Neural Networks 154 (2022) 246–254

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

Breaking CAPTCHA with Capsule Networks

Ionela Georgiana Mocanu [*], Zhenxu Yang, Vaishak Belle

University of Edinburgh, UK

A R T I C L E   I N F O

Article history:
Received 10 December 2021
Received in revised form 17 May 2022
Accepted 30 June 2022
Available online 8 July 2022

Keywords:
Capsule networks
CAPTCHA
Convolutional neural networks
Digit recognition
Spatial invariance

A B S T R A C T

Convolutional Neural Networks have achieved state-of-the-art performance in image classification. Their lack of ability to recognise the spatial relationship between features, however, leads to misclassification of the variants of the same image. *Capsule Networks* were introduced to address this issue by incorporating the spatial information of image features into neural networks. In this paper, we are interested in showcasing the digit recognition task on CAPTCHA images, widely considered a challenge for computers in relation to human capabilities. Our intention is to provide a rigorous empirical regime in which we can compare the competitive performance of Capsule Networks against the Convolutional Neural Networks. Indeed since CAPTCHA distorts images, by adjusting the spatial positioning of features, we aim to demonstrate the advantages and limitations of Capsule Networks architecture. We train the *Capsule Networks with Dynamic Routing* version and the convolutional-neural-network-based deep-CAPTCHA baseline model to predict the digit sequences on numerical CAPTCHAs, investigate the performance results and propose two improvements to the Capsule Networks model.

1. Introduction

Convolutional Neural Networks (CNNs) (LeCun, Haffner, Bottou, & Bengio, 1999) have paved the way for state-of the art performance on a number of benchmark results on *Computer Vision* tasks (Krizhevsky, Sutskever, & Hinton, 2017). Computer vision applications are widely used in domains such as automatic inspection (Neethu & Kadan, 2015), autonomous vehicles (Janai, Güney, Behl, & Geiger, 2017), biometric recognition systems (Minaee, Abdolrashidi, Su, Bennamoun, & Zhang, 2019), medicine (Shekhar, Seal, Kedia, & Guha, 2020; Thevenot, López, & Hadid, 2018) etc. About the same time, there have been concerns regarding the possibility to misclassify images, especially if faced with unfavourable lighting conditions, distortions or occlusions. This has led to a new direction of research which studies classifier's performance on adversarial examples (Koga & Takemoto, 2021; Machado, Silva, & Goldschmidt, 2020). When classifying images, CNNs appear to lack the ability to recognise the spatial relationship between features, resulting in misclassifications of variants of the same object (Alcorn, Li, Gong, Wang, Mai, Ku, & Nguyen, 2018). To overcome this, *Capsule Networks with Dynamic Routing* (Sabour, Frosst, & Hinton, 2017) (CapsNet hereafter) were proposed, which outperformed the CNN baseline in MNIST digit classification (standard database of handwritten digits used in training image processing systems LeCun, Cortes, & Burges, 2010). Since then, there has been a considerable increase in deep learning models employing CapsNet as their core element. In this paper, we are interested in testing the capabilities of these architectures further. We consider the challenge problem of CAPTCHA. CAPTCHA was first proposed by von Ahn, Blum, and Langford (2004) and is widely recognised as a type of security measure that determines whether the user is a human or a machine. Nowadays many security and login checks on websites use variants of CAPTCHAs to ensure that an individual and not a bot is using the website or service. The natural question that follows is whether *Capsule Networks* will perform well on CAPTCHA and this is not a trivial matter to check. We therefore ask, what sort of distortions in image manipulation will be necessary to comparatively assess the performance and the capabilities of *Capsule Networks* against a similar architecture such as the CNNs? In this paper, we investigate the advantages and limitations of CapsNet for the CAPTCHA task. We hypothesise that CapsNet will consistently pick up spatial information in digits to solve CAPTCHA effectively. We evaluated the performance of the CapsNet on CAPTCHA digit sequence recognition tasks and improved the CapsNet architecture within the context of these tasks. We obtained interesting results which outline CapsNet's performance after extending the task from the single-digit classification on the MNIST dataset to the digit sequence recognition on the numerical CAPTCHA datasets.

The contributions of this paper are as follows: (a) we design and generate the numerical CAPTCHA datasets (b) adapt the CapsNet to recognise the digit sequence on numerical CAPTCHA images and (c) evaluate and compare the CapsNet and the Deep-CAPTCHA on CAPTCHA sequence recognition tasks. The paper is

* Corresponding author.
E-mail address: i.g.mocanu@ed.ac.uk (I.G. Mocanu).

https://doi.org/10.1016/j.neunet.2022.06.041

structured as follows: *Preliminaries* covers the technical background of the paper, namely, the CapsNet architecture, the CAPTCHA problem and the architecture we compare our model with, the Deep-CAPTCHA. *Breaking CAPTCHA* introduces the CAPTCHA datasets and introduces the two tasks used for performance analysis and comparison. *Evaluation* section demonstrates and analyses the experiment results. *Related work* showcases previous research on *Capsule Networks*, the task of breaking CAPTCHAs and also uncertainty quantification, followed by *Conclusion*, which summarises the paper and outlines future directions.

2. Preliminaries

2.1. Capsule networks

Capsule Networks with Dynamic Routing (CapsNet) were proposed by Sabour et al. (2017). CapsNet uses neural activities that vary as the viewpoint of an object changes, which gives them the scale and position invariance properties for object representation. A CapsNet includes multiple layers of neuron groups called *capsules*. Each capsule is a vector representing the spatial properties and the presence probability of a visual entity. The direction of this vector describes instantiation parameters such as pose, deformation, texture, etc. The modulus of the vector indicates the likelihood of the visual entity being present. During the forward propagation, the output vectors from lower-level capsules are multiplied with corresponding matrices, generating prediction vectors for a capsule in the layer above.

Eq. (1) shows the steps to compute the input vector of a capsule. Suppose we have an output vector $\mathbf{u}_i$ from the ith capsule at the previous layer. We need to multiply it by a learnable weight matrix $\mathbf{W}_{ij}$ to produce a "prediction vector" $\hat{\mathbf{u}}_{j|i}$ for the jth capsule in the current layer. The input to this capsule $\mathbf{s}_j$ is a weighted sum over the prediction vectors from the previous layer's outputs.

$$\mathbf{s}_j = \sum_i c_{ij}\hat{\mathbf{u}}_{j|i}, \qquad \hat{\mathbf{u}}_{j|i} = \mathbf{W}_{ij}\mathbf{u}_i \tag{1}$$

where the c_{ij} is a coupling coefficient that will be iteratively updated by the dynamic routing. The coupling coefficient between capsule i and the capsules from the layer above it will sum to 1 and be defined by the "routing softmax" function:

$$c_{ij} = \frac{exp(b_{ij})}{\sum_k exp(b_{ik})} \tag{2}$$

where b_{ij} represent the log prior probabilities that a capsule i should be coupled to capsule j. The key point here is that the log priors are learned simultaneously as the other weights and they depend on the location and type of the two capsules rather than the current input image.

Because the modulus of the capsule output vector is the probability of the feature presence (a value between 0 and 1), a "squashing" function (Eq. (3)) is applied to the input $\mathbf{s}_j$ to generate a "potential" output $\mathbf{v}_j$ of the jth capsule. The squashing function is applied to limit the length of the temporary outputs of parent capsules between *0* and *1*.

$$\mathbf{v}_j = \frac{\|\mathbf{s}_j\|^2}{1+\|\mathbf{s}_j\|^2}\frac{\mathbf{s}_j}{\|\mathbf{s}_j\|} \tag{3}$$

We call it the "potential" output because it is subject to refinement from the dynamic routing (Algorithm 1).

Algorithm 1 represents an iterative routing-by-agreement technique, where a lower-level capsule sends its output to higher level capsules whose activity vectors have a big scalar product with the prediction coming from the lower-level capsule. In the first iteration, we weight all the prediction vectors with equal coupling coefficients and sum them up.

Algorithm 1 Dynamic Routing

```
1: procedure ROUTING($\hat{\mathbf{u}}_{j|i}$, $r$, $l$)
2:   for all capsule $i$ in layer $l$ and capsule $j$ in layer $(l+1)$ : $b_{ij} \leftarrow 0$.
3.   for $r$ iterations do
4:     for all capsule $i$ in layer $l$: $\mathbf{c}_i \leftarrow softmax(\mathbf{b}_i)$
5:     for all capsule $j$ in layer $(l+1)$: $\mathbf{s}_j \leftarrow \sum_i c_{ij}\hat{\mathbf{u}}_{j|i}$
6:     for all capsule $j$ in layer $(l+1)$: $\mathbf{v}_j \leftarrow squash(\mathbf{s}_j)$
7:     for all capsule $i$ in layer $l$ and capsule $j$ in layer $(l+1)$: $b_{ij} \leftarrow b_{ij} + \hat{\mathbf{u}}_{j|i} \cdot \mathbf{v}_j$
return $\mathbf{v}_j$
```

A higher similarity between a prediction vector and the temporary output of a parent capsule will increase the coupling coefficient for that prediction vector. In the following iterations, therefore, a larger portion of that prediction vector will be directed to the temporary output due to the increase of its coupling coefficient. Essentially, we distribute the prediction vector from a child capsule to the parent capsules based on the "agreement" between them: $a_{ij} = \hat{\mathbf{u}}_{j|i}.\mathbf{v}_j$.

The temporary output updated throughout the iterations will become the final output of the parent capsule and be fed into the next layer. Suppose that there are r routing iterations, the updates on parameters of the computational graph created by the first $r-1$ routing iterations are turned off during the backpropagation by default. CapsNet (Fig. 1) first applies a convolutional layer (256, 9×9 kernels with stride 1) and ReLU activation to the input image. The second layer (PrimaryCaps) contains another convolutional operation (256, 9×9 kernels with stride 2). Then the 256 feature maps are divided into 32 groups, each of which contains 8 feature maps. In each group, we transform the pixel values at the same spatial location across all the feature maps to an 8D primary capsule vector. Thus, the PrimaryCaps layer generates 32 groups of 6×6 8D capsules in total. Next, the model flattens the primary capsules to a list of 8D capsule vectors and distributes them with dynamic routing to the DigitCaps layer, which contains 10 digit capsules representing 10 digits. Since the length of digit capsules represents the likelihood of digit presence, we would like to encourage the digit capsule representing the correct digit to have a longer length and shorten the other capsule vectors. To train the model, we minimise the *margin loss* L_k for each digit capsule k:

$$L_k = T_k \max(0, m^+ - \|\mathbf{v}_k\|)^2 + \lambda(1-T_k)\max(0, \|\mathbf{v}_k\| - m^-)^2 \tag{4}$$

where $T_k = 1$ iff a digit of class k is present and $m^+ = 0.9$ and $m^- = 0.1$. λ is a weight to stop the initial learning from decreasing the length of all the digit capsules.

2.2. CAPTCHA

CAPTCHA (*"Completely Automated Public Turing test to tell Computers and Humans Apart"*) is a type of cybersecurity tool that determines whether the user is a human or a machine (von Ahn, Blum, Hopper, & Langford, 2003). It has been used to prevent various types of cyber security attacks, threats and penetrations of web services and websites, and lately on autonomous vehicles (Kaur & Kumar, 2014; Kulluru, Shaikh, Bidwai, Mannarwar, & Shamlani, 2016; Walch, Colley, & Weber, 2019). The security of CAPTCHA system becomes therefore a critical aspect to defending the *Internet* against automated attacks (Chen, Luo, Guo, Zhang, & Gong, 2017). The security of text-based CAPTCHA mainly depends

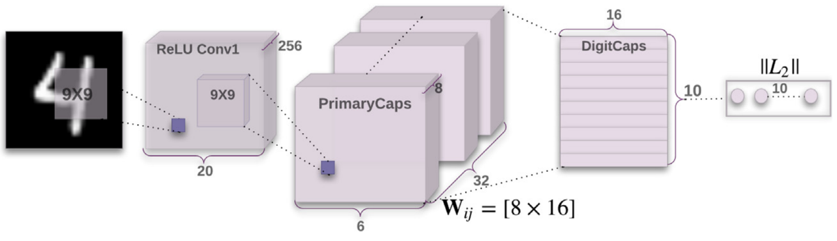
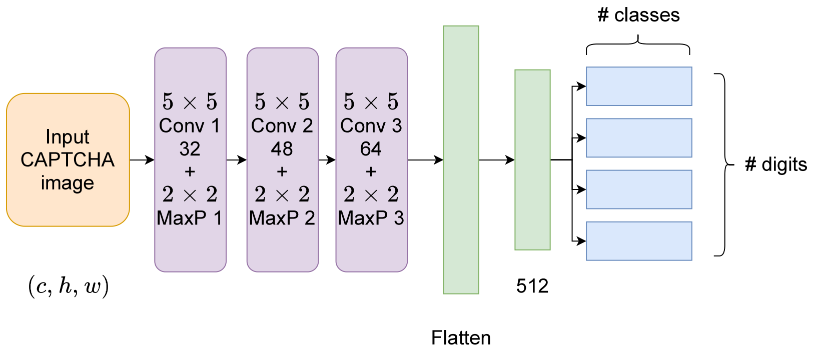


Fig. 1. The Dynamic Capsules architecture used on MNIST dataset digit classification (Sabour et al., 2017).



# classes
Input CAPTCHA image
5 × 5 Conv 1 32 + 2 × 2 MaxP 1
5 × 5 Conv 2 48 + 2 × 2 MaxP 2
5 × 5 Conv 3 64 + 2 × 2 MaxP 3
# digits
(c, h, w)
512
Flatten

Fig. 2. The Deep-CAPTCHA network architecture.

on the visual interference effects such as rotation, twisting, adhesion, and overlap. The numerical CAPTCHAs, for example, ask users to identify a random digit sequence distorted and drawn on a noisy image.

2.3. Deep-CAPTCHA

The Deep-CAPTCHA (Noury & Rezaei, 2020) is a deep CNN model applied to CAPTCHA image recognition (both numerical and alphanumerical) which then reveals the strengths and weaknesses of the existing CAPTCHA generators. We replicate the implementation of this model as our baseline (Fig. 2).

The input CAPTCHA image is passed through three consecutive Convolutional-Max-Pooling pairs (32, 48 and 64 kernels). Then the output feature maps of the third Max-Pooling layer are flattened and fed into a fully connected network with 512 neurons in the hidden layer and #digits × #classes neurons in the output layer. The output neurons are grouped into #digits vectors, each with #classes elements. The model applies the *Softmax* function (normalised exponential function used as activation) to each vector and turns the vector to the probability distribution among all classes. The class with the highest probability will be our prediction at each digit position.

3. Breaking CAPTCHA

In the current and the following sections, we focus on the empirical and competitive performances between CapsNet and CNN. Given the performance of deep-CAPTCHA, but at the same time given the capabilities of CapsNet, we therefore aim to provide the right set of distortions such that the spatial relationships can be exploited by the CapsNet and thereby obtain a better performance. We also look at identifying the right data manipulation for which CapsNet leverages its architecture advantages and outperforms the CNN baseline.

We proceed by introducing the process of adapting the CapsNet architecture for the CAPTCHA datasets. We use the `captcha` library[1] to customise numerical CAPTCHA datasets, which grant

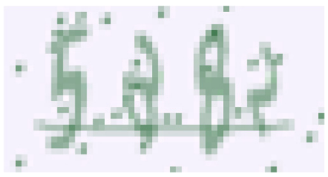


Fig. 3. An image sample from *Dataset 1* with a digit sequence "5382".

us more control over the properties of the CAPTCHA images, such as the size and the noise level of the CAPTCHA. Each digit is randomly generated and distorted with random distance between two adjacent digits. Dots and lines are randomly positioned in the image which constitutes the noise. We use these datasets to compare the CapsNet with the Deep-CAPTCHA and evaluate the improved CapsNet. For each dataset, there are 50K samples for training, 10K samples for validation and 10K samples for testing. We create a numerical CAPTCHA dataset (Dataset 1) to evaluate the performance of the CapsNet (Fig. 3).

Dataset 1. Each CAPTCHA image (28 × 54) has RGB channels and contains 4 digits. Each digit is rotated independently and randomly within a 90-degree range (clockwise or anticlockwise). We bound the rotation range to 90 degrees maximum in order to avoid confusion in classifying digits 6 and 9.

3.1. The CapsNet architecture

The architecture of the CapsNet (2.1) proposed by Sabour et al. (2017) is designed to classify 28 × 28 images with single digits from MNIST dataset (LeCun et al., 2010). The final layer (DigitCaps) contains 10 capsules representing the 10 digits. The model predicts the digit with the longest capsule length.

In the case of 4-digit CAPTCHA images, we have to predict not only the existence of four digits but also the order in which they appear in the image. Because each digit in the image is independently and randomly generated, there are 10^4 possible outcomes for digit sequences. If each outcome is represented by a capsule, the high computational power required to train such a

[1] The captcha library is available at https://pypi.org/project/captcha/.

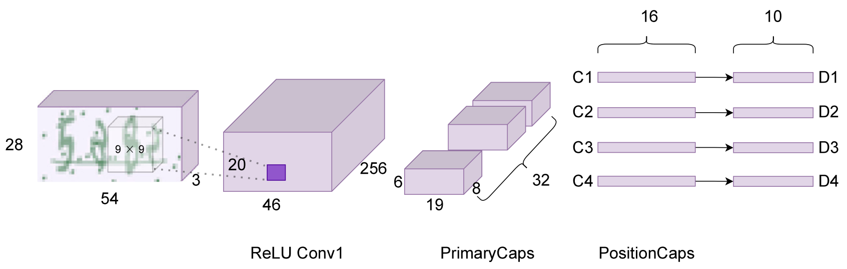


Fig. 4. The customised CapsNet architecture.

large model renders it impractical. Because the same digit may appear multiple times in an image and the order of the digits matters, it makes it difficult to only use 10 digit capsules at the final layer. Instead, we apply 4 capsules at the final layer as "position" capsules representing 4 positions instead of specific digits. The digit represented by a position capsule is not fixed but depends on the digit in that position, for example, the first position capsule represents the first digit in a CAPTCHA image.

Fig. 4 demonstrates the architecture of the adapted CapsNet.[2] Although input images are wider in size and have 3 channels, we still apply the convolutional layer and the PrimaryCaps layer as the original CapsNet. After the PrimaryCaps layer, the model produces 32 capsule blocks, each of which contains 6 × 19, 8D primary capsules. Then we serialise the primary capsules and apply dynamic routing to distribute them into 4, 16D position capsules (C1–C4). Assuming that the input CAPTCHA image contains a sequence *"5382"*, the position capsules C1–C4 will represent digits *"5"*, *"3"*, *"8"*, *"2"* respectively. Then we apply a fully connected neural network and the Softmax function to each position capsule, converting the 16D capsule vector to a 10D probability distribution vector (D1–D4). The index of the largest value in each probability distribution vector will be the predicted digit at the corresponding position. The loss of the model is the sum of the cross-entropy loss from each probability distribution vector.

In the original paper, the CapsNet reconstructs the MNIST digit from the digit capsule representing the correct digit. Here we remove the reconstruction network from our model because CAPTCHA images contain noises. In addition, using capsules to represent positions means that all capsules are always activated, so we remove the squashing function from the last iteration of the routing algorithm.

Fig. 4 depicts a vanilla version (regular feed-forward network with a single hidden layer) of the adapted CapsNet. In Section 4.1, we experiment with multiple updated versions of the adapted CapsNet, namely the CapsNet architecture with a dropout layer after the PrimaryCaps layer, and a version of CapsNet which enables certain parameter update during backpropagation.

3.2. *CAPTCHA puzzle task*

For this section, we create a new dataset and adjust the task to examine the characteristics of the CapsNet and the Deep-CAPTCHA.

Dataset 2. Each sample is a 3 × 28 × 56 noisy image, containing a 3-digit sequence with a 90-degree rotation range. Unlike the dataset in the previous section, the second digit in this dataset is always at the centre of the CAPTCHA image. Instead of using the whole CAPTCHA image as the input, we split the image from the middle into two 3 × 28 × 28 parts (Fig. 5), each of which has a portion of the second digit.

[2] The source code for experiments replication is available at: https://github.com/DeXterYa/break-captcha-with-capsule-networks.

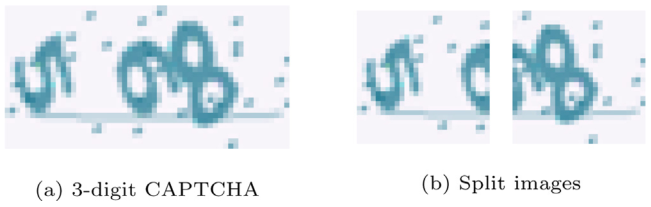


Fig. 5. A 3-digit CAPTCHA example split into two parts from the middle.

CAPTCHA puzzle task. The task is to independently extract features from the two split images and combine these features to predict the original three-digit sequence. We can think of it as completing a "puzzle" of two image pieces using the second digit as the clue. This requires the models to understand the spatial structure of the second digit instead of only seeing the second digit as a set of features.

There are three ways of feeding the image samples into the networks based on the order and direction of the two split images:

1. Two images are in the correct order and direction.
2. There is a 50% probability that two images are in reverse order. The direction of the two images remains the same as the original 3-digit CAPTCHA.
3. There is a 50% probability that two images are in reverse order. Two images are independently and randomly rotated by 0, 90, 180 or 270 degrees.

We adapt the CapsNet and the Deep-CAPTCHA models to this puzzle task as follows:

CapsNet (Fig. 6): we first feed two split images into two independent CapsNets to generate two groups of primary capsules as the feature representations for two split images. Two groups of primary capsules are concatenated to form one group as the representation of the original 3-digit CAPTCHA image. Then we apply the dynamic routing algorithm to distribute primary capsules into 3 position capsules (C1–C3). Next, we convert 16D capsule vectors to 10D probability distribution vectors (D1–D3) by applying three fully connected networks and Softmax functions. The index of the largest value in each probability distribution vector will be the predicted digit at that position. The loss of the model is computed as the sum of the cross-entropy loss from each probability distribution vector.

Deep-CAPTCHA (Fig. 7): we first feed two split images into two independent Deep-CAPTCHAs and get two vectors as feature representations. The two feature vectors are concatenated together as the overall representation of the 3-digit CAPTCHA image. Then the concatenated vector is fed into a fully connected neural network with 512 neurons in the hidden layer and 30 neurons in the output layer. The 30 output neurons are grouped into 3 vectors (D1–D3), each of which represents one digit at its corresponding position. The model applies the Softmax function

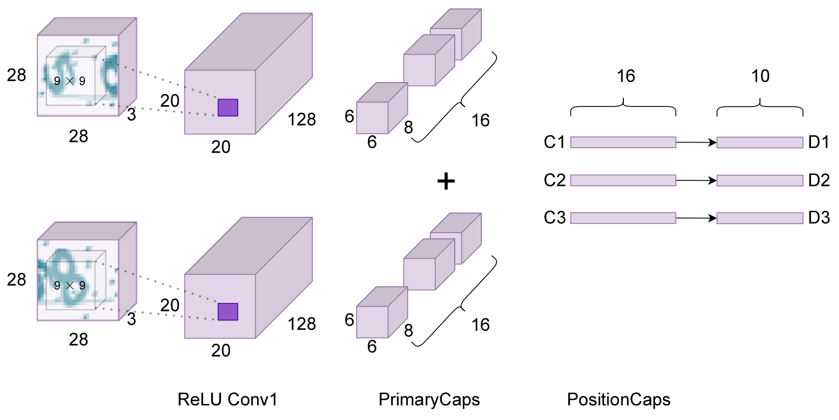


Fig. 6. The architecture of the adapted CapsNet.

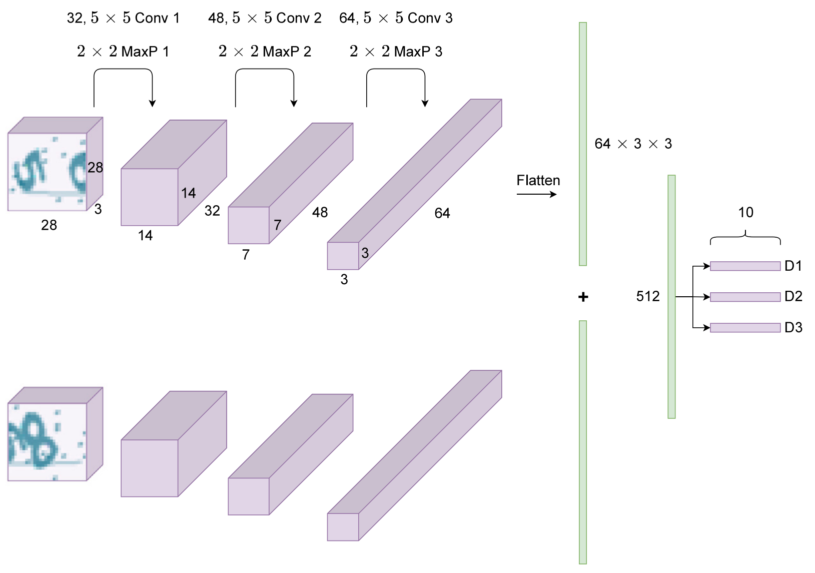


Fig. 7. The architecture of the adapted Deep-CAPTCHA.

to each vector and turns the vector into the probability distribution among 10 digits. The digit with the highest probability will be our prediction at each digit position. During the training, we minimise the sum of the cross-entropy loss at each probability distribution vector.

The purpose of the puzzle task is to essentially, compare the routing algorithm from the CapsNet and the fully connected network from the Deep-CAPTCHA. Both the CapsNet and the Deep-CAPTCHA models use convolutional layers to independently extract the features. The main difference is how the two independent sets of features are later on combined. CapsNet turns the features into capsules and combines them through the routing algorithm. The Deep-CAPTCHA uses features directly as plain vectors and combines them with a fully connected network.

4. Evaluation

We measure the model's accuracy based on digit and sequence accuracy. Digit accuracy represents the percentage of correctly classified individual digits in the whole dataset while sequence accuracy denotes the percentage of correctly recognised digit sequences in the whole dataset.

4.1. Adapt the CapsNet to the CAPTCHA recognition task

This section covers the experiment results and analysis of multiple versions of the adapted CapsNet mentioned in Section 3. In Table 1, Deep-CAPTCHA achieves high accuracy on two of three seeds but it is not trained on the third seed. CapsNet has lower accuracy but its performance is stable across the three seeds. Although the Deep-CAPTCHA leads the accuracy on the test set, the training accuracy of both the CapsNet and Deep-CAPTCHA increases to values close to 100% during the training. We interpret this as CapsNet suffering from significant overfitting, potentially due to the major differences in model complexity: CapsNet has 7.24M parameters, significantly higher than the Deep-CAPTCHA's 728K parameters. We also notice that the cube of the digit accuracy is approximately the sequence accuracy, which indicates, interestingly, that the prediction of each digit is close to independent prediction.

Table 1
Accuracy of the Deep-CAPTCHA and the CapsNets on Dataset 1.

Model	Digital accuracy	Sequence accuracy
Deep-CAPTCHA	0.9316, **0.9326**, 0.1014	0.7768, 0.7784, 0
CapsNet	0.8153	0.4882
CapsNet + dropout	0.8299	0.5211
CapsNet + gradient	**0.8706**	0.6194

Table 2
Digit accuracy of the CapsNet and the Deep-CAPTCHA with different numbers of training data.

#Training data	Deep-CAPTCHA	CapsNet
10K	**0.8587**, 0.1007, 0.1004	0.6892
20K	0.1022, **0.8893**, 0.0999	0.7494
30K	0.9149, **0.9172**, 0.1022	0.8051
40K	0.1022	**0.8371**
50K	0.9316, **0.9326**, 0.1014	0.8706

We added a dropout layer between the PrimaryCaps layer and the PositionCaps layer to reduce the impact of overfitting. The dropout layer randomly sets 25% of capsules to zeros during the training. This method effectively increases the model performance by around 2% (the third row of Table 1)[3] however it still falls significantly behind the performance of the Deep-CAPTCHA model.

Next, we investigated further improvements for the CapsNet accuracy and achieved this by allowing gradient descent on the parameters in the computational graph created by the dynamic routing process. The performance of the updated model (the last row of Table 1) is more accurate than the other versions of the CapsNet. The dynamic routing process shifts the coupling coefficient at each routing iteration. By allowing gradient descent at each step of the routing iterations, the parameters of the model could potentially receive smoother updates, due to the gradual shift of the coupling coefficient. From this point onward, we allow gradient descent inside the routing process for all the succeeding experiments in order to boost performance.

Next, we compare the *CapsNet + gradient* with the Deep-CAPTCHA on various numbers of training samples (10K–50K). We train the models for 100 epochs with three different seeds. As shown in Table 2, the fields in the table with one value contain the average accuracy from three seeds. The fields with three values list all three accuracy values from seeds 0, 1, 2. With limited data samples, the Deep-CAPTCHA is not trained on some seeds if not all, likely stuck in some local optimum during the training. In contrast, the CapsNet only witnesses a decrease in accuracy but with stable performance across all three seeds. From Tables 1 and 2, we conclude that the decrease in the training samples worsen the Deep-CAPTCHA's consistency across different parameter initialisation. By comparison, the CapsNet produces consistent accuracy across different parameter initialisation, though it suffers greater decrease in accuracy. This means that the CapsNet is potentially more suitable for computer vision tasks where the data is scarce.

4.2. CapsNet on the CAPTCHA puzzle task

This section covers the experiment results and analysis of the three scenarios of the CAPTCHA puzzle task mentioned in Section 3.2.

[3] The fields in the table with one value contain the average accuracy from three seeds, while the fields with three values list all three accuracy values from seeds 0, 1, 2. The following tables follow the same format.

Table 3
Performance of the CapsNet and the Deep-CAPTCHA with split images in the correct order and direction.

Model	Digit accuracy	Sequence accuracy
CapsNet	0.9219, 0.0986, 0.9049	0.8086, 0.0013, 0.7604
Deep-CAPTCHA	**0.9662**	**0.9084**

Table 4
Performance of CapsNet and Deep-CAPTCHA with split images in a random order but correct direction.

Model	Digit accuracy	Sequence accuracy
CapsNet	**0.8591**, 0.0986, 0.8474	**0.6612**, 0.0013, 0.6286
Deep-CAPTCHA	0.6577	0.0888

Scenario 1: Table 3 shows the experiment results when two input images are in the correct order and direction. The Deep-CAPTCHA outperforms the CapsNet by a significant margin, similar to the results shown in Table 1. Although the CapsNet achieves higher performance for some seeds, it is less stable. The parameter initialisation with seed 1 leads to the network not being trained, potentially stuck in a local optimum. The two split images in the correct order and direction mean that the relative positions of digits are unchanged after the split. The only challenge is that features extraction is done by two independent neural networks instead of one. The results show that both the fully connected layers and the routing algorithm is capable of predicting digit sequences based on features extracted independently.

Scenario 2: Table 4 shows the experiment results when two input images are in a random order but the correct direction. The CapsNet outperforms the Deep-CAPTCHA, due to its dynamic routing algorithm used in the CapsNet which can dynamically route capsules based on their spatial information. When two images are swapped, capsules will be directed by the routing algorithm to recover the correct order and "complete the puzzle" based on the second digit. The weighted sum in the Deep-CAPTCHA's fully connected network cannot adjust the weights dynamically based on the input feature vectors. This then justifies Deep-CAPTCHA's 30% decrease in accuracy.

Previously we assumed that the recognition of each digit is approximately independent and the experiments have proven that this assumption is correct. This means that the cube of the digit accuracy should be close to the corresponding sequence accuracy. However, the sequence accuracy of the Deep-CAPTCHA is significantly lower than the cube of the digit accuracy. After evaluating the digits at different positions of a sequence, we find out that the second digit has higher digit accuracy (90%) than the first and the third (40%–50%). This finding is not very intuitive at first glance since it should be more difficult to recognise the second digit which is split in half. However, a closer look at the weighted sum operation and the task itself will explain the results.

The second digit is always positioned alongside the edges of two input images after the split regardless of the order of the two images. After applying convolutional layers to an image, the features extracted from the edges of the original image remain in the edges of feature maps. When we flatten the feature maps to a vector, the features on the edge will have fixed positions in the vector. Thus, the weighted sum in the fully connected network only need to pay attention to features in those fixed positions to predict the second digit. Due to the lack of spatial information, however, the fully connected network struggles to identify the order of the other two digits based on the second digit. This results in higher accuracy for the second digit but poorer accuracy for the first and the last digit.

Table 5
Performance of the CapsNet and the Deep-CAPTCHA with split images in a random order and a random direction.

Model	Digit accuracy	Sequence accuracy
CapsNet	**0.5228**, 0.099, 0.4691	**0.1734**, 0.0013, 0.0931
Deep-CAPTCHA	0.0985	0.0016

Scenario 3: After adding random rotations to two split images, the performance gap between the CapsNet and the Deep-CAPTCHA widens. The Deep-CAPTCHA is not trained under this scenario while the CapsNet still manages to keep around 50% digit accuracy (see Table 5).

Given the experiment results under the 3 scenarios, what differentiates the Deep-CAPTCHA from the CapsNet lies in how extracted features are aggregated and passed to the next layer. The CapsNet's dynamic routing algorithm is able to utilise the spatial information stored in the capsules to "solve the CAPTCHA puzzle". More importantly, its advantage over traditional neural networks becomes more obvious when the CAPTCHA puzzle task becomes more challenging. However, we also acknowledge that the CapsNet (3.16M parameters) is more complex than the deep-CAPTCHA (841K parameters), which leads to longer training time for the CapsNet.

4.3. Further experiments

We have explored several techniques to improve the current CapsNet architecture. We compare the models on Dataset 1 but give the options to remove the noise from the background and/or reduce the number of data samples. First, we add coordinates to the feature maps before the PrimaryCaps layer or primary capsules after the PrimaryCaps layer. These two versions both outperform the vanilla version of the CapsNet in most cases. For the next improvement, we add an extra position capsule to store the noises and background information from the CAPTCHA images. The first version of this improvement simply leaves the extra capsule untrained while the second version adds an extra class to force the extra capsule to be classified as the noise or background. Similarly, both versions usually outperform the vanilla CapsNet, by a smaller margin compared to the previous improvement. It is worth noting that both improvements reach the peak performance increase when the most challenging dataset is applied (with noise and limited number of data samples).

In a further experiment, we limit the Dataset 1's digit rotation angles to one of the 7 options: [−75, −50, −25, 0, 25, 50, 75], whereas for the test set we allow the rotation angle to be a random angle of values between [−90, 90]. This task examines the model's ability to generalise from limited observation. Empirical results show a decrease in accuracy by 2%–3% on the test sets for both the CapsNet and Deep-CAPTCHA compared to the results on the normal Dataset 1. When limiting the digit rotation angles to [−80, −40, 0, 40, 80], the Deep-CAPTCHA architecture fails to train whereas CapsNet model still maintains high accuracy (>90%) (see Table 6).

5. Limitations

5.1. Cost effectiveness

One of the main limitations of CapsNet is the computational bottleneck as resulted from the iterations inside the routing algorithm. Additionally, in terms of parameter size, CapsNet uses 7.24M parameters, which is of significantly higher complexity compared to the Deep-CAPTCHA's 728K parameters. This will potentially cause more serious overfitting for CapsNet. Although it is more stable, CapsNet is less performant compared to deep-CAPTCHA with simple sequence classification. In Table 2, for example, the accuracy of Deep-CAPTCHA is above that of CapsNet.

Another limitation, of logical nature, is that the capsules under routing are "forced" to choose only one high-level capsule to vote while neglecting others, which leads severe sparsity in capsule layers. This has been mitigated by a fast version of the routing algorithm proposed by Zhang, Zhou, and Wu (2020) and also further improved by Zeng and Song (2022).

5.2. Datasets

While extensive experiments could be performed on various additional datasets, we would like to point out the main purpose of this manuscript, which is that of performing a deep comparison between the two architectures and providing the right parameters manipulation in which we compare the competitive performance of the two architectures.

Although we acknowledge that exploring different datasets would certainly bring more insight regarding the performance comparison, exploring the CAPTCHA dataset in particular for our experiments was motivated by the difficulties of the data and the significant impact of successfully "breaking" it. Thus, we have designed two CAPTCHA patterns as our datasets, and our focus has been restricted to CAPTCHA datasets.

5.3. Uncertainty quantification

CapsNets model results do not capture prediction uncertainty which comes from the uncertainty in the model weights and can therefore make the predictions unreliable. If the uncertainty in the data and the model parameters are not considered, such predictions can have a high risk of failure in actual world deployment. Uncertainty refers to situations involving unknown or imperfect knowledge and is inherent in stochastic and partially observable environments and it can be embedded in the entire analytic process (e.g., collecting, organising, and analysing big data).

In recent years, uncertainty quantification methods (Abdar et al., 2021; Begoli, Bhattacharya, & Kusnezov, 2019; Siddique, Mahmud, Keesee, Ngwira, & Connor, 2022) have become very important techniques in deep learning, where machine learning methods take one first step and provide an objective criterion that enables people to make assumptions about the prediction quality of the proposed models. Similar to other standard deep learning networks, however, CapsNets do not capture model uncertainty. Various studies have looked at uncertainty modelling for CNNs (Abdar et al., 2021; Abdar, Pourpanah et al., 2021; Abdar et al., 2021; Du, Barut, & Jin, 2021; Kontolati, Klein, Panda, & Oyen, 2021; Morrell, Hickmann, & Wilson, 2021), and recently researchers have also looked into uncertainty handling for the CapsNet architecture, such as Afshar, Mohammadi, and Plataniotis (2020), Ramírez, Cuesta-Infante, Schiavi, and Pantrigo (2020) and Ribeiro, Leontidis, and Kollias (2020). A recent study by Afshar et al. (2020), which introduced the *BayesCaps* framework, has tried to capture the epistemic uncertainty in the model. While we do acknowledge that our models do not handle uncertainty quantification at this current stage, which can prove to be insightful for model validation and reproducibility, we point out the underlying purpose of this current paper, which is of experimental analysis and the models' predictions are not used as decision support for expert knowledge, at this stage.

Table 6
Evaluation results of CapsNet and Deep-CAPTCHA models for the range of rotation angles. All Training, Validation and Test accuracy values are digit accuracies. Training and Validation runs have been performed on the indicated rotation angles in the table, whereas the testing has been done on the full range [−90, 90] degree angles.

Rotation angles	Model	Train	Val	Test
[−75, −50, −25, 0, 25, 50, 75]	CapsNet	1.0	0.9488	0.9254
[−75, −50, −25, 0, 25, 50, 75]	Deep-CAPTCHA	0.9943	0.9696	**0.9474**
[−80, −40, 0, 40, 80]	CapsNet	0.9991	0.9413	**0.9111**
[−80, −40, 0, 40, 80]	Deep-CAPTCHA	≈10%	≈10%	≈10%

6. Related work

6.1. Capsule networks

Several modifications to the original architectures have been proposed to improve performance and extend the learning tasks. *Spectral Capsule Networks* (Bahadori, 2018) provide faster convergence, compared to the EM routing method by adding spatial coincidence filters to detect entities. Similarly, another approach proposed by Li and Wang (2020), uses *cognitive consistency* to optimise the routing algorithm. Fully CapsNets (Li, Ren, & Yang, 2018) improve the equivariance of the neural networks in image segmentation, as well as *SegCaps*, proposed by LaLonde and Bagci (2018). Rawlinson, Ahmed, and Kowadlo (2018) proposed improvements for the generalisation process by unsupervised sparsening of latent capsule layers. *Siamese Capsule Networks* (Neill, 2018) are used for pairwise learning tasks and face verification on controlled and uncontrolled settings. *Path Capsule Networks* (Amer & Maul, 2019) use a deep parallel multi-path version of CapsNets, which show to reduce the number of parameters necessary for learning. *Multi-scale Capsule Network* (Xiang, Zhang, Tang, Zou, & Xu, 2018) shows better robustness by improving the dropout techniques.

6.2. Breaking CAPTCHAs

The security of CAPTCHAs is essential to defending the *Internet* against automated attacks, and in that sense, with the improvements of CAPTCHA designs, more complex and robust breaking techniques have emerged (segmentation based and non segmentation based). Researches including Garg and Pollett (2016) and Stark, Hazırbas, Triebel, and Cremers (2015) suggest using neural networks with convolutional layers to recognise the sequence. Wang and Lu (2018) use a more traditional method by segmenting the character sequence and recognising each individual character. Bursztein, Aigrain, Moscicki, and Mitchell (2014) propose to generate potential segments of CAPTCHAs and use a machine-learning algorithm to choose the best segmentation that separates each character.

Recent proposals include low-cost attacks on solving a new CAPTCHA system, called *hCaptcha* (Hossen & Hei, 2021). *hCaptcha* is a relatively new but emerging image CAPTCHA service developed by *Intuition Machines Inc*. It asks users to select images matching a category or label provided in the challenge instruction to verify that they are humans and not bots. For a more extensive review, see Chen et al. (2017), Kaur and Behal (2014), Sinha and Tarar (2016), Srivastava, Sakshi, Dutta, and Ningthoujam (2021) and Tiwari (2018).

7. Conclusion

This paper investigates the advantages and limitations of the CapsNet architecture by comparing its performance to that of the CNN-based Deep-CAPTCHA on different numerical CAPTCHA datasets and tasks. We have identified the set of data manipulation and image recognition tasks in which CapsNet outperforms its competitor classifier. For sufficiently large training samples, the Deep-CAPTCHA outperforms the vanilla CapsNet by a significant margin. However, after limiting the number of training samples, the CapsNet maintains stable accuracy while the Deep-CAPTCHA fails to complete training under certain parameter initialisation. We propose a novel CAPTCHA puzzle task to further compare the CapsNet and the Deep-CAPTCHA. When two split images are in the correct order and direction, the Deep-CAPTCHA completes the "puzzle" more efficiently. More advantage shifts towards the CapsNet as we increase the difficulty of the puzzle by rotating and changing the order of the puzzle pieces, due to the underlying dynamic routing algorithm used in CapsNet.

Furthermore, we have explored several techniques to improve the current CapsNet architecture. The first improvement proposed is the integration of the feature maps before the PrimaryCaps layer, and the second improvement is the introduction of an additional position capsule, responsible for learning the noise and the background information from the images. Under these conditions, CapsNet shows consistent performance across all parameter initialisations, whereas the Deep-CAPTCHA occasionally fails to train.

For future directions, we would like to explore other variants of the CapsNets architectures, such as Matrix Capsules with EM Routing (EM Capsules) and Stacked Capsule Autoencoders (SCAE) on the CAPTCHA tasks and also other CAPTCHA types such as alphanumerical CAPTCHAs. In particular, we are interested in evaluating the performance of a recent architecture, called Deep Capsule Networks (Rajasegaran, Jayasundara, Jayasekara, Jayasekara, Seneviratne, & Rodrigo, 2019) and comparing it with our current proposed architecture.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

We would like to thank our reviewers for the constructive feedback and suggestions which have led to a significantly improved version of this manuscript.

Zhenxu Yang conducted this work while at the University of Edinburgh. Funding: Ionela Georgiana Mocanu was supported by the Engineering and Physical Sciences Research Council (EPSRC), UK Centre for Doctoral Training in Pervasive Parallelism (grant EP/L01503X/1) at the University of Edinburgh, School of Informatics. Vaishak Belle was partly supported by a Royal Society University Research Fellowship, UK, and partly supported by a grant from the UKRI Strategic Priorities Fund, UK to the UKRI Research Node on Trustworthy Autonomous Systems Governance and Regulation (EP/V026607/1, 2020–2024).

References

Abdar, M., Fahami, M. A., Chakrabarti, S., Khosravi, A., Pławiak, P., Acharya, U. R., et al. (2021). BARF: A new direct and cross-based binary residual feature fusion with uncertainty-aware module for medical image classification. *Information Sciences*, *577*, 353–378. http://dx.doi.org/10.1016/j.ins.2021.07.024, URL https://www.sciencedirect.com/science/article/pii/S0020025521007143.

Abdar, M., Pourpanah, F., Hussain, S., Rezazadegan, D., Liu, L., Ghavamzadeh, M., et al. (2021). A review of uncertainty quantification in deep learning: Techniques, applications and challenges. *Information Fusion*, *76*, 243–297. http://dx.doi.org/10.1016/j.inffus.2021.05.008, URL https://www.sciencedirect.com/science/article/pii/S1566253521001081.

Abdar, M., Samami, M., Dehghani Mahmoodabad, S., Doan, T., Mazoure, B., Hashemifesharaki, R., et al. (2021). Uncertainty quantification in skin cancer classification using three-way decision-based Bayesian deep learning. *Computers in Biology and Medicine*, *135*, Article 104418. http://dx.doi.org/10.1016/j.compbiomed.2021.104418, URL https://www.sciencedirect.com/science/article/pii/S0010482521002122.

Afshar, P., Mohammadi, A., & Plataniotis, K. N. (2020). BayesCap: A Bayesian approach to brain tumor classification using capsule networks. *IEEE Signal Processing Letters*, *27*, 2024–2028. http://dx.doi.org/10.1109/LSP.2020.3034858.

von Ahn, L., Blum, M., Hopper, N. J., & Langford, J. (2003). CAPTCHA: Using hard AI problems for security. In E. Biham (Ed.), *Advances in cryptology — EUROCRYPT 2003* (pp. 294–311). Berlin, Heidelberg: Springer Berlin Heidelberg.

von Ahn, L., Blum, M., & Langford, J. (2004). Telling humans and computers apart automatically. *Communications of the ACM*, *47*(2), 56–60. http://dx.doi.org/10.1145/966389.966390.

Alcorn, M. A., Li, Q., Gong, Z., Wang, C., Mai, L., Ku, W., et al. (2018). Strike (with) a pose: Neural networks are easily fooled by strange poses of familiar objects. CoRR abs/1811.11553. arXiv:1811.11553. URL http://arxiv.org/abs/1811.11553.

Amer, M., & Maul, T. (2019). Path capsule networks. CoRR abs/1902.03760. arXiv:1902.03760. URL http://arxiv.org/abs/1902.03760.

Bahadori, M. T. (2018). Spectral capsule networks.

Begoli, E., Bhattacharya, T., & Kusnezov, D. F. (2019). The need for uncertainty quantification in machine-assisted medical decision making. *Nature Machine Intelligence (Online)*, *1*(1), http://dx.doi.org/10.1038/s42256-018-0004-1.

Bursztein, E., Aigrain, J., Moscicki, A., & Mitchell, J. C. (2014). The end is nigh: Generic solving of text-based CAPTCHAs. In *8th USENIX workshop on offensive technologies (WOOT 14)*. San Diego, CA: USENIX Association, URL https://www.usenix.org/conference/woot14/workshop-program/presentation/bursztein.

Chen, J., Luo, X., Guo, Y., Zhang, Y., & Gong, D. (2017). A survey on breaking technique of text-based CAPTCHA. *Security and Communication Networks*, *2017*, 1–15. http://dx.doi.org/10.1155/2017/6898617.

Du, H., Barut, E., & Jin, F. (2021). Uncertainty quantification in CNN through the bootstrap of convex neural networks. In *Proceedings of the AAAI conference on artificial intelligence, Vol. 35* (pp. 12078–12085). (13).

Garg, G., & Pollett, C. (2016). Neural network CAPTCHA crackers. In *2016 future technologies conference (FTC)* (pp. 853–861). http://dx.doi.org/10.1109/FTC.2016.7821703.

Hossen, M. I., & Hei, X. (2021). A low-cost attack against the hcaptcha system. arXiv:2104.04683.

Janai, J., Güney, F., Behl, A., & Geiger, A. (2017). Computer vision for autonomous vehicles: Problems, datasets and state-of-the-art. CoRR abs/1704.05519. arXiv:1704.05519. URL http://arxiv.org/abs/1704.05519.

Kaur, K., & Behal, S. (2014). Captcha and its techniques: A review. *International Journal of Computer Science and Information Technologies*, *5*.

Kaur, N., & Kumar, R. (2014). Survey of text CAPTCHA techniques and attacks. 5. www.ijcst.com.

Koga, K., & Takemoto, K. (2021). Simple black-box universal adversarial attacks on medical image classification based on deep neural networks. CoRR abs/2108.04979. arXiv:2108.04979. URL https://arxiv.org/abs/2108.04979.

Kontolati, K., Klein, N., Panda, N., & Oyen, D. (2021). Neural density estimation and uncertainty quantification for laser induced breakdown spectroscopy spectra. arXiv:2108.08709.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017). ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, *60*(6), 84–90. http://dx.doi.org/10.1145/3065386.

Kulluru, P. K., Shaikh, T., Bidwai, S., Mannarwar, A., & Shamlani, K. (2016). A survey on different types of CAPTCHA.

LaLonde, R., & Bagci, U. (2018). Capsules for object segmentation. arXiv:1804.04241.

LeCun, Y., Cortes, C., & Burges, C. (2010). *MNIST handwritten digit database, Vol. 2*. ATT Labs, [Online]. Available: http://yann.lecun.com/exdb/mnist.

LeCun, Y., Haffner, P., Bottou, L., & Bengio, Y. (1999). Object recognition with gradient-based learning. In *Shape, contour and grouping in computer vision* (p. 319). Berlin, Heidelberg: Springer-Verlag.

Li, S.-X., Ren, X., & Yang, L. (2018). Fully CapsNet for semantic segmentation. In *PRCV*.

Li, H., & Wang, Y. (2020). Cognitive consistency routing algorithm of capsule-network. (pp. 558–563). http://dx.doi.org/10.1007/978-3-030-17798-0_45.

Machado, G. R., Silva, E., & Goldschmidt, R. R. (2020). Adversarial machine learning in image classification: A survey towards the defender's perspective. CoRR abs/2009.03728. arXiv:2009.03728. URL https://arxiv.org/abs/2009.03728.

Minaee, S., Abdolrashidi, A., Su, H., Bennamoun, M., & Zhang, D. (2019). Biometric recognition using deep learning: A survey. CoRR abs/1912.00271. arXiv:1912.00271. URL http://arxiv.org/abs/1912.00271.

Morrell, M. C., Hickmann, K., & Wilson, B. M. (2021). Particle image velocimetry analysis with simultaneous uncertainty quantification using Bayesian neural networks. *Measurement Science & Technology*, *32*(10), Article 104003. http://dx.doi.org/10.1088/1361-6501/abf78f.

Neethu, N. J., & Kadan, A. (2015). Role of computer vision in automatic inspection systems. *International Journal of Computer Applications*, *123*, 28–31. http://dx.doi.org/10.5120/ijca2015905603.

Neill, J. O. (2018). Siamese capsule networks. arXiv:1805.07242.

Noury, Z., & Rezaei, M. (2020). Deep-CAPTCHA: a deep learning based CAPTCHA solver for vulnerability assessment. CoRR abs/2006.08296. arXiv:2006.08296. URL https://arxiv.org/abs/2006.08296.

Rajasegaran, J., Jayasundara, V., Jayasekara, S., Jayasekara, H., Seneviratne, S., & Rodrigo, R. (2019). DeepCaps: Going deeper with capsule networks. CoRR abs/1904.09546. arXiv:1904.09546. URL http://arxiv.org/abs/1904.09546.

Ramírez, I., Cuesta-Infante, A., Schiavi, E., & Pantrigo, J. J. (2020). Bayesian capsule networks for 3D human pose estimation from single 2D images. *Neurocomputing*, *379*, 64–73.

Rawlinson, D., Ahmed, A., & Kowadlo, G. (2018). Sparse unsupervised capsules generalize better. CoRR abs/1804.06094. arXiv:1804.06094. URL http://arxiv.org/abs/1804.06094.

Ribeiro, F., Leontidis, G., & Kollias, S. (2020). Capsule routing via variational Bayes. In *Proceedings of the AAAI conference on artificial intelligence, Vol. 34* (pp. 3749–3756). http://dx.doi.org/10.1609/aaai.v34i04.5785.

Sabour, S., Frosst, N., & Hinton, G. E. (2017). Dynamic routing between capsules. CoRR abs/1710.09829. arXiv:1710.09829. URL http://arxiv.org/abs/1710.09829.

Shekhar, H., Seal, S., Kedia, S., & Guha, A. (2020). *Survey on applications of machine learning in the field of computer vision* (pp. 667–678). http://dx.doi.org/10.1007/978-981-13-7403-6_58.

Siddique, T., Mahmud, M. S., Keesee, A. M., Ngwira, C. M., & Connor, H. (2022). A survey of uncertainty quantification in machine learning for space weather prediction. *Geosciences*, *12*(1), http://dx.doi.org/10.3390/geosciences12010027, URL https://www.mdpi.com/2076-3263/12/1/27.

Sinha, A., & Tarar, S. 2016. Review paper on different CAPTCHA techniques. 7, 174–176. www.ijcst.com.

Srivastava, M., Sakshi, S., Dutta, S., & Ningthoujam, C. (2021). Survey on captcha recognition using deep learning. (pp. 273–282). http://dx.doi.org/10.1007/978-981-15-7394-1_26.

Stark, F., Hazırbas, C., Triebel, R., & Cremers, D. (2015). Captcha recognition with active deep learning. In *Workshop new challenges in neural computation, Vol. 2015* (p. 94). Citeseer.

Thevenot, J., López, M. B., & Hadid, A. (2018). A survey on computer vision for assistive medical diagnosis from faces. *IEEE Journal of Biomedical and Health Informatics*, *22*(5), 1497–1511. http://dx.doi.org/10.1109/JBHI.2017.2754861.

Tiwari, A. (2018). A review on various interactive CAPTCHA techniques concerning web security.

Walch, M., Colley, M., & Weber, M. (2019). CooperationCaptcha: On-the-fly object labeling for highly automated vehicles. In *CHI EA '19*, *Extended abstracts of the 2019 CHI conference on human factors in computing systems* (pp. 1–6). New York, NY, USA: Association for Computing Machinery, http://dx.doi.org/10.1145/3290607.3313022.

Wang, Y., & Lu, M. (2018). An optimized system to solve text-based CAPTCHA. CoRR abs/1806.07202. arXiv:1806.07202. URL http://arxiv.org/abs/1806.07202.

Xiang, C., Zhang, L., Tang, Y., Zou, W., & Xu, C. (2018). MS-CapsNet: A novel multi-scale capsule network. *IEEE Signal Processing Letters*, *25*(12), 1850–1854. http://dx.doi.org/10.1109/LSP.2018.2873892.

Zeng, R., & Song, Y. (2022). A fast routing capsule network with improved dense blocks. *IEEE Transactions on Industrial Informatics*, *18*(7), 4383–4392. http://dx.doi.org/10.1109/TII.2021.3128412.

Zhang, S., Zhou, Q., & Wu, X. (2020). Fast dynamic routing based on weighted kernel density estimation. In H. Lu (Ed.), *Cognitive internet of things: Frameworks, tools and applications* (pp. 301–309). Cham: Springer International Publishing, http://dx.doi.org/10.1007/978-3-030-04946-1_30.