WGU C964

Task 2

Capstone Project

Name: John Lee

Student ID # 011740395

Date: 2/24/2025

# Part A: Letter of Transmittal

## Letter of Transmittal

2/24/2025

Mr. Ashe

Nintendont Company

4001 S 700 E #300

Millcreek, UT 84107

Dear Mr Ashe,

I am presenting to you this proposal for an machine learning-based asset categorization system designed to streamline game asset management for our development team. As game production scales, the ability to efficiently search and categorize assets is crucial. Manual tagging is often time-consuming, inconsistent, and prone to human error, creating challenges in asset retrieval. This proposal outlines a proof-of-concept model, demonstrating how a AI and automization can enhance asset organization and searchability.

Currently, our asset database contains thousands of visual assets, including characters, items, and environment objects. Our developers must manually tag assets, making it difficult to efficiently retrieve content. This slows down development cycles and increases production overhead. To address this issue, I have developed a machine learning-based image classification model, initially trained on a simplified “Pikachu vs. Not Pikachu” dataset. This prototype successfully analyzes visual features and automatically categorizes assets, demonstrating the feasibility of scaling the system for broader game development use. Future iterations will extend to multi-class classification, enabling automated tagging for diverse asset types.

With this implementation, we should expect to see reduced time and effort spent on manual labeling. With quicker asset retrieval times, we can streamline the development process and even look forward to scaling future projects. The model itself can be expanded to cover multiple asset categories and the automization will ensure that there are less human errors in tagging, avoiding duplicate or misplaced assets.

Project implementation should consist of 4 phases; data sourcing and processing, model development, deployment, and documentation. The time spent in each phase should be kept streamlined without any need for extensions. In regards to project costs and time, the estimated investments will be very reasonable. The development of the model will utilize existing open-source frameworks (TensorFlow/Keras) which cuts out any tools or libraries that would require payment for use. The expected time of completion is estimated at around 3 months, which includes the dataset sourcing, integration, development, and deployment. As a whole, we estimate that the cost of the project will be around $20,000, mainly in part due to labor fees for the engineers. In addition, as the data sourced for the project comes from publicly available websites there is no concern for data privacy infringements.

As a developer myself, with experience in both machine learning and game development. I am confident in this project and its impact on resolving our current asset tagging issues. My experience in AI-driven automation as well as game development uniquely positions me to be the perfect lead for this task. I welcome the opportunity to discuss this proposal further and explore how we can implement AI-driven asset tagging to enhance our game development workflow. Thank you for your time and consideration.

Sincerely,

John Lee

# Part B: Project Proposal Plan

## Project Summary

Game development teams face challenges in managing and searching for visual assets in large internal content databases. Manual tagging of assets, such as characters, items, and environments, is time-consuming and prone to inconsistency. This results in inefficiencies in asset retrieval, delaying development timelines.

Nintendont game development company is seeking to improve the efficiency of their asset management system. Their primary need is an automated image classification solution that can accurately tag and categorize game assets, improving searchability and accessibility within their internal database. To resolve these issues, this project will create a convolutional neural network model for that is able to tag assets automatically. Users will be instructed on how to to use the model to retrieve image predictions from the model and will be given performance reports from the program itself for documentation purposes. This should allow for manual workloads, improving the development process by reducing time spent in the asset management process.

## Data Summary

The raw data will be sourced from **Kaggle.com/datasets** and supplemented with proprietary game assets provided by the client. The dataset will be curated to ensure relevant asset classification.

During the design phase, data will be analyzed, cleaned, and augmented to ensure diversity and reduce bias. In the development phase, data will be preprocessed using normalization, augmentation, and resizing to improve model accuracy. The maintenance phase will include periodic retraining of the model with new game assets to improve classification performance.

The selected dataset includes diverse asset types, ensuring the model generalizes well to different asset styles. Any anomalies, such as mislabeled images or outliers, will be identified and corrected through data validation techniques.

All dataset sources will be verified for copyright compliance, ensuring no proprietary or copyrighted assets are used without permission. The tagging system will avoid bias by using diverse training samples.

## Implementation

The project will follow the CRISP-DM (Cross-Industry Standard Process for Data Mining) methodology, ensuring a structured and iterative approach. The process begins with business understanding, identifying asset tagging challenges. It continues with data understanding, where the dataset is analyzed and preprocessed. Data preparation follows, which involves augmenting and formatting assets for training. Modeling is the next step, where the machine learning model is developed and fine-tuned. Evaluation is conducted to assess model accuracy and performance. Finally, the system is deployed into the client’s workflow.

The project will be implemented in multiple phases. The first phase, lasting approximately one month, will involve data collection and preprocessing. The second phase, expected to take two months, will focus on model development and initial training. The third phase, lasting one month, will consist of testing and evaluation. The fourth phase, which will take two months, will involve deployment and system integration. The final phase will be an ongoing effort for monitoring, maintenance, and retraining.

## Timeline

|  |  |  |  |
| --- | --- | --- | --- |
| Milestone or deliverable | Duration  (hours or days) | Projected start date | Anticipated end date |
| Data Retrieval | 4 hours | 2/28/2025 | 2/28/2025 |
| Data Preprocessing | 4 hours | 2/28/2025 | 2/28/2025 |
| Model Creation | 3 days | 3/1/2025 | 3/4/2025 |
| Model Training/Testing | 1 day | 3/5/2025 | 3/5/2025 |
| Model Revision | 1 day | 3/6/2025 | 3/6/2025 |
| Model Deployment | 1 day | 3/7/2025 | 3/7/2025 |
| Training Users | 2 days | 3/8/2025 | 3/9/2025 |
| Documentation | 1 day | 3/10/2025 | 3/10/2025 |

## Evaluation Plan

Verification will include code reviews to ensure proper implementation of preprocessing and model training scripts. Unit testing will be conducted to verify individual model components, including data preprocessing, feature extraction, and inference accuracy. Integration testing will confirm the seamless interaction of the system with the client’s asset database.

The model will be evaluated/validated using performance metrics such as accuracy and loss. A graph will be shown to visualize the accuracy and losses during model training. Confusion matrix analysis will be performed to identify misclassifications and improve model tuning.

## Resources and Costs

Estimated costs for this project will be broken down into sections:

Hardware Costs

* Storage for Dataset and Models – ~$1,000

Software Costs

* TensorFlow/Keras (Open-Source) – $0
* Cloud Hosting for Application – ~ $1,500/year

Labor Costs

* Machine Learning Engineer (80 hours @ $50/hr) – $4,000
* Software Developer (50 hours @ $45/hr) – $2,250
* Project Manager (60 hours @ $60/hr) – $3,600

Deployment & Maintenance Costs

* Cloud Deployment & Hosting – $1,500 annually
* Model Retraining & Updates – $5,000 annually

Total Estimated Cost: $20,000

# Part C: Application

Part C is your submitted application. This part of the document can be left blank or used to include a list of any submitted files or links.

The minimal requirements of the submitted *application* are as follows:

1. **The application functions as described.** Following the ‘User Guide’ in part D, the evaluator must be able to successfully review your application on a Windows 10 machine.
2. **A mathematical algorithm applied to data,** e.g., supervised, unsupervised, or reinforced machine learning method.
3. **A “user interface.”** Following the ‘User Guide’ in part D, the client must be able to use the application towards solving the proposed problem (as described in parts A, B, and D). For example, the client can input variables, and the application outputs a prediction.
4. **Three visualizations.** The visualizations can be included separately when including them in the application is not ideal or possible, e.g., the visualizations describe proprietary data, but the application is customer-facing.
5. **Submitted files and links are static and accessible.** All data, source code, and links must be accessible to evaluators on a Windows 10 machine. If parts of the project can be modified after submission, matching source files must be submitted. For example, if the application is a website or hosted notebook, the `.html` or `.ipynb` files must be submitted directly to assessments.

Ideally, submitted applications should be reviewable using either Windows or Mac OS, e.g., Jupyter notebooks, webpages, Python projects, etc. If the source files exceed the 200 MB limit, consider providing screenshots or a Panopto video of the functioning application and contact your course instructor.

# Part D: Post-implementation Report

## Solution Summary

As a game development team, automization of game assests can be a difficult task. When working on multiple projects, a single team may work with more than 100,000 game assets. Manually going through each one and tagging them is both time-consuming and prone to inconsistency. This project aims to automate the categorization of these assets using machine learning, making asset retrieval faster and more accurate.

The project leverages a Convolutional Neural Network algorithm to classify images automatically. The application goes through the steps of dataset processing, model development, training/evaluation, and validation to provide the team an accurate classification of any given image. The project already produces an accuracy rate of ~95% and most importantly is scalable for future use. The project can be expanded to recognize multiple asset categories and brought to a point where manualy tagging will become entirely uncessary.

## Data Summary

The data used for this project was received from Kaggle.com, a popular platform for machine learning datasets. The data was created from the it’s author by web scarping the internet using Azure Bing Web Search API. This dataset, though simple, is the perfect base to test the feasibility of automated asset categorization before scaling to more complex game assets.

In its design phase the data was augmented through flipping, zooming and shearing to enhance generalization. In the development phase, the images were preprocessed by rescaling the images between a 0 and 1 value while the pixels were resized to 64x64 pixels for uniformity. The data was then split into a near 8:2 ratio, consisting of 774 images in the training set and 258 images in the test set. All images are kept in a folder labeled Dataset and broken down into subfolders for accessibility. The dataset is easily expanded by simply adding more images to the folders and is easily integrated into the CNN model.

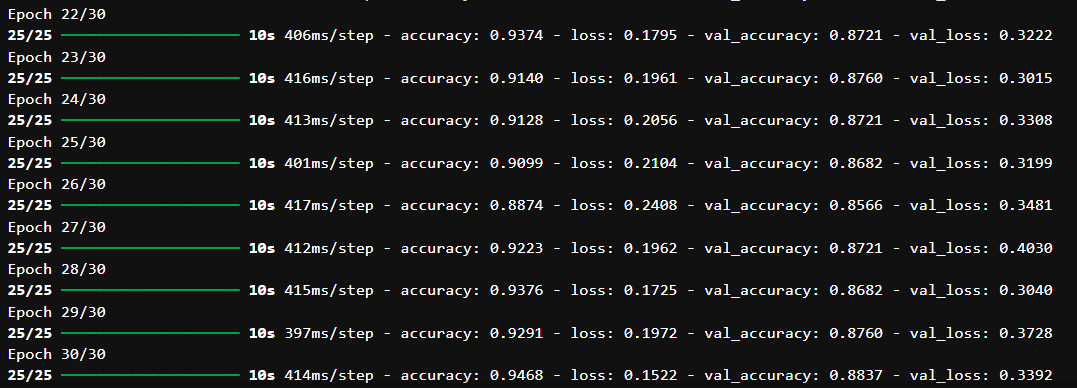
## Machine Learning

The project is a binary image classification model that distinguishes between images containing the popular character, Pikachu, and those that do not. Using TensorFlow/Keras as its core libraries, the model constructs a Convolutional Neural Network (CNN) to classify the images. A CNN works like “glasses” which allow the program to interpret visual information by focusing on specific features of the images. This “glasses” consists of multiple layers like the convolutional layer, pooling layer, and output layer that extracts features of the given images and categorizes them while returning a probability score for the binary classification.

In this project the model is given 774 training images and 258 test images to so build its classification probability. The images are processed through a ImageDataGenerator, which scales the pixel values to the [0,1] range. The data is then augmented with transformations (shear, zoom, flip). Once the data is processed the model trains by running 30 epochs which evaluates its performance on the validation set. The model returns a accuracy and loss curves and a confusion matrix for visualization to asses for model prediction accuracy and misclassifications.

This project was built using a CNN model due to the algorithm being designed for analyzing visual data like images, making it well-suited for tasks like image classification and object recognition. A CNN is able to detect edges, textures, and object structures making it the ideal algorithm for a binary classification task. The model is supplemented with an accuracy/loss graph and a confusion matrix which provides easy-to-understand measurements and insights into false positives and false negatives.

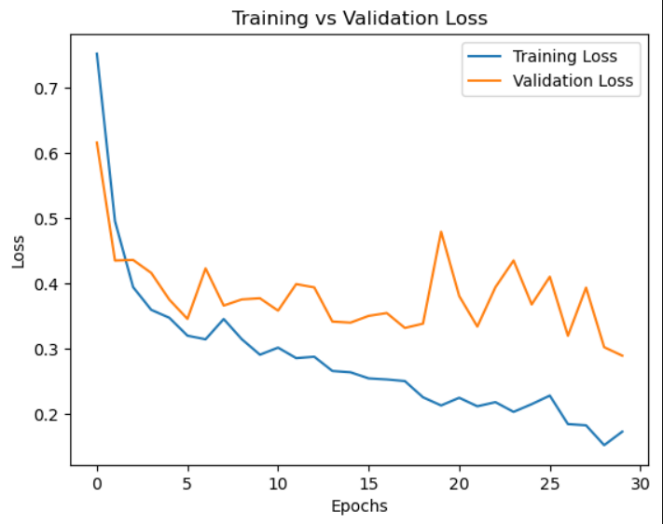
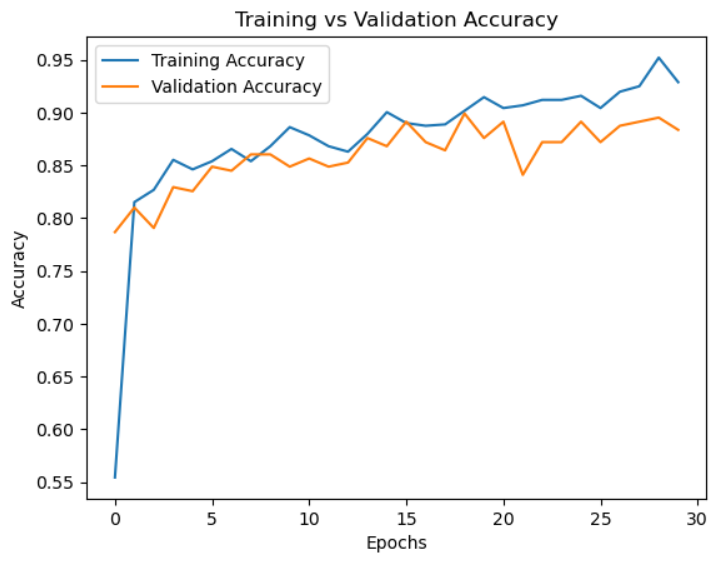
## Validation



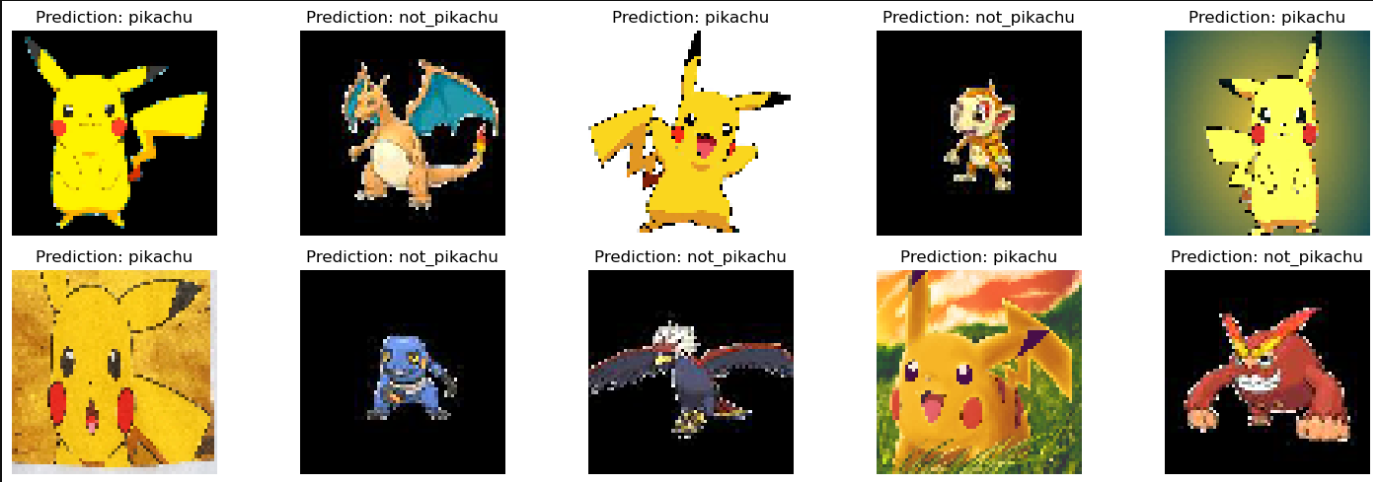
The model performance metric was tracked using accuracy\_score from sklearn.metrics and the built-in metric from TensorFlow. The training accuracy is measured during each epoch to track learning progress. The validation accuracy is measured using test data to check how well the model generalizes. These results are measured across 30 epochs to analyze performance trends. The model also uses a binary cross-entropy loss to measure the difference between predicted probabilities and actual labels. Loss values were recorded during training and validation phases. As shown in the image above, over 30 epochs, an accuracy score of around 93% and a validation accuracy of around 86% can be seen.

In addition, to these metrics, the program also provides a prediction chart and confusion matrix for visualizations (shown in the next section). The confusion matrix was used to assess classification performance beyond accuracy. The matrix is divided into 4 sections: True Positives, False Positives, True Negatives, and False Negatives. Beyond these visualizations future improvements to validate the model can be done through a F1-score and precision recall analysis. Enhancements will optimize generalization further but may only be necessary in future implementations of the program.

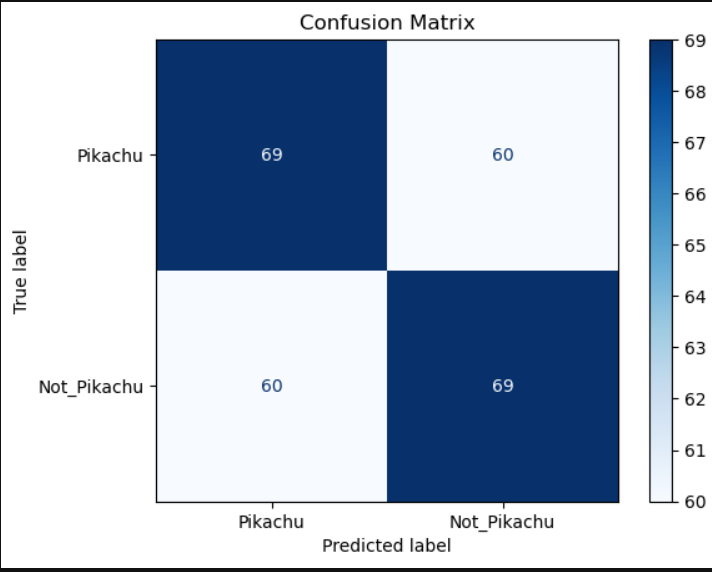
## Visualizations



* Graph of Accuracy and Loss



* Model’s predictions shown above for each image in the validation set.



* Confusion Matrix

## User Guide

Include an enumerated (steps 1, 2, 3, etc.) guide to execute and use your application.

* Include instructions for downloading and installing any necessary software or libraries.
* Give an example of how the client should use the application.