Курсовая работа

Актуальность

Нейронные сети широко применяются в системах, работающих в режиме реального времени: техническое зрение, робототехника, интеллектуальное видеонаблюдение, человеко-машинные интерфейсы. Для таких приложений важны не только точность предсказаний, но и гарантированные временные характеристики. При внедрении обученной модели в прикладную систему возникает задача её эффективного развертывания и оптимизации вывода под целевую аппаратную платформу. Классические фреймворки (например, PyTorch) удобны для разработки и обучения, однако в задачах развертывания требуется использование специализированных средств оптимизации вычислительного графа и исполнения на GPU. В экосистеме NVIDIA ключевую роль играет TensorRT, а также новый высокоуровневый фронтенд TriPy, ориентированный на производительный вывод и тесную интеграцию с TensorRT. Сравнение подходов и анализ компромисса “точность–быстродействие” позволяют сформировать практические рекомендации для задач реального времени.

Цель и задачи

Изучение технологий развертывания свёрточных нейронных сетей для приложений реального времени с использованием PyTorch и TriPy, а также сравнение точности и быстродействия моделей после экспорта в формат TensorRT.

1. Рассмотреть требования систем реального времени к выполнению вывода нейронных сетей;
2. Изучить архитектуру LeNet и адаптировать её для набора данных CIFAR-10;
3. Реализовать модель в PyTorch и TriPy и обеспечить эквивалентность вычислений;
4. Обучить модель на CIFAR-10;
5. Экспортировать реализацию в TensorRT (через PyTorch и через TriPy);
6. Сравнить точность и быстродействие.

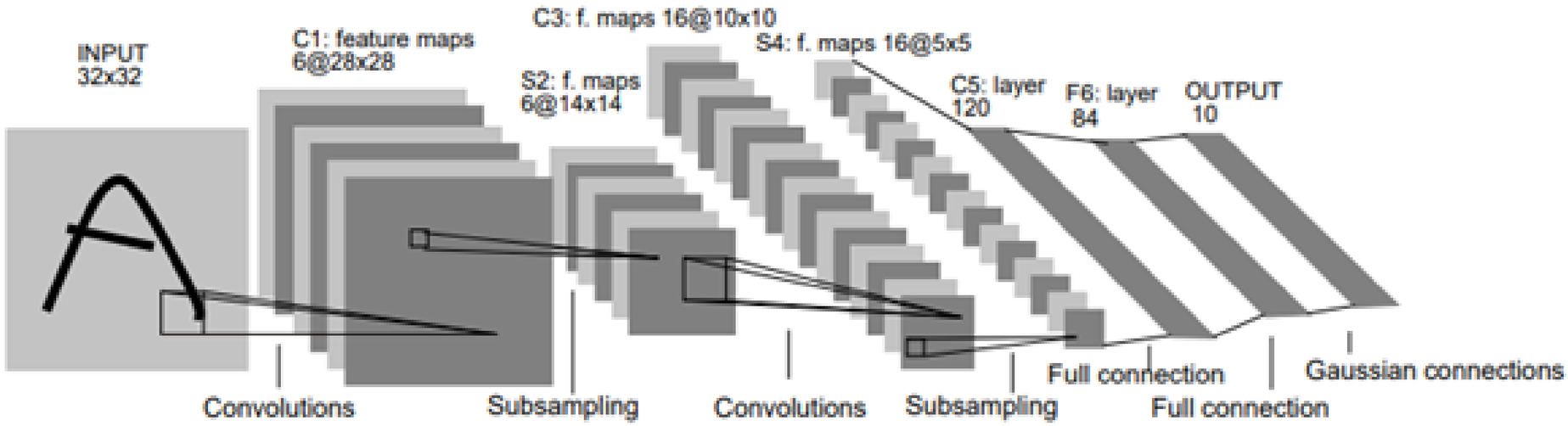
Изм.	Лист	№ докум.	Подп.	Дата	Курсовая работа		
Разраб.		Ефремов С.А.			Цель и задачи	Лит.	Масса / Масштаб
Пров.		Лычков И.И.				Лист 1	Листов 5
Т.контр.						МГТУ им. Н.Э. Баумана группа ИУ3-11М	
Н.контр.							
Утв.							

Перв. примен. | Справ. № | Подп. и дата | Инв. № дубл. | Взам. инв. № | Подп. и дата | Инв. № подл.

Копировал | Формат А3

Архитектура выбранной нейронной сети

Стандартная структура сети LeNet



Структура сети LeNet для CIFAR-10

Вход: 3×32×32 (RGB).

- Conv(3→6, k=5×5) → ReLU → MaxPool(2×2, s=2)
- Conv(6→16, k=5×5) → ReLU → MaxPool(2×2, s=2)
- Flatten(16×5×5 = 400)
- FC(400→120) → ReLU
- FC(120→84) → ReLU
- FC(84→10) (логиты классов)

В качестве базовой модели выбрана свёрточная нейронная сеть LeNet, адаптированная под изображения CIFAR-10 (цветные RGB 3×32×32, 10 классов). Архитектура является компактной (порядка 6.2×10^4 параметров), вычислительно умеренной и хорошо подходит для экспериментов по развертыванию и ускорению инференса, поскольку обеспечивает предсказуемую структуру вычислительного графа и небольшие требования к памяти.

Изм.	Лист	№ докум.	Подп.	Дата	Курсовая работа		
Разраб.		Ефремов С.А.			Архитектура LeNet	Лит.	Масса / Масштаб
Пров.		Лычков И.И.				Лист 2	Листов 5
Т.контр.						МГТУ им. Н.Э. Баумана группа ИУ3-11М	
Н.контр.							
Утв.							

Копировал

Формат А3

Перв. примен.

Справ. №

Подп. и дата

Инв. № дубл.

Взам. инв. №

Подп. и дата

Инв. № подл.

Инв. № подл. | Подп. и дата | Взам. инв. № | Инв. № дубл. | Подп. и дата | Справ. №

Перв. примен.

Копировал

Формат А3

Подготовка данных

Для обучающей выборки применялась аугментация: случайное кадрирование 32×32 с padding=4 и случайное горизонтальное отражение.
Далее выполнялись преобразование в тензор и нормализация по каналам RGB. Для тестовой выборки применялись только преобразование в тензор и нормализация.

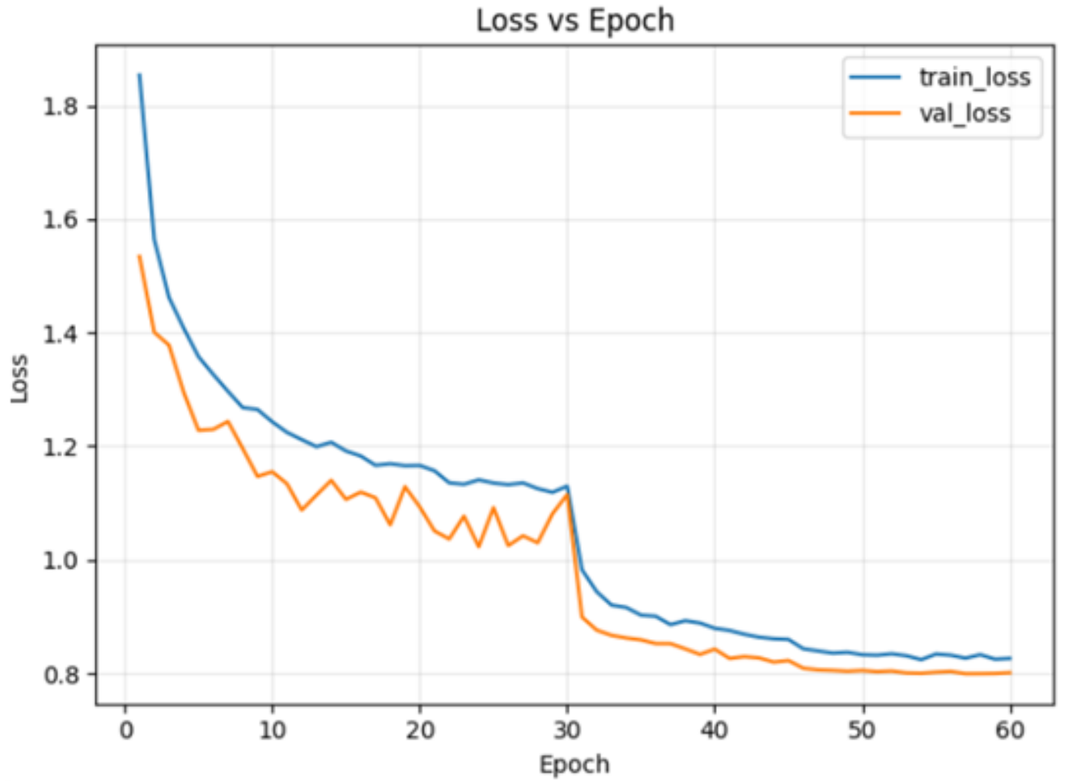


Рисунок 1 — График изменения средних потерь

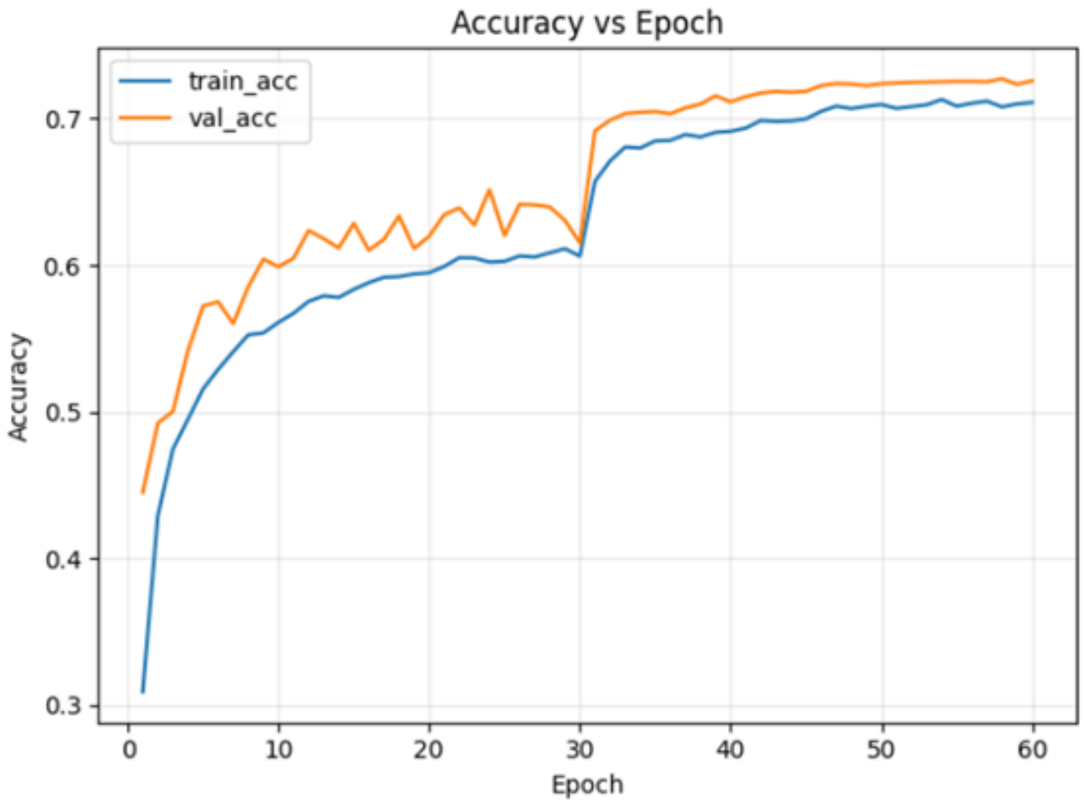


Рисунок 2 — График изменения точности

Гиперпараметры обучения

Размер батча: 200.
Оптимизатор: SGD (learning rate 0.05, momentum 0.9, weight decay 5e-4).
План изменения learning rate: уменьшение в 10 раз после 30-й и 45-й эпох.
Число эпох: 60.

Изм.	Лист	№ докум.	Подп.	Дата	Курсовая работа		
Разраб.		Ефремов С.А.			Реализация PyTorch	Лит.	Масса / Масштаб
Пров.		Лычков И.И.				Лист 3	Листов 5
Т.контр.						МГТУ им. Н.Э. Баумана группа ИУ3-11М	
Н.контр.							
Утв.							

Реализация в TriPy

TriPy используется как высокоуровневый фронтенд, ориентированный на производительный вывод и тесно интегрированный с TensorRT. Архитектура TriPy-модели структурно повторяет PyTorch-вариант LeNet: два свёрточно-пулинговых блока и полносвязная часть классификатора.

Перенос параметров из PyTorch

```
def convert_state_torch_to_tripy(torch_state):
    tripy_state = {}
    for name, param in torch_state.items():
        np_value = param.detach().cpu().numpy().astype("float32")
        tripy_state[name] = tp.Tensor(np_value)
    return tripy_state
```

Для обеспечения сопоставимости результатов используются одни и те же обученные веса. Параметры PyTorch-модели конвертируются в формат TriPy (приведение к float32 и загрузка в tp.Tensor), затем загружаются в TriPy-модель. Такой перенос позволяет получить функционально эквивалентную реализацию для дальнейшего сравнения точности и скорости.

Сравнение результатов

PyTorch: loss = 0.7994318, accuracy = 0.7270;
TriPy: loss = 0.7994541, accuracy = 0.7269.

Незначительное расхождение носит ожидаемый характер и обусловлено особенностями численной реализации вычислений и преобразований данных при переходе между средами выполнения (округления, порядок операций с плавающей запятой). В целом результаты подтверждают, что TriPy-реализация воспроизводит качество PyTorch-модели с практически полным совпадением метрик.

Изм	Лист	№ докум.	Подп.	Дата	Курсовая работа		
Разраб.		Ефремов С.А.			Реализация TriPy	Лит.	Масса / Масштаб
Пров.		Лычков И.И.				Лист 4	Листов 5
Т.контр.						МГТУ им. Н.Э. Баумана группа ИУ3-11М	
Н.контр.							
Утв.							

Копировал Формат А3

Перв. примен.

Справ. №

Подп. и дата

Инв. № дубл.

Взам. инв. №

Подп. и дата

Инв. № подл.

Изм. Лист № докум. Подп. Дата

Копировал

Формат А3

Курсовая работа

Сравнение быстродействия

Лит. Масса Масштаб

Лист 5 Листов 5

МГТУ им. Н.Э. Баумана группа ИУ3-11М

Разраб. Ефремов С.А.

Пров. Лычков И.И.

Т.контр.

Н.контр.

Утв.

Результаты сравнения быстродействия

Среда выполнения	forward, мс/пакет	forward, мс/изобр.	e2e, мс/пакет	e2e, мс/изобр.
PyTorch	0,814	0,0008	1,702	0,0017
TensorRT	0,652	0,0007	1,620	0,0016
TriPy → TensorRT	0,548	0,0005	1,476	0,0015

Методика измерений производительности

Измеряются два режима:

1. forward-only: входные данные находятся в GPU, учитывается только прямой проход модели;
2. end-to-end: включает копирование CPU→GPU, вычисления и синхронизацию.

Анализ результатов

Оптимизация вычислительной части (forward-only): TensorRT уменьшает время на ≈19.9% относительно PyTorch, а TriPy→TensorRT даёт уменьшение ≈32.7% относительно PyTorch и ≈16.0% относительно TensorRT.

В режиме end-to-end эффект слабее из-за заметной доли времени на передачу данных и синхронизацию: TensorRT уменьшает e2e на ≈4.8% относительно PyTorch, TriPy→TensorRT на ≈13.3% относительно PyTorch.

Условия эксперимента

Тестирование проводилось на CIFAR-10 (10 000 изображений) при одинаковой предобработке. Использовались warm-up 5 батчей и серия из 100 прогонов с усреднением. Оборудование: Ubuntu 24.04, CPU AMD Ryzen 5 7500F, RAM 32 ГБ DDR5, GPU NVIDIA GeForce RTX 4060 Ti 8 ГБ.