Computer Vision Pipeline
Technical Report

André Kestler
a.kestler@oth-aw.de

Marcus Haberl
m.haberl@oth-aw.de

Tobias Lettner
t.lettner@oth-aw.de

Antonio Vidos
a.vidos@oth-aw.de

Tobias Dobmeier
t.dobmeier@oth-aw.de

Tobias Weiß
t.weiss@oth-aw.de

***Zusammenfassung*—Dieser technical Report beschreibt die Softwarearchitektur des Projekts „Computer Vision Pipeline“. Das Hauptziel der Implementierung besteht darin, dem Benutzer gängige Computer-Vision Funktionen über eine benutzerfreundliche No-Code Anwendung mithilfe einer Cloud-Architektur zugänglich zu machen.**

I. Einführung und Ziele

Die digitale Bildverarbeitung spielt eine entscheidende Rolle bei der automatischen Extraktion von Informationen aus Bildern. Dabei werden verschiedene Schritte wie Filterung und Segmentierung durchgeführt, um die gewünschten Ergebnisse zu erzielen. Diese Schritte müssen für verschiedene Anwendungsfälle getestet und die Parameter angepasst werden. Um diesen Prozess zu vereinfachen, wurde die „Computer Vision Pipeline“ als Cloud-Anwendung entwickelt. Sie ermöglicht es dem Benutzer, Bilder hochzuladen und mit Hilfe einer intuitiven Benutzeroberfläche eine Verarbeitungskette zu erstellen.

Der vorliegende technical Report beschreibt die Architektur der entwickelten „Computer Vision Pipeline“. Dabei werden die allgemeine Architektur, sowie die Bausteinsicht erläutert. Des Weiteren werden die verwendeten Entwicklungswerkzeuge beschrieben. Abschließend wird ein Fazit gezogen und ein Ausblick gegeben.

II. Architektur Allgemein

Die Anwendung basiert auf einer Client-Server-Architektur, bei welcher Frontend und Backend in separaten Docker-Containern laufen. Die Kommunikation zwischen Frontend und Backend erfolgt über eine REST-Schnittstelle, die einen effizienten Informationsaustausch und eine standardisierte Kommunikation ermöglicht. Im Frontend hat der Benutzer die Möglichkeit eine Pipeline aus verschiedenen Bearbeitungsschritten (Steps) zusammenzustellen, die dann im Backend auf ein vom Benutzer hochgeladenes Bild angewendet werden. Der Benutzer hat außerdem die Möglichkeit, das Standard-Bild oder ein zufälliges Katzenbild zu laden. Die zufälligen Katzenbilder werden von einer externen API-Schnittstelle bereitgestellt. Die Anwendung wird in der AWS Cloud gehostet und nutzt einen S3 Bucket zur zuverlässigen Speicherung der (Zwischen-)Ergebnisse. Die Hosting-Funktionalität wird durch eine EC2-Instanz bereitgestellt.

III. Bausteinsicht

A. Frontend: Benutzerschnittstelle

Das in React geschriebene Frontend (siehe Abbildung 3) besteht aus einer Hauptkomponente, welche aus mehreren Kindkomponenten besteht (siehe Abbildung 1).

Header: Der *Header* bietet die Möglichkeit, die Seite neu zu laden, Informationen über die Website zu erhalten, den Lizenztext einzusehen, den Darkmode zu aktivieren und in den Developermodus zu wechseln. Im Developermodus stehen dem Benutzer nach einer Autorisierung weitere Funktionalitäten zur Verfügung.

Upload: Der Uploadbereich ermöglicht es dem Benutzer ein Bild hochzuladen oder ein vorgegebenes Beispielbild auszuwählen.

ImageView: Die *ImageView* zeigt das hochgeladene bzw. ausgewählte Bild an.

ImageDetails: Bei den *ImageDetails* werden die zum angezeigten Bild zugehörigen Informationenen in Form eines Historgrammes angezeigt.

Pipeline: Bei der Komponente *Pipeline* wird die aktuelle Verarbeitungskette angezeigt. Diese besteht initial nur aus dem hochgeladenen bzw. ausgewählten Bild und kann durch das hinzufügen weitere Bearbeitungsschritte mittels Drag and Drop erweitert werden. Ein Bearbeitungsschritt wird dabei von der Komponente *PipelineStep* repräsentiert. Dieser bietet die Möglichkeit dessen Informationen anzuzeigen, die Parameter zu konfigurieren und nach erfolgreichen Durchlauf das zugehörige Zwischenergebnis anzuzeigen.

AvailablePipelineSteps: Bei dieser Komponente werden dem Anwender alle verfügbaren Bearbeitungsschritte angezeigt. Diese können mit der implementierten *SearchBar* gefiltert werden. Ein verfügbarer Schritt, welcher mit der Komponente *AvailableStep* repräsentiert wird, kann per Drag and Drop zu der Komponente *Pipeline* gezogen werden. Dadurch wird die Verarbeitungskette um den ausgewählten Verarbeitungsschritt erweitert.

StartPipeline: Diese Komponente bietet dem Anwender die Möglichkeit, das hochgeladene bzw. ausgewählte Bild mit der erstellten Verarbeitungskette zu bearbeiten. Dabei werden alle Informationen mittels dem *Controller* ans Backend weitergeleitet.

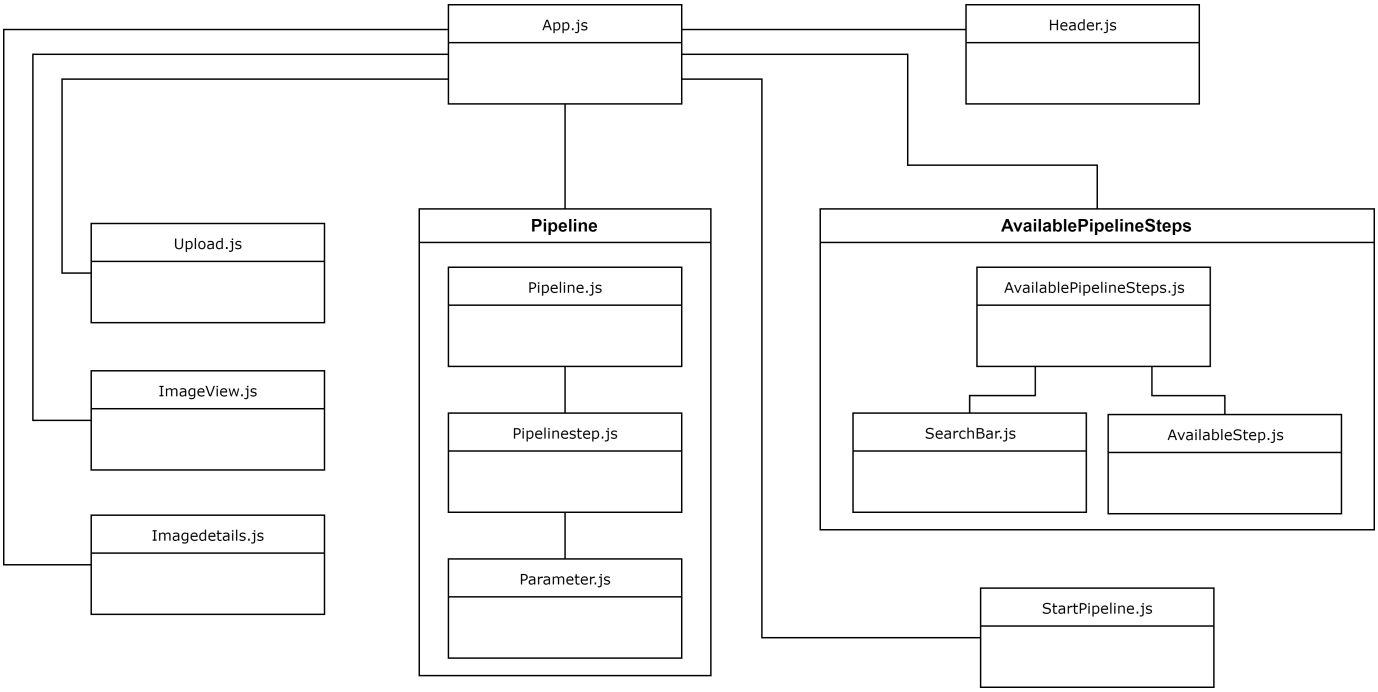

Abbildung 1. Übersicht der Komponenten im Frontend

B. Backend: Bildverarbeitung

Im Folgenden Abschnitt werden die wichtigsten Klassen und Funktionen des Backends vorgestellt. Diese sind auch in Abbildung 2 dargestellt.

routes: Hier werden alle REST-Endpunkte definiert, um die Schnittstelle zum Frontend zu ermöglichen. Diese werden im Abschnitt IV genauer erläutert.

S3Manager: Die Klasse *S3Manager* stellt Methoden zur Verfügung um mit einem S3-Bucket zu interagieren. Die Methode *getImageFromS3* kann verwendet werden um ein Bild aus dem S3-Bucket zu laden. Die Methode bekommt dafür den *objectKey*, also eine eindeutige ID, übergeben. Mit Hilfe der Methode *pushImageToS3* kann ein Bild unter einem bestimmten *objectKey* im S3-Bucket abgelegt werden. Außerdem werden die Methoden *deleteImageFromS3* und *deleteAllImagesFromS3* bereitgestellt, um ein bestimmtes oder alle Bilder aus dem S3-Bucket zu löschen.

Metadata: Die Klasse *Metadata* ist für die Erzeugung des Histogramms, sowie für die Extraktion von Metadaten aus einem Bild verantwortlich. Zu diesem Zweck stellt die Klasse die Methode *getMetadata* zur Verfügung. Diese bekommt als Eingabe ein Bild. Die Methode erzeugt ein Histogramm und speichert dieses als Bild im S3-Bucket. Zurückgegeben wird eine ID unter der das erzeugte Histogramm abgerufen werden kann, die Breite und Höhe des Bildes, sowie die Anzahl an Farbkanälen.

Pipeline: Die zentrale Klasse zum Verarbeiten der Bilder ist die Klasse *Pipeline*. Diese wird für jede Verarbeitung neu initialisiert. Bei der Initialisierung bekommt sie bereits die ID des zu verarbeitenden Bildes, sowie die Verarbeitungsschritte die Ausgeführt werden sollen mit. Beim Aufruf der Methode *start* wird über die Liste aller übergebenen Schritte iteriert und dieses mit den entsprechenden Parametern ausgeführt. Dabei sind alle Schritte von der Klasse *BaseStep* abgeleitet. Als Eingabe für den nächsten Verarbeitungsschritt dient dabei die Ausgabe des Vorherigen. Jedes Zwischenergebnis wird im S3-Bucket abgespeichert. Außerdem werden Histogramm und Metadaten für jedes Zwischenergebnis generiert. Die Methode liefert eine Liste an IDs und Metadaten für alle Zwischenergebnisse, diese können im Frontend angezeigt werden.

BaseStep: Die Klasse *BaseStep* stellt das Grundgerüst für alle implementierten Schritte dar. Alle Schritte implementieren die Methode *__call__*, die die Verarbeitung eines Bildes mit den übergebenen Parametern ausführt und das Ergebnisbild zurückliefert. Außerdem implementieren alle Schritte die Methode *describe*, diese liefert Informationen über die Benutzung des Schritts und wird verwendet, um dem Benutzer die Schritte im Frontend zu erklären und anzuzeigen (siehe Endpunkt */available-steps*)

IV. Schnittstellen

Die Kommunikation zwischen Front- und Backend erfolgt über eine REST-Schnittstelle. Die Endpunkte dieser Schnittstelle werden im Folgenden genauer erläutert.

/start-pipeline/<imageId>: Der Endpunkt */start-pipeline/<imageId>* ist dafür zuständig die Verarbeitung eines Bildes zu starten. *<imageId>* ist dabei eine eindeutige ID um das zu verarbeitende Bild aus dem S3-Bucket zu laden. Im Content des HTTP-POST Requests werden dabei die auszuführenden Verarbeitungsschritte, sowie deren Parameter übergeben. Die tatsächliche Verarbeitung des Bilds übernimmt die Klasse *Pipeline*.

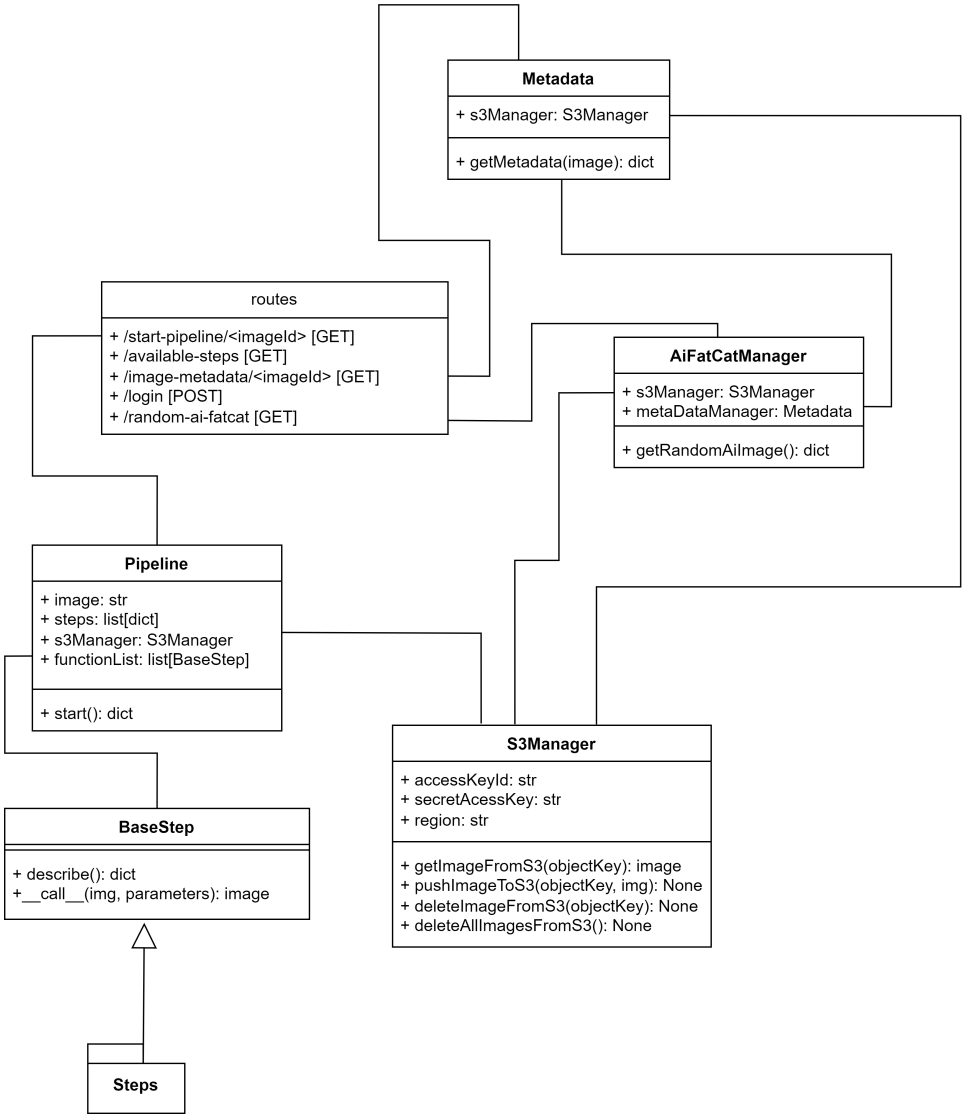

Abbildung 2. Die wichtigsten Klassen und Funktionen im Backend

/available-steps: Der Endpunkt */available-steps* liefert eine Beschreibung für alle implementierten Verarbeitungsschritte, sowie deren Parameter. Dieser Endpunkt wird verwendet um die verfügbaren Verarbeitungsschritte im Frontend anzuzeigen und dem Benutzer Informationen über deren Verwendung zu liefern.

/image-metadata/<imageId>: Mit Hilfe des Endpunkts */image-metadata/<imageId>* kann ein Histogramm für das übergebene Bild erstellt werden. Außerdem liefert der Endpunkt Informationen über die Größe des Bilds und die Anzahl an Farbkanälen. Für die Erzeugung dieser Informationen ist die Klasse *Metadata* verantwortlich.

/login: Der Endpunkt */login* bietet die Möglichkeit die im POST-Request übergebenen Parameter Benutzername und Passwort zu überprüfen. Dieser wird verwendet um dem Benutzer die Möglichkeit zu geben das Entwicklermenü im Frontend zu aktivieren und darüber alle Daten aus dem S3-Bucket zu löschen. In der Aktuellen Implementierung werden Benutzername und Passwort im Klartext unverschlüsselt über HTTP übertragen. Diese Übertragung stellt keine sichere Authentifizierung dar für den Einsatz in einem Produktivsystem muss an dieser Stelle eine sichere Authentifizierung implementiert werden.

/random-ai-fatcat: Unter dem Endpunkt */random-ai-fatcat* kann ein zufälliges Katzenbild aus dem Internet geladen werden. Dieses wird im S3-Bucket gespeichert und die ID zurückgegeben. Die Bilder stammen dabei von einer externen API [13].

A. AWS: Amazon Web Services

Im Rahmen des Projekts wurde Amazon Web Sevices (AWS) verwendet, um die Anwendung mithilfe einer EC2-Instanz zu hosten und die Daten mithilfe eines S3-Buckets zentral zu speichern. Auf den S3-Bucket wird mithilfe, der

im Frontend und Backend erstellten Klasse *S3Manager* zugegriffen. Die Klasse basiert auf den Funktionen der Bibliothek boto3 [9] und der aws-sdk [10]. Der S3-Bucket dient als gemeinsamer Speicher zum Austausch von Bildern zwischen Frontend und Backend.

Die Zugriffsrechte wurden in der AWS-Konsole über den Identity und Access Management (IAM) konfiguriert. Dabei wurden feingranulare Rechte an die Benutzer vergeben. Dies soll sicherstellen, dass nur autorisierte Benutzer auf den *bdcc-fatcat-data* Bucket zugreifen können. Diese Sicherheitsmaßnahme soll die anderen Bereiche der Cloud schützen, falls ein Bereich kompromittiert wird.

Durch die Verwendung von AWS und den beiden S3Manager Klassen konnte eine skalierbare Lösung zur Bildspeicherung und -verwaltung bereitgestellt werden.

B. Testabdeckung

Im **Frontend** wurden Unit-Tests mit Hilfe des Testframeworks *jest* [7] implementiert. Durch die Verwendung der *–coverage*-Einstellung wurde die Testabdeckung ermittelt, um sicherzustellen, dass eine möglichst umfassende Prüfung des Codes erfolgt. Dabei konnte im Frontend eine Testabdeckung von ca. $84\ \%$ erreicht werden.

Im **Backend** wurden ebenfalls Unit-Tests eingesetzt. Dazu wurde das Testframework *pytest* [8] verwendet. Auch hier wurde die Testabdeckung mithilfe des Pytest-Plugins *pytest-cov* überwacht [14]. Dabei konnte eine Testabdeckung von ca. $92\ \%$ erreicht werden.

V. Entwicklungswerkzeuge

Für die Entwicklung des Projekts wurde verschiedene Werkzeuge verwendet, um ein möglichst reibungsloses und effizientes Arbeiten zu gewährleisten.

A. Paketverwaltung

Die Paketverwaltung spielt eine entscheidende Rolle bei der Verwaltung der Abhängigkeiten und dem Versionsmanagement der einzelnen Bibliotheken.

Im Backend wurde *pip* verwendet. Dies ist die Standardpaketverwaltung für Python. *Pip* ermöglicht die einfache Installation und Aktualisierung von Python-Paketen. Im Frontend erfolgte die Paketverwaltung mithilfe von *npm* (Node Package Manager). Dadurch konnten die benötigten Pakete für das Frontend effizient verwaltet werden.

B. Frontend

Das Frontend des Projekts basiert auf React [6]. Dies ist eine leistungsstarke JavaScript-Bibliothek, die zur Entwicklung von Benutzeroberflächen dient. React ermöglicht eine modulare Gestaltung der Oberfläche, was zu einer besseren Wartbarkeit des Codes führt. Die einzelnen React-Komponenten wurden in JSX (einer Erweiterung für JavaScript) geschrieben.

C. Backend

Das Backend des Projekts basiert auf Python und dem Flask-Framework. Python bietet eine Vielzahl an Bibliotheken zur Bildverarbeitung. Flask zeichnet sich durch seine einfache Handhabung und Flexibilität bei der Implementierung von REST-Schnittstellen aus.

D. Container

Um eine effiziente Bereitstellung des Systems zu gewährleisten, wurden Container verwendet. Die Komposition der Container erfolgt dabei mit Docker-Compose [12]. Dabei existiert ein Container für das Frontend und ein Container für das Backend. Durch die Verwendung von Containern wurde sichergestellt, dass alle erforderlichen Abhängigkeiten, Bibliotheken und Konfigurationen zentral festgelegt sind und Plattform unabhängig ausgeführt werden können.

VI. Fazit und Ausblick

Das Projekt Computer Vision Pipeline bietet den Benutzern eine einfache Anwendung zur Verwendung von gängigen Bildverarbeitungsschritten.

Abschließend lässt sich festhalten, dass das Projektteam trotz der unterschiedlichen Vorkenntnisse erfolgreich zusammenarbeitete. Die Einarbeitung in das Frontend-Framework React kostete zwar Zeit, aber führte letztendlich zu einer ansprechenden Benutzeroberfläche. Auch die Implementierung von Tests erwies sich als herausfordernd. Zur Qualitätssicherung des Systems sind diese aber von großer Bedeutung.

Ein Ausblick auf das Projekt zeigt vielversprechende Möglichkeiten zur Erweiterung der Anwendung. Durch die angelegte Pipelinestruktur im Backend, können mit wenig Aufwand weitere Funktionen eingebaut werden. Da personenbezogene Daten, wie die Bilder, verarbeitet werden, sollte in Betracht gezogen werden die Anwendung um eine Authentifizierung für den Benutzer zu erweitern. Dadurch sollte sichergestellt werden, dass nur der Benutzer auf seine Bilder Zugriff hat. Außerdem sollte die Nutzung von Session-Tokens für AWS im Frontend implementiert werden. Dadurch wird es ermöglicht die AWS-Credentials (die Zugangsdaten) nur temporär an das Frontend zu übertragen. Dies wurde aufgrund der Zeit nicht in den Produktiv-Code eingefügt.

Die Kommunikation zwischen allen Komponenten erfolgt in der aktuellen Implementierung über das unverschlüsselte *HTTP*-Protokoll. Um eine sichere Datenübertragung zu gewährleisten sollte das verschlüsselte *HTTPS*-Protokoll verwendet werden. Neben der bereits beschriebenen unverschlüsselten Kommunikation zwischen Frontend und Backend, stellt auch der direkte Zugriff auf den S3-Bucket durch das Frontend in der aktuellen Implementierung ein Sicherheitsrisiko dar.

Literatur

[1] Scratch Programmierung: https://scratch.mit.edu/projects/editor/?tutorial=getStarted
[2] Python: https://www.python.org
[3] OpenCV: https://opencv.org
[4] scikit-image: https://scikit-image.org

[5] Flask: https://flask.palletsprojects.com/en/2.3.x/
[6] React: https://react.dev
[7] Jest: https://jestjs.io
[8] PyTest: https://docs.pytest.org/en/7.3.x/
[9] Boto3: https://boto3.amazonaws.com/v1/documentation/api/latest/index.html
[10] AWS-SDK: https://www.npmjs.com/package/aws-sdk
[11] Docker: https://www.docker.com
[12] Docker Compose: https://docs.docker.com/compose/
[13] The Cat API: https://thecatapi.com
[14] pytest-cov: https://pypi.org/project/pytest-cov/

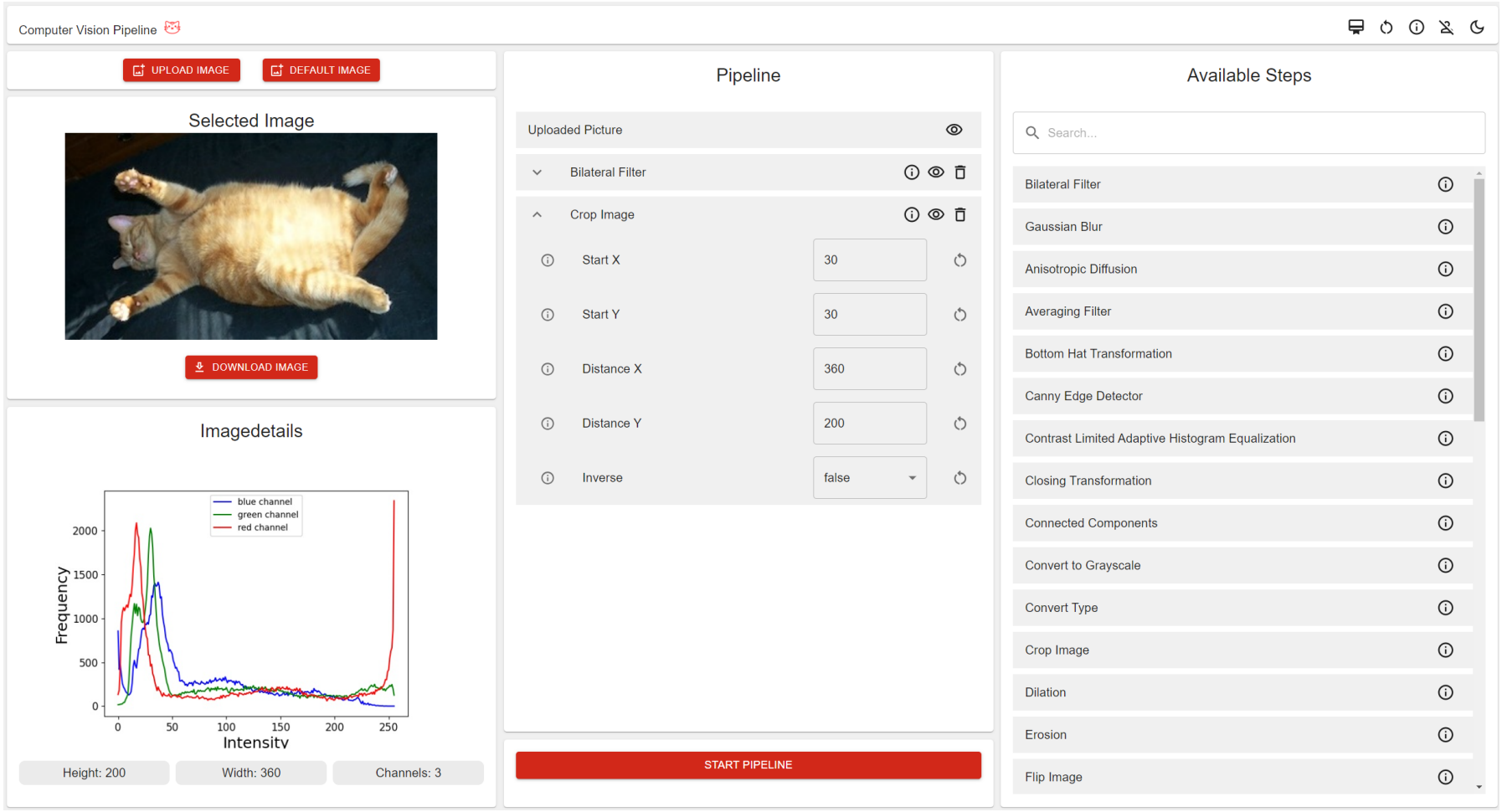

Abbildung 3. Frontend Computer Vision Pipeline