Adaptação, Desenvolvimento e Implantação de uma API para Classificação de Qualidade de Carne com Aprendizado de Máquina: Um Relato de Experiência

Vitor Gabriel B. Balsanello
UTFPR
Dois Vizinhos, Brasil
vitorbalsanello@alunos.utfpr.edu.br

Francisco Carlos M. Souza
UTFPR
Dois Vizinhos, Brasil
franciscosouza@utfpr.edu.br

***Abstract*—The implementation of Machine Learning models involves a series of specific characteristics and also faces problems inherent to the area itself. This is often due to the lack of relevant architectural standards or their low specificity. This work seeks to demonstrate experience in implementing *Machine Learning* projects, through the application of modern architectural standards, demonstrating how this type of practice can help in the implementation of projects. As a result, an API integrated into a model was implemented and through this activity, as a secondary result, an analysis of the current literature was carried out, observing a low number of related studies.**

***Keywords*—Designe Patterns; Machine Learning; API.**

***Resumo*—A implantação de modelos de Aprendizado de Maquina envolvem uma série de características específicas e também enfrentam problemas inerentes a própria área. Isso se deve, em grande parte das vezes, a falta de padrões arquiteturais relevantes ou a sua baixa especificidade. Esse trabalho busca demonstrar uma experiência na implantação de projetos de *Machine Learning*, através da aplicação de padrões arquiteturais modernos, demonstrado como esse tipo de prática pode auxiliar na implantação dos projetos. Como resultado foi realizada a implantação de uma API integrada a um modelo e através desta atividade, como um resultado secundário, foi realizada uma análise da literatura vigente, observando-se um número baixo de estudos relacionados.**

***Palavras-chave*—Designe Patterns; Machine Learning; API.**

I. Introdução

Nos últimos anos, o uso de *machine learning* como uma alternativa para solucionar problemas complexos do mundo real tornou-se cada vez mais importante. Esse tipo de aplicação tem apresentado um avanço bastante relevante, graças à evolução que os algoritmos relacionados tiveram, bem como à abundância de dados que apoia esse tipo de projeto. Aplicações focadas na geração procedural de dados, como IA's que geram imagens e artes gráficas, que trabalham com PLN e permitem a geração de texto ou até mesmo de código-fonte, e até apresentações, por exemplo, tornaram-se cada vez mais comuns e conhecidas pelo grande público.

Além disso, modelos de aprendizado de máquina são amplamente aplicados em cálculos de custos relacionados ao transporte, na priorização de tarefas em empresas, em algoritmos de recomendação e na geração procedural de elementos para jogos, por exemplo. Estes últimos estão frequentemente integrados a produtos de software mais abrangentes.

Além disso, como observado por [1], em todas as áreas relacionadas a Engenharia de Software, s projetos se tornaram mais extensos e tecnicamente exigentes, tornando-se, portanto, mais propensos a erros e desafiadores de se manterem estáveis. Projetos de *machine learning* além disso enfrentam uma grande propensão a gerar débito técnico. Essa falha se caracteriza por um acumulado técnico, graças a falta de padrões de projeto e metodologias convenientes, além de escolhas arquiteturais erradas, é possível atribuir esse problema, no contexto atual, a falta de uma literatura relevante sobre o tema, que não gera suporte suficiente para a problemática em questão.

Apesar de práticas arquiteturais, e padrões de projeto se desenvolverem cada vez mais em produtos de Engenharia de Software, a maioria das pessoas envolvidas nas equipes se desenvolve com uma ênfase técnica muito grande, não se desenvolvendo em questões arquiteturais o que contra-intuitivamente gera problemas técnicos [2].

Considerando esses fatores arquiteturais é perceptível que existe um grande problema na disponibilidades de modelos de *machine learning*, porque eles não são realmente mantidos para uso, do mesmo modo, os que são, tendem a se tornar obsoletos se não passarem por um processo de retreinamento e por fim, esse tipo de aplicação fornece um certa dificuldade para a integração por não disponibilizarem uma interface acessível.

Levando esses pontos em consideração, podemos estabelecer que descrever um modelo de arquitetura de *machine learning* é

uma necessidade aparente para esse tipo de projeto, embora seja frequentemente descartada. Observa-se a necessidade de seguir rigorosamente padrões, visando evitar demandas expressivas de manutenibilidade.

O objetivo deste trabalho é conduzir uma experiência arquitetural em um projeto de *machine learning*, mostrando os benefícios desse tipo de desenvolvimento e demonstrando quais as principais características que esses de projetos apresentam. Esses objetivos são representados pela integração e implantação de um modelo de *machine learning* existente, pela aplicação de padrões arquiteturais modernos e bem definidos, pela criação e funcionamento de uma API REST com o objetivo de consumir o modelo citado, e por fim pela implantação da solução completa.

As publicações na área de arquitetura para projetos de *machine learning* ainda são raras, porém, alguns autores apresentaram grandes avanços na temática. Em 2020 [3] publicaram um dos principais livros da área, intitulado "Machine Learning Design Patterns" o livro se estende sobre os princípios introdutórios de conceitos de arquitetura de software para *machine learning*, detalhando os principais aspectos desse campo.

Já em 2022, Software-Engineering Design Patterns for Machine Learning Applications foi publicado por [4] fazendo uma revisão da literatura e analisando oportunidades de melhoria, através de 15 diferentes padrões de projeto para *Machine Learning*.

Em 2020 [5]. Propôs em seu trabalho: A Design Pattern for Deploying Machine Learning Models to Production, um padrão de design para realizar o *deploy* de projetos de aprendizado de máquina. Conforme observado pelo autor em questão, existe uma notável discrepância entre a quantidade de modelos disponíveis e a quantidade efetivamente implementada em produção.

Em 2019, através de um visão predominante de engenharia de software [6] publicou o artigo, Studying Software Engineering Patterns for Designing Machine Learning Systems, que buscava contribuir para sanar a grande falta de documentação arquitetural que orientava o desenvolvimento de projetos de *machine learning* até aquele período.

O artigo está organizado com as seguintes sessões: Contexto, onde uma pequena análise do contexto o qual a problemática surge é abordado, que nesse caso são os numerosos projetos de *machine learning* existentes, trabalhos relacionados, onde uma pequena descrição sobre a literatura corrente do tema é definida, a sessão de proposta que busca exprimir como o estudo será conduzido e a sessão de resultados mostrando qual o comportamento dos dados observados do projeto. E conclusão que resume o artigo e comenta sobre os resultados.

II. Contexto

A. Desenvolvimento de Aplicações de Aprendizado de Máquina

Os modelos típicos de *machine learning* são criados a partir de uma estrutura comum, que em geral provem de ideias de uma área conhecida como *business intelligence*, as etapas para projetos desse tipo são descritas abaixo.

Segundo [7] em seu livro: Designing Machine Learning Systems, pode-se dividir o processo construção de projetos em algumas etapas que são geralmente comuns a maioria dos produtos:

1) Definição do Escopo do Projeto: Nesta fase inicial, o escopo do projeto atual é estabelecido, determinando seus objetivos, conceitos-chave e os principais desafios que o projeto poderá enfrentar. *Stakeholders* precisam estar envolvidos e identificados.
2) Engenharia de Dados: Modelos de *machine learning* são criados a partir do comportamento de dados conhecidos, por tanto em um segundo passo, é necessário definir qual será esse conjunto de dados analisado, depois estes precisam ser delimitados, coletados e tratados.
3) Desenvolvimento do Modelo de *Machine Learning*: Como o grupo definido de dados, é realizada a extração das características e o desenvolvimento dos modelos iniciais, utilizando-se das características mais relevantes. Essa é a etapa do projeto que requer a maior habilidade técnica da equipe, onde algoritmos são selecionados, o modelo é treinado, e também avaliado.
4) *Deployment*: Como o modelo é treinado e avaliado, é necessário colocá-lo à disposição para uso, ou seja, em produção.
5) Monitoramento: Após o modelo estar operacional, é essencial monitorar seu desempenho, observando variações na acurácia, seu comportamento em diferentes contextos e o atendimento a requisitos diversos.
6) Análise de Negócios: A nível de negócio é necessário avaliar se o modelo atinge as metas e objetivos estabelecidos, através dessa prática é possível até mesmo, conceber novas ideias de negócios relacionadas ao projeto.

B. APIs REST e sua Importância

A definição que o livro API Design for C++ concede é: Uma *Application Programming Interface*, ou API provê uma abstração para um problema e específica como aplicações clientes devem interagir com componentes de software que implementam a solução para esse problema [8].

Os componentes são tipicamente distribuídos como uma biblioteca de *software*, permitindo-os serem usados em múltiplas aplicações. Na prática API são blocos construtores reutilizáveis

que permitem partes modulares de funcionalidade serem incorporadas a aplicações.

Já o estilo arquitetural REST, que é umas das ferramentas mais disruptivas para aplicações *Web*, foi definido por Roy Thomas Fielding em sua tese de doutorado e pode ser descrita como um estilo arquitetural para sistemas de hipermídia.

Para uma API esteja definida dentro do estilo arquitetural REST é necessário que ela siga algumas restrições típicas, como descreve resumidamente [9] são elas:

1) Identificação de Recursos: Todos os recursos que são relevantes para aplicação devem ser identificados de maneira estável e única. Esses identificadores devem ser globais, de maneira que eles devem ser diferenciados independentemente do contexto.

2) Interface Uniforme: Todas as interações devem ser construídas através de uma interface uniforme a qual apoia todas as interações com recursos, fornecendo um conjunto de métodos suficientes para a aplicação.

3) Mensagens Meta-Descritiva: Para a interação de recursos com interfaces uniformes, o estilo arquitetural REST exige representações de recursos que representam aspectos importantes desses.

4) Hypermedia Drive Application State: As representações que são trocadas devem ser linkadas de maneira que uma aplicação que entende a representação, vai estar apta a encontrar essa ligação, as partes interessadas tenham um completo entendimento dos recursos ou dos estados relevantes apenas inspecionando suas representações.

5) Interações sem estado definido: Essa definição delimita a interação do cliente e servidor determinando que cada uma delas deve ser completamente contida nessa relação, não havendo "estado do cliente" o que é frequentemente nomeado como sessão, mantido no servidor. Essa definição garante que nem uma interação dependa da troca da representação e da sessão associada ao cliente.

Atualmente API's do tipo REST são fundamentais para a internet e desempenham um papel fundamental na forma como aplicações são criadas, apesar de suas grandes diferenças internas

C. Padrões de Projeto em Desenvolvimento de APIs

Como qualquer produto de software novo o desenvolvimento de API's exige que padrões e estilos arquiteturais convenientes sejam definidos, embora o padrão REST não seja estritamente prescritivo em relação à forma de criar aplicações, ainda é essencial dedicar atenção específica à sua implementação.

Os principais pontos a serem considerados incluem a eficiência com a qual esses projetos podem ser integrados por outras aplicações, que é o objetivo de produtos desse tipo, e a escalabilidade, principalmente em relação ao grande número de usuários que podem fazer requisições ao mesmo tempo. Para alcançar esses objetivos, é necessário seguir alguns padrões comuns.

Esses padrões, também chamados de *design patterns* se dividem em 3 grupos distintos, segundo o livro Use a Cabeça! Padrões de Projeto, escrito por [10] os primeiros se referem a padrões criacionais, o segundo a padrões estruturais e o terceiro padrões comportamentais. Considerando o contexto deste trabalho, serão abordados os padrões criacionais. Existe a possibilidade de classificar alguns que apresentam uma grande importância para o desenvolvimento de software moderno, principalmente para o desenvolvimento de API'S Rest. esses padrões são amplamente utilizados tanto na indústria quando em pesquisas, é possível elencar:

1) Singleton: Esse padrão assegura que uma classe tenha uma única instância, controlando o acesso a um recurso compartilhado. Isso é particularmente útil na criação de instâncias de bases de dados, por exemplo.

2) Abstract Factory: Permite criar famílias de objetos parecidos sem ter que criar uma classe específica para cada um, considerando suas pequenas diferenças, quando um novo objeto parecido mas com diferenças mínimas precisa ser criado ele herda a classe principal.

3) Builder: Padrão necessário quando é preciso criar uma série de objetivos parecidos com pequenas diferenças, normalmente isso geraria várias subclasses, mas com esse padrão classes para cada parte do novo objeto são criadas e conectadas.

4) Factory Method: A ideia do *factory method* é delegar para uma subclasse a criação de um novo objeto, por exemplo: se o código em questão tem a capacidade de criar um objeto do tipo "estudante de biologia", e um outro tipo de estudante for solicitado ao sistema, a subclasse é projetada de modo que seja possível, alterar o tipo do objeto.

5) Prototype: é um padrão de projeto utilizado para alocar objetos da mesma classe sem que seja necessário consultar todos os campos do objeto, isso pode ser um problema já que alguns campos podem estar inacessíveis. A ideia é delegar a clonagem ao próprio objeto que será clonado.

D. Implantação de Modelos de Aprendizado de Máquina

O processo de implantação de modelos de *machine learning* é bastante linear. Após o treinamento e teste do modelo criado, ele precisa ser colocado em produção para ser utilizado, em geral essa tarefa pode ser realizada utilizando uma API REST como é o caso deste trabalho. Para isso, o modelo precisa ser salvo através alguma ferramenta específica, em Python está disponível a biblioteca pickle para esse fim, com uma chamada simples é possível gerar um arquivo com o modelo de classificação já salvo.

Após essa etapa, a utilização do modelo pode ser realizada importando o módulo por meio do *framework* Flask. Dessa forma, o modelo de classificação pode ser carregado novamente, conforme necessário, utilizando a biblioteca Pickle. Isso pode ser feito dentro da própria API, como demonstrado neste estudo, ou por meio de uma solicitação ao modelo em um ambiente diferente.

Apesar de se tratar de um *pipeline* relativamente simples, os projetos de *machine learning* podem apresentar alguns problemas relativamente comuns, como por exemplo: é necessário avaliar o modelo constantemente de forma que a precisão não decaia ao longo do tempo, e afete os resultados das previsões.

Caso isso aconteça é necessário que o modelo seja treinado novamente e passe por um processo de reavaliação. Esses tipos de inconsistências pode ser corrigidas e ou identificadas através de um conjunto de requerimentos como cita [7], em Designing Machine Learning Systems:

1) Confiabilidade: O software deve se manter com um nível consistente, não apresentando um decaimento nos seus resultados, conforme problemas de hardware ou software, ou mesmo humanos.

2) Escalabilidade: Sistemas que utilizam aprendizado de máquina podem aumentar muito de tamanho, isso em vários aspectos, em primeiro lugar em complexidade, um modelo que começa seu ciclo de vida com 1 GB pode terminá-lo em 10 GB. Em outro ângulo, ele pode crescer em relação ao tráfego, no início o modelo pode conter 10.000 requisições de predição por dia, conforme o tempo passa esse número pode subir para 10.000.000 por exemplo. Além disso, modelos podem crescer de forma que apenas um único, não seja o suficiente para atender a demanda, exigindo que um novo seja treinando, aumentando a especificidade e complexidade do sistema em questão.

3) Manutenibilidade: Considerando que existam vários profissionais diferentes trabalhando em determinados projetos, que possuem níveis diferentes de conhecimento e também diferentes em no que se trata a áreas de atuação é necessário criar uma estrutura de manutenção que permita que esses profissionais consigam se desenvolver de maneira confortável desempenhando bem seus papéis. O código deve ser versionado, documentado e mesmo sem a ajuda dos autores originais o código deve ser suficientemente fácil de ser entendido.

4) Adaptabilidade: O modelo deve ser adaptável a dados e requisitos de negócios, sendo um requisito importante a capacidade de evoluir de forma rápida e contínua.

III. Trabalhos Relacionados

A sessão atual busca apresentar uma série de trabalhos que serviram como base para o trabalho corrente, abordando tanto padrões de projeto relacionados a *machine learning* quanto outras experiências de implantação.

Em 2016 [11], descreveu a necessidade de se definir bons padrões arquiteturais em projetos de software, focando sua análise em WEB SERVICES da arquitetura REST. A pesquisa foi realizada entrevistando 46 profissionais da área de TI, e registrando seus conhecimentos e opiniões sobre padrões arquiteturais.

[12], realizou a implantação de um projeto de arquitetura de API REST, voltado para o monitoramento de redes ópticas, o objetivo do trabalho era implantar uma aplicação robusta e confiável, que tivesse perspectiva de melhoramento futuro.

Em 2015 [13] e uma série de autores, descreveram e sumarizam a evolução de API's relacionadas a *machine learning*, desde o começo dos primeiros projetos da área de ML em 1950, aos dias de hoje. Além disso compararam as problemáticas passadas com as atuais, e a maneira com que a teoria de implementação dessas ferramentas se diferencia da prática.

Em sua dissertação de mestrado [14] propôs uma plataforma para o gerenciamento de modelos de *machine learning* em produção, o que forneceu contribuição para a experimento com o processo de implantação. A tese descreve uma série metodologias necessárias para implantação dos projetos e uma proposta de arquitetura manual.

Em 2012 [15] realizou a implantação de um projeto de *machine learning*, com objetivos médicos, facilitando a pesquisa na busca por questões específicas de saúde, driblando o problema do grande custo de trabalho que esses profissionais tem para sanar suas dúvidas e encontrar soluções.

A. Comparativo

A seguir é apresentada uma pequena tabela comparando os objetivos dos trabalhos relacionados com o atual.

TABELA I
Comparativo

Estudos	Objetivos
[11]	Pesquisa sobre padrões arquiteturais
[12]	Implantação de API
[13]	Sumarização dos projetos de *machine learning*
[14]	Gerenciamento de modelos em produção
[15]	Implantação de modelo
Atual	Implantação de modelo, API, e de padrões arquiteturais

IV. Proposta

A. Refatoração do Script de Classificação de Qualidade de Carne

O ponto principal da pesquisa envolveu a obtenção de um modelo de classificação já criado, extraído da plataforma

Kaggle [16], esse modelo tem o objetivo de classificar imagens de carnes submetidas. O modelo atribuiu duas classes possíveis à foto em questão, *"Spoiled"*, ou seja estragada, ou *"Fresh"*, fresca ou consumível.

Apesar de ser um modelo relativamente simples de classificação, o código fonte escrito em Python, não apresentava um estilo arquitetural bem definido, estava desorganizado e escrito de maneira linear, em forma de *notebook*, como é normalmente fornecido.

B. Refatoração e Organização do Código

O primeiro passo foi organizar o código em funções, os principais métodos criados foram os seguintes:

1) Função de Compilação: Função de tratamento dos dados, nesse caso ajuste do tamanho das imagens fornecidas para o treinamento.

2) Função de Classificação: Essa função ficou responsável por treinar o modelo com as imagens fornecidas através de um *dataset* localizado localmente, o *dataset* foi fornecido na plataforma Kaggle.

3) Função de Exibição da Matriz de Confusão: Função que exibia uma matriz de confusão do modelo. Demonstrando em números absolutos a acurácia do modelo.

4) Função de Exibição dos resultados: Esta função exibe uma matriz 4x4 com os resultados do modelo, apresentando a classificação original da imagem e a classificação obtida pelo modelo.

5) Função de Plotagem: Exibia um gráfico com os resultados de cada geração executada.

A partir da realização da primeira refatoração o código foi modificado em quatro diferentes módulos, levando em conta o ciclo de vida de projetos de *machine learning*. O primeiro deles é o arquivo principal do modelo que apenas fazia a chamada para as demais classes e funções, o segundo arquivo ficou responsável pelo tratamento e compilação dos dados, o terceiro, pela criação do modelo e plotagem dos dados do *dataset* e o último pelo teste do modelo. Além disso, cada um dos arquivos foi delimitado a uma única classe aumentando o nível de desacoplamento.

C. Aplicação de Design Patterns

Foram utilizados três *design patterns* diferentes durante a segunda refatoração. Primeiro a aplicação principal deixou de conhecer os dados sobre o *dataset*, isso facilita a expansão do projeto e a escalabilidade pois somente a classe referente a captação e tratamento dos dados conhece a base em questão *(Adapter Pattern)*.

Em segundo houve a refatoração a nível de tipo de dado. O sistema não depende de um tipo de dado específico para funcionar, ou seja, apesar de utilizar fotos de carnes ele pode ser facilmente adaptado para a classificação de outras imagens desde que a classificação seja binária. Apenas a classe de classificação se importa com qual tipo de dado está sendo trabalhado *(Factory Pattern)*.

No nível de algoritmo, o código foi refatorado para que o algoritmo de classificação fosse conhecido apenas pela classe de classificação, assim nem uma outra parte do código depende de qual estratégia de classificação está sendo utilizada (*Strategy Pattern*). Isso facilita em possíveis futuras mudanças do projeto.

Todas essas estratégias facilitam a escalabilidade do modelo e além de atividades de teste e manutenção, criando modularidade ajudando na integração com outras aplicações caso necessário.

As imagens seguintes mostram a aplicação da refatoração e dos padrões de projeto no carregamento dos dados para o modelo, esse padrão é conhecido como *Adapter Pattern*, a aplicação principal não tem mais conhecimento da base de dados e apenas a classe de carregamento tem responsabilidade sobre ela.

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
import seaborn as sns

sample_size = 500
width = 100
height = 100

files = ['Fresh', 'Spoiled']
adress = 'C:/Users/admin/Desktop/{}'
data = {}
for f in files:
    data[f]=[]
for col in files:
    os.chdir(adress.format(col))
    for i in os.listdir(os.getcwd()):
        if i.endswith('.jpg'):
            data[col].append(i)
```

Fig. 1. Antes da refatoração

D. Criação da API REST

A API funciona de uma forma bastante linear, considerando a arquitetura REST, a API foi desenvolvida através do *Framework* Flask, que é bastante conveniente para esse tipo de aplicação, já que levando em consideração outras tecnologias como Django por exemplo, elas apresentam uma robustez que não é exigida pelo projeto, com a instalação de um série de bibliotecas que não seriam utilizadas durante a execução, além disso o Flask fornece uma sintaxe simples e é fácil de ser instalado e utilizado.

```python
from IPython.display import clear_output

class DataLoader:
    def __init__(self, sample_size=500, width=100, height=100):
        self.sample_size = sample_size
        self.width = width
        self.height = height
        self.files = ['Fresh', 'Spoiled']
        self.adress = 'C:/Users/admin/Desktop/{}'
        self.data = {}
        for f in self.files:
            self.data[f] = []

    def collect_data(self):
        for col in self.files:
            os.chdir(self.adress.format(col))
            for i in os.listdir(os.getcwd()):
                if i.endswith('.jpg'):
                    self.data[col].append(i)
```

Fig. 2. Depois da refatoração

A API apresenta uma única rota: /media/upload essa rota recebe a imagem enviada de um formulário, em uma página principal através do método de envio POST. A API recebe a imagem verificando se o tipo de arquivo corresponde realmente com os de tipo relacionado a imagens, caso o arquivo enviado não corresponda a uma imagem, a API retorna uma mensagem de erro, caso o usuário faço o envio sem nem uma imagem a API também retorna também uma mensagem de erro.

Como a imagem correta enviada, ela é guardada em uma constante e submetida a classificação pelo modelo, quando o modelo faz a análise, o resultado volta na resposta e é informado ao usuário.

E. Deploy e Testes

O *deploy* do modelo foi realizado na plataforma Render, que é um serviço gratuito para esse tipo de aplicação, e que suporta uma série de tecnologias. Para colocar em produção, são necessárias apenas algumas etapas, seguindo os tutoriais da plataforma específicos para cada linguagem de programação, além disso é necessário que os arquivos enviados, além dos criados pelo usuário, tenham também um arquivo .txt especificando as tecnologias utilizadas no projeto.

Dentre as plataformas consideradas para realizar essa etapa, também estavam Fly.io que fornece serviço parecido e a bohr.io que é uma plataforma brasileira e *serveless*.

Durante a realização da refatoração do modelo foram realizados principalmente teste de integração do tipo *big-bang*, esses testes em geral apresentam um custo elevado de tempo, pois é necessário que a geração do modelo seja executada várias vezes, assim elevando o tempo de conclusão do projeto. Porém eles eram necessários, considerando o tamanho do escopo.

Testes de unidade foram realizados nas funções durante a refatoração, mas em geral elas apresentaram um bom desempenho já que, o modelo já funcionava corretamente anteriormente as refatorações.

Durante a criação da API foram realizados testes de unidade, principalmente na função principal de alocação da imagem, essa função também apresentou um bom desempenho logo no início, devido suas características simples. A integração do modelo com a API também não mostrou muitos problemas. Pois a única ação necessária era fornecer a foto enviada ao modelo desserializado com o *Pickle*. A próxima imagem exibe o código da única rota da API

```python
@app.route('/classificacao', methods=['POST'])
def upload_media():
        if 'file' not in request.files:
            return jsonify({'error': 'Media não fornecida'}), 400
        file = request.files['file']
        if file.filename == '':
            return jsonify({'error': 'Nem um arquivo selecionado'}), 400
        if file and allowed_file(file.filename):
            filename = secure_filename(file.filename)
            file.save(os.path.join(app.config['UPLOAD_FOLDER'], filename))

        with open('C:/Users/admin/Desktop/classificacao_de_carne.pkl', 'rb') as f:
            model = pickle.load(f)

        img = cv2.imread(os.path.join(app.config['UPLOAD_FOLDER'], filename))
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        img = cv2.resize(img, (100, 100))
        img_array = image.img_to_array(img)
        img_array = np.expand_dims(img_array, axis=0)
        img_array = img_array / 255.0

        resultado = model.predict(img_array)
        classe_predita = np.argmax(resultado)

        if classe_predita == 1:
            return jsonify({'Resultado da Carne': 'Carne estragada!'})
        elif classe_predita == 0:
            return jsonify({'Resultado da Carne': 'Carne Fresca!'})

        return jsonify({'error': 'Erro ao processar a imagem'}), 400
```

Fig. 3. Rota Principal da API

V. Resultados

A. Refatoração do Código e Aplicação de Design Patterns

O processo de refatoração foi o mais bem sucedido entre as atividades realizadas, pois conseguiu atingir o objetivo de maneira bastante clara, além disso é perceptível o quanto a qualidade do código e a facilidade em que quesitos como manutenção e escalabilidade aumentaram.

Organizar o código em classes permite uma série de benefícios, principalmente na leitura e modificação. Além disso, o modelo se tornou mais flexível e apresenta uma grande possibilidade de expansão e melhoramento.

B. Funcionalidade da API REST

A API se mostrou bastante eficiente na manipulação das imagens para o envio, primeiro devido a sua simplicidade, e o número pequeno de *endpoints*. A API pode retornar quatro respostas distintas: a primeira ocorre quando o arquivo enviado está vazio, a segunda quando o arquivo enviado não corresponde ao tipo específico de imagem, a terceira quando a imagem é enviada corretamente e, por fim, após o processamento do modelo, a classificação da imagem como estragada ou comestível.

C. Implantação e Testes Bem-Sucedidos

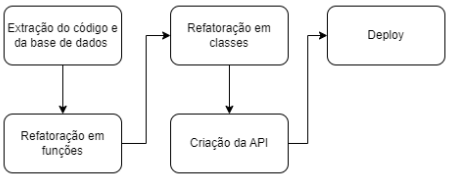

Fig. 4. Objetivos atingidos pelo estudo

A implantação total do modelo com a API, também gerou um resultados interessantes, as plataformas elencadas oferecem uma série de ferramentas para o processo de *deploy* principalmente o Render que foi utilizado no projeto, apesar de possuir algumas limitações na versão gratuita como nesse caso. Adicionalmente, o processo de implantação é ágil, embora seja frequente encontrar alguns erros durante o percurso.

VI. Conclusão

A partir dos estudo dos temas elencados e da análise de projetos parecidos, fica claro a necessidade de se estabelecer bons padrões arquiteturais para projetos de *machine learning* e o quanto eles são importantes para o sucesso do projeto final. Além disso o quanto o processo de refatoração pode ser uma ferramenta útil para o melhoramento de aplicações do tipo.

As características muito próprias desse tipo de projeto, que os diferencia de outros produtos finais de software, exige que exista uma esforço consciente para a criação de estratégias que permitam uma melhor manutenção e escalabilidade.

Todas as ferramentas estudadas foram muito eficientes na realização do projeto, pois conseguiram se enquadrar muito bem no escopo de um modelo simples e experimental. Principalmente o Flask que se mostrou uma ferramenta de fácil aprendizagem e prática. Desenvolver uma API é uma tarefa descomplicada no âmbito do *framework*. Além disso o Python oferece uma ampla abordagem para *machine learning* o que torna a parte mais técnica de implementação dos algoritmos, uma atividade praticamente trivial.

Há uma grande disparidade entre o número de modelos treinados e os efetivamente disponibilizados, o que pode se tornar problemático, uma vez que esses dois cenários são discrepantes. Isso resulta em inexperiência arquitetural, como evidenciado na literatura atual, e, como consequência, pode levar a problemas nos softwares finais ou até mesmo à inviabilidade de implantação real.

Referências

[1] B. W. Boehm, "Machine learning for technical debt identification," p. 15, 1981.
[2] A. C. D. K. A. A. T. A. L. A. Dimitrios Tsoukalas, Nikolaos Mittas, "Machine learning for technical debt identification," p. 15, 2022.
[3] M. M. Valliappa Lakshmanan, Sara Robinson, *Machine Learning Design Patterns*. O'Reilly Media, 2021.
[4] H. Washizaki, "Software engineering patterns for machine learningapplication," *27th Conference on Pattern Languages of Programs*, vol. 55, no. 7, p. 10, 2022.
[5] R. Xu, "A design pattern for deploying machine learning models to production," Master's thesis, California State University San Marcos, California - Estados Unidos, 2020.
[6] H. Washizaki, "Studying software engineering patterns for designing machine learning systems," *10th International Workshop on Empirical Software Engineering in Practice*, p. 6, 2019.
[7] C. Huyen, *Designing Machine Learning Systems*. O'Reilly Media, 2022.
[8] M. Reddy, *API design for C++*. Elsevier, 2011.
[9] E. Wilde, *REST from research to practice*. Springer, 2011.
[10] E. F. Eric Freeman, *Use a cabeça! Padrões de Projeto*. Alta Books, 2022.
[11] L. R. da Silva, "Necessidade de se utilizar boas práticas arquiteturais e padrões de projeto no desenvolvimento de web service baseado na arquitetura restful," p. 70, 2016.
[12] L. C. Jorge, "Projeto e arquitetura de api rest para sistema de monitoramente de des óticas," p. 57, 2020.
[13] J. A. O. P. P. Atakan Cetinsoy, Francisco J. Martin, "The past, present, and future of machine learning apis," *JMLR: Workshop and Conference Proceedings*, p. 7, 2015.
[14] L. C. de Paula e Silva, "Flowi: Uma plataforma para desenvolvimento e gerenciamente de modelos de aprendizado de máquina," Master's thesis, Universidade de São Paulo - USP, São Paulo - Brasil, 2022.
[15] C. E. B. J. L. T. A. T. Byron C. Wallace, Kevin Small, "Deploying an interactive machine learning system in an evidence-based practice center: abstrackr," Master's thesis, California State University San Marcos, Tufts University and Tufts Medical Center, 2015.
[16] O. Ulucan, D. Karakaya, and M. Turkan, "Meat quality assessment based on deep learning," in *2019 Innovations in Intelligent Systems and Applications Conference (ASYU)*. IEEE, 2019, pp. 1–5.