CITY College
International Faculty of the University of

Sheffield Athens Tech

Computer Science Department

FINAL YEAR INDIVIDUAL PROJECT

Cat breed image classification

This report is submitted in partial fulfilment of the requirement for the degree of Bachelor of Science with Honours in Computer Science by

Iakovos Chaliasas

June 4, 2023

Approved

____

Dr. Thanos Tagaris

Cat breed image classification

by

Iakovos Chaliasas

Supervisor

Dr. Thanos Tagaris

ABSTRACT

This research introduces a method for distinguishing among various cat breeds using a specially tailored machine learning model. The process of developing this model involved a cycle of experiments and fine-tuning to maximize its prediction capabilities. Our initial step was to establish clear goals for this project, followed by an in-depth exploration of artificial intelligence, focusing on subcategories such as machine learning, deep learning, and image classification. We then examined a range of topics crucial to our study, including convolutional neural networks, the ResNet model, the ImageNet dataset, and transfer learning, along with other fundamental machine learning concepts. After laying a strong theoretical foundation, we moved on to explain our experimental methodology, including how we handled the dataset called "Cats and Dogs Breeds Classification Oxford Dataset." We used this dataset to train two different models, with the first model reaching an accuracy rate of 31.5%, and the second one, which employed the ResNet architecture and transfer learning, achieving a remarkable accuracy rate of 92%. We also discussed the development of our application, including the creation of both front-end and back-end components. The final sections of our study shed light on how we tested our product and evaluated its performance. Overall, this research contributes a valuable solution for accurate cat breed identification using machine learning.

DECLARATION

All sentences or passages quoted in this thesis from other people's work have been specifically acknowledged by clear cross referencing to author, work and page(s). I understand that failure to do this, amounts to plagiarism and will be considered grounds for failure in this thesis and the degree examination as a whole.

It is also my understanding that intellectual rights of this work are shared equally between myself (the author of the dissertation), the supervisor of the project and The University of Sheffield International Faculty of CITY College, and that any kind of exploit of this work will need the consent of all parties involved.

☐ I agree that my dissertation can be used as a model/example by future undergraduate students, for educational purposes only.

Name: IAKOVOS CHALIASAS

Signed: .... Date: June 4, 2023

Acknowledgements

I want to take a moment to express my deep gratitude to Dr. Thanos Tagaris for his unwavering support and guidance throughout my individual project. His vital insights and assistance have been key in reaching my objectives, for which I am immensely grateful. My sincere thanks also go to Dr. Dimitris Iracleous, the Academic Director of Projects and Dissertations, and Dr. Odysseas Efremidis, who both provided expert advice and invaluable perspectives during the project. Their readiness to address any queries I had during the academic year has been greatly appreciated. Lastly, I want to acknowledge Mrs. Maria Georgiou, our College Administrator, whose assistance has been pivotal throughout my entire college journey.

Table of Contents

List of Figures

1. Introduction

Since the 1940s, computers have become an integral part of our lives, and the field of computer science has evolved tremendously. In the early days, the creation of computers like ENIAC and others created a need for computer science, and Columbia University established one of the first academic classes in 1946 [1] that offered credits in computer science. Later on, in 1953 [2], Cambridge University offered the first computer science degree. Fast forward to today, we are living in a time where technology is evolving at an unprecedented rate. AI is one of the technologies that has blossomed, and new products are being released every day that push the boundaries of what we thought was possible. From image-generating software to text and voice generation, AI has the potential to change our lives in many ways. However, along with the many advancements come new challenges. The ethical implications of AI have become a prominent concern, with many problems that need to be addressed. While some of these challenges are academic, such as the emergence of chatGPT and its uses, others are more pervasive, like the ease of creating fake videos and the ethical concerns surrounding AI intelligence. One important area of AI is image classification, which has many practical applications in fields such as agriculture and academia. As we move forward, it is crucial to consider the ethical implications of AI and find ways to address these challenges so that we can continue to reap the benefits of this powerful technology [3].

1.1. Aims

The aims of this project are firstly to create a machine learning algorithm that can identify cat breeds given an image and secondly to create a web application to showcase the capabilities of the machine learning algorithm.

1.2. Objectives

The objects of this project are divided into two parts. The first part is concerning the development of the ML algorithm:

- Identify, download, and preprocess dataset for cat breed classification.
- Experimentation with custom and pre-trained CNN architectures for image classification to identify the best performing one.
- Fine-tune model on above dataset.
- Evaluation of final model to see if its performance is acceptable.

The second part is concerning the development of the software application:

- Development of machine learning service that utilizes the above trained model.
- Development of REST API that will expose the above machine learning service to the web.
- Development of web app that the user can access the machine learning service through.
- Testing of individual components as well as the whole app.

1.3. Motivation

I was drawn to this project due to my fascination with the concept of machine learning and its impact on the world. With its applications in medicine, construction, research, and many other fields, machine learning is transforming our lives every day. I was eager to delve deeper into this field, learn about its intricacies, and begin my journey in this incredible world. Choosing to work on the cat breed classification project was a natural fit for me. My love for animals, particularly cats, was a driving force behind my decision. I was curious about how image classification works and the potential impact it could have in fields such as medicine, such as detecting skin cancer and other diseases through image analysis. Through this project, I was able to explore this technology and its capabilities in a hands-on way. This project allowed me to not only develop my technical skills but also deepened my understanding of the power and potential of machine learning. As I continue my journey in this field, I am excited to explore further the countless possibilities of machine learning in solving complex problems and contributing to a better world.

1.4.Main Research Domains

1.4.1. Artificial Intelligence

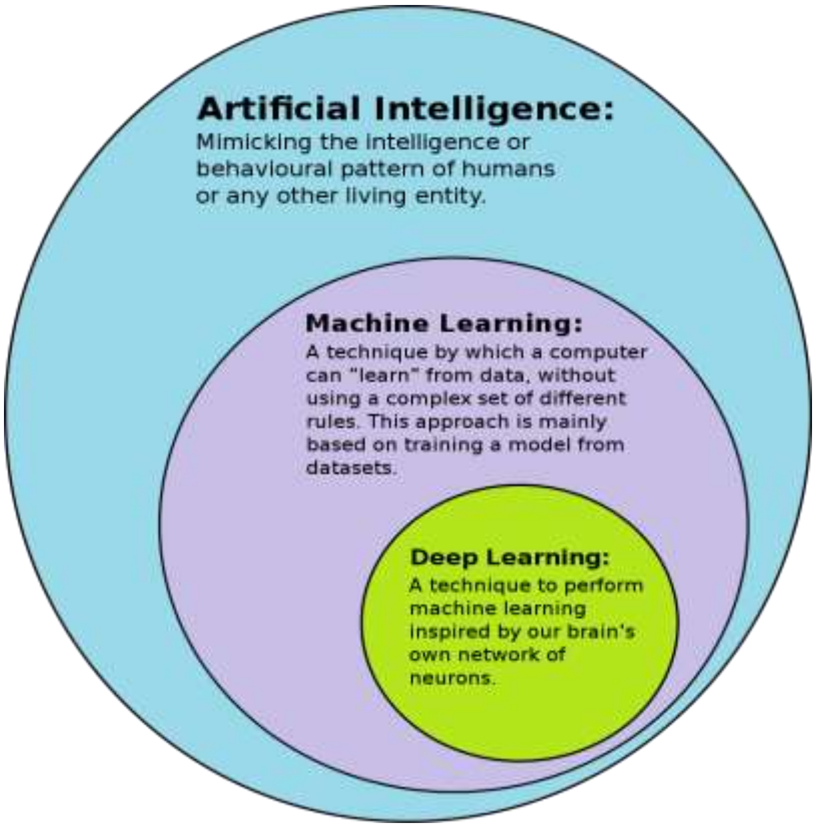
Artificial Intelligence (AI) represents a vital component of contemporary computer science, anchoring its purpose in the conception of intelligent systems designed to perform tasks typically requiring human cognitive faculties. These tasks encompass a broad spectrum of activities including natural language processing, pattern recognition, complex problem-solving, and the execution of informed decisions. AI is divided into two primary categories. The first, termed "narrow AI", is primarily concerned with executing relatively simpler tasks, examples of which include image and speech recognition. In contrast, "general AI" embodies a more ambitious aspiration - to emulate human cognitive abilities across a wide array of sophisticated and intellectually demanding tasks. It's essential to recognize the underlying subsets that constitute AI. Machine learning is one such subset, a discipline that revolves around the training of algorithms on data, and subsequently learning and making decisions based on the acquired information. This subset provides the foundation for another, known as deep learning. Deep learning utilizes neural networks to model and identify complex patterns within large datasets. Recent developments in AI have introduced concepts such as explainable AI (XAI), which seeks to develop systems that offer transparent and understandable decision-making processes for human experts. This is going to significantly benefit industries such as healthcare and autonomous transportation, where comprehensible decision-making systems are crucial. Natural Language Processing (NLP) is another expanding field within AI that has witnessed considerable advancements in recent years. Its focus lies in the facilitation of interactions between computers and human languages. The rise of transformer architectures has catapulted NLP into a new era, enabling machines to generate text that mirrors human composition, exemplified by applications like ChatGPT, Bard etc. Image Classification, empowered by the implementation of deep neural networks, has also made significant strides. It's a field that is having a profound influence on sectors such as healthcare and security, revolutionizing the way we diagnose illnesses and secure our systems, thereby spotlighting AI's pivotal role in modern society [34].

1.4.2. Machine Learning

Machine learning, a crucial component of artificial intelligence, branches out into three primary types: supervised learning, unsupervised learning, and reinforcement learning. The focus of this project lies within the realm of supervised learning, where the model receives training on pre-labeled data, and the objective is to predict the class or category of the input data. In contrast, unsupervised learning, the model is exposed to unlabeled data, and its mission is to discover patterns within the given data set. Common applications of this approach include clustering, where the aim is to group similar data together, and anomaly detection, which focuses on identifying unusual data. Reinforcement learning, the third type, is rather interactive. In this scenario, the model learns by carrying out actions and receiving feedback in the form of rewards for desirable actions and penalties for undesirable ones. The ultimate goal of this approach is to maximize the accumulation of rewards, which consequently encourages the model to act in the desired manner. Moreover, there are several other subtypes of machine learning, formed by interweaving these three primary ones like semi-supervised learning, deep learning and more [34].

1.4.3. Deep Learning

Deep learning, which falls under the larger umbrella of machine learning, is a fascinating technology that attempts to mimic the learning process of the human brain in computers. It's a vital technology that is gaining momentum, especially in modern applications such as self-driving cars and virtual assistants. At the core of deep learning, you find an extensive neural network. This network adapts and changes its internal structure, or 'weights', throughout the training phase. By learning from substantial datasets, it's as if we're creating a miniature model of a human-like brain. This process endows the system with a certain sense of "intelligence", making it capable of understanding and responding to complex scenarios, much like a human would [34].


Artificial Intelligence:
Mimicking the intelligence or behavioural pattern of humans or any other living entity.
Machine Learning:
A technique by which a computer can "learn" from data, without using a complex set of different rules. This approach is mainly based on training a model from datasets.
Deep Learning:
A technique to perform machine learning inspired by our brain's own network of neurons.

1.1 Artificial Intelligence Umbrella [35]

1.4.4. Image Classification

Image classification is a fundamental process in machine learning that allows us to categorize images into different classes automatically. The process of image classification generally involves three main steps. Firstly, data preparation, where a dataset of images is created and prepared for use in a machine learning algorithm. This may include resizing images, normalizing them, splitting the dataset into training and testing sets, and labeling each image with the correct class. The second step is to create a convolutional neural network, which is a deep learning algorithm that can learn the features of the dataset to identify patterns in the images. Finally, the third step involves using the trained model to correctly classify the images into their respective classes. Image classification has gained a significant amount of attention in recent years due to its numerous applications in various fields. This technique has the ability to analyze large datasets of images automatically, allowing for the identification of patterns that would be difficult for humans to identify. It is a powerful technique that has found a wide range of real-world applications in fields such as medicine, security, and autonomous driving. In medicine, image classification is used to diagnose diseases by analyzing medical images. In security, it can be used to detect and identify individuals based on facial recognition. In autonomous driving, it can be used to detect and avoid obstacles and traffic signs. With the advances happening in machine learning and deep learning algorithms every year we can expect image classification to be used in many more parts of our lives to find solutions to problems we face in our everyday life [34].

2. Theoretical Background

2.1. Convolutional Neural Networks

Convolutional Neural Networks or CNNs, represent an integral component of artificial neural networks. They emulate the human brain's process of interpreting visual stimuli by mimicking the structure of the animal visual cortex, the portion of the brain tasked with processing visual data [4]. CNNs consist of interconnected artificial neurons, arranged in layers. The interconnections are such that the output of some neurons becomes input for others, thereby mirroring the synaptic connections observed in the brain. These layers, within a typical CNN architecture, are primarily grouped into three types: convolutional, pooling, and fully connected layers. The convolutional layer, the first layer in most CNNs, filters the input data for specific visual features. The pooling layer then simplifies this information, reducing computational demand by eliminating less important details. Finally, the fully connected layer interprets these features and makes a final decision about the input image, similar to how our brain concludes what we see. CNNs are powerful tools in machine learning and deep learning, used extensively across various fields. Their ability to learn intricate patterns in visual data, while minimizing preprocessing steps, has led to breakthroughs in numerous applications, from image and video processing to medical imaging, and autonomous vehicles [34].

2.2. ResNet

Res-Net, short for "Residual Network," is a type of Convolutional Neural Network (CNN) architecture that was introduced in 2015 by a team of researchers from Microsoft Asia, including Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun [5]. Since its introduction, Res-Net has been evaluated on many benchmark datasets and has achieved remarkable results, including winning 1st place on the ILSVRC 2015 classification task with an accuracy of 3.57% on the ImageNet problem using ResNet-152. One of the main challenges faced by deep CNN architectures is the vanishing gradient problem. When training deep networks, the gradients of the loss function with respect to the weights can become very small, making it difficult for the network to learn effectively. This problem occurs more when using many convolutional layers, where the gradients can vanish altogether. Res-Net addresses this challenge by introducing skip connections, which enable the network to skip one or more layers and create a shortcut from the input layer to the output layer. By doing so, the network can learn residual functions that capture the difference between the input and output blocks, without requiring the entire mapping to be learned from scratch. The skip connections also allow the gradient to flow more freely through the network, mitigating the vanishing gradient problem. The key innovation introduced by Res-Net is the use of identity mappings, which ensure that the gradient flows freely through the network. This is achieved through a new building block called a "residual block," which includes two convolutional layers with batch normalization, a ReLU activation, and a skip connection that adds the input to the output block. The residual block allows the network to learn residual functions, which capture the difference between the input and output blocks. By using residual functions, the network can learn to make incremental changes to the input, rather than having to learn the entire mapping from scratch. Overall, Res-Net is an impressive model that has enabled the development of deeper CNN architectures with many layers. By introducing skip connections and residual functions, Res-Net has pushed the boundaries of what's possible in deep learning, and its ideas have since been widely adopted in the field. Thanks to Res-Net, researchers can now train much deeper CNNs, which can achieve even better accuracy on a wide range of computer vision tasks.

2.3. ImageNet

Large datasets are a critical component in machine learning, providing the data needed to train algorithms and evaluate their performance. Some of the most widely known and used datasets include CIFAR, a collection of small image datasets created by the Canadian Institute for Advanced Research, MNIST [6] which consists of hand-written digits from 0 to 9 in black and white and ImageNet [7] one of the most groundbreaking datasets in the field of computer vision. ImageNet contains over 14 million images and more than 20,000 classes, making it one of the largest datasets in existence. The paper detailing the dataset was published in 2009 by Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei-Fei Li. It has since been cited over 52,000 times by papers in the field of machine learning and computer science in general, reflecting its profound impact on the field. Numerous well-known machine learning algorithms, such as AlexNet [8], VGG [9], ResNet [5], EfficientNet [10], and many others, have been trained on the ImageNet dataset. The dataset is hierarchically structured based on the WordNet hierarchy, a database of English words. Each category corresponds to a synset that is a set of one or more synonyms or phrases in WordNet, which provides a more organized structure to the dataset. To create the dataset, the authors first had a team of humans create labels for a large number of photos, and then used algorithms that were trained with the subset of data that the humans labeled to assign labels automatically. This combination of human and machine input ensured both accuracy and scalability. ImageNet is an essential tool for benchmarking and training machine learning models and has played a significant role in advancing the field of computer vision. Its hierarchical organization based on WordNet and large scale have made it an indispensable resource for researchers and practitioners in the field of artificial intelligence.

2.4. Transfer Learning

Transfer learning is a powerful technique that has gained increasing attention in the field of machine learning. Its ability to leverage pre-existing knowledge from one problem domain and apply it to another has the potential to greatly improve the performance of machine learning models in various domains. To apply transfer learning, a pre-trained model is used to extract general features and patterns from a large dataset. This model is then adapted to a new problem domain to improve its performance on a new, smaller dataset. By reusing the knowledge learned from the pre-trained model, transfer learning can help overcome the limitations of small datasets and reduce the computational costs of training new models from scratch. One popular approach to transfer learning is model-based transfer learning, which involves using the pre-trained model as a starting point for the new model, and then fine-tuning the weights of the pre-trained model on the new dataset. In this way, the pre-trained model provides a head start for the new model, helping it to learn more quickly and effectively. This project has employed model-based transfer learning to develop a cat classification model. The pre-trained ResNet-50 model has being used, which had been trained on a large dataset of images and fine-tuned its weights on a smaller dataset of cat images. By leveraging the pre-existing knowledge of the ResNet-50 model, my cat classification model was able to achieve high accuracy with a relatively small amount of training data. Overall, transfer learning has significant potential for improving the performance of machine learning models in various domains. It can help overcome the limitations of small datasets, reduce the computational costs of training new models, and speed up the learning process. As such, transfer learning is likely to play an increasingly important role in the development of machine learning applications in the future [11], [12].

2.5. Core Learning Concepts

Creating a machine learning algorithm is a complex process that involves numerous elements. Some of these key components include:

2.5.1. Epochs

These represent the total number of times the learning model will cycle through the entire dataset for both training and evaluation [34].

2.5.2. Batch Size

This term refers to the specific number of training examples after which the model updates its internal parameters, or weights. These gradual updates help enhance the model's predictive accuracy over time [34].

2.5.3. Optimizers

These crucial tools adjust the model's weights after each iteration of the batch size. This adjustment effectively alters the model's output, improving its performance. For the transfer learning model of the project, we have used the Stochastic Gradient Descent (SGD) optimizer [13] due to its speed and superior results. In our specific model, we used the Adam optimizer [14], as it delivers the best performance.

2.5.4. Learning Rate

This is the speed at which the optimizer modifies the model's weights after completing each batch size. It's an essential parameter because a poor choice can either slow down the training process and destabilize it or cause the model to overfit [34].

2.5.5. Loss Function

This metric evaluates the effectiveness of a classification model. Since our models are addressing an image classification problem, we utilize the CrossEntropyLoss algorithm [15]. This algorithm produces a loss score ranging from zero to one, with zero indicating a flawless model. The objective is to minimize this score as much as possible.

2.5.6. Weights

These are the values that the model assigns to each input or feature. They help the model understand the importance of each feature, thus facilitating more accurate predictions based on the image's conveyed features [34].

2.5.7. Convolutional Layers

These are essential components of a Convolutional Neural Network (CNN). They contain the parameters needed for the model to learn. Usually, the size of a convolutional layer is smaller than the actual image, and the image size typically decreases with each successive convolutional layer [16].

$$\text{out}(N_i, C_{\text{out}_j}) = \text{bias}(C_{\text{out}_j}) + \sum_{k=0}^{C_{\text{in}}-1} \text{weight}(C_{\text{out}_j}, k) \star \text{input}(N_i, k)$$

2.1 Convolution Mathematical Formula

2.5.8. ReLU Activations

The Rectified Linear Activation Function (ReLU) [17] introduces non-linearity to the model and addresses the vanishing gradients problem.

$$\text{ReLU}(x) = (x)^+ = \max(0, x)$$

2.2 ReLU Mathematical Formula

2.5.9. Max Pooling

This process selects the most valuable features from the convolutional layers. It helps filter and retain the most crucial features [18].

$$out(N_i, C_j, h, w) = \max_{m=0,\ldots,kH-1} \max_{n=0,\ldots,kW-1} \text{input}(N_i, C_j, \text{stride}[0] \times h + m, \text{stride}[1] \times w + n)$$

2.3 Max Pooling Mathematical Formula

2.5.10. Linear Layer

This layer performs a linear transformation [19] on the input data and learns the weights of the neural network, connecting all the input neurons to the output neuron.

$$: y = xA^T + b$$

2.4 Linear Mathematical Formula

2.5.11. Softmax

The Softmax function [20] transforms a vector of raw scores, or logits, into a vector of probabilities, which is essential for multiclass classification problems.

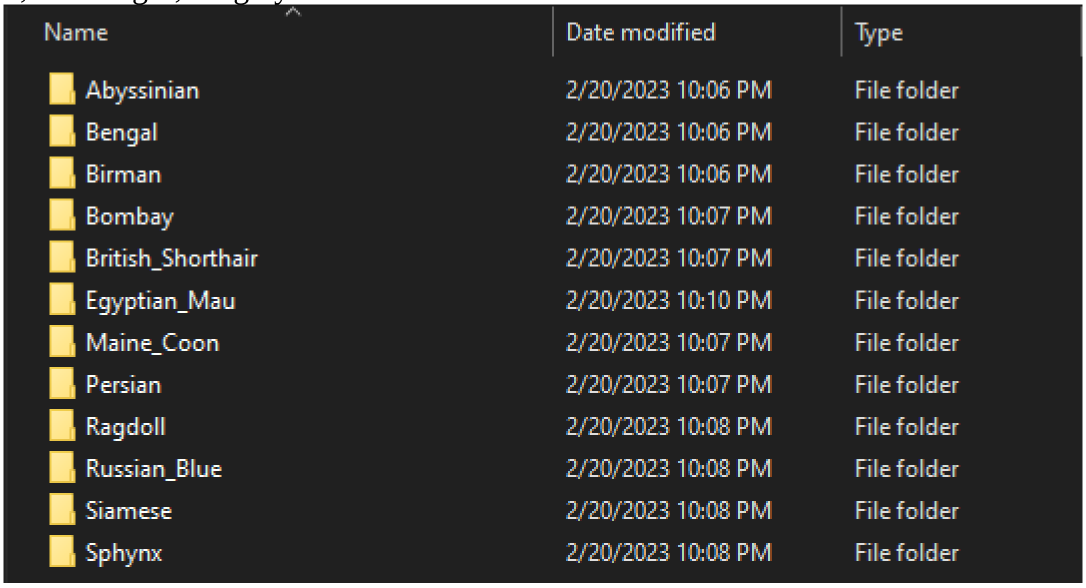
$$\text{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

2.5 Softmax Mathematical Formula

3. Experimental Methodology

3.1. Dataset

The dataset used in this dissertation project is the "Cats and Dogs Breeds Classification Oxford Dataset"[21]. This dataset was created through funding from the UK India Education and Research Initiative (UKIERI) and the ERC Grant VisRec. It consists of 37 categories of pets and its size is roughly 800MB, including both cats and dogs, with 200 images for each category. The dataset provides a large variation of images in terms of scale, pose, and lighting. All the images in the dataset are named after the breed of the cat or dog, followed by a number. To conduct this research, only cat breed images were needed, which accounted for 12 categories and a total of 2,400 images. The cat breed images were distinguished from the dog breed images in the dataset as they started with a capital letter, whereas the dog breed images started with a lower-case letter. To separate the cat breed images from the dataset, a Python script was created. The script performed file manipulation, firstly deleting all the images that started with a lower-case letter to remove the dog breed images. Then, 12 folders were created, each named after one of the cat breeds. Another code block was implemented in Python to move the cat images into their respective breed folders. This was done by deleting the number at the end of each image name and then moving it to the folder with the same name as the remaining part of the image name. Once all the cat breed images were sorted into their respective folders, all the images were opened to ensure they were working correctly. Some images failed to open, however the number of faulty images was relatively low. To prevent an imbalanced dataset, it was decided to delete some photos from each breed category. The breed with the largest number of failed image openings had seven faulty images, therefore seven images were removed from each breed category. After this step, each cat breed category contained 193 images totaling 2,316 images, roughly 200MB.



Name	Date modified	Type
Abyssinian	2/20/2023 10:06 PM	File folder
Bengal	2/20/2023 10:06 PM	File folder
Birman	2/20/2023 10:06 PM	File folder
Bombay	2/20/2023 10:07 PM	File folder
British_Shorthair	2/20/2023 10:07 PM	File folder
Egyptian_Mau	2/20/2023 10:10 PM	File folder
Maine_Coon	2/20/2023 10:07 PM	File folder
Persian	2/20/2023 10:07 PM	File folder
Ragdoll	2/20/2023 10:08 PM	File folder
Russian_Blue	2/20/2023 10:08 PM	File folder
Siamese	2/20/2023 10:08 PM	File folder
Sphynx	2/20/2023 10:08 PM	File folder

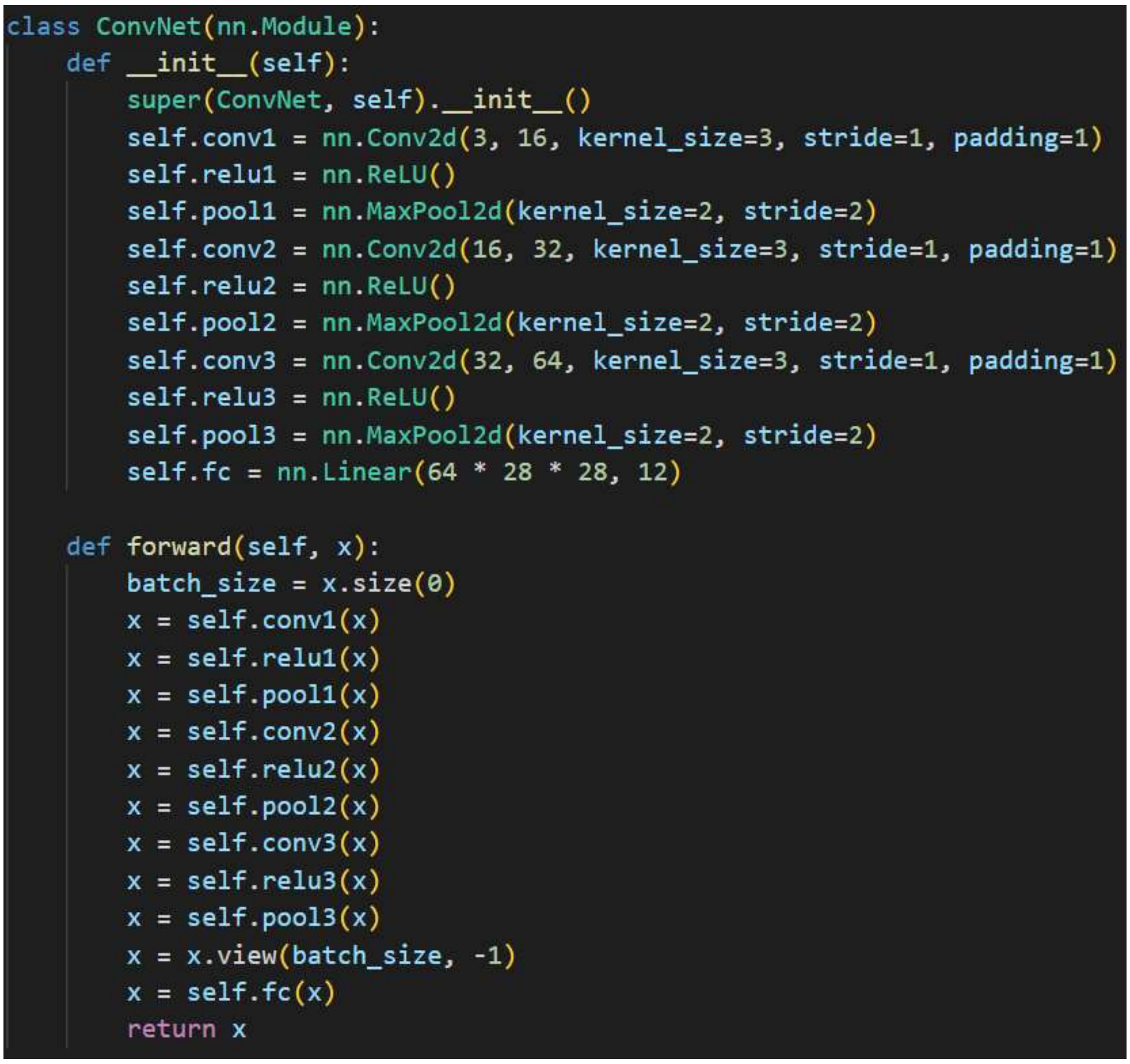
3.1 Dataset File System

3.2 Dataset Sample Cat Image (Abyssinian Breed)

3.3 Dataset Sample Cat Image (Ragdoll Breed)

3.2. Architectures

This project utilizes two machine learning models, each having its distinct architecture. The first architecture is a design I developed myself, and the second one employs the well-established ResNet-50 [5] architecture. Starting with my custom design, it's based on a convolutional neural network (CNN) composed of 10 layers. These include three convolutional layers, which are integral for detecting the various features in the input images, three ReLU (Rectified Linear Unit) activation layers to introduce non-linearity into the network, and three max pooling layers to reduce the spatial dimensions, thus simplifying the computational requirements. The final layer is a fully connected layer which outputs a set of 12 labels, corresponding to the 12 different cat breeds we aim to identify in this project. The model begins with an image input of 224 by 224 pixels, and by the time the image has been processed through all the layers, it has been reduced to a 28 by 28 pixel output.

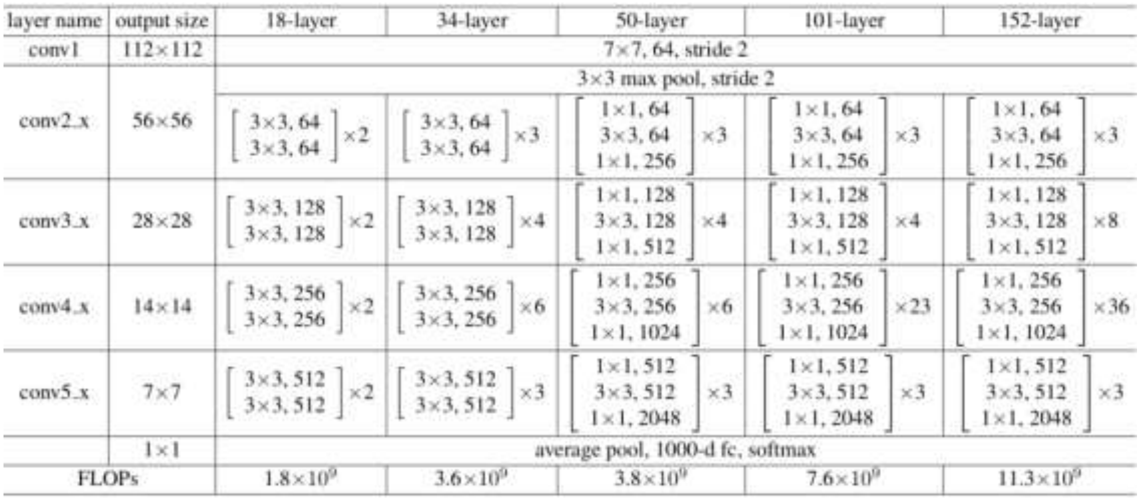


```python
class ConvNet(nn.Module):
    def __init__(self):
        super(ConvNet, self).__init__()
        self.conv1 = nn.Conv2d(3, 16, kernel_size=3, stride=1, padding=1)
        self.relu1 = nn.ReLU()
        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)
        self.conv2 = nn.Conv2d(16, 32, kernel_size=3, stride=1, padding=1)
        self.relu2 = nn.ReLU()
        self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2)
        self.conv3 = nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1)
        self.relu3 = nn.ReLU()
        self.pool3 = nn.MaxPool2d(kernel_size=2, stride=2)
        self.fc = nn.Linear(64 * 28 * 28, 12)

    def forward(self, x):
        batch_size = x.size(0)
        x = self.conv1(x)
        x = self.relu1(x)
        x = self.pool1(x)
        x = self.conv2(x)
        x = self.relu2(x)
        x = self.pool2(x)
        x = self.conv3(x)
        x = self.relu3(x)
        x = self.pool3(x)
        x = x.view(batch_size, -1)
        x = self.fc(x)
        return x
```

3.4 Convolutional Neural Network Architecture (10 Layers)

On the other hand, the ResNet-50 architecture, as the name implies, is composed of 50 layers and is also a convolutional neural network. It starts with an initial convolutional layer, followed by 16 'residual block' layers. Each residual block is unique to ResNet and introduces a 'shortcut' or 'skip connection' which allows the model to refer back to earlier layers, thus aiding in the mitigation of issues like vanishing gradients in deep networks. Every residual block incorporates three convolutional layers, which, when combined with the down sampling process, constitutes 48 layers. The final piece of the architecture is a fully connected layer that outputs 1000 categories, matching the number of labels in the ImageNet dataset on which ResNet is trained. The model begins with an image input of 224 by 224 pixels, and by the time the image has been processed through all the layers, it has been reduced to a 7 by 7 pixel output.

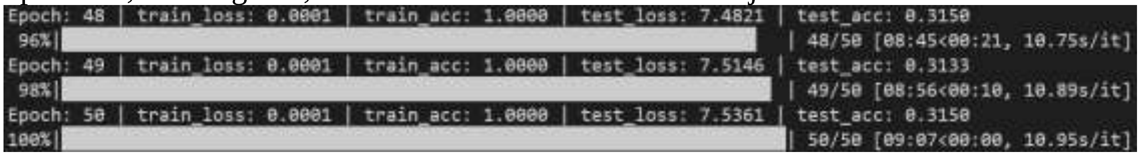


layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	[3×3, 64; 3×3, 64] ×2	[3×3, 64; 3×3, 64] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3
conv3_x	28×28	[3×3, 128; 3×3, 128] ×2	[3×3, 128; 3×3, 128] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×8
conv4_x	14×14	[3×3, 256; 3×3, 256] ×2	[3×3, 256; 3×3, 256] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×23	[1×1, 256; 3×3, 256; 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512; 3×3, 512] ×2	[3×3, 512; 3×3, 512] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

3.5 ResNet Architectures [38]

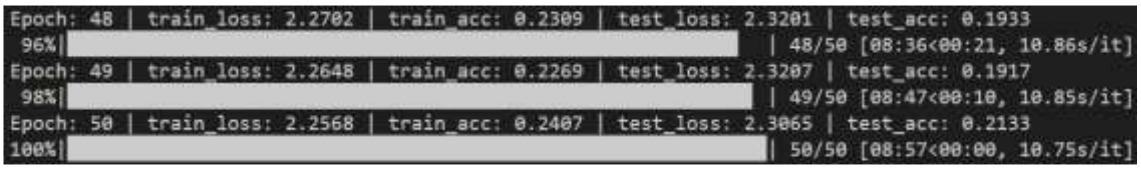
3.3. Training Process and Evaluation

In this study, training and evaluation occur simultaneously. These are two distinct functions invoked within another function, designed to clearly present what's happening. They have differences, for instance, evaluation uses a separate dataset. This approach ensures our model doesn't overfit that means that it avoids learning only from the training images, instead detecting patterns for better identification. Another difference is that during evaluation, the model goes into evaluation mode. This means the model's weights don't change; we use it just for image identification. As training and evaluation happened concurrently, we could monitor each experiment's results in real-time. This allowed us to fine-tune our parameters, ensuring we achieved the best possible outcome. For the training phase, various architectures were tested to determine the best fit for our model. Along with these, adjustments were made to the model's peripherals like the optimizer, learning rate, and batch sizes. Some of these adjustments were:



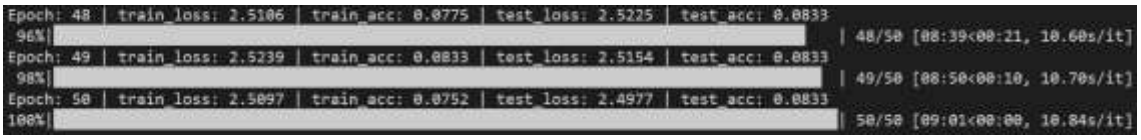
```
Epoch: 48 | train_loss: 0.0001 | train_acc: 1.0000 | test_loss: 7.4821 | test_acc: 0.3150
 96%|                                                              | 48/50 [08:45<00:21, 10.75s/it]
Epoch: 49 | train_loss: 0.0001 | train_acc: 1.0000 | test_loss: 7.5146 | test_acc: 0.3133
 98%|                                                              | 49/50 [08:56<00:10, 10.89s/it]
Epoch: 50 | train_loss: 0.0001 | train_acc: 1.0000 | test_loss: 7.5361 | test_acc: 0.3150
100%|                                                              | 50/50 [09:07<00:00, 10.95s/it]
```

3.6 10 Layer CNN, Adam Optimizer 0.001 Learning Rate



```
Epoch: 48 | train_loss: 2.2702 | train_acc: 0.2309 | test_loss: 2.3201 | test_acc: 0.1933
 96%|                                                              | 48/50 [08:36<00:21, 10.86s/it]
Epoch: 49 | train_loss: 2.2648 | train_acc: 0.2269 | test_loss: 2.3207 | test_acc: 0.1917
 98%|                                                              | 49/50 [08:47<00:10, 10.85s/it]
Epoch: 50 | train_loss: 2.2568 | train_acc: 0.2407 | test_loss: 2.3065 | test_acc: 0.2133
100%|                                                              | 50/50 [08:57<00:00, 10.75s/it]
```

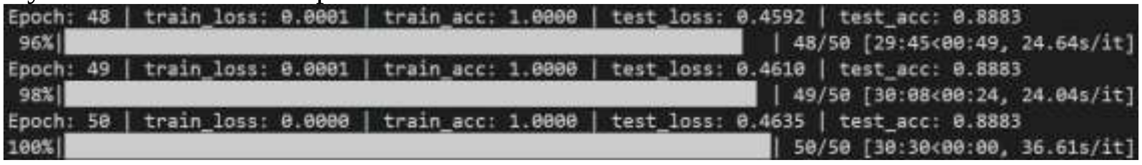
3.7 10 Layer CNN, SDG Optimizer 0.001 Learning Rate



```
Epoch: 48 | train_loss: 2.5106 | train_acc: 0.0775 | test_loss: 2.5225 | test_acc: 0.0833
 96%|                                                              | 48/50 [08:39<00:21, 10.60s/it]
Epoch: 49 | train_loss: 2.5239 | train_acc: 0.0833 | test_loss: 2.5154 | test_acc: 0.0833
 98%|                                                              | 49/50 [08:50<00:10, 10.70s/it]
Epoch: 50 | train_loss: 2.5097 | train_acc: 0.0752 | test_loss: 2.4977 | test_acc: 0.0833
100%|                                                              | 50/50 [09:01<00:00, 10.84s/it]
```

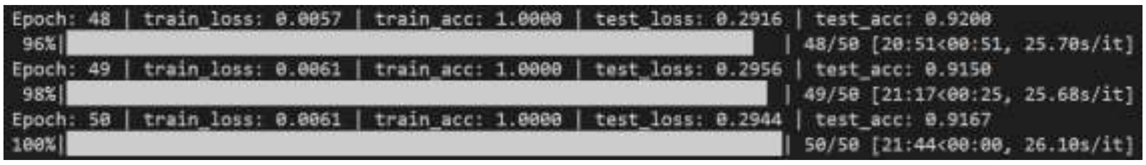
3.8 10 Layer CNN, SDG Optimizer 0.01 Learning Rate

Looking at the 'test_acc' field in our results, we notice varying degrees of accuracy across the three models we have as sample. The most effective model and the one we decided to use in our final product achieved an accuracy rate of 31.5%. This means that out of every 100 predictions made by this model, approximately 31 are correct, which indicates a robust level of performance in this complicated classification problem. When we worked with the ResNet-50, we couldn't alter its architecture. So, we modified only its final layer to output 12 classes instead of the original 1000. Further tinkering happened on the peripherals, similar to the adjustments made on our own model. The only difference was that we alternated between freezing and unfreezing the model's layers. Some of these experiments were:



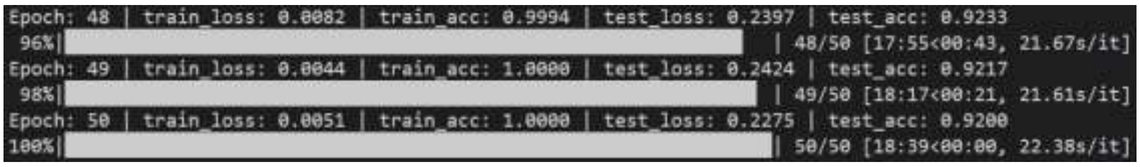
```
Epoch: 48 | train_loss: 0.0001 | train_acc: 1.0000 | test_loss: 0.4592 | test_acc: 0.8883
 96%|██████████████████████████████████████████    | 48/50 [29:45<00:49, 24.64s/it]
Epoch: 49 | train_loss: 0.0001 | train_acc: 1.0000 | test_loss: 0.4610 | test_acc: 0.8883
 98%|███████████████████████████████████████████   | 49/50 [30:08<00:24, 24.04s/it]
Epoch: 50 | train_loss: 0.0000 | train_acc: 1.0000 | test_loss: 0.4635 | test_acc: 0.8883
100%|██████████████████████████████████████████████| 50/50 [30:30<00:00, 36.61s/it]
```

3.9 50 Epochs, Adam Optimizer



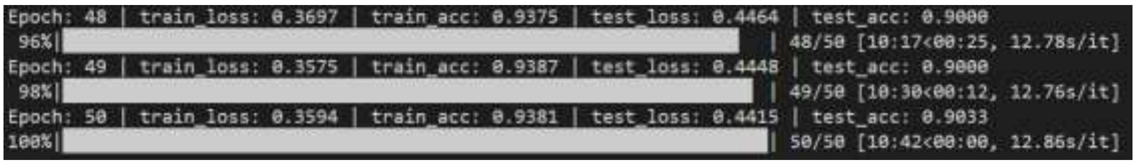
```
Epoch: 48 | train_loss: 0.0057 | train_acc: 1.0000 | test_loss: 0.2916 | test_acc: 0.9200
 96%|██████████████████████████████████████████    | 48/50 [20:51<00:51, 25.70s/it]
Epoch: 49 | train_loss: 0.0061 | train_acc: 1.0000 | test_loss: 0.2956 | test_acc: 0.9150
 98%|███████████████████████████████████████████   | 49/50 [21:17<00:25, 25.68s/it]
Epoch: 50 | train_loss: 0.0061 | train_acc: 1.0000 | test_loss: 0.2944 | test_acc: 0.9167
100%|██████████████████████████████████████████████| 50/50 [21:44<00:00, 26.10s/it]
```

3.10 50 Epochs, SDG Optimizer, Momentum False



```
Epoch: 48 | train_loss: 0.0082 | train_acc: 0.9994 | test_loss: 0.2397 | test_acc: 0.9233
 96%|██████████████████████████████████████████    | 48/50 [17:55<00:43, 21.67s/it]
Epoch: 49 | train_loss: 0.0044 | train_acc: 1.0000 | test_loss: 0.2424 | test_acc: 0.9217
 98%|███████████████████████████████████████████   | 49/50 [18:17<00:21, 21.61s/it]
Epoch: 50 | train_loss: 0.0051 | train_acc: 1.0000 | test_loss: 0.2275 | test_acc: 0.9200
100%|██████████████████████████████████████████████| 50/50 [18:39<00:00, 22.38s/it]
```

3.11 50 Epochs, SDG Optimizer, Momentum True



```
Epoch: 48 | train_loss: 0.3697 | train_acc: 0.9375 | test_loss: 0.4464 | test_acc: 0.9000
 96%|██████████████████████████████████████████    | 48/50 [10:17<00:25, 12.78s/it]
Epoch: 49 | train_loss: 0.3575 | train_acc: 0.9387 | test_loss: 0.4448 | test_acc: 0.9000
 98%|███████████████████████████████████████████   | 49/50 [10:30<00:12, 12.76s/it]
Epoch: 50 | train_loss: 0.3594 | train_acc: 0.9381 | test_loss: 0.4415 | test_acc: 0.9033
100%|██████████████████████████████████████████████| 50/50 [10:42<00:00, 12.86s/it]
```

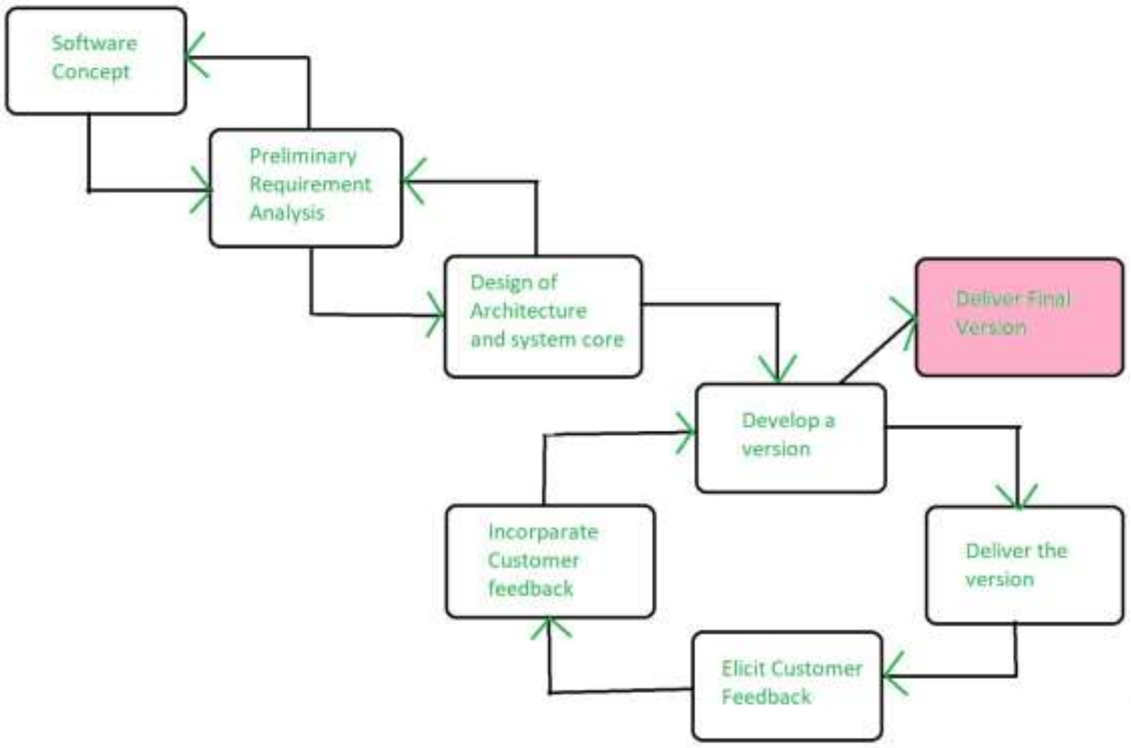
3.12 50 Epochs, SDG Optimizer, Frozen Layers

Looking at the four different test instances, it's clear that the ResNet architecture delivers consistently strong results, with accuracy rates that are relatively close to each other. What sets them apart is the duration of their training times, which vary from as short as 10 minutes to as long as 30 minutes. Impressively, the top-performing model achieved an outstanding accuracy rate of 92%. This highlights the significant potential and power of such machine learning algorithms.

4. Application Development

4.1. Software Development Process

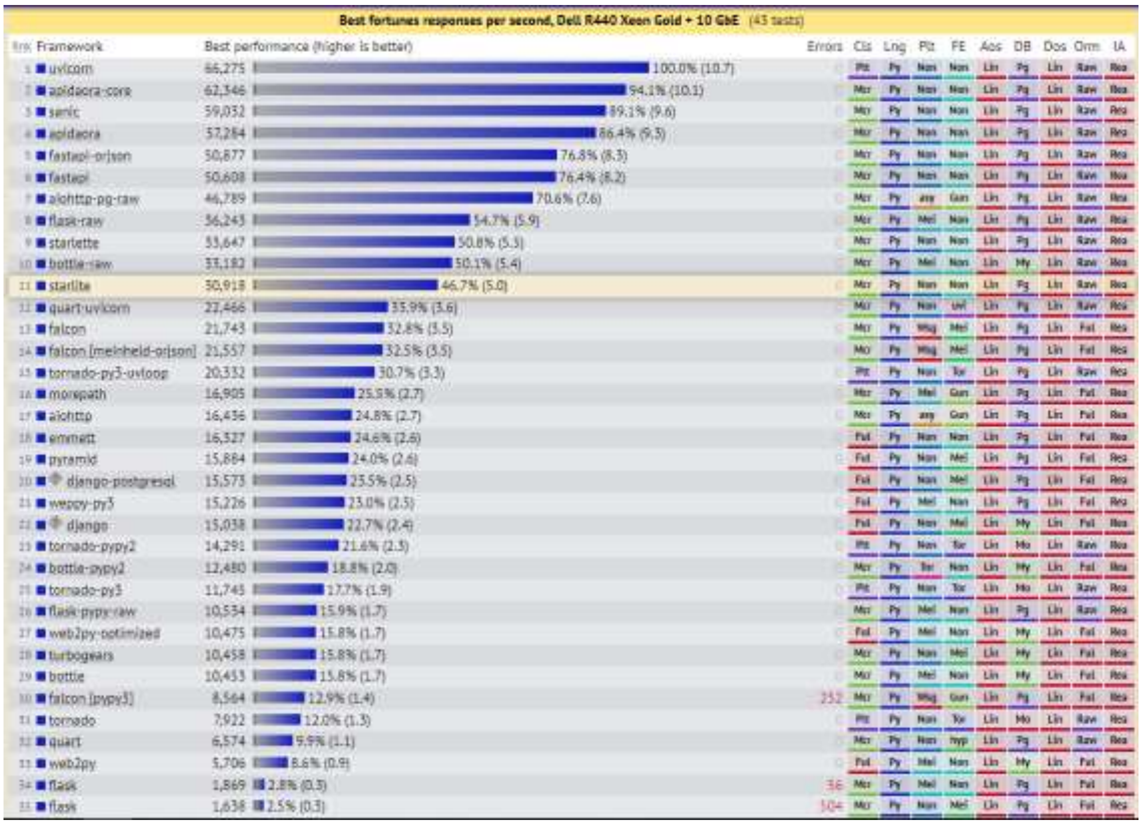
In the development of this project, we employed the evolutionary software development model. This model is a hybrid of the iterative and incremental models, implying that the project doesn't culminate in a single release. Instead, it undergoes numerous minor releases leading up to the final one. Adopting this model requires certain prerequisites. First, a clear and precise understanding of the project's end goal is essential to avoid significant alterations during the production phase. Second, this approach carries a certain degree of risk due to the continuous feedback loop with the client, necessitating a high standard of work to meet client expectations. Lastly, this model is particularly beneficial when working with unfamiliar technology, as it allows for a learning curve. The model's cycle begins with a thorough understanding of the specifications, followed by the design of the core system. The next step involves developing a version, delivering it to the client in this case, the supervisor and receiving feedback. This feedback is then incorporated into the project. If the project is complete, it is published; otherwise, the final step is repeated. The primary advantage of this approach is the establishment of a foundational model, which is then progressively built upon. This method allows for continuous modifications based on the supervisor's feedback, while also providing time for familiarization with the tools used in the project.


Software Concept
Preliminary Requirement Analysis
Design of Architecture and system core
Deliver Final Version
Develop a version
Incorparate Customer feedback
Deliver the version
Elicit Customer Feedback

4.1 Evolutionary Software Development Model Steps [36]

4.2. Tools

The creation of the program for this dissertation involved the development of three key components: the backend, the front-end, and the machine learning model. Each of these components required careful consideration and evaluation to ensure that the program would be effective and efficient. Given that the machine learning model was the cornerstone of the program, and that it was going to be developed using Python, which has a strong library support for machine learning, it was decided to use Python for the entire stack. This choice was made after careful consideration of other options, and it was deemed that Python was the most suitable language for the task at hand. When it came to selecting the machine learning framework, PyTorch and TensorFlow Keras were the two options that were considered. PyTorch was created by Meta on September 1, 2016 [22], and TensorFlow was created by Google on November 9, 2015 [23]. Both of these frameworks are widely used and highly regarded in the field of AI with both of them having more than 55,000 [24], [25] stars on GitHub, with both of them having strengths and weaknesses. After conducting research and consulting with the project supervisor, it was decided that PyTorch was the better choice for the program. This decision was based on several factors, including the fact that PyTorch is gaining momentum and is more future proof than TensorFlow Keras. Additionally, PyTorch is widely used by Facebook, with 93% of the models they trained in 2021 using PyTorch [26]. For the backend, two tools were considered: FastAPI and Flask. Flask has been around since April 16, 2010 [27], and has a large and active community behind it with more than 62,000 [28] stars on GitHub, while FastAPI is a newer framework created on December 5, 2018 [29], that has gained popularity due to its speed and ease of use and has more than 57,000 [30] stars on GitHub. After careful consideration and consultation with the project supervisor, FastAPI was selected as the better choice for the program. While Flask has a larger community, the speed and popularity of FastAPI were deemed more important for the project.



Best fortunes responses per second, Dell R440 Xeon Gold + 10 GbE (43 tests)

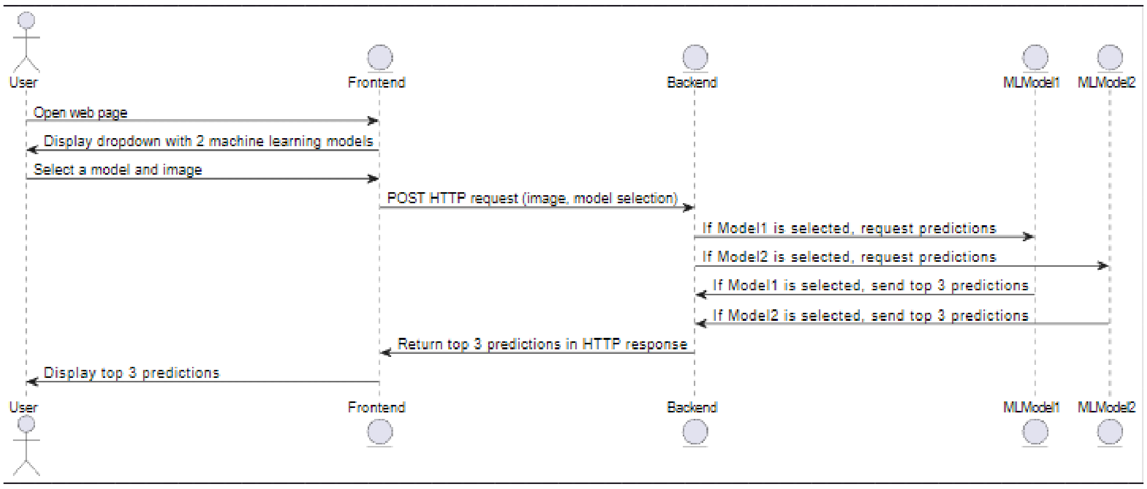
Rnk	Framework	Best performance (higher is better)		Errors	Cls	Lng	Plt	FE	Aos	DB	Dos	Orm	IA
1	uvicorn	66,275	100.0% (10.7)		Plt	Py	Non	Non	Lin	Pg	Lin	Raw	Rea
2	apidaora-core	62,346	94.1% (10.1)		Mcr	Py	Non	Non	Lin	Pg	Lin	Raw	Rea
3	sanic	59,032	89.1% (9.6)		Mcr	Py	Non	Non	Lin	Pg	Lin	Raw	Rea
4	apidaora	57,284	86.4% (9.3)		Mcr	Py	Non	Non	Lin	Pg	Lin	Raw	Rea
5	fastapi-orjson	50,877	76.8% (8.3)		Mcr	Py	Non	Non	Lin	Pg	Lin	Raw	Rea
6	fastapi	50,608	76.4% (8.2)		Mcr	Py	Non	Non	Lin	Pg	Lin	Raw	Rea
7	aiohttp-pg-raw	46,789	70.6% (7.6)		Mcr	Py	asy	Gun	Lin	Pg	Lin	Raw	Rea
8	flask-raw	36,243	54.7% (5.9)		Mcr	Py	Mei	Non	Lin	Pg	Lin	Raw	Rea
9	starlette	33,647	50.8% (5.3)		Mcr	Py	Non	Non	Lin	Pg	Lin	Raw	Rea
10	bottle-raw	33,182	50.1% (5.4)		Mcr	Py	Mei	Non	Lin	My	Lin	Raw	Rea
11	starlite	30,918	46.7% (5.0)		Mcr	Py	Non	Non	Lin	Pg	Lin	Raw	Rea
12	quart-uvicorn	22,466	33.9% (3.6)		Mcr	Py	Non	Uvi	Lin	Pg	Lin	Raw	Rea
13	falcon	21,743	32.8% (3.5)		Mcr	Py	Wsg	Mei	Lin	Pg	Lin	Ful	Rea
14	falcon [meinheld-orjson]	21,557	32.5% (3.5)		Mcr	Py	Wsg	Mei	Lin	Pg	Lin	Ful	Rea
15	tornado-py3-uvloop	20,332	30.7% (3.3)		Plt	Py	Non	Tor	Lin	Pg	Lin	Raw	Rea
16	morepath	16,905	25.5% (2.7)		Mcr	Py	Mei	Gun	Lin	Pg	Lin	Ful	Rea
17	aiohttp	16,436	24.8% (2.7)		Mcr	Py	asy	Gun	Lin	Pg	Lin	Ful	Rea
18	emmett	16,327	24.6% (2.6)		Ful	Py	Non	Non	Lin	Pg	Lin	Ful	Rea
19	pyramid	15,884	24.0% (2.6)		Ful	Py	Non	Mei	Lin	Pg	Lin	Ful	Rea
20	django-postgresql	15,573	23.5% (2.5)		Ful	Py	Non	Mei	Lin	Pg	Lin	Ful	Rea
21	weppy-py3	15,226	23.0% (2.5)		Ful	Py	Mei	Non	Lin	Pg	Lin	Ful	Rea
22	django	15,038	22.7% (2.4)		Ful	Py	Non	Mei	Lin	My	Lin	Ful	Rea
23	tornado-pypy2	14,291	21.6% (2.3)		Plt	Py	Non	Tor	Lin	Mo	Lin	Raw	Rea
24	bottle-pypy2	12,480	18.8% (2.0)		Mcr	Py	Tor	Non	Lin	My	Lin	Ful	Rea
25	tornado-py3	11,745	17.7% (1.9)		Plt	Py	Non	Tor	Lin	Mo	Lin	Raw	Rea
26	flask-pypy-raw	10,534	15.9% (1.7)		Mcr	Py	Mei	Non	Lin	Pg	Lin	Raw	Rea
27	web2py-optimized	10,475	15.8% (1.7)		Ful	Py	Mei	Non	Lin	My	Lin	Ful	Rea
28	turbogears	10,458	15.8% (1.7)		Mcr	Py	Non	Mei	Lin	My	Lin	Ful	Rea
29	bottle	10,453	15.8% (1.7)		Mcr	Py	Mei	Non	Lin	My	Lin	Ful	Rea
30	falcon [pypy3]	8,564	12.9% (1.4)	252	Mcr	Py	Wsg	Gun	Lin	Pg	Lin	Ful	Rea
31	tornado	7,922	12.0% (1.3)		Plt	Py	Non	Tor	Lin	Mo	Lin	Raw	Rea
32	quart	6,574	9.9% (1.1)		Mcr	Py	Non	Hyp	Lin	Pg	Lin	Raw	Rea
33	web2py	5,706	8.6% (0.9)		Ful	Py	Mei	Non	Lin	My	Lin	Ful	Rea
34	flask	1,869	2.8% (0.3)	36	Mcr	Py	Mei	Non	Lin	Pg	Lin	Ful	Rea
35	flask	1,638	2.5% (0.3)	104	Mcr	Py	Non	Mei	Lin	Pg	Lin	Ful	Rea

4.2 REST API Tools Benchmark [37]

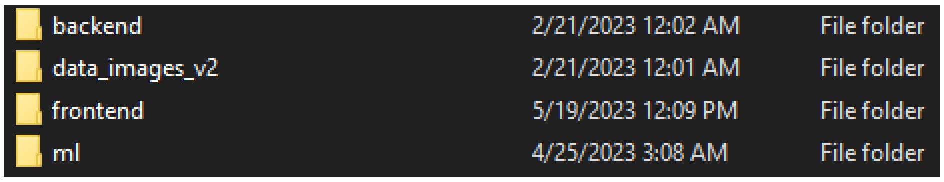
Finally, the front-end component required a tool that was easy to use and visually appealing. After exploring various options and consulting with the supervisor, Streamlit was selected as the best choice. Streamlit was created on April 26, 2019 [31], and has more than 24,000 [32] stars on GitHub. It is a simple and elegant front-end tool that meets all the requirements for the project. It was chosen because of its ease of use, its visual appeal, and its ability to showcase the machine learning model effectively.

4.3. System Architecture

Our system is comprised of three significant sections: one located on the user-facing front-end and two on the server-side back-end. These elements work together seamlessly, initiating a series of interconnected events when a user interacts with the main button on the front-end interface. Specifically, when this button is clicked, the front-end sends an image and the model selection to the back-end via a POST HTTP request to our REST API. In return, the system provides the top three prediction results produced by the selected model. This interaction ensures a smooth and interactive user experience while harnessing the power of our machine learning algorithm.


User
Frontend
Backend
MLModel1
MLModel2
Open web page
Display dropdown with 2 machine learning models
Select a model and image
POST HTTP request (image, model selection)
If Model1 is selected, request predictions
If Model2 is selected, request predictions
If Model1 is selected, send top 3 predictions
If Model2 is selected, send top 3 predictions
Return top 3 predictions in HTTP response
Display top 3 predictions

4.3 User Interaction and ML Model Selection Sequence Diagram



backend	2/21/2023 12:02 AM	File folder
data_images_v2	2/21/2023 12:01 AM	File folder
frontend	5/19/2023 12:09 PM	File folder
ml	4/25/2023 3:08 AM	File folder

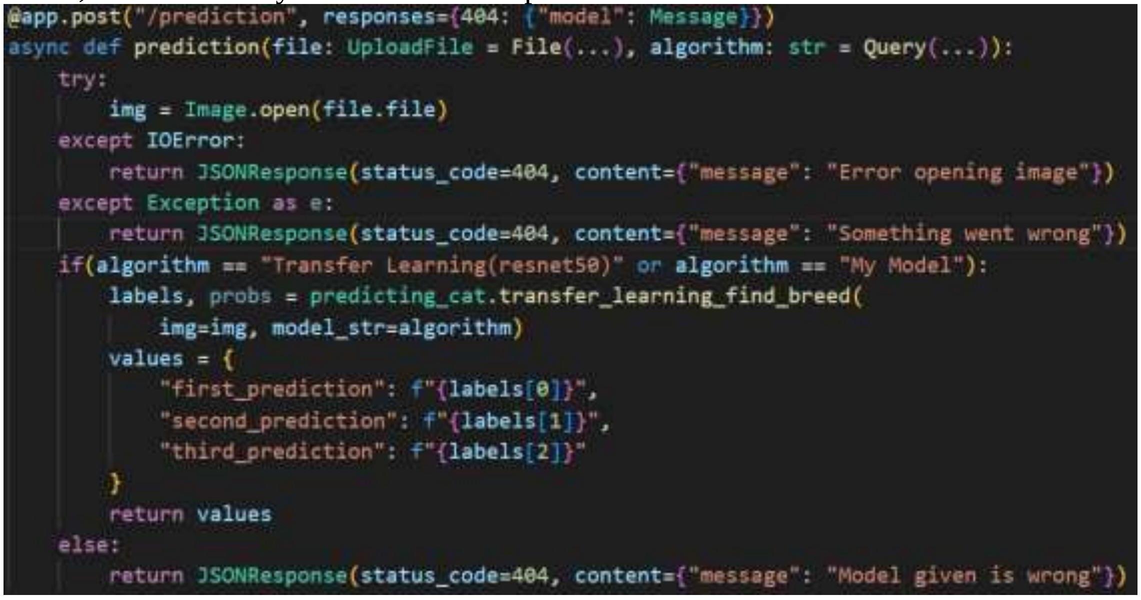
4.4 Project File System

4.4. Machine Learning Service

The machine learning component of this project was organized within five main files, each having distinct roles and functionalities. The first file, "edit_data", contains scripts responsible for managing the database. This file handles tasks such as deleting undesirable images and sorting the remaining images into folders named after the respective cat breeds. The second and third files, "cat_cnn" and "transfer_learning_cat", are dedicated to model creation. These files begin by setting all necessary parameters and loading the images, which are resized to 224 by 224 pixels and labeled appropriately. A total of 144 images are designated for training, and 50 for evaluation, both sets being randomly sampled. The image data is then fed to a DataLoader, configured with batch size parameters and shuffling enabled. Both these files also list the 12 recognizable breeds and a save parameter that preserves the model's state after training for future usage. The "cat_cnn" file is programmed to create a convolutional neural network model comprising 10 layers, which is then dispatched to the GPU for expedited computations. A cross-entropy loss function is incorporated for image classification, followed by the utilization of an SGD optimizer. The training and evaluation loops are triggered through a function call to another file. The "transfer_learning_cat" file largely mirrors the structure of "cat_cnn", with the key distinction being its approach to model creation. Instead of manually constructing layers, this file procures pre-trained weights and layers from PyTorch [33], freezes them to modify the final layer from 1000 (the number of ImageNet labels) to 12 (the number of our project's labels), and then unfreezes them. The fourth file, "train_models", comprises three key functions. The first one is a wrapper function that invokes the remaining two functions while outputting appropriate messages. The second function cycles through the specified number of epochs to train our model, and the third function examines the testing data to evaluate the model's accuracy. Finally, the "predicting_cat" file contains a single function that is invoked by the REST API. This function examines an input image and offers the top three breed predictions for the depicted cat.

4.5. REST API Service

This project involved the creation of a REST API (back-end), which, while not initially required, was ultimately deemed beneficial after consultation with the project supervisor. The decision to establish an API was driven by three primary considerations: it facilitated cleaner, more readable code, allowing for easy maintenance and changes to the front-end, it provided a ready-to-use endpoint for any user interested in employing this API in their own applications, and it made the application scalable meaning that if we have more traffic and we need to scale the application we can scale the parts of the application as needed. To construct the REST API, we utilized FastAPI, a python library known for its speed and ease of use. This library simplifies the back-end development process, making it more accessible. The developed API includes two endpoints that operate on port 8000 and are hosted locally. The first endpoint is the default one, accessible at (/). It employs a GET method and returns a welcoming message - "Welcome to FastAPI". The second endpoint, found under (/prediction), uses a POST method, and accepts two input parameters: a file containing an image and a string specifying the model to be utilized. Upon receiving these inputs, a method dedicated to cat breed prediction is invoked, using the supplied image and model to generate predictions. This method then returns the top three predictions in the form of labels, which are relayed back to the endpoint as in a JSON format.

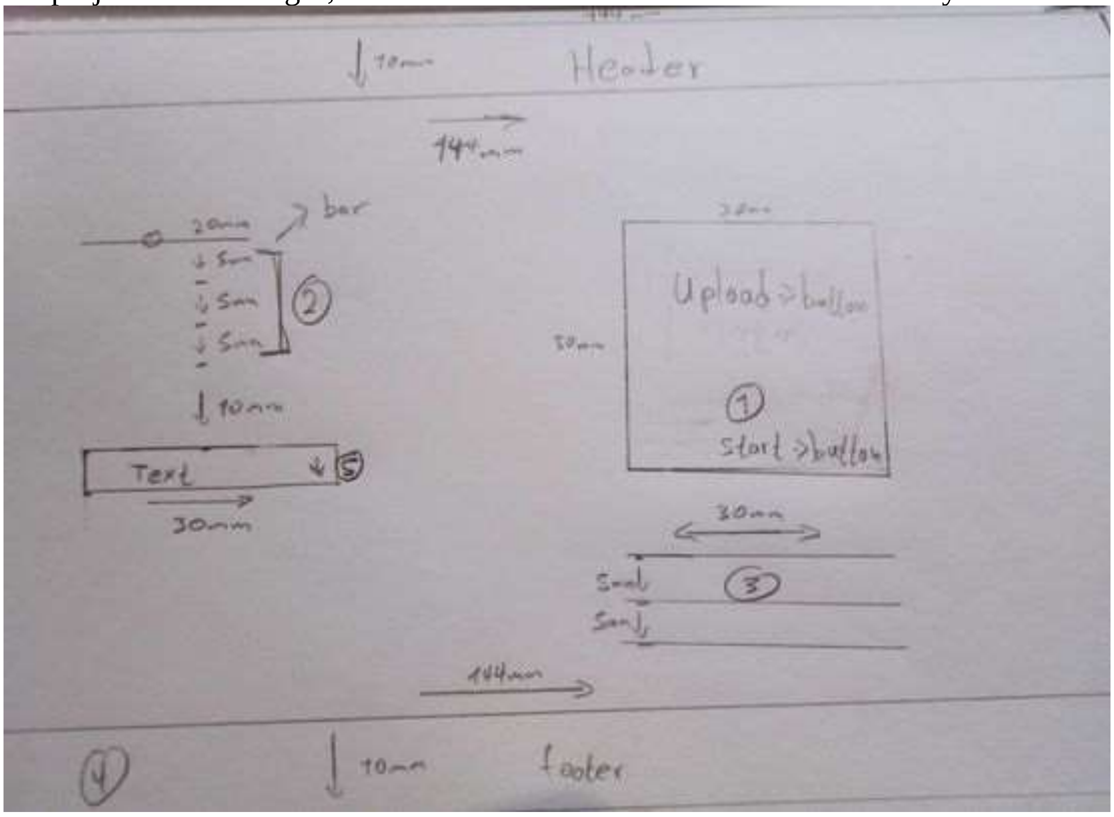


```python
@app.post("/prediction", responses={404: {"model": Message}})
async def prediction(file: UploadFile = File(...), algorithm: str = Query(...)):
    try:
        img = Image.open(file.file)
    except IOError:
        return JSONResponse(status_code=404, content={"message": "Error opening image"})
    except Exception as e:
        return JSONResponse(status_code=404, content={"message": "Something went wrong"})
    if(algorithm == "Transfer Learning(resnet50)" or algorithm == "My Model"):
        labels, probs = predicting_cat.transfer_learning_find_breed(
            img=img, model_str=algorithm)
        values = {
            "first_prediction": f"{labels[0]}",
            "second_prediction": f"{labels[1]}",
            "third_prediction": f"{labels[2]}"
        }
        return values
    else:
        return JSONResponse(status_code=404, content={"message": "Model given is wrong"})
```

4.5 /prediction endpoint

4.6. Web UI Service

This project necessitated the creation of an intuitive and stylish web user interface (UI) to demonstrate the capabilities of the developed AI model. The user-friendly front-end was facilitated by leveraging Streamlit, a Python library known for assisting in crafting aesthetically pleasing UIs. Despite its limited customization options, Streamlit offers a robust suite of features, including import widgets, media elements, layouts, and containers. In this project Streamlit runs on port 8501 it is hosted locally and there is a file created named config.toml that contains some settings for this particular project. In the project's initial stages, I sketched an outline to visualize the desired layout of the UI.



4.6 Initial Hand Drawn UI Layout

The sketch included a welcoming header, and the page was split into two segments. The left segment of the page initially housed a bar, designed to accommodate sliders allowing users to train the dataset directly from the website. However, this feature was too demanding in terms of GPU usage and resulted in extended waiting periods (approximately 30-60 minutes) for model updates. Therefore, this option was discarded. The left side of the UI also featured a dropdown box to select pre-trained models. Incorporated into the final design, this dropdown now offers a choice between two models, enabling the user to identify cat breeds based on their preferred model. The right-hand side of the initial design proposed a large box for image upload. However, an existing Streamlit widget offering similar functionality was adopted instead. This rectangular widget allows users to browse and select their desired photo. Beneath the image widget, the sketch also displayed text lines indicating the top three predictions, a feature implemented in the final design. The final UI was enhanced with three additional components. The first is a text display on the left side of the UI above the dropdown, detailing the breeds that the models can recognize. Secondly, a button has been placed beneath the dropdown, redirecting users to the project's GitHub repository. This repository houses the program files, the dissertation paper, and a readme file explaining the project's objectives and outcomes. The final two additions reside on the right side of the UI below the image widget: one is a button for users to press once they've selected an image, and the other is a widget displaying the selected image upon pressing the button. This interaction enables users to view the image used for the model's prediction. On pressing the button, the UI communicates with the REST API by sending a file with the image and a string containing the user-selected model. The system responds with a JSON object encapsulating the top three prediction labels.

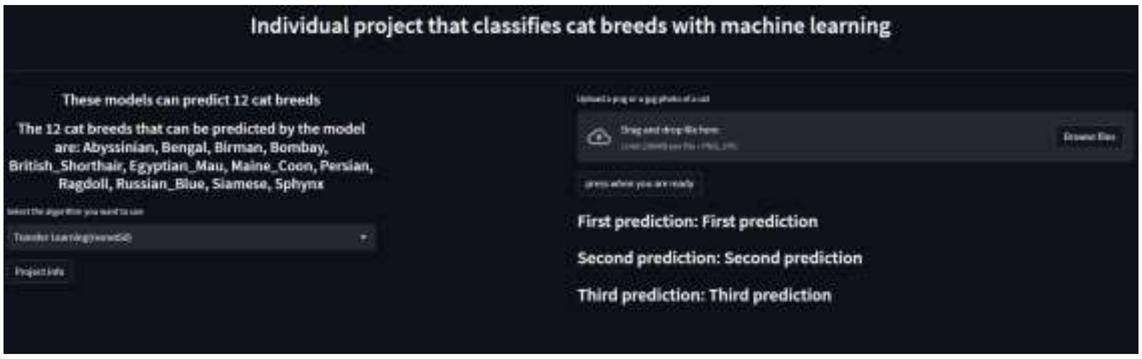
5. Results

5.1. Software Testing

The testing process for each component of this software application was conducted using a hands-on, trial-and-error approach. This means that no specific testing software was used; instead, each component was individually tested through direct interaction and manipulation. The user interface (UI), the front-end of the application, consists of three buttons and a dropdown menu. Streamlit's design ensures that two of the three buttons and the dropdown menu are inherently safe, as they limit the user to expected actions. The first button redirects the user to the project's GitHub repository, the second button enables the user to upload an image in either jpg or png format, and the dropdown menu allows the user to select from two models, preventing any alternative selections. The third button, which triggers the prediction process, requires more careful handling. When pressed, this button initiates an API call and updates the prediction text with the actual predictions. To ensure the safety of this process, several conditions and exceptions have been implemented to catch potential errors and unexpected user actions. For instance, the application checks if the user has uploaded an image before proceeding, and if an API call fails, an appropriate error message is displayed. The backend of the project, which includes the API, was tested using the same trial-and-error method, supplemented with Postman for API call testing. Given that the API is intended for public use, a try-catch statement has been implemented to ensure that the uploaded file is an image and that the input string matches the expected parameters. The machine learning component of the project was tested purely through trial and error. If an error occurred, we would trace its root cause and fix it using print statements and conditional logic to pinpoint the source of the error. This hands-on approach allowed us to thoroughly test and refine each component of the software application.

5.2. Final Application

The final output of this project comprises a user-friendly web interface, a responsive REST API in the back-end, and two functioning models also located in the back-end. The interaction process is straightforward the user uploads an image via the interface and then clicks the prediction button to determine the breed of the cat featured in the image. This action triggers a request to the back-end, which in turn provides the top three potential cat breeds based on the image. The results are then promptly displayed to the user.


Upload a png or a jpg photo of a cat
Drag and drop file here
Limit 200MB per file • PNG, JPG
Browse files
press when you are ready
Select the algorithm you want to use
Transfer Learning(resnet50)
Project info

Individual project that classifies cat breeds with machine learning

These models can predict 12 cat breeds

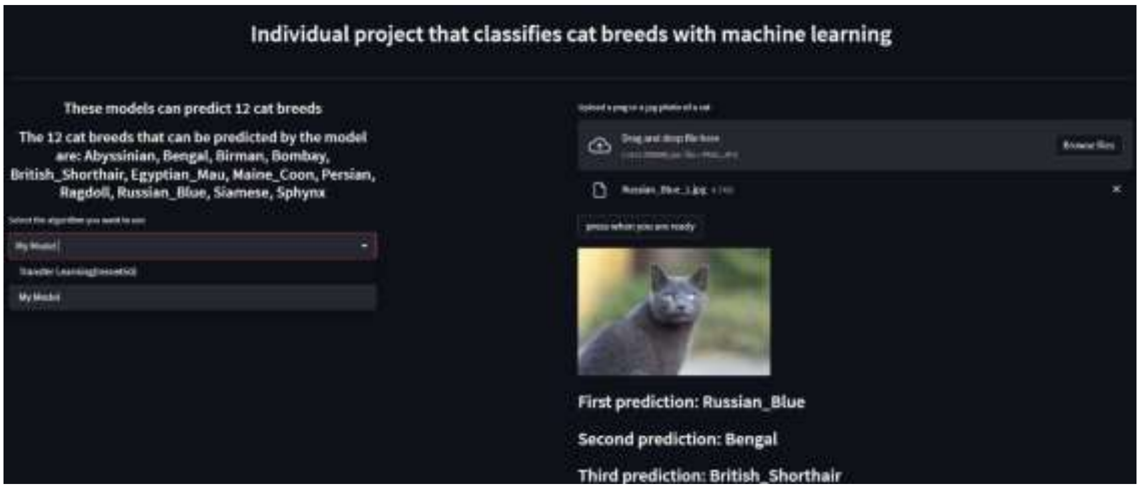
The 12 cat breeds that can be predicted by the model are: Abyssinian, Bengal, Birman, Bombay, British_Shorthair, Egyptian_Mau, Maine_Coon, Persian, Ragdoll, Russian_Blue, Siamese, Sphynx

First prediction: First prediction

Second prediction: Second prediction

Third prediction: Third prediction

5.1 Final Web Landing UI


Upload a png or a jpg photo of a cat
Drag and drop file here
Limit 200MB per file • PNG, JPG
Browse files
Russian_Blue_1.jpg
press when you are ready
Select the algorithm you want to use
My Model
Transfer Learning(resnet50)
My Model

Individual project that classifies cat breeds with machine learning

These models can predict 12 cat breeds

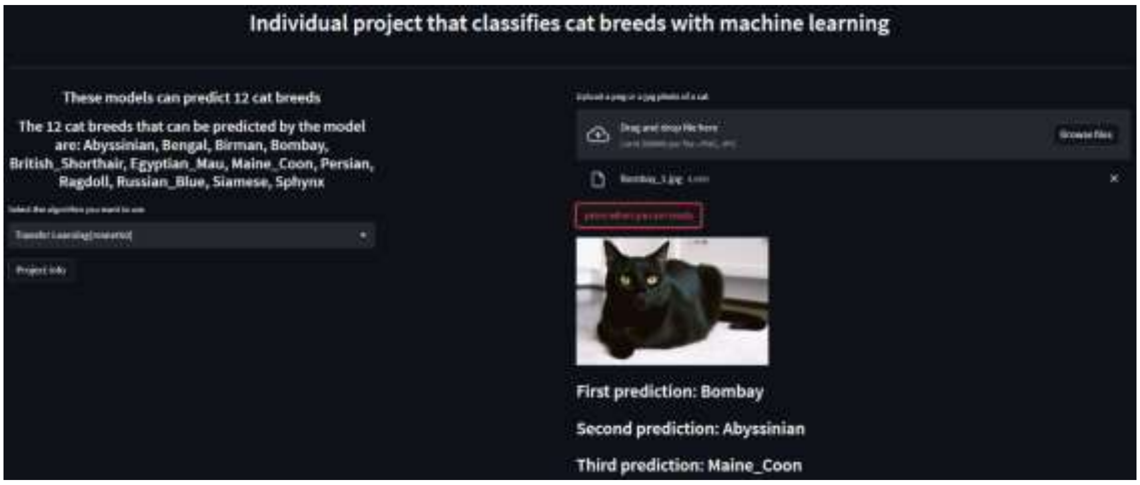
The 12 cat breeds that can be predicted by the model are: Abyssinian, Bengal, Birman, Bombay, British_Shorthair, Egyptian_Mau, Maine_Coon, Persian, Ragdoll, Russian_Blue, Siamese, Sphynx

First prediction: Russian_Blue

Second prediction: Bengal

Third prediction: British_Shorthair

5.2 My Model Cat Breed Prediction


Upload a png or a jpg photo of a cat
Drag and drop file here
Limit 200MB per file • PNG, JPG
Browse files
Bombay_1.jpg
press when you are ready
Select the algorithm you want to use
Transfer Learning(resnet50)
Project info

Individual project that classifies cat breeds with machine learning

These models can predict 12 cat breeds

The 12 cat breeds that can be predicted by the model are: Abyssinian, Bengal, Birman, Bombay, British_Shorthair, Egyptian_Mau, Maine_Coon, Persian, Ragdoll, Russian_Blue, Siamese, Sphynx

First prediction: Bombay

Second prediction: Abyssinian

Third prediction: Maine_Coon

5.3 ResNet-50 Cat Breed Prediction

6. Conclusions

6.1. Main Conclusions

To conclude, this research offers a strong method for telling cat breeds apart using a carefully adjusted machine learning model. Using the revised dataset called "Cats and Dogs Breeds Classification Oxford Dataset" we could achieve high accuracy numbers in the cat breed classification problem. Our model hitting a 31.5% accuracy and the ResNet model using the transfer learning technique hitting a 92% accuracy. It also provides a basic guide to understanding artificial intelligence and what's needed to create such a system. This development opens up new possibilities for using artificial intelligence, especially when detailed recognition tasks are needed. However, we should note that if we had access to more powerful graphics processing units (GPUs), the model could improve even more. This additional power would allow us to develop a more intricate artificial neural network, further enhancing our results.

6.2. Future Plan

This project has made significant strides, but there are additional enhancements that could further improve its performance. Firstly, the creation of a more comprehensive dataset, encompassing a greater number of images and cat breeds, could enhance the training process and enable the recognition of a wider variety of breeds. Secondly, optimizing my model further could potentially yield higher performance scores without sacrificing computer power like what the ResNet model could achieve with fifty layers on a mid to high-range computer. Lastly, incorporating real-time training could provide users with the ability to train the model directly via the webpage. This feature would offer a more interactive and dynamic user experience, further enhancing the utility of the project.

7. References

[1] “IBM100 - The Origins of Computer Science.” [Online]. Available: https://www.ibm.com/ibm/history/ibm100/us/en/icons/compsci/. [Accessed: 02-May-2023].

[2] “Some EDSAC statistics.” [Online]. Available: https://www.cl.cam.ac.uk/events/EDSAC99/statistics.html. [Accessed: 02-May-2023].

[3] P. J. Denning *et al.*, “Computing as a discipline,” *Computer (Long. Beach. Calif).*, vol. 22, no. 2, p. 63, 1989.

[4] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2323, 1998.

[5] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, vol. 45, no. 8, pp. 770–778.

[6] L. Deng, “The MNIST database of handwritten digit images for machine learning research,” *IEEE Signal Process. Mag.*, vol. 29, no. 6, pp. 141–142, 2012.

[7] J. Deng, W. Dong, R. Socher, L.-J. Li, Kai Li, and Li Fei-Fei, “ImageNet: A large-scale hierarchical image database,” in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, vol. 20, no. 11, pp. 248–255.

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” *Commun. ACM*, vol. 60, no. 6, pp. 84–90, May 2017.

[9] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *3rd Int. Conf. Learn. Represent. ICLR 2015 - Conf. Track Proc.*, pp. 1–14, 2015.

[10] M. Tan and Q. V. Le, “EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks,” *Open J. Model. Simul.*, vol. 09, no. 03, pp. 259–274, May 2019.

[11] K. Weiss, T. M. Khoshgoftaar, and D. D. Wang, *A survey of transfer learning*, vol. 3, no. 1. Springer International Publishing, 2016.

[12] L. Torrey and J. Shavlik, “Transfer Learning,” in *Handbook of Research on Machine Learning Applications and Trends*, IGI Global, 2010, pp. 242–264.

[13] L. Bottou, “Stochastic gradient descent tricks,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 7700 LECTU, no. 1, pp. 421–436, 2012.

[14] D. P. Kingma and J. L. Ba, “Adam: A method for stochastic optimization,” *3rd*

Int. Conf. Learn. Represent. ICLR 2015 - Conf. Track Proc., pp. 1–15, 2015.

[15] “CrossEntropyLoss — PyTorch 2.0 documentation.” [Online]. Available: https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html. [Accessed: 29-May-2023].

[16] “Conv2d — PyTorch 2.0 documentation.” [Online]. Available: https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html. [Accessed: 29-May-2023].

[17] “ReLU — PyTorch 2.0 documentation.” [Online]. Available: https://pytorch.org/docs/stable/generated/torch.nn.ReLU.html. [Accessed: 29-May-2023].

[18] “MaxPool2d — PyTorch 2.0 documentation.” [Online]. Available: https://pytorch.org/docs/stable/generated/torch.nn.MaxPool2d.html. [Accessed: 29-May-2023].

[19] “Linear — PyTorch 2.0 documentation.” [Online]. Available: https://pytorch.org/docs/stable/generated/torch.nn.Linear.html. [Accessed: 29-May-2023].

[20] “Softmax — PyTorch 2.0 documentation.” [Online]. Available: https://pytorch.org/docs/stable/generated/torch.nn.Softmax.html. [Accessed: 29-May-2023].

[21] “The Oxford-IIIT Pet Dataset.” [Online]. Available: https://www.robots.ox.ac.uk/~vgg/data/pets/. [Accessed: 19-May-2023].

[22] “Release alpha-1 release · pytorch/pytorch · GitHub.” [Online]. Available: https://github.com/pytorch/pytorch/releases/tag/v0.1.1. [Accessed: 06-May-2023].

[23] “Google Just Open Sourced TensorFlow, Its Artificial Intelligence Engine | WIRED.” [Online]. Available: https://www.wired.com/2015/11/google-open-sources-its-artificial-intelligence-engine/. [Accessed: 07-May-2023].

[24] “GitHub - pytorch/pytorch: Tensors and Dynamic neural networks in Python with strong GPU acceleration.” [Online]. Available: https://github.com/pytorch/pytorch. [Accessed: 06-May-2023].

[25] “GitHub - keras-team/keras: Deep Learning for humans.” [Online]. Available: https://github.com/keras-team/keras. [Accessed: 06-May-2023].

[26] “PyTorch builds the future of AI and machine learning at Facebook.” [Online]. Available: https://ai.facebook.com/blog/pytorch-builds-the-future-of-ai-and-machine-learning-at-facebook/. [Accessed: 06-May-2023].

[27] “Flask · PyPI.” [Online]. Available: https://pypi.org/project/Flask/#history. [Accessed: 07-May-2023].

[28] “GitHub - pallets/flask: The Python micro framework for building web

applications.” [Online]. Available: https://github.com/pallets/flask. [Accessed: 07-May-2023].

[29] “Commits · tiangolo/fastapi · GitHub.” [Online]. Available: https://github.com/tiangolo/fastapi/commits/master?after=eddbae948f04e13fe412dc45a569d10e34b698a4+1990&branch=master. [Accessed: 07-May-2023].

[30] “GitHub - tiangolo/fastapi: FastAPI framework, high performance, easy to learn, fast to code, ready for production.” [Online]. Available: https://github.com/tiangolo/fastapi. [Accessed: 07-May-2023].

[31] “Changelog - Streamlit Docs.” [Online]. Available: https://docs.streamlit.io/library/changelog#version-0350. [Accessed: 07-May-2023].

[32] “GitHub - streamlit/streamlit: Streamlit — A faster way to build and share data apps.” [Online]. Available: https://github.com/streamlit/streamlit. [Accessed: 07-May-2023].

[33] “ResNet — Torchvision main documentation.” [Online]. Available: https://pytorch.org/vision/main/models/resnet.html. [Accessed: 31-May-2023].

[34] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[35] “File:AI-ML-DL.svg - Wikimedia Commons.” [Online]. Available: https://commons.wikimedia.org/wiki/File:AI-ML-DL.svg. [Accessed: 29-May-2023].

[36] “Software Engineering | Evolutionary Model - GeeksforGeeks.” [Online]. Available: https://www.geeksforgeeks.org/software-engineering-evolutionary-model/. [Accessed: 01-Jun-2023].

[37] “Round 21 results - TechEmpower Framework Benchmarks.” [Online]. Available: https://www.techempower.com/benchmarks/#section=data-r21&hw=ph&test=fortune&l=zijzen-64f. [Accessed: 01-Jun-2023].

[38] “ResNet-50: The Basics and a Quick Tutorial.” [Online]. Available: https://datagen.tech/guides/computer-vision/resnet-50/. [Accessed: 01-Jun-2023].