
Quantum Ghost Imaging

Project Report – Summer Internship Program 2023



Submitted by

Akhilesh Dubey
Kirori Mal College, University of Delhi

Supervisor

Dr. Shashi Prabhakar
AMOPH Division,
Physical Research Laboratory,
Ahmedabad, India

JULY 2023

Summer Internship Program 2023

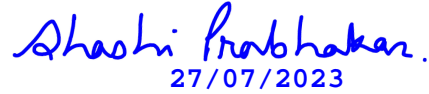
A project report entitled
Quantum Ghost Imaging

by



Akhilesh Dubey
27/07/2023

Akhilesh Dubey
(SIP 2023 Student)



Shashi Prabhakar.
27/07/2023

Dr. Shashi Prabhakar
(Project supervisor)

Thesis printed on: July 27, 2023
Place: PRL, Ahmedabad

Contents

1 Introduction	1
2 Imaging Methods	1
3 Computational ghost imaging	2
4 Scanning Methods	3
5 Machine Learning	4
5.1 Convolutional Neural Networks (CNNs)	5
5.2 Model architecture of VGG16	5
6 Results and discussion	7
6.1 Image generation & exploration	7
6.2 Training the model	7
6.3 Testing images using the trained model	8
6.4 Character recognition	9
7 Conclusion and future outlook	11
8 References	12
9 Appendix	12
9.1 Code for generating training images	12
9.2 Code for generating testing images	14
9.3 Code for training the model	15
9.4 Code for testing the imaging and saving the probabilities in .txt file	17
9.5 Code for reading the .txt file and generating the probability predictions in .png format	19

1 Introduction

Imaging and object identifications are the two integral parts used in several applications. One example is the camera on a traffic signal, identifying the numbers from the license plates. This method generally works in broad light. The imaging is performed using a camera which collects sufficient amount of light in the day and adequate enough from street lights at night, and the object/character identification is performed computationally using character recognition. The challenge arrives when the light collected by the sensor to record an image is less, not enough from the street light.

If the imaging part is not sufficient, the character recognition technique will also not be efficient. There are several economic challenges. Computational ghost imaging helps in this situation, where an image is built using a scanning mask and a high-efficient single-pixel detector. The scanning mask strategically collects light from several parts of the object, providing spatial information to the intensity recorded by the single-pixel detector, which speeds up reconstruction times. Further, the machine learning algorithm using a neural network enhances the prediction of the object recognized after each measurement. Thereby reducing the image reconstruction time by reducing the number of measurements needed.

For this Summer Internship Program 2023 PRL, we combined two techniques of computational ghost imaging and machine learning to identify the object in effectively less amount of time. The machine learning algorithms in a two-step approach – training the model and testing the image. We harnessed the power of supervised machine learning algorithms to reduce the number of measurements required to identify the object. In our case, we used the alphabets (A-Z) and 0-9 as our objects. Our method enhances the recognition algorithm to predict the confidence level of object recognition after each measurement.

This report is organized as follows. The two imaging methods are discussed in Section [2](#), computational ghost imaging in Section [3](#), and several scanning methods in Section [4](#). The various aspects related to machine learning are in Section [5](#). We discuss the results obtained from the project in Section [6](#). We finally conclude in Section [7](#) and present the future outlook too.

2 Imaging Methods

Any 2D image has two parts – 2D space and their corresponding intensities. This information is captured conventionally using a 2D array of detectors, like a 2D sensor of any camera. The light from the object reaches the sensor using two lenses, the first being the collection lens and the other being the condenser lens. The spatial intensity distribution of the image reaches the 2D array of detectors and accurately forms the image. This imaging method is depicted in Figure [1a](#)). The advantage of this method is that the complete image is recorded in one frame of necessary exposure time. The disadvantage being the 2D sensors doesn't function well in low-light applications, due to low gain and high cross-talk in the readout process.

Considering a situation where the sensor is only a bucket/single-pixel detector. These detectors have conventionally high detection efficiency, which make it suitable for low-light applications. In this situation, the sensor cannot obtain spatial information. In this situation, a scanning mask is placed between the two lenses. The correlation between the total light

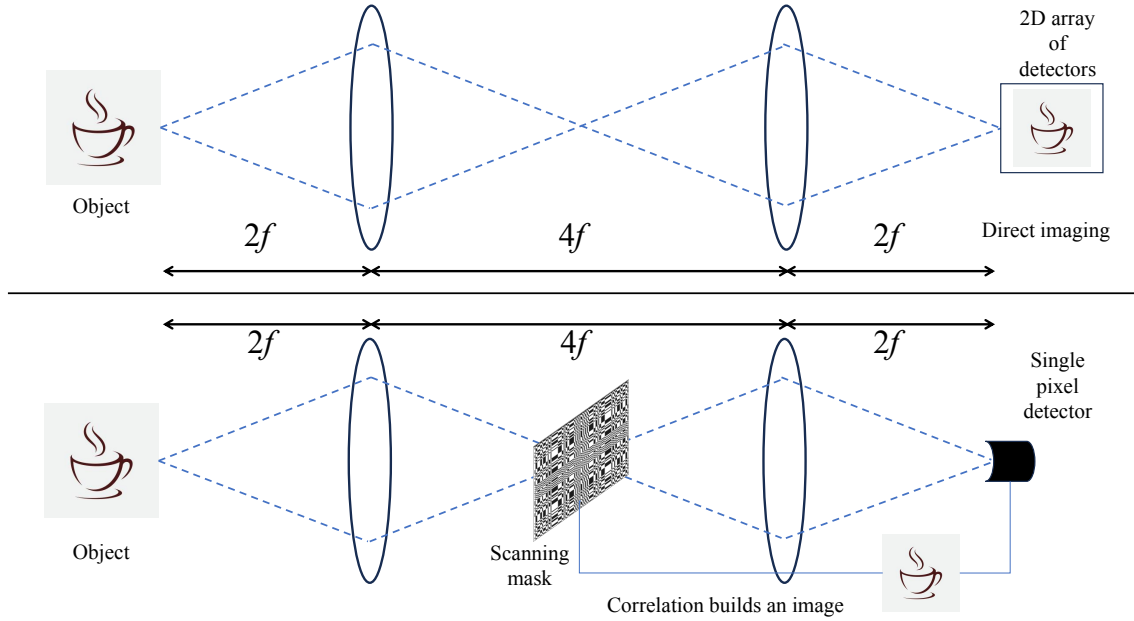


Figure 1: Two types of imaging method. a) Imaging with a 2D detector, b) Imaging with a single pixel detector and using mask.

collected by the sensor and the scanning mask provides the image of an object. This imaging method is depicted in Figure 1b). This method has the advantage that a single-pixel detector can also obtain an image. The disadvantage is that the image will always be flawed due to the noise in the single pixel's intensity. Several scanning techniques are presented in section 4

In this report, we focused mainly on the imaging using single pixel detectors. This method will find applications in quantum sensing where the source has low intensity, of the order of few thousand photons/second. This method is commonly known as Computational ghost imaging, leading to quantum ghost imaging, and we will focus mainly on this topic.

3 Computational ghost imaging

Ghost imaging is an alternative image acquisition technique that uses a correlation between a spatial mask, and the photons/intensity collected using a single-pixel detector. It was first demonstrated as a quantum entanglement phenomenon generated from the non-linear spontaneous parametric down-conversion (SPDC) process. But it is also shown that classical correlation can similarly be used in a ghost imaging experiment. It was thought to produce higher-quality images, but it has been demonstrated both in quantum and classical cases, and images of almost identical quality are produced. As an advantage, using quantum light allows for low light levels imaging applications where it is beneficial to reduce the risk of photo-damage to light-sensitive matter. The primary aim of ghost imaging is to reconstruct the image with a more efficient higher resolution, and the time taken for it should be minimal. Still, ghost imaging faces inefficient imaging speed since imaging speed depends on many measurements needed to form an image that works on quadratic scales with the required resolution (pixel²), which limits getting a higher resolution image. Traditionally, the image is reconstructed as a

linear combination of all masks weighted by the coincidences and is expressed as

$$I_N = \sum_{i=1}^N c_i M_i, \quad (1)$$

where I_N is the image after N measurements, c_i and M_i are the intensity recorded by the single-pixel detector and the corresponding mask for the i^{th} measurement, respectively.

4 Scanning Methods

The various scanning methods, not limited to, are shown in Figure 2 and discussed below. The dark spots have no transmission, while white spots full transmit the light. This is how, the spatial information is transferred to the single-pixel detectors.

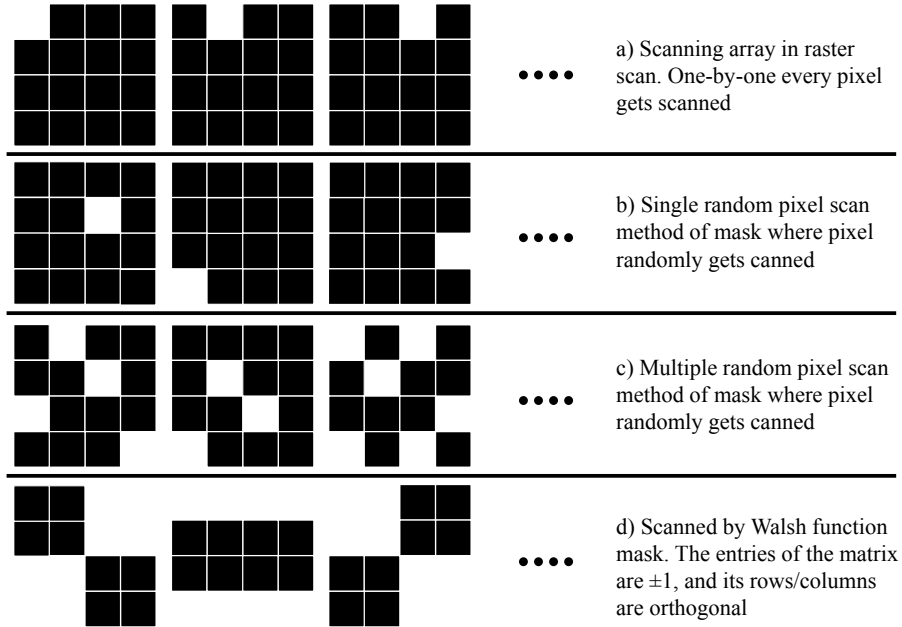


Figure 2: Four types of masks having 4×4 pixels, not limited to, used for scanning: Raster scan, Single random pixel scan, multiple random pixels scan, and Walsh matrix scan. Only three combinations of each case is shown for reference.

- a) **Raster scan:** This is a common technique used in imaging, where a beam of light scans across the object, point by point, horizontally and then then vertically. The intensity of the transmitted light from the corresponding mask is measured, producing a 2D image of the object. The complete image can be formed only after scanning a pixel^2 number of measurements.
- b) **Single random pixel scan:** This scanning method involves using a mask with only one random pixel transmitting light. An approximate image can be formed in fewer measurements; however, an image with complete contrast may not be obtained even in many measurements.

- c) **Multiple random pixels scan:** This scanning has a random distribution of transparent and opaque areas, which allows for obtaining the intensity. The ratio of transmitting and blocking pixels can be manipulated depending on the application. An approximate image can be formed in fewer measurements; however, complete contrast may not be obtained even in many measurements.
- d) **Walsh matrix scan:** This is a complex scanning method. The Walsh matrix, a Hadamard matrix, is a square matrix with either ± 1 entries. This matrix is used as a scanning function in a defined manner. Using the Walsh matrix as a scanning mask can increase the quality of the obtained images and the efficiency of the scanning process.

In terms of the experiments, these masks and methods can be used in various combinations. For instance, a random mask can provide broad, non-specific information about the object, while raster scanning may allow for imaging with higher accuracy. The Walsh matrix scan, being the most complex method, could be used for more advanced imaging tasks where the highest resolution or specific spatial information about the object is required. A sample image of character ‘1’ is shown in Figure 3

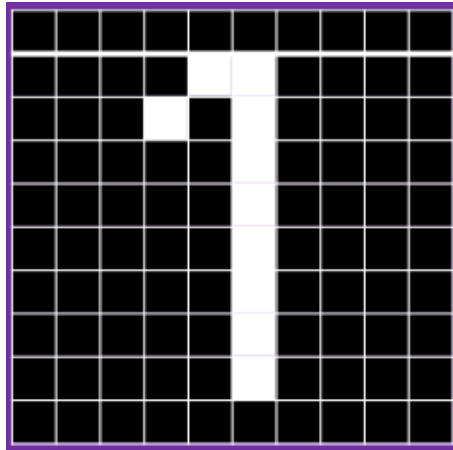


Figure 3: A sample image of ‘1’ constructed using the raster scan method with no noise. The image is highly pixelated, however the character ‘1’ is easily identified.

As we are now aware of the advantages of the ghost imaging technique and the project requires character recognition, we employed a machine learning algorithm to identify it sooner, i.e., minimum value of N in Eq. 1. This would help us to perform a minimum number of measurements with noise and obtain the object accurately.

5 Machine Learning

Machine learning is the science and art of programming computers so they can *learn from the data*. In a more engineering-oriented way, a computer program is said to learn from experience E for some task T and some performance measure P , if its performance on T , as measured by P , and improves E (by Tom Mitchell. 1997). Figure 4 shows a schematic of the basic machine learning approach.

In summary, we can write that the technique of machine learning is a great use for

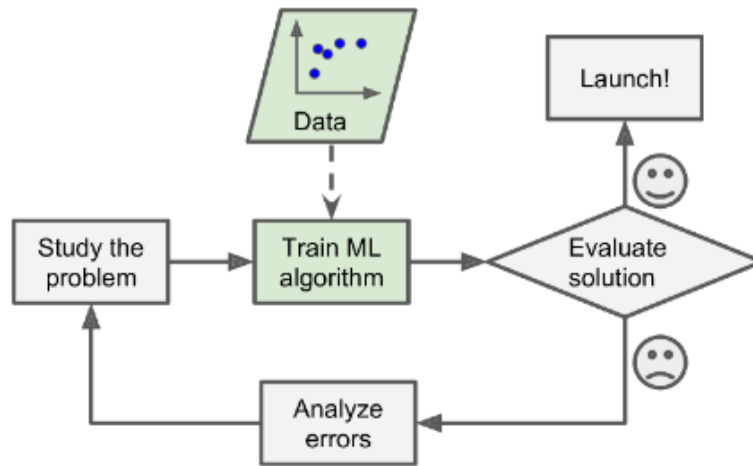


Figure 4: A schematic of machine learning approach.

- problems for which existing solutions require lot of hand-tuning or long lists of rules: one Machine Learning algorithm can often simplify code and perform better
- complex problems for which there is no good solution at all using a traditional approach: the best Machine learning techniques can find a solution
- fluctuating environments: a machine learning system can adapt to new data
- getting insights about complex problems and large amounts of data

5.1 Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNNs) are a type of artificial neural network designed to process data with a grid-like topology, such as an image. CNNs use three basic types of layers convolutional layers, pooling layers, and fully connected layers refer to Figure 5. Neurons in the first convolutional layer are not connected to every pixel in the input image, but only to pixels in their receptive fields. In turn, each neuron in the second convolutional layer is connected only to neurons located within a small rectangle in the first layer. Also, neurons in a convolutional layer are not connected to every single output of the previous layer, but only to a subset of them, defined by the dimensions of the convolutional kernel. This substantially reduces the number of parameters and helps the network to focus on localized features.

This architecture allows the network to concentrate on small low-level features in the first hidden layer, assemble them into larger higher-level features in the next hidden layer, and so on. This hierarchical structure is common in real-world images, which is one of the reasons why CNNs work so well for image recognition.

5.2 Model architecture of VGG16

We used the VGG16 (Visual Graphics Group 16) model for image classification tasks. VGG16 is a pre-trained model as the base model with 16 layers (13 convolution and 3 fully connected layers). It is widely used in the image net scale visual recognition using a pre-trained model, allowing us to use the pre-existing architecture and its trained weights to train our model

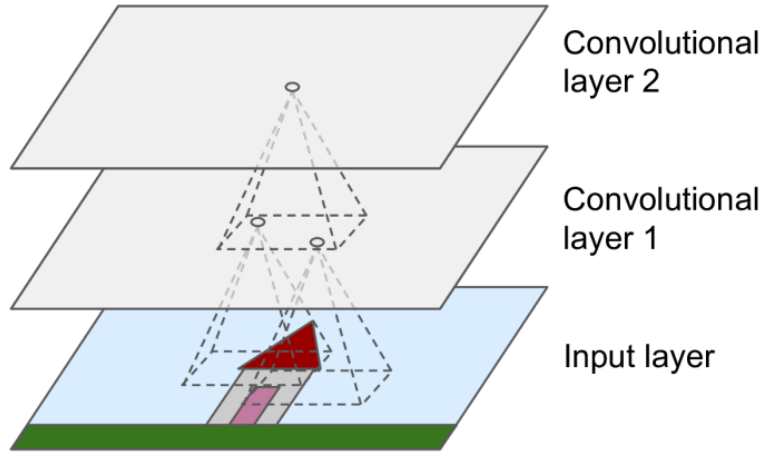


Figure 5: CNN layers with rectangular local receptive fields.

further, thereby reducing training time considerably. After the base model, the GlobalAveragePooling2D layer is added to average the spatial information, thus reducing the number of parameters in the model. Dense layers are added, fully connected, and responsible for feature extraction. In the subsequent layers, dropouts are added to randomly set a fraction of the input units to '0' at each update during training time, which helps to avoid over-fitting. We use the softmax activation function in the output layer, which is ideal for multi-class classification.

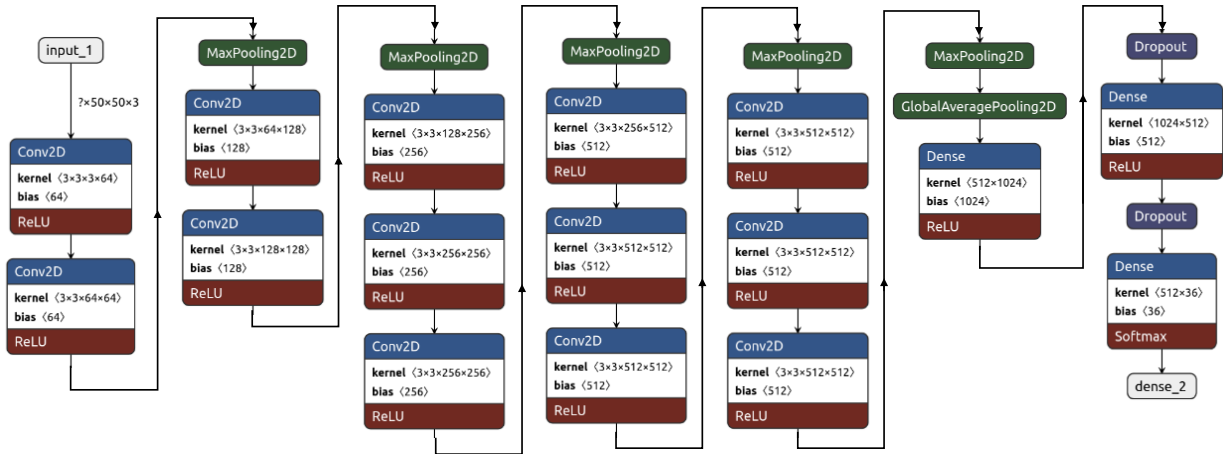


Figure 6: Flowchart for VGG16 CNN model used in this project.

Figure 6 shows the steps involved in training using the VGG16 model. The first step is to collect a data set. This data set should include images of as many different types of game buttons as possible. The images should be properly labeled with the kind of button they represent. Once the data set is generated, it needs to be pre-processed. This involves resizing the images to a standard size and normalizing the pixel values. The pre-processed images are then used to train the VGG16 model. The VGG16 model is a convolution neural network that consists of 16 layers. The first 13 layers of the model are convolutional layers, which extract features from the images. The last three layers of the model are fully connected layers, which classify the images into different classes. The VGG16 model is trained using a technique called back-propagation. Back-propagation is an iterative algorithm that adjusts the model weights to minimize the loss function, which measures how well the model performs. Once the VGG16

model is trained, it can be used to classify new images. The new images are then passed through the model, and the output is used to determine the class represented in the image.

6 Results and discussion

In this project, we applied the convolutional neural network model for image classification and recognition using the Keras library and VGG16 base model on the ghost images. The target is to minimize the value of N in Eq. 1. The various parts of the results are presented in the following subsections.

For this project, we used 36 alphanumeric characters of $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z\}$, and the resolution of the images are fixed for 50×50 pixels. Before performing the experiment, using the parameters from the experiment, and using the computational resources, we wrote the Python code to generate the possible images for both training and model and subsequently for testing the images and study their performance. All the Python codes are included in the appendix of this report.

6.1 Image generation & exploration

For training purposes, we generated 50×50 pixel for $N = 100$ images for each class. 'ObjectImage' 2D array is created with a clear representation of characters at the center with 25% noise (worst case scenario from experiment). The generated text character is converted to grayscale images, which is then normalized. We used the scanning mask type c) Multiple random pixels scan, as discussed earlier. The probability chosen for blocking:transmitting pixel is 95%. We trained the model using this image-set.

In testing purposes and to find the performance of the model, we generated only $N = 100$ images for each class with six possible orientations – normal, rotation by 90° , rotation by 180° , rotation by 270° , horizontal flip and vertical flip. The plan was to test the model with all these orientations.

6.2 Training the model

The model is trained using Adam optimizer. The adaptive learning rate approach helps in training the model efficiently. Since dealing with multi-class classification, we have used categorical cross entropy as a loss function. The batch size of 32 from our ImageDataGenerator was used, which signifies the model updates weights after training on 32 samples. The 'fit' method then trains the model for a fixed number of epochs by iterating through the complete image-set. The steps per epoch ensures the network sees all samples in each training epoch. Here we have observed that when we increase our epochs so that our model gets more iterations and time it is sufficiently gets more trained and it's performance gets enhanced on predicting the probability classes.

Here in our project for generating results, we are giving training data images for our model total of 3600 images, we are training our model for simple orientation images for 50 epochs. We have a total of 36 classes, each containing 100 images. The total time taken for training

over 36 classes on 3600 images of 50×50 pixel size is ≈ 30 minutes for 50 Epochs. Some of the images for some classes for training are shown in Figure 7

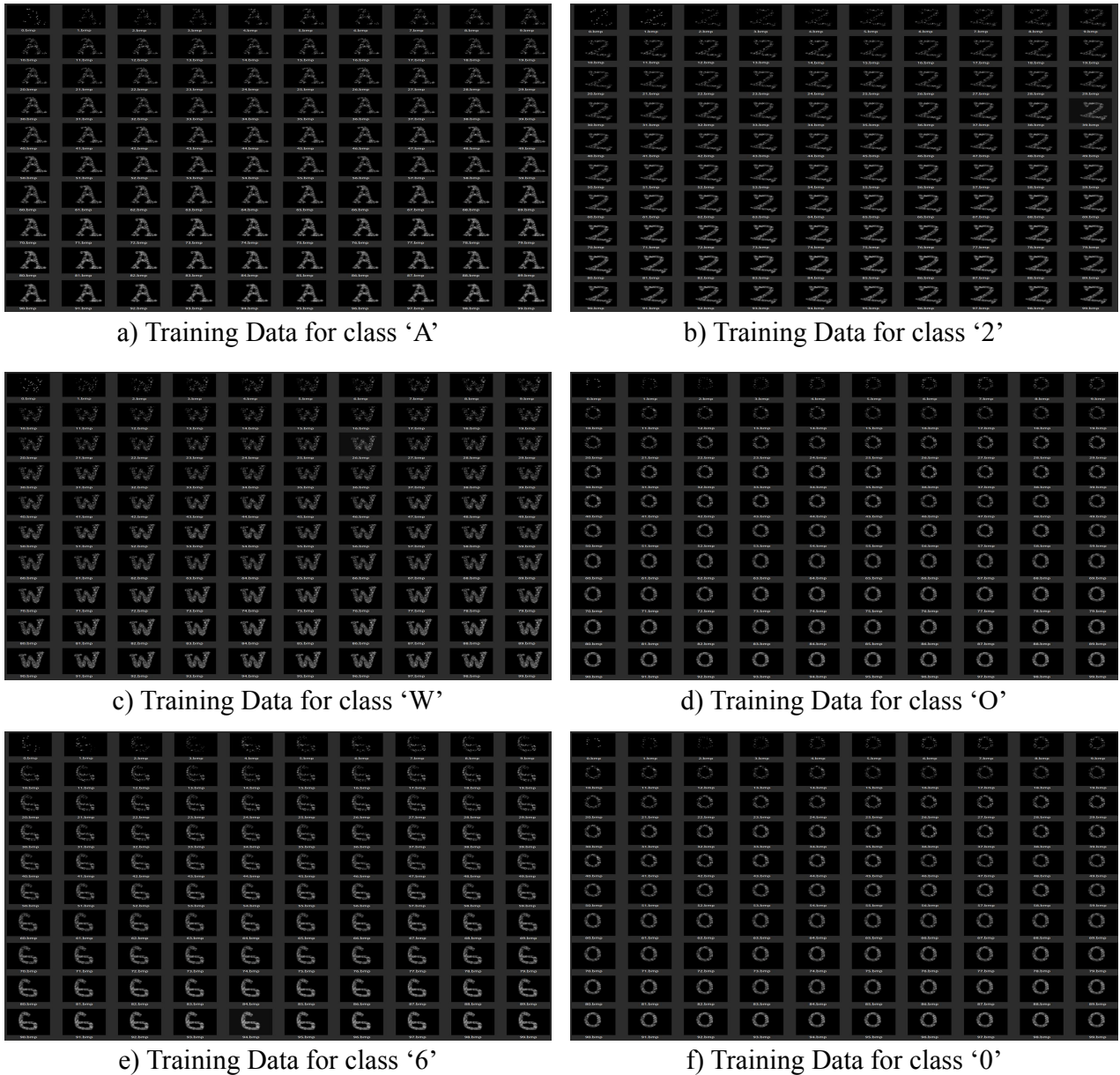


Figure 7: Some training images for few classes.

The model was trained without error, and the loss and accuracy over epochs were plotted and analyzed in Figure 8. Model accuracy improved over time while loss decreased, indicative of the model learning the underlying patterns in the training data. The model was saved and reloaded to confirm its integrity and usability for future predictions. The model used demonstrate the ability to classify images based on their contents accurately. The training and loss curve for over 500 epochs which took around 5.45 hours to run with accuracy of approximately 75%.

6.3 Testing images using the trained model

While testing our model, we tested it on for up to 50% of noise, and the results got slightly distorte, but it is satisfactory. Obviously, if we increase the noise on our testing images, it could

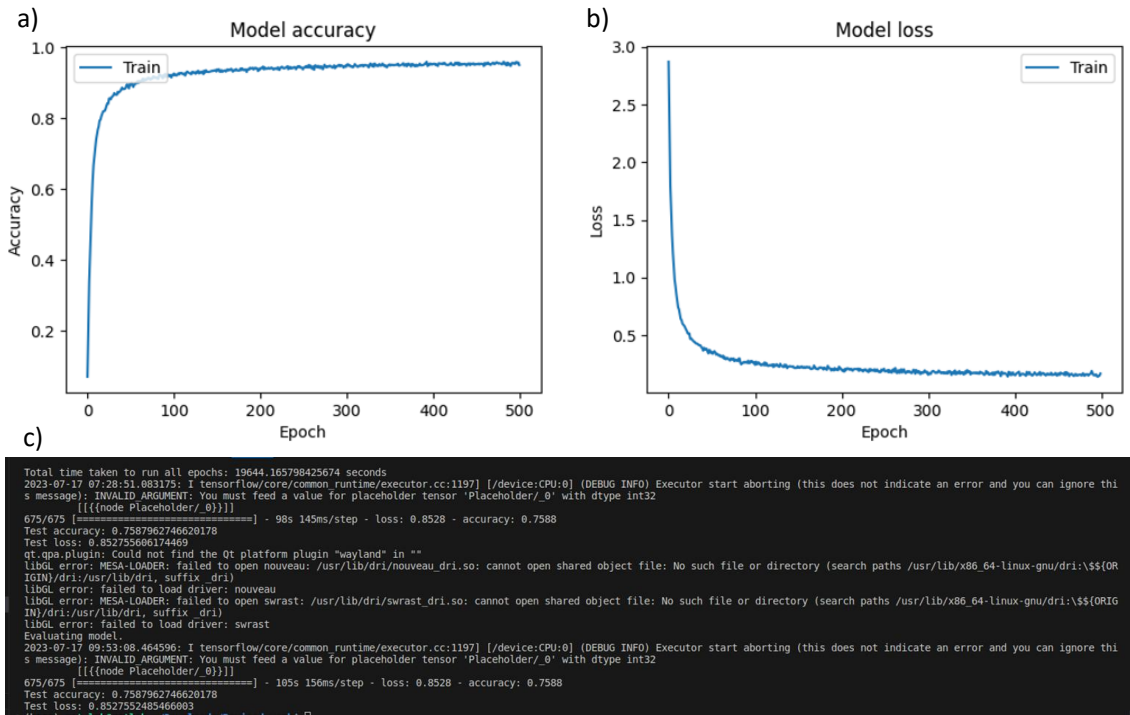


Figure 8: Model accuracy a) and model loss b) vs epoch, with accuracy results and total time take for training model c).

be challenging to recognize images since we had trained our model with 25% of noise. Here we are taking into consideration of diffusive noise in our images.

Using the trained model for 50 epochs on 25% noise, we analyzed the testing image set. The testing image set is prepared in six possible orientations. Figure 9 shows the complete set of six orientations for the class 'J'. For 50×50 pixel image for every 36 classes having six orientations, we have tested our model in 21600 images. We are testing our images for different orientations because we are analyzing at what level our model has been trained by applying rotation and flip in ImageDataGenerator function in our codes. When we tested these 36 classes of images with different noises up to 50% noise. The model could predict the class due to the robust testing model, even if the information is lost due to larger noise. The testing time for more noise-containing images is slightly longer as our trained model finds complexity in analyzing these noisy images. To maintain the higher efficiency and uniform nature, we had to keep the aspect ratio consistent, 50×50 pixel. Also, while loading the images, they are re-scaled by dividing the pixel value by 255.

6.4 Character recognition

Some of the characters in our class set are similar to our trained model, like '6' and '9', '1' and 'l', 'O' and '0', 'N' and flipped 'Z', and others. However, using a suitable font can make it possible for the model to differentiate between them successfully. Figure 10 shows the probability distribution for various orientations of class '2', '6', 'A', and 'V'. The bright white line in increasing order shows that the images on which we are testing our model initially are not fully constructed. Over the number of iterations, the images attain their clear picture, as can also be seen from Figure 9. For class '2', we can see some other probability results are also



a) Test Data for class 'J' for normal orientation



b) Test Data for class 'J' for 90° orientation



c) Test Data for class 'J' for horizontal flip



d) Test Data for class 'J' for vertical flip



e) Test Data for class 'J' for 180° rotation



f) Test Data for class 'J' for 270° rotation

Figure 9: Some testing images for 'J' classes in various orientations.

coming because our model is instigating the superposition with 'S'. This superposition of other probability images is also coming for '6' and 'G'. Since the learning rate depends on how we train our data, we are training our model with 25% of noise and testing it on 25% of noise images. The increase in epochs of the model will help in reducing the superpositions for image recognition. The total time taken for testing 36 classes, each containing six orientations and a total of 21600 images, is ≈ 19 minutes.

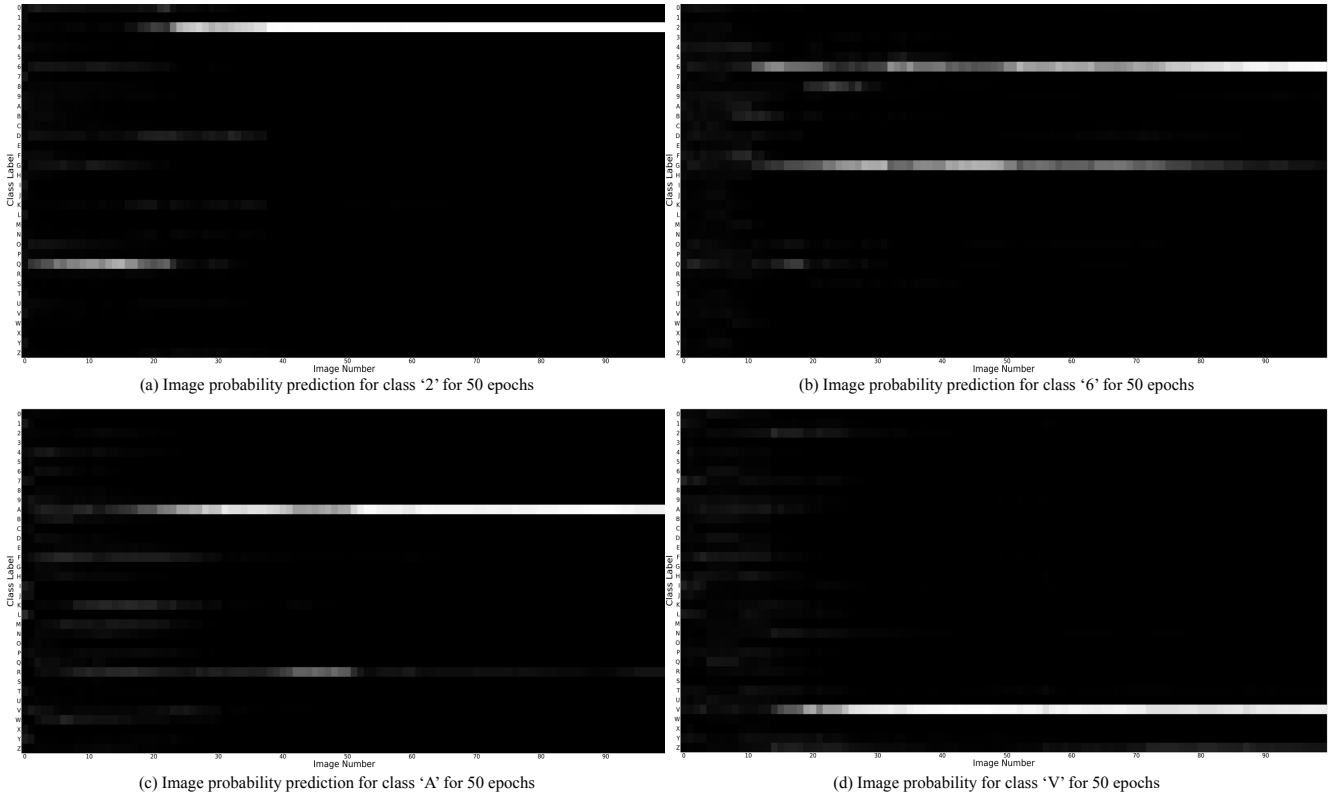


Figure 10: Probability plot. Y-axis represent the class and X-axis represent the number of images.

7 Conclusion and future outlook

In the future, we plan to test our pre-trained models to see if we can achieve better performance with more pixel images, like 100×100 . We also plan to fine-tune our model further by adjusting parameters such as learning rate, batch size, and the number of epochs to boost its accuracy and efficiency. Image pre-processing and feature extraction techniques could also be explored to optimize the model's performance. We are also considering ways to integrate this model into a real-time application and reduce the time taken; I will modify these codes, which are compatible to run on PRL VIKRAM-1000 HPC (high-performance computing) for faster results. This would demonstrate our work's practical usefulness and generate valuable user feedback for further improvements.

In summary, this image classification project has been a great learning experience in applying deep learning techniques to real-world data. With each step and improvement to our model, we are getting closer to a fully functional image classifier with the potential to impact various sectors significantly. The project has relevant applications across several domains, from autonomous vehicle systems and object-tracking surveillance systems to medical imaging diagnostics and automated image tagging in social media platforms.

Acknowledgements

I would like to express my profound gratitude to Dr. Shashi Prabhakar, my supervisor at Physical Research Laboratory, for his invaluable guidance, unwavering support, and construc-

tive criticism throughout the duration of my summer internship project on Quantum Imaging. His immense knowledge, innovative thinking, and strong expertise have been a great source of inspiration for me.

I am deeply thankful for his encouragement during the trials and tribulations of this project, his patience as I grappled with complex concepts, and his faith in my abilities. His precious advice and insights have empowered me to learn more, grow professionally, and develop a broader understanding of the field of Quantum Imaging. The experience and skills that I have acquired during this project have played a decisive role in shaping my career aspirations and future professional endeavors. I am privileged and honored to have had this unique opportunity to work under his guidance and mentor ship.

To all members of the Physical Research Laboratory, I extend my appreciation for providing a conducive work atmosphere, progressive learning environment, and advanced resources that have greatly aided the successful completion of my project.

Lastly, I would like to extend my heartfelt thanks to my colleagues and friends for their companionship, cooperation, and shared enthusiasm during this summer internship. This project would not have been possible without the help and support of these wonderful individuals. My gratitude goes to each one of them. I am truly humbled by the entire experience.

Thank you all.

8 References

References

- [1] Chané Moodley, and Andrew Forbes. “Does Quantum Ghost Imaging Need a Camera to Image an Object?” In Laser Beam Shaping XXII, 12218:7–17.// SPIE, 2022. <https://doi.org/10.1117/12.2637900>.
- [2] Chané Moodley, and Andrew Forbes. “Quantum imaging gets super smart” In Laser Beam Shaping XXII, 12218:7–17. SPIE, 2022. <https://doi.org/10.1117/12.2632911>.
- [3] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. ”Deep Learning”. Deep learning publication, 2019
- [4] Aurelion Geron, Hands-on Machine Learning with Scikit-Learn, Keras & TensorFlow, 2013
- [5] TensorFlow library by python (*https://www.tensorflow.org/api_docs/python/tf/keras/utils/image_dataset_from_directory*)

9 Appendix

This section contains Python codes developed during this project.

9.1 Code for generating training images


```

1 # Here this code generates 50 cross 50 pixel of 100 images in a specific
   directory for all the 36 classes which we needed. Here we uses
   Hadamard function for generating random pixel images which is replica
   of hadamard mask which we used in our experiment.
2
3 import numpy as np
4 from PIL import Image, ImageDraw, ImageFont
5 import os
6 def CreateObject(sz, objj):
7     img = Image.new('RGB', (sz, sz), color=(0, 0, 0))
8     d = ImageDraw.Draw(img)
9     fnt = ImageFont.truetype('42.ttf', 40)
10    d.text((12, 0), objj, font=fnt, fill=(255, 255, 255))
11    gray = img.convert('L')
12    npdata = np.array(gray)
13    npdata1 = npdata / np.amax(npdata)
14    return npdata1
15 def HadamardMatrixDefine(sz, prob):
16    num0 = int(prob * sz * sz)
17    num1 = sz * sz - num0
18    Array0 = np.zeros(num0)
19    Array1 = np.ones(num1)
20    Array01 = np.concatenate((Array0, Array1))
21    return Array01
22 def HadamardMatrixShuffle(SA, sz):
23    np.random.shuffle(SA)
24    SM = np.reshape(SA, (sz, sz))
25    return SM
26 def main():
27    Size = 50
28    prob01 = 0.95
29    classes = "0123456789ABCDEFGHIJKLMNPOQRSTUVWXYZ"
30    base_dir = "kk"
31    for c in classes:
32        ObjectImage = CreateObject(Size, str(c))
33        dir_path = os.path.join(base_dir, str(c))
34        if not os.path.exists(dir_path):
35            os.makedirs(dir_path)
36        RandomArray = HadamardMatrixDefine(Size, prob01)
37        GhostImage = np.zeros((Size, Size))
38        for i in range(100):
39            RandomMask = HadamardMatrixShuffle(RandomArray, Size)
40            Correlation = ObjectImage * RandomMask
41            CC = sum(sum(Correlation)) * np.random.rand() * 1.25
42            GhostImage = GhostImage + CC * Correlation
43            GhostImage_to_save = (GhostImage) / (np.amax(np.amax(GhostImage)
44            )) * 255
45            GhostImage_to_save = GhostImage_to_save.astype(np.uint8)
46            img = Image.fromarray(GhostImage_to_save)
47            img = img.convert('L')
48            img.save(os.path.join(dir_path, '{}.bmp'.format(i)))
49 if __name__ == "__main__":
   main()

```


9.2 Code for generating testing images

```
1 #This coede is used for generating images for 50 cross 50 pixel size in six
   different orientation which is going to used in for testing and
   generating the probability prediction on that.
2
3 import numpy as np
4 from PIL import Image, ImageDraw, ImageFont
5 import os
6 def CreateObject(sz, objj):
7     img = Image.new('RGB', (sz, sz), color=(0, 0, 0))
8     d = ImageDraw.Draw(img)
9     fnt = ImageFont.truetype('42.ttf', 40)
10    d.text((12, 0), objj, font=fnt, fill=(255, 255, 255))
11    gray = img.convert('L')
12    npdata = np.array(gray)
13    npdata1 = npdata / np.amax(npdata)
14    return npdata1
15 def HadamardMatrixDefine(sz, prob):
16    num0 = int(prob * sz * sz)
17    num1 = sz * sz - num0
18    Array0 = np.zeros(num0)
19    Array1 = np.ones(num1)
20    Array01 = np.concatenate((Array0, Array1))
21    return Array01
22 def HadamardMatrixShuffle(SA, sz):
23    np.random.shuffle(SA)
24    SM = np.reshape(SA, (sz, sz))
25    return SM
26 def main():
27    Size = 100
28    prob01 = 0.95
29    classes = "0123456789ABCDEFGHIJKLMNPOQRSTUVWXYZ"
30    base_dir = "50cross50Test"
31    transformations = {"00": lambda img: img,
32                      "01": lambda img: img.transpose(Image.FLIP_LEFT_RIGHT
33                      ),
34                      "02": lambda img: img.rotate(-90),
35                      "03": lambda img: img.rotate(90),
36                      "04": lambda img: img.rotate(180),
37                      "05": lambda img: img.transpose(Image.TRANSPOSE)}
38    for c in classes:
39        ObjectImage = CreateObject(Size, str(c))
40        for trans_id, trans_func in transformations.items():
41            dir_path = os.path.join(base_dir, str(c), trans_id)
42            if not os.path.exists(dir_path):
43                os.makedirs(dir_path)
44            RandomArray = HadamardMatrixDefine(Size, prob01)
45            GhostImage = np.zeros((Size, Size))
46            for i in range(100):
47                RandomMask = HadamardMatrixShuffle(RandomArray, Size)
48                Correlation = ObjectImage * RandomMask
49                CC = sum(sum(Correlation)) * np.random.rand() * 1.25
```

```

49         GhostImage = GhostImage + CC * Correlation
50         GhostImage_to_save = (GhostImage) / (np.amax(np.amax(
GhostImage))) * 255
51         GhostImage_to_save = GhostImage_to_save.astype(np.uint8)
52         img = Image.fromarray(GhostImage_to_save)
53         img = img.convert('L')
54         img_transformed = trans_func(img)
55         img_transformed.save(os.path.join(dir_path, '{}.bmp'.format(
i)))
56 if __name__ == "__main__":
57     main()

```

9.3 Code for training the model

```

1 # This code is for 50 epochs for training the model over 50 cross 50 pixel
image size. In which we have used VGG16 transfer learning model with
CNN and Adam optimizer with Keras and tensorflow environment.
2 import os
3 import numpy as np
4 import matplotlib.pyplot as plt
5 from keras.applications.vgg16 import VGG16
6 from keras.models import Sequential
7 from keras.layers import Dense, Dropout, Flatten
8 from keras.layers import Conv2D, MaxPooling2D
9 from keras.preprocessing.image import ImageDataGenerator
10 from keras.optimizers import Adam
11 from keras.models import load_model
12 from keras.models import Model
13 from keras.layers import Dense, GlobalAveragePooling2D
14 from tensorflow.keras.preprocessing import image
15 from PIL import Image
16 import time
17 def check_image_size(path, size=(50, 50)):
18     img = Image.open(path)
19     if img.size != size:
20         print(f"Incorrect size: {path}")
21         return False
22     else:
23         return True
24 def check_image_with_pil(path):
25     try:
26         img = Image.open(path)
27         return True
28     except:
29         return False
30 def remove_corrupted_images(dir):
31     for subdir, dirs, files in os.walk(dir):
32         for file in files:
33             file_path = os.path.join(subdir, file)
34             if not check_image_with_pil(file_path):
35                 print(f"Removing corrupted image: {file_path}")
36                 os.remove(file_path)

```

```

37         elif not check_image_size(file_path):
38             print(f"Removing incorrectly sized image: {file_path}")
39             os.remove(file_path)
40 train_directory = '/home/qst-lab/Downloads/Revised_work/Downloads/
    Revised_work/kk'
41 test_directory = '/home/qst-lab/Downloads/Revised_work/Test_Image'
42 remove_corrupted_images(train_directory)
43 remove_corrupted_images(test_directory)
44 train_datagen = ImageDataGenerator(
45     rescale=1./255,
46     rotation_range=180,
47     zoom_range=0.15,
48     width_shift_range=0.2,
49     height_shift_range=0.2,
50     shear_range=0.15,
51     horizontal_flip=True,
52     vertical_flip=True,
53     fill_mode="nearest"
54 )
55 training_set = train_datagen.flow_from_directory(
56     train_directory,
57     target_size=(50, 50),
58     batch_size=32,
59     class_mode='categorical'
60 )
61 test_datagen = ImageDataGenerator(rescale=1./255)
62 test_set = test_datagen.flow_from_directory(
63     test_directory,
64     target_size=(50, 50),
65     batch_size=32,
66     class_mode='categorical'
67 )
68 base_model = VGG16(weights="imagenet", include_top=False, input_shape=(50,
    50, 3))
69 for layer in base_model.layers:
70     layer.trainable = False
71 x = base_model.output
72 x = GlobalAveragePooling2D()(x)
73 x = Dense(1024, activation="relu")(x)
74 x = Dropout(0.5)(x)
75 x = Dense(512, activation='relu')(x)
76 x = Dropout(0.5)(x)
77 predictions = Dense(training_set.num_classes, activation='softmax')(x)
78 model = Model(inputs=base_model.input, outputs=predictions)
79 model.compile(optimizer=Adam(), loss='categorical_crossentropy', metrics=['
    accuracy'])
80 print("Starting training.")
81 start = time.time()
82 history = model.fit(training_set, steps_per_epoch=len(training_set), epochs
    =50)
83 end = time.time()
84 total_time = end - start
85 print(f"Total time taken to run all epochs: {total_time} seconds")

```

```

86 test_loss, test_accuracy = model.evaluate(test_set)
87 print(f"Test accuracy: {test_accuracy}")
88 print(f"Test loss: {test_loss}")
89 plt.figure(figsize=(12, 4))
90 plt.subplot(1, 2, 1)
91 plt.plot(history.history['accuracy'])
92 plt.title('Model accuracy')
93 plt.ylabel('Accuracy')
94 plt.xlabel('Epoch')
95 plt.legend(['Train'], loc='upper left')
96 plt.subplot(1, 2, 2)
97 plt.plot(history.history['loss'])
98 plt.title('Model loss')
99 plt.ylabel('Loss')
100 plt.xlabel('Epoch')
101 plt.legend(['Train'], loc='upper right')
102 plt.show()
103 model.save('my_model_VGG16.h5')
104 model = load_model('my_model_VGG16.h5')
105 print("Evaluating model.")
106 loss, accuracy = model.evaluate(test_set)
107 print(f"Test accuracy: {accuracy}")
108 print(f"Test loss: {loss}")
109 class_labels = training_set.class_indices
110 class_labels = {v: k for k, v in class_labels.items()}
111 def predict_possible_labels(img_path, top_n=2):
112     img = image.load_img(img_path, target_size=(50, 50))
113     x = image.img_to_array(img)
114     x = np.expand_dims(x, axis=0)
115     images = np.vstack([x])
116     images /= 255.
117     probs = model.predict(images, batch_size=10)
118     pred_probs_indices = np.argsort(probs[0])[-top_n:][::-1]
119     pred_labels_and_probs = [(class_labels[i], probs[0][i]) for i in
120                             pred_probs_indices]
121     return pred_labels_and_probs

```

9.4 Code for testing the imaging and saving the probabilities in .txt file

```

1 # This code generates a .txt file which is referring for prediction results
  # probability of 21600 images on a pretrained model which is saved in .
  # h5 format.
2 from keras.models import load_model
3 from tensorflow.keras.preprocessing import image
4 import numpy as np
5 import os
6 import time
7 def predict_all_labels(img_path):
8     img = image.load_img(img_path, target_size=(50, 50))
9     x = image.img_to_array(img)

```

```

10     x = np.expand_dims(x, axis=0)
11     images = np.vstack([x])
12     images /= 255.
13     probs = model.predict(images, batch_size=10)[0]
14     return probs
15 def sort_func(folder_name):
16     try:
17         return int(folder_name)
18     except ValueError:
19         return 1000000 + ord(folder_name)
20 model_path = 'my_model_VGG16.h5'
21 if not os.path.isfile(model_path):
22     print(f'Model file at "{model_path}" does not exist or is not a file.')
23     exit(1)
24 model = load_model('my_model_VGG16.h5')
25 base_img_dir = '/home/qst-lab/Downloads/Step_1_Work/50cross50Test'
26 if not os.path.isdir(base_img_dir):
27     print(f'Base image directory "{base_img_dir}" does not exist or is not a
28     directory.')
29     exit(1)
30 results_file = 'Step_1_Work_for__VGG16_Revised_Train_50_epoch_noise25%
31     _Test_25%_without_rounding_off.txt'
32 start = time.time()
33 with open(results_file, 'w') as f:
34     for folder_name in sorted(os.listdir(base_img_dir), key=sort_func):
35         main_folder_path = os.path.join(base_img_dir, folder_name)
36         print(f'Accessing folder: {folder_name}')
37         if os.path.isdir(main_folder_path):
38             order = ['00', '01', '02', '03', '04', '05']
39             for subfolder_name in order:
40                 subfolder_path = os.path.join(main_folder_path,
41                 subfolder_name)
42                 print(f'Accessing subfolder: {subfolder_name}')
43                 if os.path.isdir(subfolder_path):
44                     for img_file in sorted(os.listdir(subfolder_path), key=
45                     lambda x: int(os.path.splitext(x)[0])):
46                         img_path = os.path.join(subfolder_path, img_file)
47                         if img_path.endswith('.bmp'):
48                             print(f'Predicting for image: {img_file}')
49                             predictions = predict_all_labels(img_path)
50                             predictions_str_list = [str(pred) for pred in
51                             predictions]
52                             predictions_text = ",".join(predictions_str_list
53                             )
54                             f.write(f'{folder_name},{subfolder_name},{
55                             img_file},{predictions_text}\n')
56 end = time.time()
57 total_time = end - start
58 print(f'Predictions have been written to {results_file}.')
59 print(f'Total execution time: {total_time} seconds.')

```

9.5 Code for reading the .txt file and generating the probability predictions in .png format

```
1 # This codes analyses the .txt file and generates the probabily into images
   results as shown in Figure 9.
2 import numpy as np
3 import matplotlib.pyplot as plt
4 from collections import defaultdict
5 def extract_data(line):
6     data = line.split(',')
7     try:
8         class_label = int(data[0])
9     except ValueError:
10        class_label = ord(data[0]) - 55
11    folder_label = data[1]
12    probabilities = np.array(data[3:], dtype=float)
13    return class_label, folder_label, probabilities
14 def create_plot(probability_matrix, class_label, folder_label, path_to_save)
    :
15    plt.figure(figsize=(100,50))
16    plt.imshow(probability_matrix, cmap='gray', aspect='auto')
17    class_labels = [str(i) for i in range(10)]+list('
        ABCDEFGHIJKLMNOPQRSTUVWXYZ')
18    plt.xlabel('Image Number',fontsize=90)
19    plt.ylabel('Class Label',fontsize=90)
20    plt.xticks(np.arange(0, probability_matrix.shape[1], step=10),fontsize
        =60)
21    plt.yticks(np.arange(0, probability_matrix.shape[0], step=1), labels=
        class_labels,fontsize=60)
22    plt.tight_layout()
23    save_path = f'{path_to_save}/plot_{class_label}_{folder_label}.png'
24    plt.savefig(save_path)
25    plt.close()
26 data_dict = defaultdict(list)
27 with open('Step_1_Work__VGG16_Revised_Train_50_epoch_noise25%_Test_25%
        _without_rounding_off.txt', 'r') as file:
28     for line in file:
29         class_label, folder_label, probabilities = extract_data(line)
30         data_dict[(class_label, folder_label)].append(probabilities)
31 path_to_save = "/home/qst-lab/Downloads/Step_1_Work/Raw_Images_3"
32 for (class_label, folder_label), probabilities in data_dict.items():
33     probability_matrix = np.array(probabilities).T
34     create_plot(probability_matrix, class_label, folder_label, path_to_save)
```
