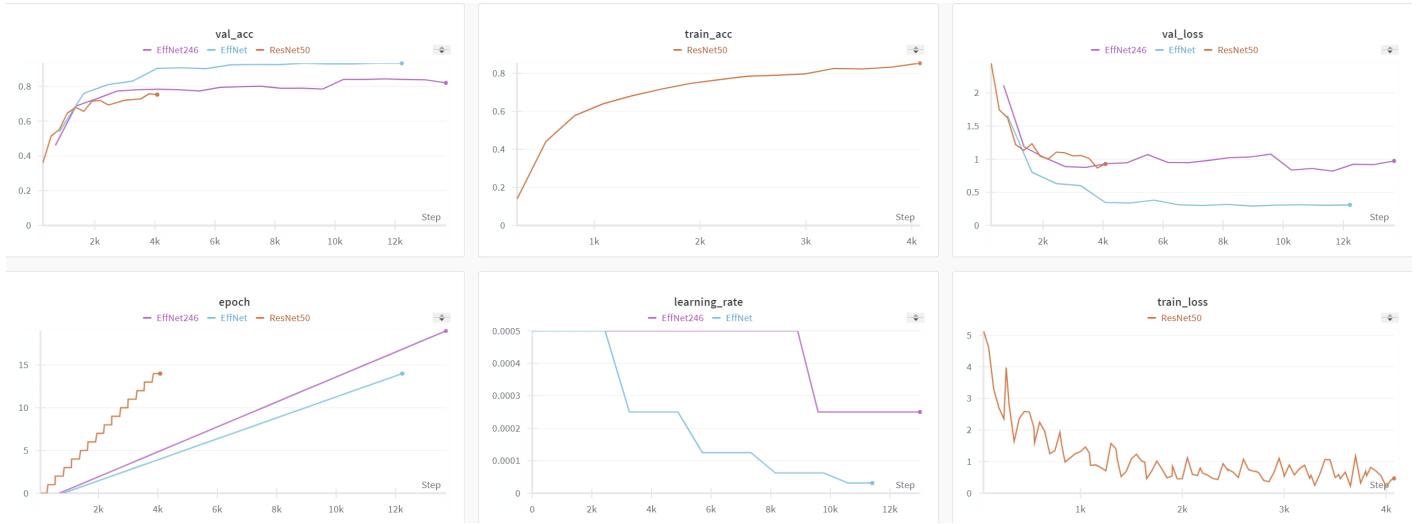


Рис. 12: Метрики процесса обучения всех моделей, автоматически созданные с помощью библиотеки wandb

Описание графиков:

- **График *val_acc* (верхний левый):** Отображает точность (accuracy) на валидационном наборе данных для моделей EffNet246, EffNet и ResNet50 по мере увеличения числа шагов (steps) или эпох (epochs). Видно, что точность на валидационном наборе увеличивается по мере обучения модели, достигая высоких значений.
- **График *train_acc* (верхний центральный):** Отображает точность на тренировочном наборе данных для модели ResNet50 по мере увеличения числа шагов. Точность на тренировочном наборе также увеличивается по мере обучения, достигая значений около 0.85.

- **График *val_loss* (верхний правый):** Отображает значение функции потерь (loss) на валидационном наборе данных для моделей EffNet246, EffNet и ResNet50 по мере увеличения числа шагов. Значение потерь уменьшается по мере обучения модели, что указывает на улучшение её производительности на валидационных данных.
- **График *epoch* (нижний левый):** Отображает номер текущей эпохи для моделей EffNet246, EffNet и ResNet50 по мере увеличения числа шагов. Видно, что обучение моделей продолжается до 15-й эпохи.
- **График *learning_rate* (нижний центральный):** Отображает изменение скорости обучения (learning rate) для моделей EffNet246 и EffNet по мере увеличения числа шагов. Видно, что скорость обучения уменьшается ступенчато в течение обучения.
- **График *train_loss* (нижний правый):** Отображает значение функции потерь на тренировочном наборе данных для модели ResNet50 по мере увеличения числа шагов. Значение потерь на тренировочных данных снижается, что указывает на то, что модель обучается правильно. Рассчитывайте по этой формуле:

$$J(W, b) = -\frac{1}{m} \sum_{i=1}^m (y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i))$$

где m - количество примеров в обучающей выборке, y_i - истинная метка класса

для i -го примера, \hat{y}_i - предсказанная моделью вероятность принадлежности к классу.

Эти графики вместе позволяют анализировать процесс обучения моделей:

- **Сходимость:** Уменьшение функции потерь на тренировочных и валидационных данных указывает на то, что модели успешно обучаются.
- **Обучение и валидация:** Сравнение точности и потерь на тренировочных и валидационных данных позволяет оценить, не происходит ли переобучение моделей. Если валидационные метрики (точность и потери) сильно отличаются от тренировочных, это может указывать на переобучение.
- **Эффективность моделей:** Высокие значения точности и низкие значения потерь показывают, что модели достигают хорошей производительности.

8 Использование модели

В данной главе рассмотрим использование модели и выявим несколько закономерностей. Протестируем фотографии разных размеров на двух моделях:

- EfficientNet на датасете из 196 марок
- EfficientNet на датасете из 246 марок

Начнем с простого. В датасете есть Ferrari 458 Italia Coupe 2012, протестируем модель на фотографии Ferrari 458 Italia Coupe 2013

```
#!/usr/bin/python3
# checkpoint_path = "/content/drive/MyDrive/Colab Notebooks/MoscowIC_project/model_196_cars-epoch=13-val_acc=0.9328ckpt"
model = EfficientNet.load_from_checkpoint(checkpoint_path, num_classes=196)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)

checkpoint.load_from_checkpoint(checkpoint_path, map_location=device)
intersecting_keys = set(model.state_dict().keys()) & set(checkpoint['state_dict'].keys())
state_dict = {k: v for k, v in checkpoint['state_dict'].items() if k in intersecting_keys}
model.load_state_dict(state_dict)

image_path = "/content/drive/MyDrive/Colab Notebooks/MoscowIC_project/recr/moscow/1.jpg"
predicted_class_index = predict_class(image_path, model, device)
train_data_dir = "/content/drive/MyDrive/Colab Notebooks/MoscowIC_project/car_data/train196"
classes = ImageFolder(train_data_dir).classes
predicted_class_name = classes[predicted_class_index]
print("Predicted class:", predicted_class_name)

# Отправляем изображение и выводим его
from IPython.display import Image
display(Image(image_path))

if __name__ == '__main__':
    main()

INFO:pytorch_lightning.utilities.migration.utils:Lightning automatically upgraded your loaded checkpoint from v1.2.0 to v2.2.4.
Loaded pretrained weights for efficientnet-b5
Predicted class: Ferrari 458 Italia Coupe 2012
```

Рис. 13: Демонстрация работы модели EfficientNet на датасете из 196 марок

Обе модели легко справились с этой задачей.

Усложним задачу. Возьмем фотографию из auto.ru. А так же возьмем по новее год 2015 (так как в датасете марки автомобилей максимум 2012 год).

```
# checkpoint_path = "/content/drive/MyDrive/Colab Notebooks/MoscowIC_project/model_246_cars-epoch=16-val_acc=0.8423ckpt"
model = EfficientNet.load_from_checkpoint(checkpoint_path, num_classes=246)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)

checkpoint.load_from_checkpoint(checkpoint_path, map_location=device)
intersecting_keys = set(model.state_dict().keys()) & set(checkpoint['state_dict'].keys())
state_dict = {k: v for k, v in checkpoint['state_dict'].items() if k in intersecting_keys}
model.load_state_dict(state_dict)

image_path = "/content/drive/MyDrive/Colab Notebooks/MoscowIC_project/recr/moscow/2.jpg"
predicted_class_index = predict_class(image_path, model, device)
train_data_dir = "/content/drive/MyDrive/Colab Notebooks/MoscowIC_project/car_data/train196"
classes = ImageFolder(train_data_dir).classes
predicted_class_name = classes[predicted_class_index]
print("Predicted class:", predicted_class_name)

# Отправляем изображение и выводим его
from IPython.display import Image
width = 400
height = 350
display(Image(filename=image_path, width=width, height=height))

if __name__ == '__main__':
    main()

INFO:pytorch_lightning.utilities.migration.utils:Lightning automatically upgraded your loaded checkpoint from v1.2.0 to v2.2.4.
Loaded pretrained weights for efficientnet-b5
Predicted class: BMW 3 Series Sedan 2012
```

Рис. 15: Демонстрация работы модели EfficientNet на датасете из 196 марок

```
# Путь к сохраненному чекпоинту модели
checkpoint_path = "/content/drive/MyDrive/Colab Notebooks/MoscowIC_project/model_246_cars-epoch=16-val_acc=0.8423ckpt"
model = EfficientNet.load_from_checkpoint(checkpoint_path, num_classes=246)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)

checkpoint.load_from_checkpoint(checkpoint_path, map_location=device)
intersecting_keys = set(model.state_dict().keys()) & set(checkpoint['state_dict'].keys())
state_dict = {k: v for k, v in checkpoint['state_dict'].items() if k in intersecting_keys}
model.load_state_dict(state_dict)

image_path = "/content/drive/MyDrive/Colab Notebooks/MoscowIC_project/recr/moscow/1.jpg"
predicted_class_index = predict_class(image_path, model, device)
train_data_dir = "/content/drive/MyDrive/Colab Notebooks/MoscowIC_project/car_data/train196"
classes = ImageFolder(train_data_dir).classes
predicted_class_name = classes[predicted_class_index]
print("Predicted class:", predicted_class_name)

# Отправляем изображение и выводим его
from IPython.display import Image
display(Image(image_path))

if __name__ == '__main__':
    main()

INFO:pytorch_lightning.utilities.migration.utils:Lightning automatically upgraded your loaded checkpoint from v1.2.0 to v2.2.4.
Loaded pretrained weights for efficientnet-b5
Predicted class: Ferrari 458 Italia Coupe 2012
```

Рис. 14: Демонстрация работы модели EfficientNet на датасете из 246 марок

Обе модели легко справились с этой задачей.

Усложним задачу. Возьмем фотографию из auto.ru. А так же возьмем по новее год 2015 (так как в датасете марки автомобилей максимум 2012 год).

```
# checkpoint_path = "/content/drive/MyDrive/Colab Notebooks/MoscowIC_project/model_196_cars-epoch=13-val_acc=0.9328ckpt"
model = EfficientNet.load_from_checkpoint(checkpoint_path, num_classes=196)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)

checkpoint.load_from_checkpoint(checkpoint_path, map_location=device)
intersecting_keys = set(model.state_dict().keys()) & set(checkpoint['state_dict'].keys())
state_dict = {k: v for k, v in checkpoint['state_dict'].items() if k in intersecting_keys}
model.load_state_dict(state_dict)

image_path = "/content/drive/MyDrive/Colab Notebooks/MoscowIC_project/recr/moscow/2.jpg"
predicted_class_index = predict_class(image_path, model, device)
train_data_dir = "/content/drive/MyDrive/Colab Notebooks/MoscowIC_project/car_data/train196"
classes = ImageFolder(train_data_dir).classes
predicted_class_name = classes[predicted_class_index]
print("Predicted class:", predicted_class_name)

# Отправляем изображение и выводим его
from IPython.display import Image
width = 400
height = 350
display(Image(filename=image_path, width=width, height=height))

if __name__ == '__main__':
    main()

INFO:pytorch_lightning.utilities.migration.utils:Lightning automatically upgraded your loaded checkpoint from v1.2.0 to v2.2.4.
Loaded pretrained weights for efficientnet-b5
Predicted class: BMW 3 Series Sedan 2012
```

Рис. 15: Демонстрация работы модели EfficientNet на датасете из 196 марок

Рис. 16: Демонстрация работы модели EfficientNet на датасете из 246 марок

Все две модели справились.

Усложним задачу. Возьмем фотографию из auto.ru. И возьмем год 2022

```
checkpoint_path = "/content/drive/MyDrive/Colab Notebooks/МорозовДС_проект/model_246_cars_epoch=16_val_acc=0.8423.ckpt"
model = EfficientNet.from_checkpoint(checkpoint_path, num_classes=246)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)

checkpoint = torch.load(checkpoint_path, map_location=device)
intersecting_keys = set(model.state_dict().keys()) & set(checkpoint['state_dict'].keys())
state_dict = {k: v for k, v in checkpoint['state_dict'].items() if k in intersecting_keys}
model.load_state_dict(state_dict)

image_path = "/content/drive/MyDrive/Colab Notebooks/МорозовДС_проект/тест модели/3.jpg"
predicted_class_index = predict_class(image_path, model, device)
print("Predicted class:", predicted_class_index)
classes = ImageFolder(train_data_dir).classes
predicted_class_name = classes[predicted_class_index]
print("Predicted class:", predicted_class_name)

# Открываем изображение и выводим его
from IPython.display import Image
width = 400
height = 350
display(Image(filename=image_path, width=width, height=height))

if __name__ == '__main__':
    main()
```

INFO:pytorch_lightning.utilities.migration.utils:Lightning automatically upgraded your loaded checkpoint from v1.2.0 to v2.2.0
Loaded pretrained weights for efficientnet-b5
Predicted class: BMW X6 SUV 2012



Рис. 17: Демонстрация работы модели EfficientNet на датасете из 196 марок

```
checkpoint_path = "/content/drive/MyDrive/Colab Notebooks/МорозовДС_проект/model_196_cars_epoch=13_val_acc=0.9328.ckpt"
model = EfficientNet.from_checkpoint(checkpoint_path, num_classes=196)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)

checkpoint = torch.load(checkpoint_path, map_location=device)
intersecting_keys = set(model.state_dict().keys()) & set(checkpoint['state_dict'].keys())
state_dict = {k: v for k, v in checkpoint['state_dict'].items() if k in intersecting_keys}
model.load_state_dict(state_dict)

image_path = "/content/drive/MyDrive/Colab Notebooks/МорозовДС_проект/тест модели/3.jpg"
train_data_dir = "/content/drive/MyDrive/Colab Notebooks/МорозовДС_проект/car_data/train196"
classes = ImageFolder(train_data_dir).classes
predicted_class_name = classes[predicted_class_index]
print("Predicted class:", predicted_class_name)

# Открываем изображение и выводим его
from IPython.display import Image
width = 400
height = 350
display(Image(filename=image_path, width=width, height=height))

if __name__ == '__main__':
    main()
```

INFO:pytorch_lightning.utilities.migration.utils:Lightning automatically upgraded your loaded checkpoint from v1.2.0 to v2.2.0
Loaded pretrained weights for efficientnet-b5
Predicted class: BMW M3 Coupe 2012



Рис. 18: Демонстрация работы модели EfficientNet на датасете из 246 марок

Здесь уже видим, что модель, у которой точность хоть и выше выдала не верный предикт, но тем временем модель с большим количеством марок и меньшей точностью справилась правильно.

Теперь можно попробовать сфоткать автомобиль на улице и загрузить

```

checkpoint_path = "/content/drive/MyDrive/Colab Notebooks/Морозов\С_проект/model_196_cars-epoch=13-val_acc=0.9328.ckpt"
model = EffNet.load_from_checkpoint(checkpoint_path, num_classes=196)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
checkpoint = torch.load(checkpoint_path, map_location=device)
intersecting_keys = set(model.state_dict().keys()) & set(checkpoint['state_dict'].keys())
state_dict = {k: v for k, v in checkpoint['state_dict'].items() if k in intersecting_keys}
model.load_state_dict(state_dict)
image_path = "/content/drive/MyDrive/Colab Notebooks/Морозов\С_проект/тест модели/4.jpg"
predicted_class_index = predict_class(image_path, model, device)
train_data_dir = "/content/drive/MyDrive/Colab Notebooks/Морозов\С_проект/car_data/train196"
classes = ImageFolder(train_data_dir).classes
predicted_class_name = classes[predicted_class_index]
print("Predicted class:", predicted_class_name)

# Открываем изображение и выводим его
from IPython.display import Image
width = 400
height = 350
display(Image(filename=image_path, width=width, height=height))

if __name__ == '__main__':
    main()

```

INFO:pytorch_lightning.utilities.migration.utils:Lightning automatically upgraded your loaded checkpoint from v1.2.0 to v2.0
Loaded pretrained weights for efficientnet-b5
Predicted class: MINI Cooper Roadster Convertible 2012

Рис. 19: Демонстрация работы модели EfficientNet на датасете из 196 марок

```

checkpoint_path = "/content/drive/MyDrive/Colab Notebooks/Морозов\С_проект//model_246_cars-epoch=16-val_acc=0.8423.ckpt"
model = EffNet.load_from_checkpoint(checkpoint_path, num_classes=246)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
checkpoint = torch.load(checkpoint_path, map_location=device)
intersecting_keys = set(model.state_dict().keys()) & set(checkpoint['state_dict'].keys())
state_dict = {k: v for k, v in checkpoint['state_dict'].items() if k in intersecting_keys}
model.load_state_dict(state_dict)
image_path = "/content/drive/MyDrive/Colab Notebooks/Морозов\С_проект/тест модели/4.jpg"
predicted_class_index = predict_class(image_path, model, device)
train_data_dir = "/content/drive/MyDrive/Colab Notebooks/Морозов\С_проект/car_data/train"
classes = ImageFolder(train_data_dir).classes
predicted_class_name = classes[predicted_class_index]
print("Predicted class:", predicted_class_name)

# Открываем изображение и выводим его
from IPython.display import Image
width = 400
height = 350
display(Image(filename=image_path, width=width, height=height))

if __name__ == '__main__':
    main()

```

INFO:pytorch_lightning.utilities.migration.utils:Lightning automatically upgraded your loaded checkpoint from v1.2.0 to v2.0
Loaded pretrained weights for efficientnet-b5
Predicted class: MINI Cooper Roadster Convertible 2012

Рис. 20: Демонстрация работы модели EfficientNet на датасете из 246 марок

Все две модели справились отлично.

Попробуем сфоткать для модели более сложный кадр.

```

checkpoint_path = "/content/drive/MyDrive/Colab Notebooks/Морозов\С_проект//model_246_cars-epoch=16-val_acc=0.8423.ckpt"
model = EffNet.load_from_checkpoint(checkpoint_path, num_classes=246)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
checkpoint = torch.load(checkpoint_path, map_location=device)
intersecting_keys = set(model.state_dict().keys()) & set(checkpoint['state_dict'].keys())
state_dict = {k: v for k, v in checkpoint['state_dict'].items() if k in intersecting_keys}
model.load_state_dict(state_dict)
image_path = "/content/drive/MyDrive/Colab Notebooks/Морозов\С_проект/тест модели/12.jpg"
predicted_class_index = predict_class(image_path, model, device)
train_data_dir = "/content/drive/MyDrive/Colab Notebooks/Морозов\С_проект/car_data/train"
classes = ImageFolder(train_data_dir).classes
predicted_class_name = classes[predicted_class_index]
print("Predicted class:", predicted_class_name)

# Открываем изображение и выводим его
from IPython.display import Image
width = 400
height = 350
display(Image(filename=image_path, width=width, height=height))

if __name__ == '__main__':
    main()

```

INFO:pytorch_lightning.utilities.migration.utils:Lightning automatically upgraded your loaded checkpoint from v1.2.0 to v2.0
Loaded pretrained weights for efficientnet-b5
Predicted class: Suzuki Kizashi Sedan 2012

Рис. 21: Демонстрация работы модели EfficientNet на датасете из 196 марок

```

checkpoint_path = "/content/drive/MyDrive/Colab Notebooks/Морозов\С_проект/model_196_cars-epoch=13-val_acc=0.9328.ckpt"
model = EffNet.load_from_checkpoint(checkpoint_path, num_classes=196)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
checkpoint = torch.load(checkpoint_path, map_location=device)
intersecting_keys = set(model.state_dict().keys()) & set(checkpoint['state_dict'].keys())
state_dict = {k: v for k, v in checkpoint['state_dict'].items() if k in intersecting_keys}
model.load_state_dict(state_dict)
image_path = "/content/drive/MyDrive/Colab Notebooks/Морозов\С_проект/тест модели/12.jpg"
predicted_class_index = predict_class(image_path, model, device)
train_data_dir = "/content/drive/MyDrive/Colab Notebooks/Морозов\С_проект/car_data/train196"
classes = ImageFolder(train_data_dir).classes
predicted_class_name = classes[predicted_class_index]
print("Predicted class:", predicted_class_name)

# Открываем изображение и выводим его
from IPython.display import Image
width = 400
height = 350
display(Image(filename=image_path, width=width, height=height))

if __name__ == '__main__':
    main()

```

INFO:pytorch_lightning.utilities.migration.utils:Lightning automatically upgraded your loaded checkpoint from v1.2.0 to v2.0
Loaded pretrained weights for efficientnet-b5
Predicted class: Hyundai Veracruz SUV 2012

Рис. 22: Демонстрация работы модели EfficientNet на датасете из 246 марок

Мы можем заметить, что модель с 93% точностью правильно предиктовала название автомобиля в то время, как модель с более большим датасетом сделала это не корректно.

Вот еще несколько тестов:

```
[42] checkpoint_path = "/content/drive/MyDrive/Colab Notebooks/MoposonIC_проект//model_246_cars_epoch=16-val_acc=0.8423.ckpt"
model = EfficientNet.load_From_checkpoint(checkpoint_path, num_classes=246)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
checkpoint = torch.load(checkpoint_path, map_location=device)
intersecting_keys = set(model.state_dict().keys()) & set(checkpoint['state_dict'].keys())
state_dict = {k: v for k, v in checkpoint['state_dict'].items() if k in intersecting_keys}
model.load_state_dict(state_dict)
image_path = "/content/drive/MyDrive/Colab Notebooks/MoposonIC_проект//rect_модели/34.jpg"
predicted_class_index = predict_class(image_path, model, device)
train_data_dir = '/content/drive/MyDrive/Colab Notebooks/MoposonIC_проект//car_data/train'
classes = ImageFolder(train_data_dir).classes
predicted_class_name = classes[predicted_class_index]
print("Predicted class: ", predicted_class_name)

# Очищаем кэш памяти и выведем его
from IPython.display import Image
width = 400
height = 350
display(Image(filename=image_path, width=width, height=height))

if __name__ == '__main__':
    main()

INFO:pytorch_lightning.utilities.migration.utils:Lightning automatically upgraded your loaded checkpoint from v1.2.0 to v2.2
Loaded pretrained weights for efficientnet-b5
Predicted class: Toyota

```

Рис. 23: Демонстрация работы модели EfficientNet на датасете из 196 марок

```
[43] checkpoint_path = "/content/drive/MyDrive/Colab Notebooks/MoposonIC_проект//model_196_cars_epoch=13-val_acc=0.9328.ckpt"
model = EfficientNet.load_From_checkpoint(checkpoint_path, num_classes=196)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
checkpoint = torch.load(checkpoint_path, map_location=device)
intersecting_keys = set(model.state_dict().keys()) & set(checkpoint['state_dict'].keys())
state_dict = {k: v for k, v in checkpoint['state_dict'].items() if k in intersecting_keys}
model.load_state_dict(state_dict)
image_path = "/content/drive/MyDrive/Colab Notebooks/MoposonIC_проект//rect_модели/34.jpg"
predicted_class_index = predict_class(image_path, model, device)
train_data_dir = '/content/drive/MyDrive/Colab Notebooks/MoposonIC_проект//car_data/train196'
classes = ImageFolder(train_data_dir).classes
predicted_class_name = classes[predicted_class_index]
print("Predicted class: ", predicted_class_name)

# Отправляем изображение и выводим его
from IPython.display import Image
width = 400
height = 350
display(Image(filename=image_path, width=width, height=height))

if __name__ == '__main__':
    main()

INFO:pytorch_lightning.utilities.migration.utils:Lightning automatically upgraded your loaded checkpoint from v1.2.0 to v2.2
Loaded pretrained weights for efficientnet-b5
Predicted class: Dodge Journey SUV 2012

```

Рис. 24: Демонстрация работы модели EfficientNet на датасете из 246 марок

```
[44] checkpoint_path = "/content/drive/MyDrive/Colab Notebooks/MoposonIC_проект//model_246_cars_epoch=16-val_acc=0.8423.ckpt"
model = EfficientNet.load_From_checkpoint(checkpoint_path, num_classes=246)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
checkpoint = torch.load(checkpoint_path, map_location=device)
intersecting_keys = set(model.state_dict().keys()) & set(checkpoint['state_dict'].keys())
state_dict = {k: v for k, v in checkpoint['state_dict'].items() if k in intersecting_keys}
model.load_state_dict(state_dict)
image_path = "/content/drive/MyDrive/Colab Notebooks/MoposonIC_проект//rect_модели/28.jpg"
predicted_class_index = predict_class(image_path, model, device)
train_data_dir = '/content/drive/MyDrive/Colab Notebooks/MoposonIC_проект//car_data/train'
classes = ImageFolder(train_data_dir).classes
predicted_class_name = classes[predicted_class_index]
print("Predicted class: ", predicted_class_name)

# Очищаем кэш памяти и выведем его
from IPython.display import Image
width = 400
height = 350
display(Image(filename=image_path, width=width, height=height))

if __name__ == '__main__':
    main()

INFO:pytorch_lightning.utilities.migration.utils:Lightning automatically upgraded your loaded checkpoint from v1.2.0 to v2.2
Loaded pretrained weights for efficientnet-b5
Predicted class: Maserati

```

Рис. 25: Демонстрация работы модели EfficientNet на датасете из 196 марок

```
[45] checkpoint_path = "/content/drive/MyDrive/Colab Notebooks/MoposonIC_проект//model_196_cars_epoch=13-val_acc=0.9328.ckpt"
model = EfficientNet.load_From_checkpoint(checkpoint_path, num_classes=196)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
checkpoint = torch.load(checkpoint_path, map_location=device)
intersecting_keys = set(model.state_dict().keys()) & set(checkpoint['state_dict'].keys())
state_dict = {k: v for k, v in checkpoint['state_dict'].items() if k in intersecting_keys}
model.load_state_dict(state_dict)
image_path = "/content/drive/MyDrive/Colab Notebooks/MoposonIC_проект//rect_модели/28.jpg"
predicted_class_index = predict_class(image_path, model, device)
train_data_dir = '/content/drive/MyDrive/Colab Notebooks/MoposonIC_проект//car_data/train196'
classes = ImageFolder(train_data_dir).classes
predicted_class_name = classes[predicted_class_index]
print("Predicted class: ", predicted_class_name)

# Отправляем изображение и выводим его
from IPython.display import Image
width = 400
height = 350
display(Image(filename=image_path, width=width, height=height))

if __name__ == '__main__':
    main()

INFO:pytorch_lightning.utilities.migration.utils:Lightning automatically upgraded your loaded checkpoint from v1.2.0 to v2.2
Loaded pretrained weights for efficientnet-b5
Predicted class: BMW M6 Convertible 2018

```

Рис. 26: Демонстрация работы модели EfficientNet на датасете из 246 марок

```

checkpoint_path = "/content/drive/MyDrive/Colab Notebooks/МорозовDC_проект/model_246_cars-epoch=16-val_acc=0.8423.ckpt"
model = EffNet.load_from_checkpoint(checkpoint_path, num_classes=246)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
checkpoint = torch.load(checkpoint_path, map_location=device)
intersecting_keys = set(model.state_dict().keys()) & set(checkpoint['state_dict'].keys())
state_dict = {k: v for k, v in checkpoint['state_dict'].items() if k in intersecting_keys}
model.load_state_dict(state_dict)
image_path = "/content/drive/MyDrive/Colab Notebooks/МорозовDC_проект/тест модели/14.jpg"
predicted_class_index = predict_class(image_path, model, device)
train_data_dir = "/content/drive/MyDrive/Colab Notebooks/МорозовDC_проект/car_data/train"
classes = ImageFolder(train_data_dir).classes
predicted_class_name = classes[predicted_class_index]
print("Predicted class:", predicted_class_name)

# Открываем изображение и выводим его
from IPython.display import Image
width = 400
height = 350
display(Image(filename=image_path, width=width, height=height))

if __name__ == '__main__':
    main()

```

INFO:pytorch_lightning.utilities.migration.utils:Lightning automatically upgraded your loaded checkpoint from v1.2.0 to v2.0
Loaded pretrained weights for efficientnet-b5
Predicted class: Mazda

Рис. 27: Демонстрация работы модели EfficientNet на датасете из 196 марок

```

checkpoint_path = "/content/drive/MyDrive/Colab Notebooks/МорозовDC_проект/model_196_cars-epoch=13-val_acc=0.9328.ckpt"
model = EffNet.load_from_checkpoint(checkpoint_path, num_classes=196)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
checkpoint = torch.load(checkpoint_path, map_location=device)
intersecting_keys = set(model.state_dict().keys()) & set(checkpoint['state_dict'].keys())
state_dict = {k: v for k, v in checkpoint['state_dict'].items() if k in intersecting_keys}
model.load_state_dict(state_dict)
image_path = "/content/drive/MyDrive/Colab Notebooks/МорозовDC_проект/тест модели/14.jpg"
predicted_class_index = predict_class(image_path, model, device)
train_data_dir = "/content/drive/MyDrive/Colab Notebooks/МорозовDC_проект/car_data/train196"
classes = ImageFolder(train_data_dir).classes
predicted_class_name = classes[predicted_class_index]
print("Predicted class:", predicted_class_name)

# Открываем изображение и выводим его
from IPython.display import Image
width = 400
height = 350
display(Image(filename=image_path, width=width, height=height))

if __name__ == '__main__':
    main()

```

INFO:pytorch_lightning.utilities.migration.utils:Lightning automatically upgraded your loaded checkpoint from v1.2.0 to v2.0
Loaded pretrained weights for efficientnet-b5
Predicted class: BMW X3 SUV 2012

Рис. 28: Демонстрация работы модели EfficientNet на датасете из 196 марок

```

checkpoint_path = "/content/drive/MyDrive/Colab Notebooks/МорозовDC_проект/model_246_cars-epoch=16-val_acc=0.8423.ckpt"
model = EffNet.load_from_checkpoint(checkpoint_path, num_classes=246)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
checkpoint = torch.load(checkpoint_path, map_location=device)
intersecting_keys = set(model.state_dict().keys()) & set(checkpoint['state_dict'].keys())
state_dict = {k: v for k, v in checkpoint['state_dict'].items() if k in intersecting_keys}
model.load_state_dict(state_dict)
image_path = "/content/drive/MyDrive/Colab Notebooks/МорозовDC_проект/тест модели/18.jpg"
predicted_class_index = predict_class(image_path, model, device)
train_data_dir = "/content/drive/MyDrive/Colab Notebooks/МорозовDC_проект/car_data/train"
classes = ImageFolder(train_data_dir).classes
predicted_class_name = classes[predicted_class_index]
print("Predicted class:", predicted_class_name)

# Открываем изображение и выводим его
from IPython.display import Image
width = 400
height = 350
display(Image(filename=image_path, width=width, height=height))

if __name__ == '__main__':
    main()

```

INFO:pytorch_lightning.utilities.migration.utils:Lightning automatically upgraded your loaded checkpoint from v1.2.0 to v2.0
Loaded pretrained weights for efficientnet-b5
Predicted class: Rolls-Royce Ghost Sedan 2012

Рис. 29: Демонстрация работы модели EfficientNet на датасете из 196 марок

```

checkpoint_path = "/content/drive/MyDrive/Colab Notebooks/МорозовDC_проект/model_196_cars-epoch=13-val_acc=0.9328.ckpt"
model = EffNet.load_from_checkpoint(checkpoint_path, num_classes=196)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
checkpoint = torch.load(checkpoint_path, map_location=device)
intersecting_keys = set(model.state_dict().keys()) & set(checkpoint['state_dict'].keys())
state_dict = {k: v for k, v in checkpoint['state_dict'].items() if k in intersecting_keys}
model.load_state_dict(state_dict)
image_path = "/content/drive/MyDrive/Colab Notebooks/МорозовDC_проект/тест модели/18.jpg"
predicted_class_index = predict_class(image_path, model, device)
train_data_dir = "/content/drive/MyDrive/Colab Notebooks/МорозовDC_проект/car_data/train196"
classes = ImageFolder(train_data_dir).classes
predicted_class_name = classes[predicted_class_index]
print("Predicted class:", predicted_class_name)

# Открываем изображение и выводим его
from IPython.display import Image
width = 400
height = 350
display(Image(filename=image_path, width=width, height=height))

if __name__ == '__main__':
    main()

```

INFO:pytorch_lightning.utilities.migration.utils:Lightning automatically upgraded your loaded checkpoint from v1.2.0 to v2.0
Loaded pretrained weights for efficientnet-b5
Predicted class: Dodge Charger Sedan 2012

Рис. 30: Демонстрация работы модели EfficientNet на датасете из 246 марок

Из этих тестов мы можем сделать вывод, что датасет влияет на предиктовку больше, чем точность модели.

9 Заключение

Проект показал, что глубокие нейронные сети могут эффективно решать задачу распознавания марок автомобилей по фотографиям. Достигнутые результаты подтверждают правильность выбранного подхода, однако есть возможности для дальнейшего улучшения модели.

В ходе проекта была разработана и обучена модель нейронной сети для классификации изображений автомобилей. Дальнейшие исследования и улучшения могут включать более сложные методы аугментации данных, более новую архитектуру модели такие как EfficientNetB7 настройку гиперпараметров и расширение датасета для улучшения производительности модели.

Список использованных источников

1. EfficientNet-B5. — <https://paperswithcode.com/paper/high-performance-large-scale-image/review/>.
2. Eliminate training loops with PyTorch Lightning. — <https://coderzcolumn.com/tutorials/artificial-intelligence/pytorch-lightning-eliminate-training-loops>.
3. Getting started with PyTorch. — <https://pytorch.org/get-started/previous-versions>.
4. GitHub Repository for AI Experiments. — <https://github.com/krasnoteh/AI-experiments>.
5. Introduction to Weights and Biases. — <https://medium.com/polimi-data-scientists/introduction-to-weights-biases-72ba61220523>.
6. *Mooney P. T.* Stanford Cars Dataset with Fastai v1. — <https://www.kaggle.com/code/paultimothymooney/stanford-cars-dataset-with-fastai-v1/input>.
7. OpenAI ChatGPT. — <https://chatgpt.com>.
8. PyTorch Hub - ResNet50. — <https://pytorch.org/vision/main/models/generated/torchvision.models.resnet50.html>.
9. Torchvision Models - ResNet152. — <https://pytorch.org/vision/main/models/generated/torchvision.models.resnet152.html>.
10. Weights and Biases. — <https://wandb.ai>.
11. Weights and Biases YouTube Playlist. — <https://www.youtube.com/playlist?list=PLD80i8An1OEGajeVo15ohAQYF1Ttle0lk>.
12. YouTube Channel for AI Experiments. — <https://www.youtube.com/@user-gj9lg4fg4d/featured>.
13. YouTube video on GPU usage in PyTorch. — <https://www.youtube.com/watch?v=zuRCXIjiSB4>.
14. YouTube Video on Image Classification. — <https://www.youtube.com/watch?v=1quKqf47V6s>.