Designing a Test System for Pretrained Decision Algorithms

Master of Engineering
Information Technology
Abbad Zafar
1360066
abbad.zafar@stud.fra-uas.de

***Abstract*— Individuals compose the logic that interacts with the data in order to achieve the desired behavior in traditional software-based decision-making systems. They can now be used in automated decision-making systems due to advancements in machine learning (ML) algorithms. However, in terms of how they generate predictions from input data, these algorithms remain a 'black box.' Furthermore, these algorithms require large, high-quality training datasets. In this paper, a GUI-based test system comprised of multiple ML algorithms is developed to evaluate the accuracy and efficiency of various decision algorithms by providing test data.**

Keywords— GUI, algorithms, machine learning algorithms, decision-making systems, accuracy, efficiency, dataset

I. Introduction

For many years, artificial intelligence (AI) was primarily a theoretical field with few applications with real-world impact. This has shifted dramatically over the last decade, as a combination of more powerful machines, improved learning algorithms, and easier access to massive amounts of data enabled advances in Machine Learning (ML) and led to widespread industrial adoption [1]. Deep Learning methods [2] began to dominate accuracy benchmarks around 2012, achieving superhuman results and improving further in subsequent years. As a result, many real-world problems in various domains, ranging from retail and banking [3,4] to medicine and healthcare [5-7], are now being addressed using machine learning models.

However, increased model complexity is frequently used to achieve this improved predictive accuracy. The deep learning paradigm, which is at the heart of most cutting-edge machine learning systems, is an excellent example. It enables machines to automatically discover, learn, and extract the hierarchical data representations required for detection or classification tasks. This hierarchy of increasing complexity, combined with the fact that vast amounts of data are used to train and develop such complex systems, while increasing the systems' predictive power in most cases, inherently reduces their ability to explain their inner workings and mechanisms. As a result, the reasoning behind their decisions becomes difficult to understand, making their predictions difficult to interpret.

There is a clear trade-off between a machine learning model's performance and its ability to generate explainable and interpretable predictions. On the one hand, there are black-box models like deep learning [2] and ensembles [8-10].

White-box or glass-box models, on the other hand, produce explainable results, with common examples including linear [11] and decision tree based [12] models. Although more explainable and interpretable, the latter models are less powerful and fall short of achieving cutting-edge performance when compared to the former. Their poor performance and ability to be well-interpreted and easily explained stem from the same possible cause: their frugal design.

Despite the machine learning industry's progress in developing solutions to assist data teams and practitioners in operationalizing their machine learning models, testing these models to ensure they work as intended remains one of the most difficult aspects of putting them into production [13].

Furthermore, different machine learning models perform differently on different data sets. Finding the best machine learning model for a given data set is thus a difficult task. A test system with various machine algorithms can be used as a solution to determine the best learning algorithm for the data set. Furthermore, as a solution to this problem, various machine learning algorithms can be compared. The system with multiple learning approaches can also lead to ensemble learning in order to predict optimal outcomes.

In this paper, we present a system for systematically testing machine learning algorithms with their respective datasets. Unlike other testing systems, we intend to generate a report and confusion matrix of the results of the tested algorithm as well.

A. Related Work

Different types of testing systems are designed for specific applications. A testing system, for example, was designed for systematic testing of a CNN model for autonomous driving [14]. Similarly, monotonicity is important in machine learning models for prediction and overall performance, so a system was created to test the monotonicity of machine learning models [15].

Furthermore, some machine learning applications are designed to learn data set properties where the correct answers are not already known to human users. It is difficult to test such ML software because there is no dependable test oracle. We describe a software testing strategy for addressing this issue [16].

B. Contribution

This study focuses on building a gui-based testing system for machine learning or decision algorithms in which users may input model of algorithm together with test data and the system should be able to provide a report and confusion matrix demonstrating how the model fared to the supplied test.

C. Paper Organization

The rest of the paper is structured as follows. Section II describes the information and operation of several decision algorithms that may be tested on our system. Section III provides the working of our system for the user. Section IV presents an overview of the testing system's code. Section V comprises testing several ML models with the testing system.

II. Decision Algorithms

ML is the branch of science that studies how computers can learn without being explicitly programmed. Thus, we may describe ML as the discipline of computer science in which machines that can program themselves can be created [18].

The learning process is simply learning from past work experience or observations, such as examples, or instruction, to seek for patterns in data and with the assistance of examples, supplied the system can make better judgments. The fundamental goal of ML is to teach computers to learn automatically without human intervention and to change their behavior accordingly [18,19].

Fig. 1 depicts the ML procedure. Past data is utilized to train the model, which is then used to test fresh data and forecast the future. The performance of the trained ML model is assessed using a subset of the available historical data (which is not present during training). This is commonly known as the validation procedure. The ML model is assessed for performance measures such as accuracy during this procedure. Accuracy measures the ML model's performance over unknown data by dividing the number of properly predicted features by the total number of accessible features to be predicted.

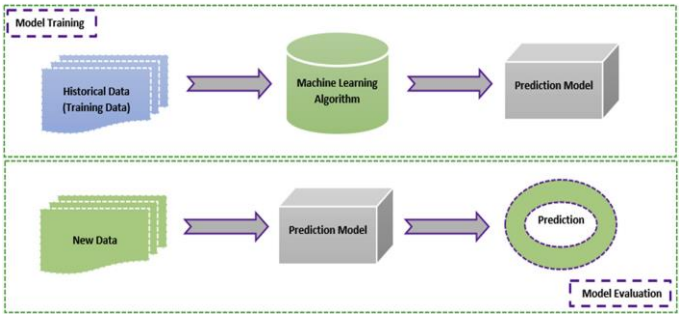

Fig. 1 Machine Learning process [17]

ML algorithms can be divided into supervised or unsupervised learning:

a) Supervised ML algorithms: This is a form of ML approach that may be used to generate new data from labeled data and predict future events or labels based on what has already been learnt. A supervisor (labels) is there to guide or correct in this form of learning. The known training set is used for this first analysis, and then the output values are predicted using the learning process. The learning system's output may be compared to the actual output; if inaccuracies are found, they can be corrected, and the model adjusted appropriately [18].

b) Unsupervised ML algorithms: There is no supervisor to guide or correct in this case. When unlabeled or unclassified data is provided, this sort of learning algorithm is employed to train the system. The system does not specify the proper output, but it examines the data such that it may draw inferences (rules) from datasets and identify hidden structures in unlabeled data [18-20].

c) Semi supervised ML algorithms: They are algorithms that fall between supervised and unsupervised learning. Thus, for training purposes, this form of learning algorithm employs both unlabeled and labeled data, with a small quantity of labeled data and a big number of unlabeled data. This sort of technique is utilized to increase learning accuracy [18-20].

d) Reinforcement ML algorithm: It is a sort of learning approach that assigns rewards or punishment based on the system's performance. If we teach the system to accomplish a certain task and it fails, the system may be penalized; if it succeeds flawlessly, the system will be rewarded. It normally operates on the 0 and 1 scales, with 0 indicating punishment and 1 indicating reward.

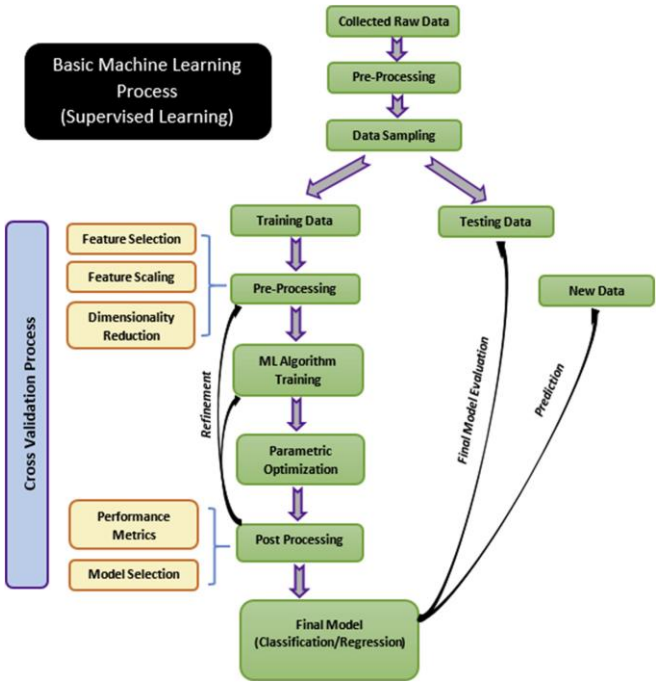

Fig. 2 Machine learning (ML) process flowchart

A. Convolutional Neural Networks (CNN)

The CNN algorithm is the most well-known and widely used in the area of DL [21, 22-26]. The fundamental advantage of CNN over its predecessors is that it

automatically finds significant elements without the need for human intervention [27]. CNNs have been widely used in a variety of domains, including computer vision [28], audio processing [29], face recognition [30], and so on. CNNs, like traditional neural networks, were inspired by neurons in human and animal brains. More precisely, the visual cortex in a cat's brain is formed by a complicated series of cells, which is mimicked by the CNN [31]. According to Goodfellow et al. [32], the CNN provides three main benefits: similar representations, sparse interactions, and parameter sharing. In contrast to typical fully connected (FC) networks, shared weights and local connections are used in CNN to fully use 2D input-data structures such as picture signals. This method employs an incredibly minimal number of parameters, which simplifies and speeds up the network training process. This is similar to what happens in visual cortex cells. Notably, these cells detect just tiny areas of a scene rather than the entire picture (i.e., these cells spatially extract the local correlation available in the input, like local filters over the input).

A typical version of CNN, comparable to the multi-layer perceptron (MLP), consists of several convolution layers before sub-sampling (pooling) layers, with FC layers at the end. Figure 3 is an example of CNN architecture for image categorization. Each layer's input x in a CNN model is arranged in three dimensions: height, width, and depth, where the height (m) equals the width. The depth is also known as the channel number. In an RGB image, for example, the depth (r) is three. Several kernels (filters) are accessible in each convolutional layer and have three dimensions (n n q), comparable to the input image. [33].

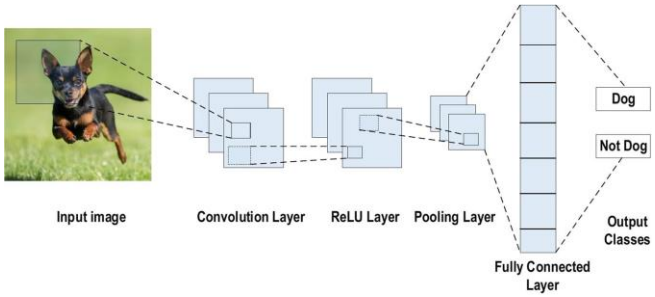

Fig. 3 An example of CNN architecture for image classification [33].

However, n must be less than m, and q must be equal to or less than r. Furthermore, the kernels serve as the foundation for the local connections, which, as previously stated, share comparable characteristics (bias bk and weight Wk) for producing k feature maps hk with a size of (m,n,1) each and are convolved with input. The convolution layer, like NLP, produces a dot product between its input and the weights, but the inputs are undersized portions of the initial picture size. Next, we derive the following by adding nonlinearity or an activation function to the convolution-layer output:

$$h^k = f(W^k * x + b^k) \qquad (1)$$

The following step is to down sample each feature map in the sub-sampling layers. This reduces the network parameters, which speeds up the training process and allows for the treatment of the overfitting problem. The pooling function (e.g., max or average) is applied to a neighboring region of size p x p, where p is the kernel size, for all feature maps. Finally, like in a conventional neural network, the FC layers receive the mid- and low-level features and generate the high-level abstraction, which represents the final-stage layers. The classification scores are produced by the last layer [for example, support vector machines (SVMs) or SoftMax]. Every score for a particular instance reflects the likelihood of a specific class. [33].

The following are the advantages of employing CNNs over other standard neural networks in the computer vision environment:

1. The weight sharing feature, which decreases the amount of trainable network parameters and so allows the network to improve generalization and avoid overfitting, is the primary reason to adopt CNN.

2. Learning the feature extraction layers and the classification layer concurrently results in a model output that is both highly ordered and heavily reliant on the extracted features.

3. CNN makes large-scale network deployment considerably easier than other neural networks.

The CNN architecture is made up of several levels (or so-called multi-building blocks). Each layer of the CNN design is detailed in depth below, including its purpose.

a) Convolutional Layer

The convolutional layer is the most important component in CNN architecture. It is made up of a number of convolutional filters (so-called kernels). The input picture is convolved with these filters to form the output feature map, which is represented as N-dimensional metrics..

1) Kernel definition: The kernel is described by a grid of discrete integers or values. Each number is referred to as the kernel weight. At the start of the CNN training process, random numbers are assigned to act as kernel weights. Furthermore, there are numerous approaches for initializing the weights. These weights are then modified during each training epoch, allowing the kernel to learn to extract significant characteristics.

2) Convolutional Operation: The vector format is the standard neural network's input, whereas the multichannel picture is the CNN's input. The gray-scale picture format, for example, is single-channel, but the RGB image format is three-channeled.

3) Sparse Connectivity: In FC neural networks, each neuron in a layer communicates with all neurons in the layer below. In contrast, just a few weights are accessible between two neighboring layers in CNNs. As a result, the number of needed weights or connections is limited, as is the memory required to store these

weights; hence, this strategy is memory effective. Furthermore, matrix operations are substantially more computationally expensive than dot (.) operations in CNN.

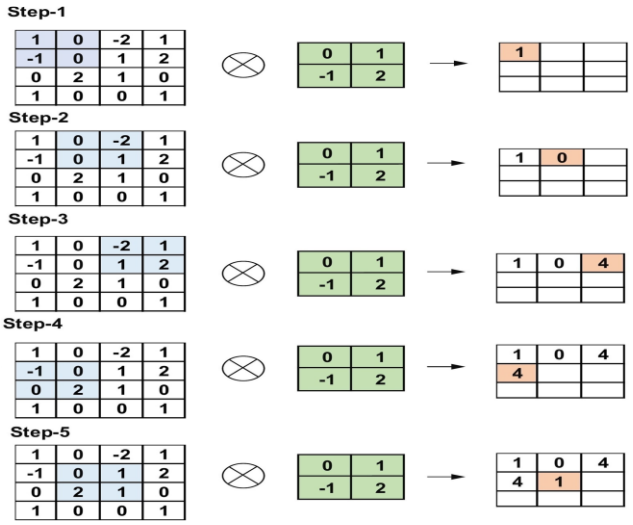

Fig. 4 The primary calculations executed at each step of convolutional layer [33].

b) Pooling Layer

The pooling layer's primary function is to subsample the feature maps. These maps are created by using convolutional procedures. In other words, this method condenses large-scale feature maps into smaller feature maps. At the same time, it keeps the bulk of the dominating information (or characteristics) in every stage of the pooling process. Before the pooling process, both the stride and the kernel are size-assigned in the same way as the convolutional operation is. Pooling techniques of various sorts are available for use in various pooling levels. Tree pooling, gated pooling, average pooling, min pooling, max pooling, global average pooling (GAP), and global max pooling are examples of these approaches.

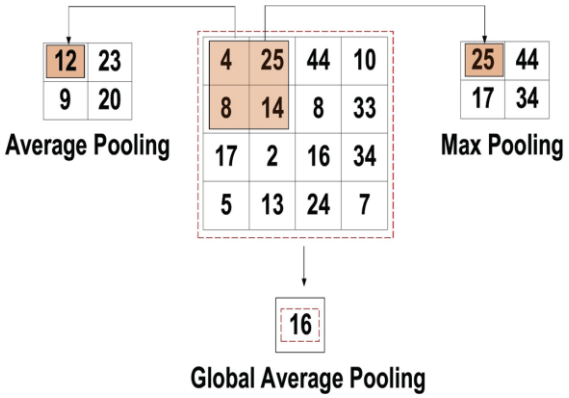

Fig. 5 Three types of pooling operations [33].

c) Activation Function (non-linearity)

The primary purpose of all sorts of activation functions in all types of neural networks is mapping the input to the output. The input value is computed by taking the weighted sum of the neuron input and its bias (if present). This implies that the activation function decides whether or not to fire a neuron in response to a certain input by producing the matching output.

In CNN architecture, non-linear activation layers are used after all weighted layers (so-called learnable layers, such as FC layers and convolutional layers). The activation layers' non-linear performance implies that the mapping of input to output will be non-linear; also, these layers enable the CNN to learn extra-complicated things. The activation function must also be able to distinguish, which is a critical characteristic since it enables error back-propagation to be utilized to train the network. The activation functions listed below are the most often employed in CNN and other deep neural networks.

1) Sigmoid: The input of this activation function is a real number, and the output is limited to a range of zero to one. The sigmoid function curve is S-shaped and formally expressed by Eq. 2.

$$f(x)_{sigm} = \frac{1}{1+e^{-x}} \qquad (2)$$

2) Tanh: It is similar to the sigmoid function in that it accepts real numbers as input, but its output is limited to values between 1 and 1. Eq. 3 shows its mathematical representation.

$$f(x)_{tanh} = \frac{e^{x}+e^{-x}}{e^{x}+e^{-x}} \qquad (3)$$

3) ReLU: In the CNN context, this is the most widely utilized function. It transforms the input's full values to positive integers. The key advantage of ReLU over the others is its lower computational load. Eq. 4 shows its mathematical representation.

$$f(x)_{ReLU} = \max(0, x) \qquad (4)$$

d) Fully Connected Layer

This layer is often found near the ending of each CNN architecture. Each neuron in this layer is linked to all neurons in the previous layer, using the Fully Connected (FC) technique. It serves as the CNN classifier. As a feed-forward ANN, it follows the fundamental method of the standard multiple-layer perceptron neural network. The FC layer receives its input from the previous pooling or convolutional layer. This input takes the form of a vector, which is generated after flattening the feature maps. As shown in Fig. 6, the FC layer output reflects the final CNN output [33].

e) Loss Functions:

The preceding section discussed various layer types of CNN architecture. In addition, the final classification is done from the output layer, which represents the last layer of the CNN design. In the CNN model, certain loss functions are used in

the output layer to determine the expected error generated over the training samples. This mistake shows the discrepancy between the actual and projected output. The CNN learning procedure will then be used to optimize it [33].].

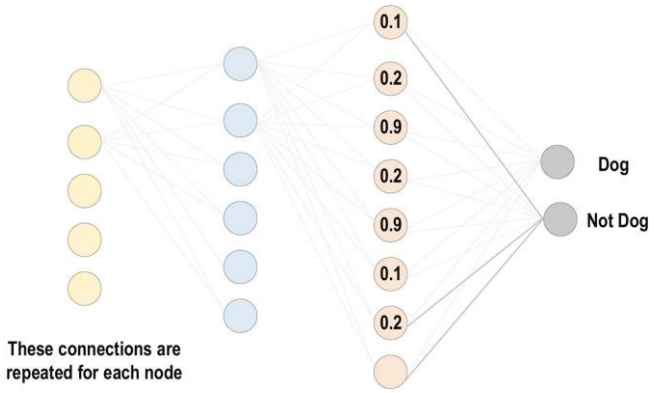

Fig. 6 Fully connected layer of CNN [33].

B. AlexNet

The origins of deep CNNs may be traced back to the emergence of LeNet [34]. The CNNs were limited to handwritten digit identification tasks at the time, which cannot be scaled to all picture classes. AlexNet is highly regarded in deep CNN architecture [21], having produced groundbreaking achievements in image recognition and classification. AlexNet was initially proposed by Krizhevesky et al. [21], who then improved the CNN learning capabilities by increasing its depth and adopting multiple parameter optimization algorithms. The basic concept of the AlexNet architecture is seen in Figure 7.

Due to hardware constraints, the deep CNN's learning potential was restricted at the time. To address these hardware constraints, AlexNet was trained using two GPUs (NVIDIA GTX 580) in parallel. Furthermore, the number of feature extraction steps in AlexNet was expanded from five in LeNet to seven in order to improve the CNN's adaptability to diverse picture categories. Despite the fact that depth improves generalization for a variety of picture resolutions, overfitting was the primary disadvantage of depth.. To overcome this issue, Krizhevesky et al. adopted Hinton's concept [35, 36]. He randomly passes over various transformational units during the training step to guarantee that the characteristics learnt by the system were extra resilient.

Furthermore, by alleviating the vanishing gradient problem, ReLU [37] might be used as a non-saturating activation function to speed up convergence [38]. To improve generalization by reducing overfitting, local response normalization and overlapping subsampling were also used. Other improvements were made to improve on the performance of prior networks, including the use of large-size filters (5 5 and 11 11) in the early layers. AlexNet has had a significant impact on current CNN generations, as well as kicking off an inventive research age in CNN applications. [33]

Model	Main finding	Depth	Dataset	Error rate	Input size
AlexNet	Utilizes Dropout and ReLU	8	ImageNet	16.4	227 × 227 × 3
GoogLeNet	Increased depth, block concept, different filter size, concatenation concept	22	ImageNet	6.7	224 × 224 × 3
ResNet	Robust against overfitting due to symmetry mapping-based skip links	152	ImageNet	3.57	224 × 224 × 3
DenseNet	Blocks of layers; layers connected to each other	201	CIFAR-10, CIFAR-100, ImageNet	3.46, 17.18 ,5.54	224 × 224 × 3
NIN	New layer, called ‘mlpconv’, utilizes GAP	3	CIFAR-10, CIFAR-100, ImageNet	10.41.35. 68, 0.45	32 × 32 × 3
ZfNet	Visualization idea of middle layers	8	ImageNet	11.7	224 × 224 × 3

Table 1. Brief overview of CNN architectures [33].

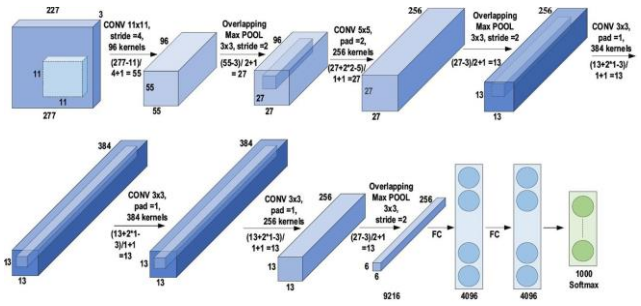

Fig. 7 The architecture of AlexNet [33].

C. LeNet

The Convolutional Neural Network LeNet was employed in the tests. LeNet is made up of layers, not counting the input, all of which have trainable parameters and weights.
A x pixel image is used as input. At most x pixels centered in an x eld, this is significantly larger than the biggest character in the database. The rationale for this is because

possible distinguishing characteristics such as stroke endpoints or corners should show in the center of the receiving field of the highest-level feature detectors. In LeNet, the collection of centers of the final convolutional layer C, as shown below, forms an x region in the center of the x input. The input pixel values are nor changed so that the background level white corresponds to and the foreground black corresponds to This approximates the mean input and the variance, which speeds up learning [39].

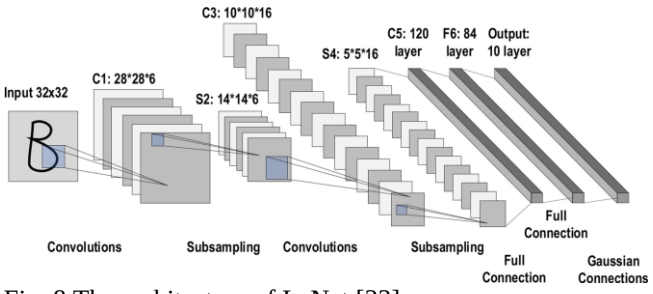

Fig. 8 The architecture of LeNet [33]

The first layer is the input layer, which is not considered a network layer because nothing is learned in this layer. The input layer is designed to accept 32x32 pictures, which are then passed on to the following layer. Those who are familiar with the MNIST dataset will recognize that the pictures in the MNIST dataset have the size 28x28. The 28x28 pictures are padded to make the MNIST images fit the criteria of the input layer [40].

The grayscale pictures used in the study had their pixel values normalized from 0 to 255, yielding values ranging from -0.1 to 1.175. Normalization is used to guarantee that the batch of pictures has a mean of 0 and a standard deviation of 1. The benefits of this are visible in the decrease of training time. In the image categorization with LeNet-5 example below, we will normalize the picture pixel values to take on values ranging from 0 to 1.

The LeNet-5 design employs two types of layer structures: a) Convolutional layers, b) Sub-sampling layers

Convolutional layers are labeled as ‘Cx’ in the research paper and the table below, while subsampling layers are identified as ‘Sx’, where ‘x’ represents the layer's sequential position within the architecture. The symbol ‘Fx’ denotes completely linked layers. The graphic above depicts this method of layer identification.

The official first layer convolutional layer C1 has a kernel size of 5x5 and provides 6 feature maps as output. The window containing the weight values that are used during the convolution of the weight values with the input values is referred to as the kernel/filter. 5x5 also represents the size of each unit or neuron within a convolutional layer's local receptive field. The first convolution layer generates six feature maps with size of 28x28.

The ‘C1’ layer is followed by a subsampling layer 'S2.' The ‘S2’ layer reduces the dimension of the feature maps received from the preceding layer by half; this is referred to as down sampling.

The 'S2' layer additionally generates six feature maps, one for each of the feature maps supplied as input from the preceding layer. More information on subsampling layers may be found at this site [40].

The following table summarizes the essential elements of each layer:

Layer	Layer Type	Feature Maps	Kernel	Feature Map Size
Input	Image	-	-	32x32
C1	Convolution	6	5x5	28x28
S2	Sub Sampling	6	2x2	14x14
C3	Convolution	16	5x5	10x10
S4	Sub Sampling	16	2x2	5x5
C5	Convolution	120	5x5	1x1
F6	Fully Connected	-	-	82
Output	Fully Connected	-	-	10

Table 2. LeNet layer structure [40]

D. ResNet

He et al. [41] created ResNet (Residual Network), which won the ILSVRC 2015. In comparison to earlier networks, their goal was to construct an ultra-deep network that was free of the vanishing gradient issue. Depending on the number of layers, several types of ResNet were created (starting with 34 layers and going up to 1202 layers). ResNet50 was the most prevalent, with 49 convolutional layers and a single FC layer. The total number of network weights was 25.5 million, whereas the total number of MACs was 3.9 million. ResNet's original innovation is its usage of the bypass pathway concept, as shown in Fig. 10, which was used in Highway Nets in 2015 to overcome the difficulty of training a deeper network. This is seen in Fig. 9, which displays the basic ResNet block diagram. This is a feedforward network with a residual link. The residual layer output is recognized as the (l 1)th outputs supplied by the preceding layer (xl 1). F(xl 1) is the result of several procedures [such as convolution with variable-size filters or batch normalization before applying an activation function like ReLU on (xl 1)]. The final residual output is xl, which may be expressed mathematically as in Eq. 5 [33].

$$x_l = \mathrm{F}(x_l - 1) + x_l - 1 \qquad (5)$$

The residual network has a large number of fundamental residual blocks. The operations in the residual block vary depending on the kind of residual network architecture [41].

ResNet has the ability to avoid gradient decreasing concerns since the shortcut connections (residual links) promote deep network convergence. ResNet won the 2015-ILSVRC championship with 152 layers of depth, which is eight times the depth of VGG and twenty times the depth of AlexNet. It has lower computational complexity than VGG, even with increased depth [33].

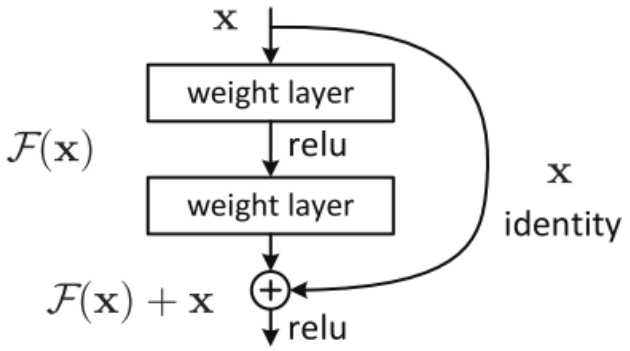

Fig. 9 Residual Block Structure [40]

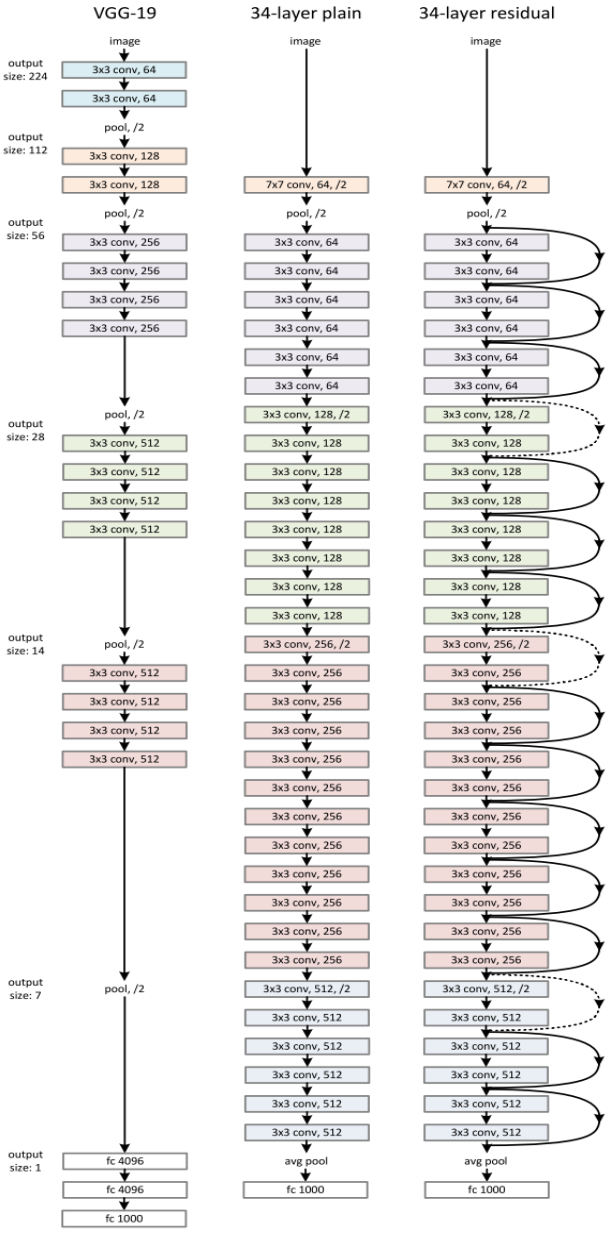

Fig. 10 ResNet architecture [33]

E. *ResNext*

The ResNext Network is an improved version of the Inception Network [42]. The Aggregated Residual Transform Network is another name for it. Cardinality, a novel word introduced by [42], made efficient use of the divide, transform, and merge topology. It represents the transformation's size as an additional dimension [43-45]. However, the Inception network handles network resources more effectively while also improving the traditional CNN's learning capabilities. Different spatial embeddings (for example, 5x5, 3x3, and 1x1) are employed in the transformation branch. As a result, each layer must be customized independently. ResNext, on the other hand, takes its cues from ResNet, VGG, and Inception. It combined the VGG deep homogeneous topology with GoogleNet's fundamental architecture, using 3x3 filters as spatial resolution inside the blocks of split, transform, and merge. The ResNext building blocks are seen in Figure 11. ResNext made use of multi-transformations within the divide, transform, and merge blocks, as well as describing such transformations in cardinality terms. As Xie et al. shown, raising the cardinality improves performance dramatically. ResNext's complexity was controlled by using 1x1 filters (low embeddings) before a 3x3 convolution. Skipping connections, on the other hand, are employed for efficient training [42].

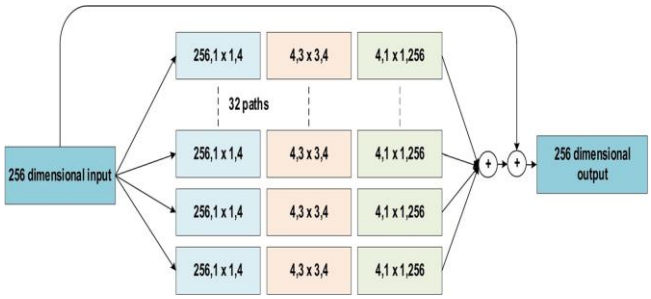

Fig. 11 The ResNext building blocks' fundamental block diagram [33].

F. *MLP (Multilayer Perceptron)*

Although the Perceptron is now generally known as an algorithm, it was originally designed as an image recognition system. It receives its name from its ability to execute human-like functions such as sensing, seeing, and identifying pictures. The neuron accepts input and generates a random set of weights. These are incorporated in a weighted total, and the activation function, ReLU, determines the output value [46].

Fig. 12 Neuron model of perceptron's (left) and activation function (right) [46].

Although it was claimed that the Perceptron could represent any circuit or logic, the most serious objection was that it couldn't represent the XOR gate, exclusive OR, which only returns 1 if the inputs are different. This was

demonstrated over a decade later, in 1969[47], by Minsky and Papert, highlighting the fact that the Perceptron, with only one neuron, cannot be used to non-linear data [46].

To address this restriction, the Multilayer Perceptron was created. It is a neural network with a non-linear mapping between inputs and outputs. A Multilayer Perceptron contains input and output layers, as well as one or more hidden layers made up of numerous neurons layered on top of each other. While neurons in a Perceptron must have an activation function that enforces a threshold, such as ReLU or sigmoid, neurons in a Multilayer Perceptron can employ any arbitrary activation function [46].

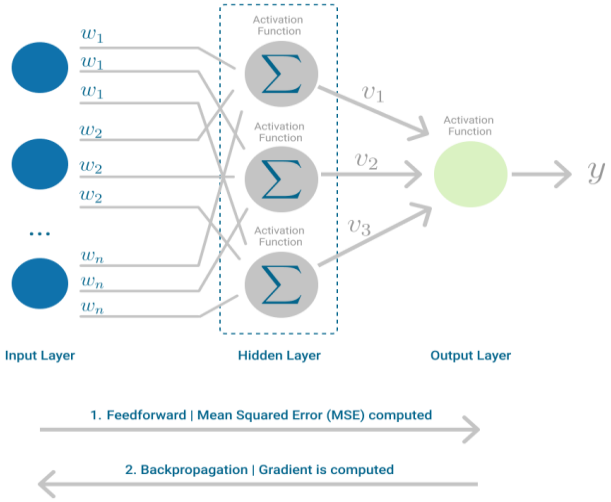

Fig. 13 Feedforward and Backpropagation phases of a multilayer perceptron [46].

Because inputs are merged with the starting weights in a weighted sum and applied to the activation function, the Multilayer Perceptron falls under the category of feedforward algorithms. However, each linear combination is transferred to the following layer.

Each layer feeds the outcome of their calculation, their internal representation of the data, to the next. This continues through the hidden layers to the output layer.

The method would be unable to learn the weights that optimize the cost function if it just computed the weighted sums in each neuron, transmitted the findings to the output layer, and then stopped. There would be no learning if the algorithm simply calculated one iteration.

Backpropagation [48] comes into play here. Backpropagation is the learning process that enables the Multilayer Perceptron to iteratively alter the network weights in order to minimize the cost function.

Backpropagation must meet one strict criteria in order to function effectively. In a neuron, the function that mixes inputs and weights, such as the weighted sum, and the threshold function, such as ReLU, must be differentiable. Because Gradient Descent is often employed as the optimization function in Multilayer Perceptron, these functions must have a limited derivative.

After the weighted sums are sent through all layers in each iteration, the gradient of the Mean Squared Error is computed over all input and output pairs. The weights of the first hidden layer are then modified with the gradient value to propagate it back. That is how the weights are transmitted back to the neural network's starting point!

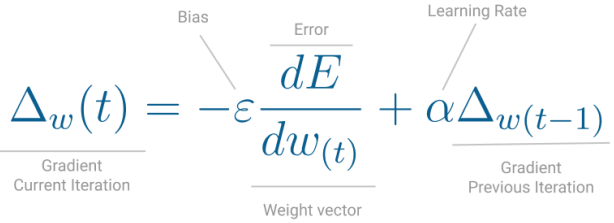

Fig. 14 Gradient Descent iteration one [46]

G. SVM (Support Vector Machines)

Support Vector Machines (SVM) are generally thought to be a classification technique, however they may be used in both classification and regression issues. It can handle a large number of continuous and categorical variables with ease. To differentiate various classes, SVM creates a hyperplane in multidimensional space. SVM iteratively develops ideal hyperplanes, which are used to minimize errors. SVM's core notion is to determine the maximum marginal hyperplane (MMH) that optimally divides a dataset into classes. [49]

SVM is an excellent classification algorithm. It is a supervised learning method that is mostly used to categorize data. SVM is trained using label data. The fundamental benefit of SVM is that it can be applied to both classification and regression issues. To divide or classify two classes, SVM constructs a decision boundary, which is a hyperplane between them. SVM is also utilized in picture classification and object detection [49].

The data points nearest to the hyperplane are called support vectors. By computing margins, these points will better define the separation line. These points are more significant to the classifier's creation. A hyperplane is a decision plane that divides objects of various class memberships [49].

A margin is the distance between two lines at the nearest class points. This is determined as the perpendicular distance between the line and the nearest support vectors or points. A bigger gap between classes is regarded a good margin; a smaller margin is considered a bad margin [49].

The main goal is to separate the provided dataset as effectively as possible. The margin is the distance between the two nearest locations. The goal is to find the hyperplane with the greatest feasible margin between support vectors in the provided dataset.

In practice, the core SVM method is implemented using a kernel. A kernel is responsible for transforming an input data space into the desired format. SVM employs a method known as the kernel trick. In this case, the kernel turns a low-dimensional input space into a higher-dimensional one. In other words, it changes no separable issues into separable problems by adding extra dimensions to them. It is very effective in nonlinear separation problems. The kernel technique allows you to create a more accurate classifier. [49].

1) Linear Kernel : Any two supplied observations can be normal dot producted using a linear kernel. The total of the

multiplications of each pair of input values is the product of two vectors.

$$K(x, x1) = sum(x * x1) \quad (6)$$

2) Polynomial Kernel : A polynomial kernel is a broader variant of the linear kernel. The polynomial kernel can tell the difference between curved and nonlinear input space.

$$K(x, x1) = 1 + sum(x * x1)^d \quad (7)$$

Where d is the polynomial's degree. The linear transformation is analogous to d=1. The degree must be explicitly entered into the learning process.

3) Radial Basis Function Kernel: The Radial basis function kernel is a well-known kernel function in support vector machine classification. RBF can map an input space to an unlimited number of dimensions.

$$K(x, x1) = e^{(-gamma * sum((x - xi^2))} \quad (8)$$

Gamma is a parameter with a value ranging from 0 to 1. A larger gamma value will precisely match the training dataset, resulting in over-fitting. Gamma=0.1 is regarded as a decent default value. In the learning procedure, the value of gamma must be explicitly provided [49].

III. Algorithm Testing System

The proposed algorithm testing system can test all decision algorithms such as CNN, SVM, MLP, ResNet, AlexNet and so on. ML models may be evaluated using a GUI developed on the PyQt Framework. The GUI assists the user in making decisions about models, input photos, input size, input/output directories, and so on. Finally, a report is provided along with a confusion matrix that gives facts about the model and tested pictures as well as whether the prediction made by the utilized model was right or not.

A. PyQt

When dealing with Python, there are various GUI frameworks that may be used; one of the most actively developed and well supported is PyQt (now PyQt5).

Qt is a cross-platform application framework (Windows, Linux, OS X, Android, and others) that does more than just produce UI widgets and dialogs, though it is one of its primary use cases. Qt was created by many businesses (Trolltech, Nokia, and Digia) and is now a fairly mature library with a big user base. Since previously stated, Qt is capable of much more than just developing GUI components, as it also supports networking, multimedia, SQL (database), and Webkit (web content rendering), plus add-on modules for Bluetooth and NFC, hardware sensors, and GPS locating service access. [50]

B. Qt Designer

Qt Designer is a Qt toolkit that allows you to develop GUIs for your PyQt applications using a what-you-see-is-what-you-get (WYSIWYG) user interface. You construct GUIs with this tool by dragging and dropping QWidget objects into an empty form. Then, using various layout managers, you may organize them into a logical GUI.

Qt Designer also allows you to examine your GUIs in various styles and resolutions, link signals and slots, construct menus and toolbars, and do a variety of other things. Qt Designer is agnostic of platform and programming language. It does not generate code in any programming language, but it does generate ".ui" files. These are XML files that include thorough instructions for creating Qt-based GUIs.

Using pyuic5, a command-line tool included with PyQt, we may convert the content of ".ui" files into Python code. The Python code may then be used in your GUI apps. You may also directly read ".ui" files and import their contents to construct the accompanying GUI [51].

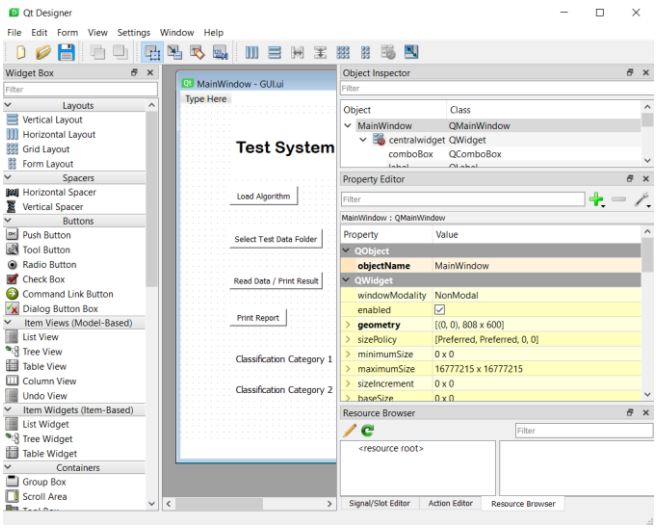

Fig. 15 Qt designer gui demonstration

C. Testing System GUI

The GUI that we built was created using Qt Designer, and any further modifications were made in GUI Python code. Several buttons and text fields are included in the graphical user interface. The following describes how the GUI works, from initialization to report and confusion matrix compilation.

1. Selecting ML Model

The first step in testing the algorithm is to choose the model to be tested. For illustration, there is a drop-down menu in the graphical interface where we may choose from a variety of models, and after picking one, we must hit the "Save Model Name" button so that the program knows which model is going to be tested.

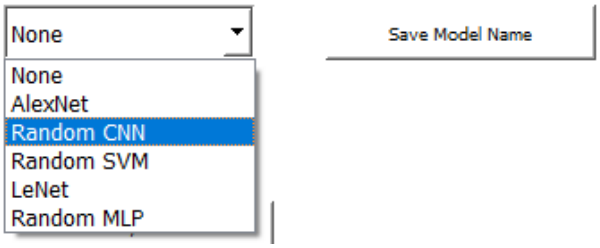

Fig. 16 Choosing a Model Name

2. Loading Model

After selecting the model, we must select the "Load Algorithm" button so that software may load the model that we already created in the code. It may take some time to load the model, but once finished, the interface window will show "Model Loaded."

Fig. 17 Loading the desired model/algorithm

3. Input Image Size

The next step is to choose the image size for the model that will be evaluated. The size of the input photos (width and height) changes depending on the model. For example, in the case of AlexNet, the input size is 224x224, but in the case of LeNet, it is 32x32. As a result, we must adjust the input size based on the loaded model. We have included a text box called “Input Size" where the user can specify the model's input size and a button next to it called "Save Image Size" which will save the image size for model testing.

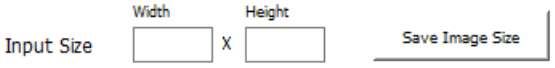

Fig. 18 Specifying the model's input size

4. Selecting Input Folder

Now we need to pick a folder to do testing on. To do so, click the "Select Input Folder" option and navigate to the folder containing the test images. The application will load the folder, cycle over each file one by one, and then output the expected outcome according to loaded models results.

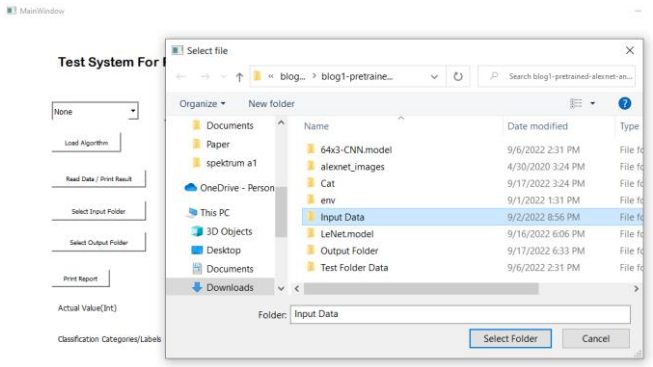

Fig. 19 Selecting input folder for the model

5. Selecting Output Folder

After selecting the input folder, we must also specify the output folder, where the resulting report and confusion matrix will be located. This requires the same procedure as selecting the Input folder. We must select the "Select Output Folder" button.

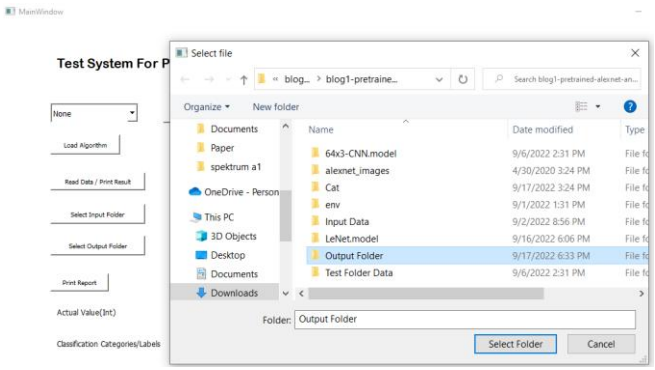

Fig. 20 Selecting output folder for the model

6. Selecting Categories/Output Labels

There are two ways for the user to add Output Labels. 1) Entering comma separated labels in textbox 2) Adding labels through text file. If the user has more than two labels, add a label.txt file. The label can be inputted into the "Classification Categories/Label" text box by the user. To store the labels, the user can enter keywords such as "dog, cat" and hit the "Save Categories" button.

By hitting the "Select Categories File" button, you may load through a file. The labels should be line separated, with one label following another on the next line till the end.

We must also provide the model with the actual index number of the label so that we can determine if the model prediction is right or not, which will aid in the construction of the confusion matrix. As a result, we may write that in the "Actual Value(int)" text area. For example, if the prediction of Dog is correct in the scenario of labels “cat, dog”, we should write "1" in the actual value text area.

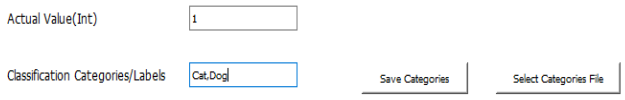

Fig. 21 Defining and choosing model output labels and actual int

7. Read Data and Begin Testing

The most crucial step is to begin testing. Once all of the preceding procedures have been completed, we may hit the "Read Data/ Print Results" button. By selecting this button, the application will begin checking each file one by one and produce a report file with the projected value from the model and the user's actual value. The confusion matrix is created with the assistance of the generated report.

8. Print Report and Confusion Matrix

After all of the processes have been completed, the final step is to create the report and confusion matrix. To do so, click the "Print Report" button, and a confusion matrix is created from the report, demonstrating clear representation of model findings.

IV. Code Overview

The system is developed in Python, and image libraries such as TensorFlow, keras, matplotlib, NumPy, sklearn, torch, and others were utilized. The gui went through several stages, including adding buttons and text fields, labeling buttons, and fields, adding functionality to each button, and so on. The steps are as follows, with an example:

A. Creating a button, label and text field

This interface utilizes four widgets: buttons, labels, a combo box (drop down menu), and line edit (text field). In the gui configuration, we must describe the button and text fields, as well as any additional widgets that will be used in the main gui. To make a pushbutton, we must use the Pyqt library of qt widgets. Following the creation of a button, we must define its shape, color, orientation in the gui window, and font in the gui window.

```
def setupUi(self, MainWindow):
    MainWindow.setObjectName("MainWindow")
    MainWindow.resize(800, 600)
    MainWindow.setStyleSheet("background-color: rgb(255, 255, 255);")
    self.centralwidget = QtWidgets.QWidget(MainWindow)
    self.centralwidget.setObjectName("centralwidget")
    self.pushButton = QtWidgets.QPushButton(self.centralwidget)
    self.pushButton.setGeometry(QtCore.QRect(80, 180, 111, 31))
    self.pushButton.setObjectName("pushButton")
```

Fig. 22 Creating a pushbutton

In the same way, we can define a label in gui. Again, we must utilize the “QTwidgets” library and specify a label. Then we may choose the label font, color, geometry, and position in the gui window.

```
self.label = QtWidgets.QLabel(self.centralwidget)
self.label.setGeometry(QtCore.QRect(90, 20, 631, 91))
font = QtGui.QFont()
font.setFamily("Arial Rounded MT Bold")
font.setPointSize(16)
font.setBold(False)
font.setWeight(50)
self.label.setFont(font)
self.label.setObjectName("label")
```

Fig. 23 Creating a label

Then, similarly, line edit is defined using “QTwidgets”. Line edit is used to enter text that will be utilized in the GUI to add output labels. We just need to specify line edit, its shape, and where it will be put in the GUI.

```
self.lineEdit = QtWidgets.QLineEdit(self.centralwidget)
self.lineEdit.setGeometry(QtCore.QRect(280, 500, 113, 22))
self.lineEdit.setObjectName("lineEdit")
```

Fig. 24 Creating a line edit (text field)

Finally, define a combo box or drop-down menu for model name selection. We design a combo box, then add items to the menu for the number of options we want, and then specify geometry and font.

```
self.comboBox = QtWidgets.QComboBox(self.centralwidget)
self.comboBox.addItem("")
self.comboBox.addItem("")
self.comboBox.addItem("")
self.comboBox.addItem("")
self.comboBox.addItem("")
self.comboBox.addItem("")
self.comboBox.setObjectName("comboBox")
self.comboBox.setGeometry(QtCore.QRect(80, 130, 141, 31))
font2 = QtGui.QFont()
font2.setPointSize(10)
self.comboBox.setFont(font2)
```

Fig. 25 Creating a combo box (drop down menu)

B. Naming buttons, labels and combobox items

The software's "retranslateUi" function handles all widget names. We choose the widget and give it a name by using the “setText” or “setItemtext” functions.

```
self.pushButton_8.setText(_translate("MainWindow", "Save Model Name"))
self.pushButton_9.setText(_translate("MainWindow", "Save Image Size"))

self.label_4.setText(_translate("MainWindow", "Classification Categories/Labels"))
self.label_5.setText(_translate("MainWindow", "Actual Value(Int)"))
self.label_6.setText(_translate("MainWindow", "Input Size"))
self.label_7.setText(_translate("MainWindow", "Width"))
self.label_8.setText(_translate("MainWindow", "Height"))
self.label_9.setText(_translate("MainWindow", "X"))



self.pushButton_5.setText(_translate("MainWindow", "Save Categories"))

self.comboBox.setItemText(0, _translate("MainWindow", u"None", None))
self.comboBox.setItemText(1, _translate("MainWindow", u"AlexNet", None))
self.comboBox.setItemText(2, _translate("MainWindow", u"Random CNN", None))
self.comboBox.setItemText(3, _translate("MainWindow", u"Random SVM", None))
self.comboBox.setItemText(4, _translate("MainWindow", u"LeNet", None))
self.comboBox.setItemText(5, _translate("MainWindow", u"Random MLP", None))
```

Fig. 26 Naming all the widgets in the interface

C. Connecting button with a function

The most important aspect of the program is connecting the designated buttons with functions so that each button has its own unique purpose. We utilize the “clicked.connect” function for this, when the button is clicked, the function to which the button is attached is called. In the example below, the pushbutton (Load Model button) is linked to the "button clicked" function, which is used to load the model selected via a drop-down menu or combo box. When a button is clicked, the model is loaded using the button clicked function.

```
self.pushButton.clicked.connect(self.button clicked)

def button_clicked(self):
    print("you pressed the button")

    self.label_2.setText("Loading Model")

    global model, model_name

    print("Model name is ", model_name)

    #
    ####prediction = model.predict([prepare('C:/Users/Abbad/Desktop/Sommer
    # 22/Project/doggo.jpg')])

    # model=pickle.load(open('SVM_model.p','rb'))
```

Fig. 27 Connection of button with a function (top), Function of loading model (bottom)

D. Defining functions

We have to develop many functions because each button performs a separate purpose. For this, various buttons and functions were built, with each function serving a distinct purpose. Now we'll go through each function and what it's supposed to do in system software.

1. Image size function

When the save image size button is hit, this method is invoked. It remembers the picture size given by the user in the text fields by reading the data and uses it when the model is tested.

```
def button_clicked9(self):#Image Size

    print("you pressed the input size save")
    global size1,size2


    #["Cat", "Dog"]


    if size1 != '':
        size1 = self.lineEdit_3.text()
        size2 = self.lineEdit_4.text()

    print(size1)
    print(size2)

    self.label_2.setText("Size Saved")
```

Fig. 28 Saving image size function

2. Input/Output folder selection function

Since both input and output folder selection are utilized to point out folders, they have comparable functions but distinct buttons assigned to them because two unique functions were developed.

```
def button_clicked4(self): #Input Folder

    print("you pressed the button4")
    global fname

    fname = QFileDialog.getExistingDirectory(None, 'Select file')
    print(fname)
    self.label_2.setText("Input Folder Selected")

def button_clicked6(self): #Output Folder

    print("you pressed the button6")
    global fname2

    fname2 = QFileDialog.getExistingDirectory(None, 'Select file')
    print(fname2)
    self.label_2.setText("Output Folder Selected")
```

Fig. 29 Function for selecting Input and output folders

3. Adding Output label function

Input labels or categories can be defined using a text field or by attaching a label file. So, we built two new functions: one for saving categories via file and another for saving categories specified in a line edit or text field.

```
def button_clicked7(self):#Categories File Added

    print("you pressed the Class name")
    global class_name

    class_name = QFileDialog.getOpenFileName(None, 'Select file')
    print("class_name", class_name)
    print("class_name[0]", class_name[0])\

    global loaded_file


    loaded_file = 1

    self.label_2.setText("Categories File Selected")


def button_clicked5(self): #saving Categories

    print("you pressed the Categories save")



    text1 = self.lineEdit.text()
    text2 = self.lineEdit_2.text()
    #["Cat", "Dog"]
    print(text1)
    print("text2", text2)


    global CATEGORIES, actual_int

    global loaded_file

    if text1 != NULL:
        try:
```

Fig. 30 Function for loading category file (top), Function of saving categories from line edit (bottom)

4. Reading data and printing result function

This function is crucial to our system. This function takes each input image from the given folder and tests it on the chosen algorithm before saving a report of each file in an excel file that shows the user how the algorithm performed on test photos.

```
def button_clicked2(self): ## Read Data

    def prepare(filepath):
        global size1,size2
        IMG_SIZE = 100  # 50 in txt-based

        if size1 == 0:
            size1 = 100
            size2 = 100

        if model_name == "LeNet":
            size1 = 32
            size2 = 32

        size1 = int(size1)
        size2 = int(size2)
```

Fig. 31 Function for reading data and printing results

5. Generating Confusion matrix function

Once the report has been generated, the next step is to construct a confusion matrix from it. To do this, we have created a button that is linked to a function that reads the excel file and generates the confusion matrix.

```
def button_clicked3(self): #Confusion Matrix

    print("you pressed the button3")


    def make_confusion_matrix(cf,
                              group_names=None,
                              categories='auto',
                              count=True,
                              percent=True,
                              cbar=True,
                              xyticks=True,
                              xyplotlabels=True,
                              sum_stats=True,
                              figsize=None,
                              cmap='Blues',
                              title=None):
```

Fig. 32 Function for generating confusion matrix

V. TESTING

The system is now being demonstrated and tested using several decision algorithms. First, we begin with AlexNet Model testing and proceed through all of the processes required to effectively test the system and software.

A. *AlexNet Model*

We are now testing a pre-trained AlexNet Model with our system by giving them with a model and input images, and the application generates a confusion matrix and an excel report. The steps we took are as follows:

1. Selecting ML Model

The first step in testing the algorithm is to choose the model to be tested. For this, we choose "AlexNet" from the drop-down option, then click "Save Model Name," and the model is selected.

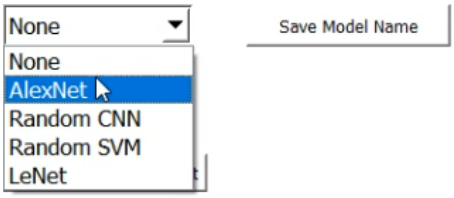

Fig. 33 Selecting AlexNet model from the drop-down menu

2. Loading Model

After picking the model, we must hit the "Load Algorithm" button, and the window will display AlexNet Model Loaded.

Fig. 34 Loading AlexNet model

3. Input Image Size

The next step is to choose the image size for the model that will be evaluated. We enter these numbers and hit the "Save Image Size" button since AlexNet has an input size of 224x224.

Fig. 35 Defining input size for AlexNet model

4. Selecting Input Folder

Now we must choose a folder to do testing with. To do so, click the "Pick Input Folder" button and select "Cat Folder." There are several photos of dogs and cats in this folder on which the model may be evaluated.

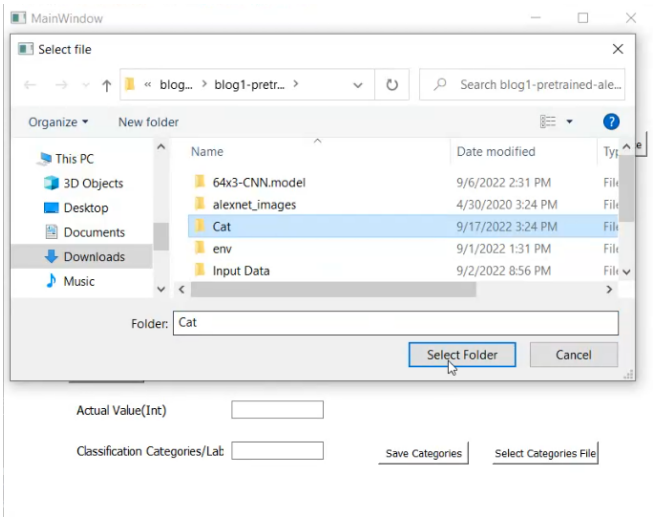

Fig. 36 Selecting Input folder where images are located

5. Selecting Output Folder

After selecting the input folder, we must also pick the output folder, where the created report and confusion matrix will be saved. This requires the same procedure as selecting the Input folder. We must select the "Select Output Folder" option.

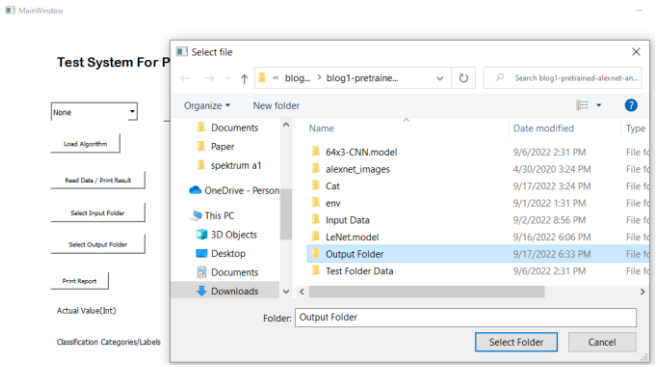

Fig. 37 Selecting Output folder where result and confusion matrix will be stored

6. Selecting Categories/Output Labels

Considering AlexNet has over 1000 output labels, it is hard to input them manually, so we import them through a label file named "class name," and because we want true case only when the label index is "281" we enter that index number in the actual value text box and select "Save Categories."

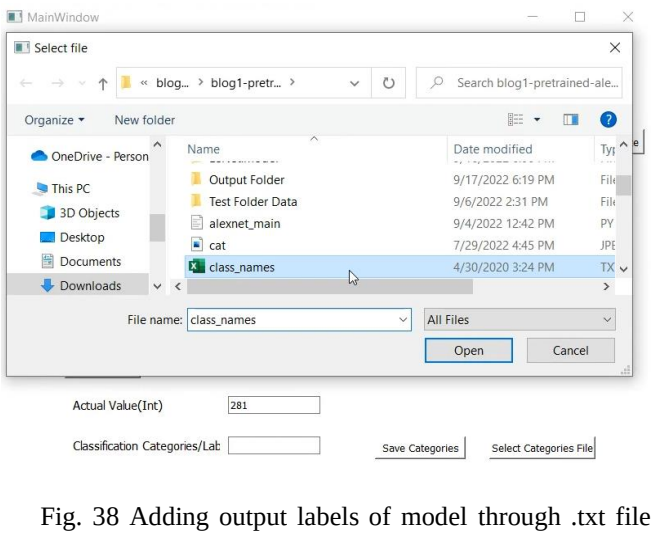

Fig. 38 Adding output labels of model through .txt file and adding actual value of label

7. Read Data and Begin Testing

The next step is to hit the "Read Data/ Print Result" button, which loads the algorithm and tests each file in the input folder, creating a report in the output folder.

8. Print Report and Confusion Matrix

When the testing is finished, the window should say "Evaluation Done." We may produce the confusion matrix by clicking on "Print Report."

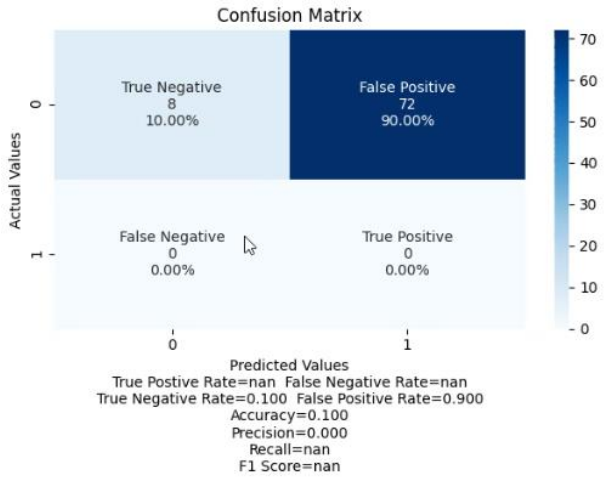

Fig. 39 Generated Confusion matrix of the AlexNet model

B. CNN Model

We are now evaluating a pre-trained CNN Model using our system. This model has been trained to differentiate between cats and dogs. The steps we took are as follows:

1. Selecting ML Model

The model we chose was "Random CNN," and we pressed the "Save Model Name" button.

Fig. 40 Selecting Random CNN model from the drop-down menu

2. Loading Model

After selecting the model, we load the algorithm by hitting the "Load Algorithm" button, and the window displays "Random CNN Loaded."

Fig. 41 Loading Random CNN model

3. Input Image Size

The next step is to define the input image size. Because we know the model was trained for image sizes of 100x100, we enter this image size and hit the "Save Image Size" button.

Fig. 42 Defining input size for CNN model

4. Selecting Input Folder

The input folder is then selected by tapping the "Select Input Folder" button and selecting "Cat Folder." This collection contains several photographs of dogs and cats for which the model has been trained.

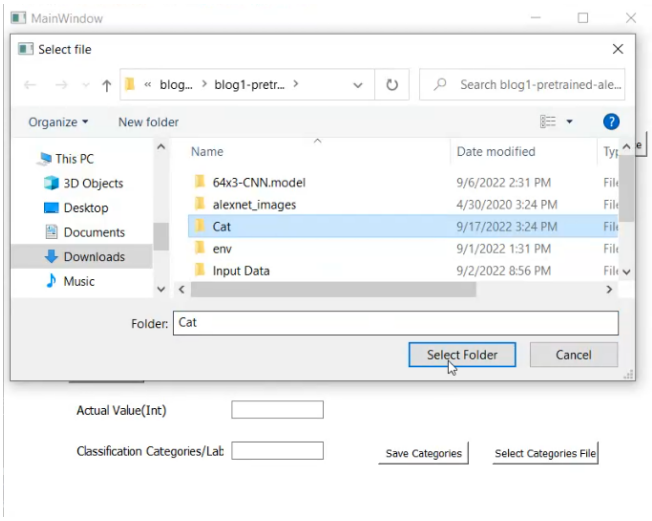

Fig. 43 Selecting Input folder where images are located

5. Selecting Output Folder

Next, we click the "Select Output Folder" button and navigate to the folder where we want the report and confusion matrix saved.

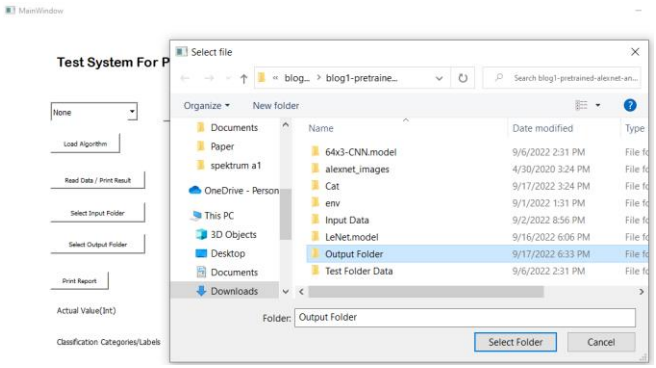

Fig. 44 Selecting Output folder where result and confusion matrix will be stored

6. Selecting Categories/Output Labels

We may now enter output labels. Because the models were trained on two categories/labels, "Cat and Dogs," we can enter them in the classification categories/label field, and because we want correct prediction when the actual value in integer form of label is 0, we enter that in the Actual value field, and then press the "Save Categories" button.

Fig. 45 Adding output labels of model and actual value of label

7. Read Data and Begin Testing

The next step is to hit the "Read Data/ Print Result" button, which loads the algorithm and tests each file in the input folder, creating a report in the output folder.

8. Print Report and Confusion Matrix

When the testing is finished, the window should say "Evaluation Done." We may produce the confusion matrix by clicking on "Print Report."

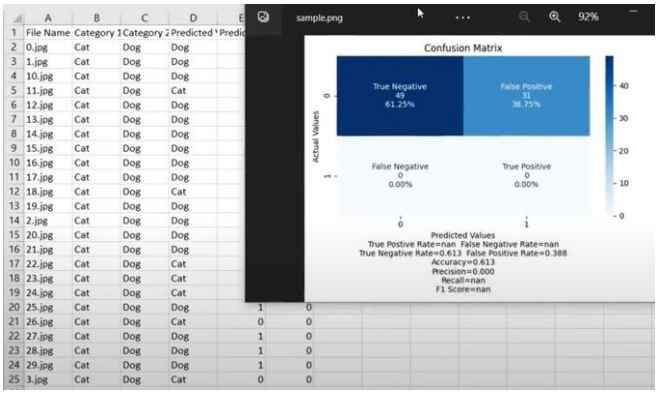

Fig. 46 Generated report and Confusion matrix of the CNN model

C. *SVM Model*

Using our system, we are now analyzing a pre-trained SVM Model. This model has been trained to distinguish between cats and dogs. The following are the steps we took:

1. Selecting ML Model

We selected "Random SVM" as the model and clicked the "Save Model Name" button.

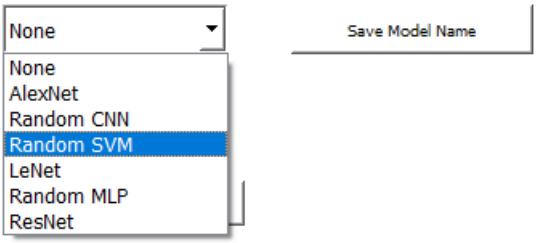

Fig. 47 Picking an SVM model from the drop-down menu

2. Loading Model

We load the algorithm after picking the model by clicking the "Load Algorithm" button, and the window shows "Random SVM Loaded."

Fig. 48 Loading Random SVM model

3. Input Image Size

The next step is to specify the size of the input image. We enter this image size and press the "Save Image Size" button since we know the model was trained for image sizes of 150x150.

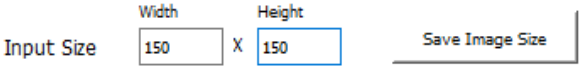

Fig. 49 Defining input size for SVM model

4. Selecting Input Folder

After that, pick the input folder by hitting the "Select Input Folder" button and selecting "Cat Folder." This collection includes images of dogs and cats for which the model has been trained.

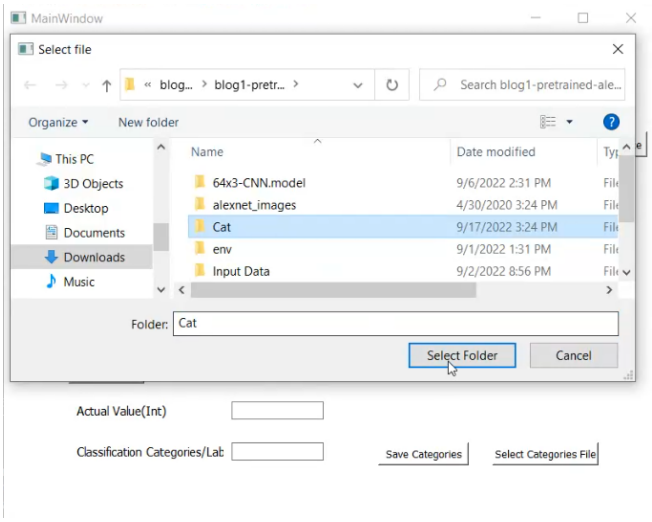

Fig. 50 Choosing the Input folder where the photos are stored

5. Selecting Output Folder

Afterwards we click the "Select Output Folder" button and go to the directory where we want the report and confusion matrix saved.

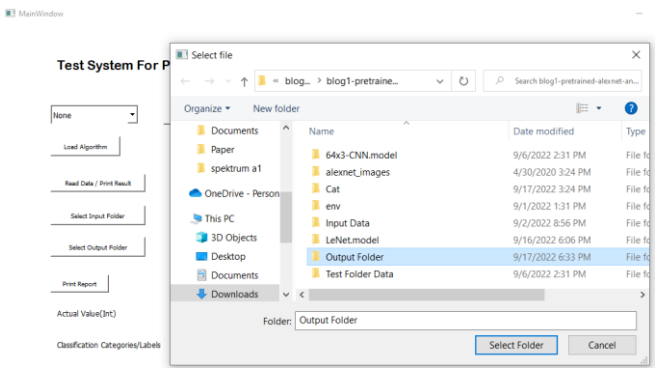

Fig. 51 Choosing an output folder to save the results and confusion matrix

6. Selecting Categories/Output Labels

We can now add output labels. Since the models were trained on two categories/labels, "Cat and Dogs," we can enter them in the classification categories/label field, and because we want correct prediction when the actual value of the label in integer form is 0, we enter that in the Actual value field, and then press the "Save Categories" button.

Fig. 52 Adding output labels of model and actual value of label

7. Read Data and Begin Testing

The next step is to click the "Read Data/ Print Result" button to load the algorithm and test each file in the input folder, producing a report in the output folder.

Fig. 53 Adding output labels of model and actual value of label

8. Print Report and Confusion Matrix

When the testing is completed, the window should display "Evaluation Complete." We may generate the confusion matrix by selecting "Print Report."

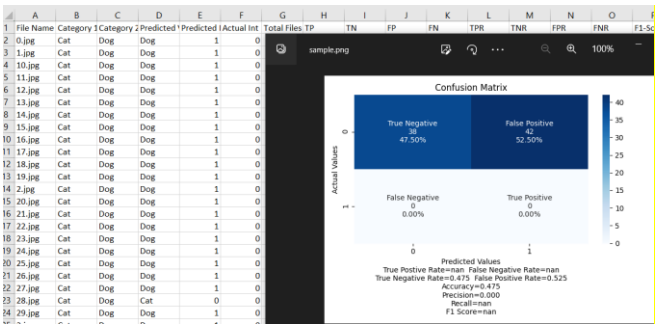

Fig. 54 Generated report and Confusion matrix of the SVM model

D. Lenet Model

We are now analyzing a pre-trained LeNet Model using our system. This model has been trained to distinguish between cats and dogs. We took the following steps:

1. Selecting ML Model

We selected "LeNet" as the model and clicked the "Save Model Name" button.

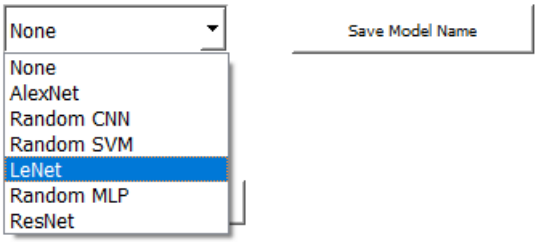

Fig. 55 Selecting LeNet model from the drop-down menu

2. Loading Model

We load the algorithm after selecting the model by clicking the "Load Algorithm" button, and the window shows "LeNet Loaded."

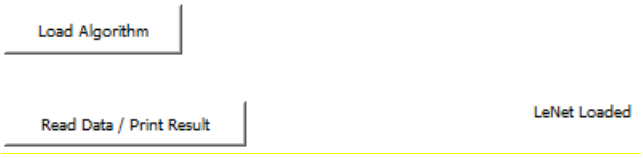

Fig. 56 Loading LeNet model

3. Input Image Size

The following step is to specify the size of the input picture. Because we know the model was trained for picture sizes of 32x32, we enter this image size and click the "Save Image Size" option.

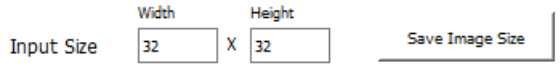

Fig. 57 Defining input size for LeNet model

4. Selecting Input Folder

The input folder is then chosen by pressing the "Select Input Folder" button and selecting "Cat Folder." This collection includes images of dogs and cats for which the model has been trained.

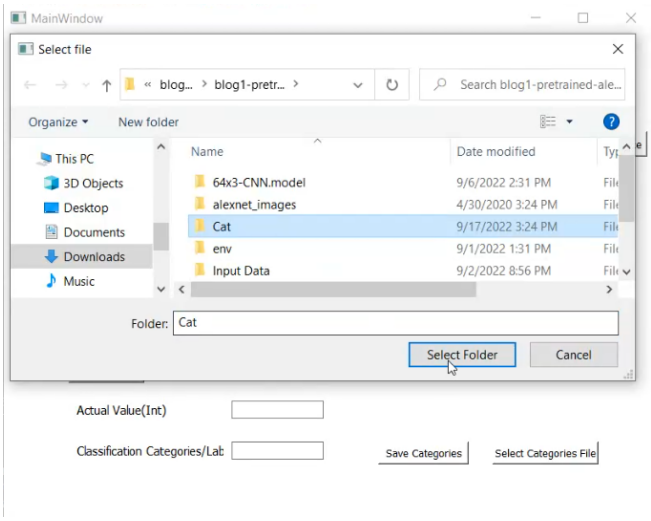

Fig. 58 Selecting Input folder where images are located

5. Selecting Output Folder

Following that, we click the "Select Output Folder" option and go to the folder where we want the report and confusion matrix saved.

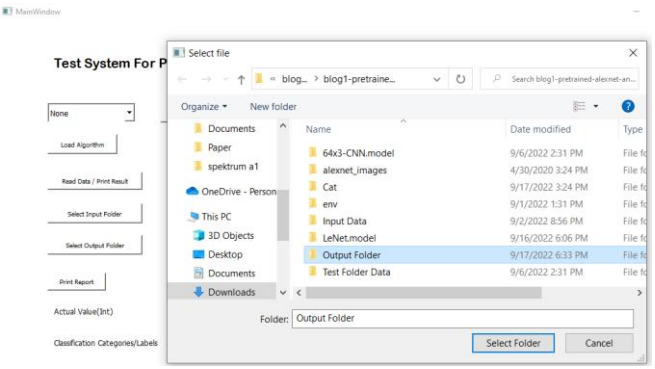

Fig. 59 Selecting Output folder where result and confusion matrix will be stored

6. Selecting Categories/Output Labels

Now, we may enter output label names Since the models were trained on two categories/labels, "Cat and Dogs," we are able to enter them within the classification categories/label field, as a result, we want correct prediction, we enter that in the Actual value field and after that press the "Save Categories" button.

Fig. 60 Adding output labels of model and actual value of label

7. Read Data and Begin Testing

The next step is to press the "Read Data/ Print Result" button, which loads the algorithm and tests each file in the input folder, producing a report in the output folder.

Fig. 61 Adding output labels of model and actual value of label

8. Print Report and Confusion Matrix

Now, we may produce the confusion matrix by clicking on "Print Report."

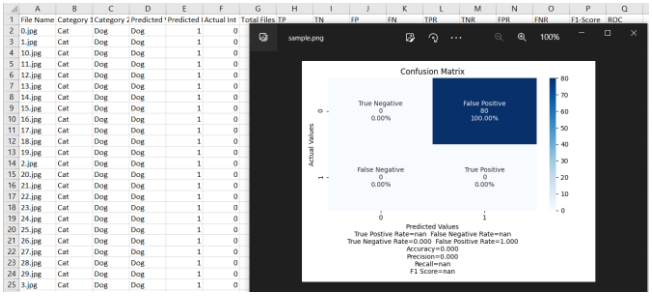

Fig. 62 Generated report and Confusion matrix of the LeNet model

E. MLP Model

We are now utilizing our system to evaluate a pre-trained MLP Model. This model was trained to distinguish between cats and dogs. Here are the steps we took:

1. Selecting ML Model

We called the model "Random MLP" and clicked the "Save Model Name" button.

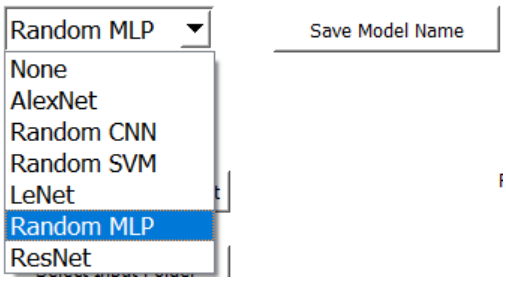

Fig. 63 Selecting Random MLP model from the drop-down menu

2. Loading Model

After selecting the model, we load the algorithm by clicking the "Load Algorithm" button, and the window shows "Random MLP Loaded."

Fig. 64 Loading Random MLP model

3. Input Image Size

The next step is to specify the dimensions of the input image. We enter this image size and press the "Save Image Size" button since we know the model was trained for 100x100 images.

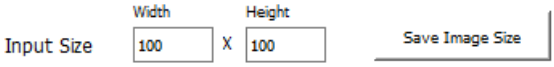

Fig. 65 Defining input size for MLP model

4. Selecting Input Folder

The input folder is then selected by tapping the "Select Input Folder" button and selecting "Cat Folder."

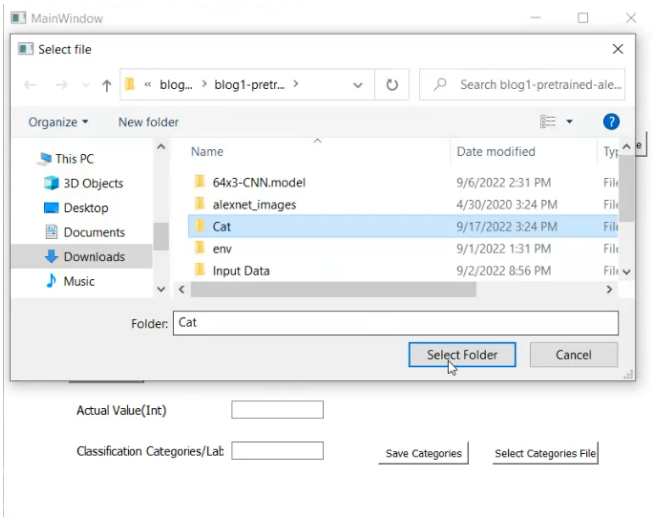

Fig. 66 Selecting Input folder where images are located

5. Selecting Output Folder

Following that, we click the "Select Output Folder" option and go to the folder where we want the report and confusion matrix recorded.

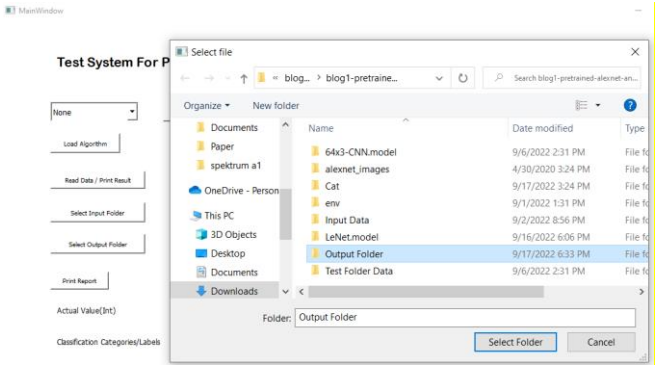

Fig. 67 Selecting Output folder

6. Selecting Categories/Output Labels

Now we repeat the same process as done in previous test by adding output labels, actual int and then press the "Save Categories" button.

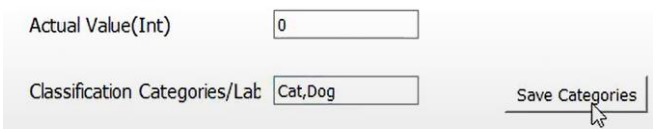

Fig. 68 Adding output labels of model and actual value of label

7. Read Data and Begin Testing

Now click the "Read Data/ Print Result" button, which loads the algorithm and tests each file in the input folder, producing a report in the output folder.

8. Print Report and Confusion Matrix

When the testing is completed, the panel should display "Evaluation Complete." We may generate the confusion matrix by selecting "Print Report."

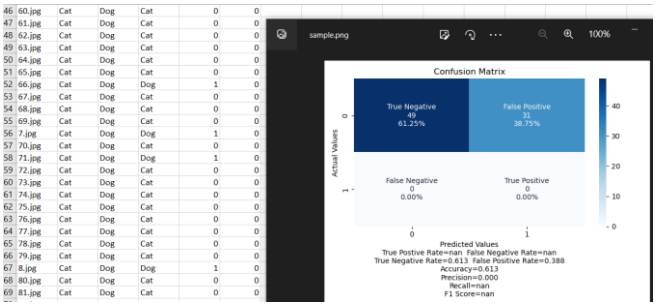

Fig. 69 Generated report and Confusion matrix of the MLP model

F. ResNet Model

Finally, we analyze the ResNet Model, which has been used to discriminate between various types of flowers such as tulips, roses, sunflowers, and so on. The following are the steps we took:

1. Selecting ML Model

The model we chose was "ResNet," and we pressed the "Save Model Name" button.

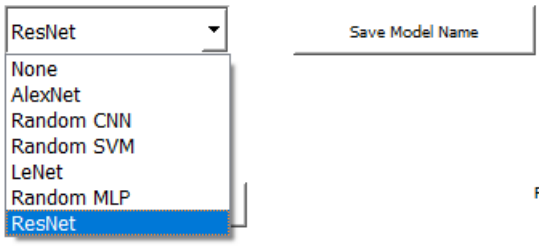

Fig. 70 Selecting ResNet model from the drop-down menu

2. Loading Model

After selecting the model, we load the algorithm by hitting the "Load Algorithm" button, and the window displays "ResNet Loaded."

Fig. 71 Loading Resnet model

3. Input Image Size

Next, we specify the input size of images which are 180x180 and hit the "Save Image Size" button.

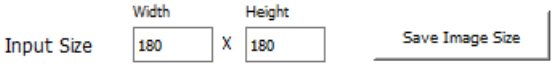

Fig. 72 Defining input size for ResNet model

4. Selecting Input Folder

The input folder is then selected by tapping the "Select Input Folder" button and selecting "Flower Input." Folder. This contains images of sunflower and roses.

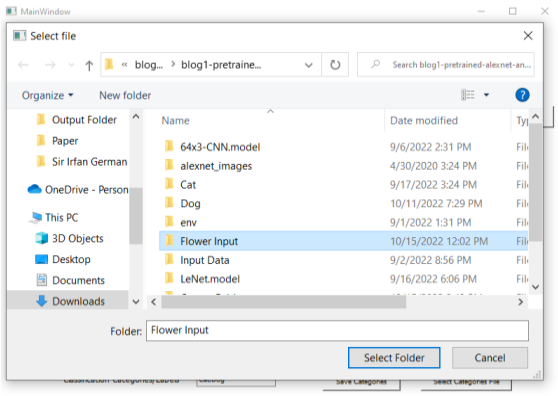

Fig. 73 Selecting Input folder where images are located

5. Selecting Output Folder

Next, we click the "Select Output Folder" button and navigate to the folder where we want the report and confusion matrix saved.

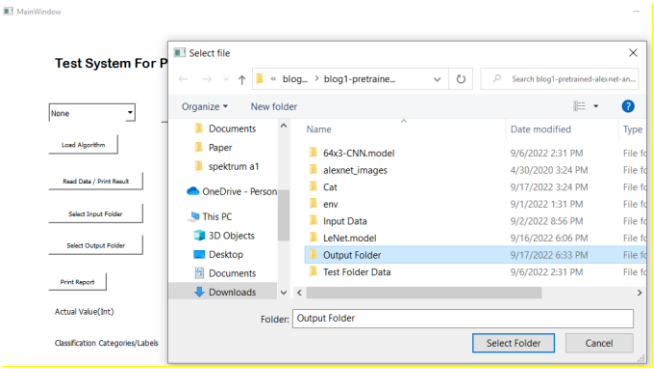

Fig. 74 Selecting Output folder where result and confusion matrix will be stored

6. Selecting Categories/Output Labels

We may now enter output labels such as roses, daisy, roses etc. and enter actual int as well and then at the end press “Save Categories” button.

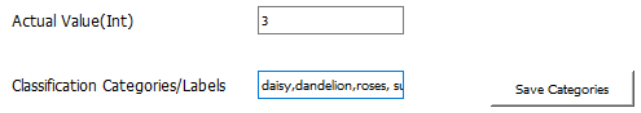

Fig. 75 Adding output labels of model and actual value of label

7. Read Data and Begin Testing

The next step is to press the "Read Data/ Print Result" button, which loads the algorithm and tests each file in the input folder, producing a report in the output folder.

Fig. 76 Adding output labels of model and actual value of label

8. Print Report and Confusion Matrix

When the evaluation has been completed, the screen should display "Evaluation Complete." We may generate the confusion matrix by selecting "Print Report."

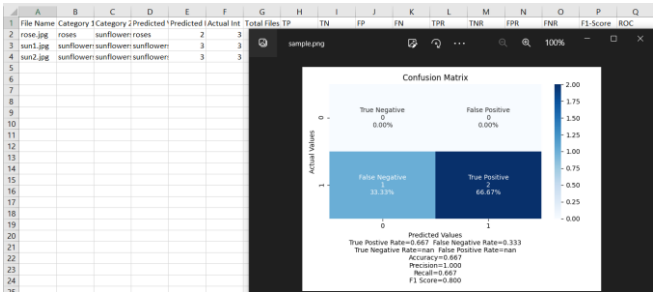

Fig. 77 Generated report and Confusion matrix of the ResNet model

VI. Conclusion

In conclusion, we were able to develop a GUI-based test system for pretrained decision algorithms. The system is able to load images, load models, define model input image size and then generate a report and confusion matrix based on selected input images and model.

VII. Acknowledgment

I would like to thank Prof. Dr. Andreas Pech for providing me an opportunity to work on this project and provided me proper guidance while working on this project and paper.

VIII. References

[1] Jordan, M.I.; Mitchell, T.M. Machine learning: Trends, perspectives, and prospects. Science 2015, 349, 255–260.

[2] LeCun, Y.; Bengio, Y.; Hinton, G. Deep learning. Nature 2015, 521, 436–444.

[3] Khandani, A.E.; Kim, A.J.; Lo, A.W. Consumer credit-risk models via machine-learning algorithms. J. Bank. Financ. 2010, 34, 2767–2787.

[4] Le, H.H.; Viviani, J.L. Predicting bank failure: An improvement by implementing a machine-learning approach to classical financial ratios. Res. Int. Bus. Financ. 2018, 44, 16–25.

[5] Dua, S.; Acharya, U.R.; Dua, P. Machine Learning in Healthcare Informatics; Springer: Berlin/Heidelberg, Germany, 2014; Volume 56

[6] Esteva, A.; Robicquet, A.; Ramsundar, B.; Kuleshov, V.; DePristo, M.; Chou, K.; Cui, C.; Corrado, G.; Thrun, S.; Dean, J. A guide to deep learning in healthcare. Nat. Med. 2019, 25, 24–29

[7] Callahan, A.; Shah, N.H. Machine learning in healthcare. In Key Advances in Clinical Informatics; Elsevier: Amsterdam, The Netherlands, 2017; pp. 279–291.

[8] Chen, T.; Guestrin, C. Xgboost: A scalable tree boosting system. In Proceedings of the 22nd ACM Sigkdd International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, 13–17 August 2016; pp. 785–794

[9] Liaw, A.;Wiener, M. Classification and regression by randomForest. R News 2002, 2, 18–22

[10] Polikar, R. Ensemble learning. In Ensemble Machine Learning; Springer: Berlin/Heidelberg, Germany, 2012; pp. 1–34.

[11] Weisberg, S. Applied Linear Regression; JohnWiley & Sons: Hoboken, NJ, USA, 2005; Volume 528.

[12] Safavian, S.R.; Landgrebe, D. A survey of decision tree classifier methodology. IEEE Trans. Syst. Man Cybern. 1991, 21, 660–674.

[13] ML Model Testing: 4 Teams Share How They Test Their Models accessed 16 October 2022,<https://neptune.ai/blog/ml-model-testing-teams-share-how-they-test-models>

[14] Dreossi, T., Ghosh, S., Sangiovanni-Vincentelli, A., and Seshia, S. A., “Systematic Testing of Convolutional Neural Networks for Autonomous Driving”, 2017.

[15] Sharma, Arnab and Heike Wehrheim. “Testing Monotonicity of Machine Learning Models.” ArXiv abs/2002.12278 (2020): n. pag.

[16] Murphy, Chris & Kaiser, Gail & Arias, Marta. (2007). An Approach to Software Testing of Machine Learning Applications. 167-.

[17] Painuli D, Mishra D, Bhardwaj S, Aggarwal M. Forecast and prediction of COVID-19 using machine learning. Data Science for COVID-19. 2021.

[18] Gavin Edwards discusses about Machine Learning: An Introduction. Available from: https:// towardsdatascience.com/machine-learning-an-introduction-23b84d51e6d0

[19] D. Gupta, A. Julka, S. Jain, T. Aggarwal, A. Khanna, N. Arunkumar, V.H.C. De Albuquerque, Optimized cuttlefish algorithm for diagnosis of Parkinson’s disease, Cognit. Syst. Res. 52 (2018) 36e48.

[20] R. Quinlan, C4.5: Programs for Machine Learning, Morgan Kaufmann Publishers, San Mateo, CA, 2014.

[21] Krizhevsky A, Sutskever I, Hinton GE. Imagenet classification with deep convolutional neural networks. Commun ACM. 2017;60(6):84–90.

[22] Zhou DX. Theory of deep convolutional neural networks: downsampling. Neural Netw. 2020;124:319–27.

[23] Jhong SY, Tseng PY, Siriphockpirom N, Hsia CH, Huang MS, Hua KL, Chen YY. An automated biometric identification system using CNN-based palm vein recognition. In: 2020 international conference on advanced robotics and intelligent systems (ARIS). IEEE; 2020. p. 1–6.

[24] Al-Azzawi A, Ouadou A, Max H, Duan Y, Tanner JJ, Cheng J. Deepcryopicker: fully automated deep neural network for single protein particle picking in cryo-EM. BMC Bioinform. 2020;21(1):1–38.

[25] Wang T, Lu C, Yang M, Hong F, Liu C. A hybrid method for heartbeat classification via convolutional neural networks, multilayer perceptrons and focal loss. PeerJ Comput Sci. 2020;6:324.

[26] Li G, Zhang M, Li J, Lv F, Tong G. Efficient densely connected convolutional neural networks. Pattern Recogn. 2021;109:107610.

[27] Gu J, Wang Z, Kuen J, Ma L, Shahroudy A, Shuai B, Liu T, Wang X, Wang G, Cai J, et al. Recent advances in convolutional neural networks. Pattern Recogn. 2018;77:354–77.

[28] Fang W, Love PE, Luo H, Ding L. Computer vision for behaviour-based safety in construction: a review and future directions. Adv Eng Inform. 2020;43:100980.

[29] Palaz D, Magimai-Doss M, Collobert R. End-to-end acoustic modeling using convolutional neural networks for hmm-based automatic speech recognition. Speech Commun. 2019;108:15–32.

[30] Li HC, Deng ZY, Chiang HH. Lightweight and resource-constrained learning network for face recognition with performance optimization. Sensors. 2020;20(21):6114.

[31] Hubel DH, Wiesel TN. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. J Physiol. 1962;160(1):106.

[32] Goodfellow I, Bengio Y, Courville A, Bengio Y. Deep learning, vol. 1. Cambridge: MIT press; 2016.

[33] Alzubaidi, L., Zhang, J., Humaidi, A.J. et al. Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. J Big Data 8, 53 (2021)

[34] LeCun Y, Jackel LD, Bottou L, Cortes C, Denker JS, Drucker H, Guyon I, Muller UA, Sackinger E, Simard P, et al. Learning algorithms for classification: a comparison on handwritten digit recognition. Neural Netw Stat Mech Perspect. 1995;261:276.

[35] Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R. Dropout: a simple way to prevent neural networks from overfitting. J Mach Learn Res. 2014;15(1):1929–58.

[36] Dahl GE, Sainath TN, Hinton GE. Improving deep neural networks for LVCSR using rectified linear units and dropout In: 2013 IEEE international conference on acoustics, speech and signal processing. IEEE; 2013. p. 8609–13.

[37] Xu B, Wang N, Chen T, Li M. Empirical evaluation of rectified activations in convolutional network; 2015. arXiv preprint arXiv: 1505. 00853.

[38] Hochreiter S. The vanishing gradient problem during learning recurrent neural nets and problem solutions. Int J Uncertain Fuzziness Knowl Based Syst. 1998;6(02):107–16.

[39] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," in Proceedings of the IEEE, vol. 86, no. 11, pp. 2278-2324, Nov. 1998, doi: 10.1109/5.726791.

[40] An Overview of ResNet and its Variants accessed 16 October 2022,<https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035>

[41] He K, Zhang X, Ren S, Sun J. Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition; 2016. p. 770–8.

[42] Xie S, Girshick R, Dollár P, Tu Z, He K. Aggregated residual transformations for deep neural networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition; 2017. p. 1492–500.

[43] Su A, He X, Zhao X. Jpeg steganalysis based on ResNeXt with gauss partial derivative filters. Multimed Tools Appl. 2020;80(3):3349–66.

[44] Yadav D, Jalal A, Garlapati D, Hossain K, Goyal A, Pant G. Deep learning-based ResNeXt model in phycological studies for future. Algal Res. 2020;50:102018.

[45] Han W, Feng R, Wang L, Gao L. Adaptive spatial-scale-aware deep convolutional neural network for high-resolution remote sensing imagery scene classification. In: IGARSS 2018-2018 IEEE international geoscience and remote sensing symposium. IEEE; 2018. p. 4736–9.

[46] Multilayer Perceptron Explained with a Real-Life Example and Python Code: Sentiment Analysis accessed 16 October 2022,<https://towardsdatascience.com/multilayer-perceptron-explained-with-a-real-life-example-and-python-code-sentiment-analysis-cb408ee93141>

[47] Minsky M. L. and Papert S. A. 1969. Perceptrons. Cambridge, MAMIT Press.

[48] D. Rumelhart, G. Hinton, and R. Williams. Learning Representations by Back-propagating Errors. Nature 323 (6088): 533–536 (1986).

[49] Support Vector Machines with Scikit-learn Tutorial accessed 16 October 2022,<https://www.datacamp.com/tutorial/svm-classification-scikit-learn-python>

[50] Creating Graphical User Interfaces (GUIs) with QT in Python accessed 16 October 2022, <https://biomedicalhub.github.io/python-data/pyqt.html>

[51] Qt Designer and Python: Build Your GUI Applications Faster accessed 16 October 2022, <https://realpython.com/qt-designer-python/>