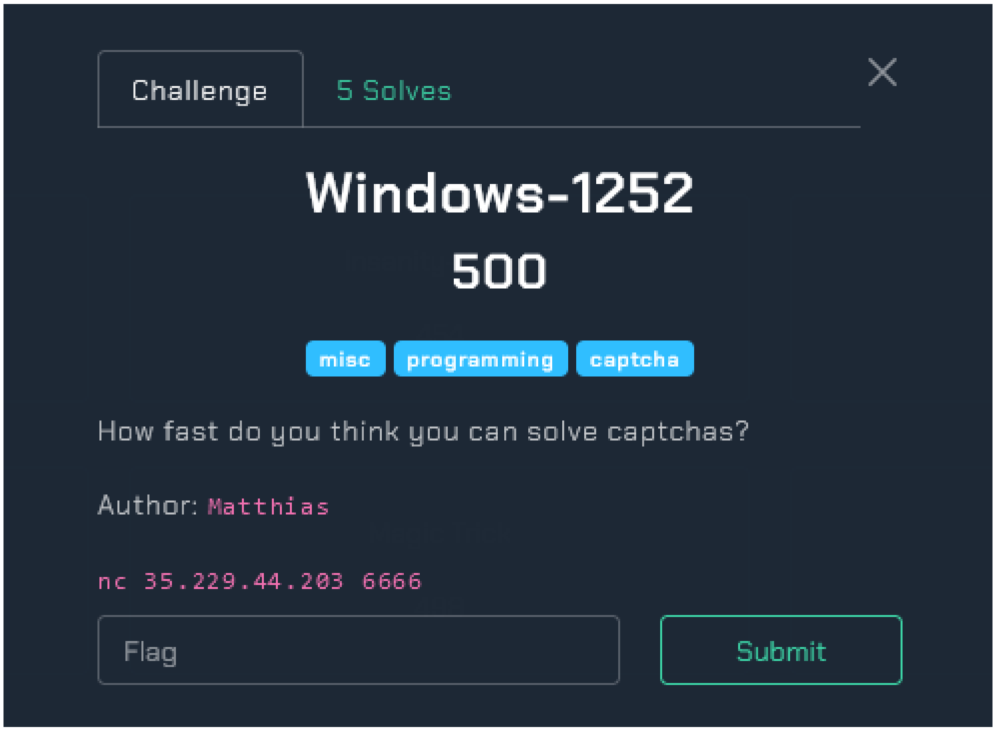
Windows 1252 Walkthrough



Challenge created by **Matthias**
Write-up by **VivisGhost**

Introduction

The Windows-1252 challenge involves solving CAPTCHAs, which are used to distinguish between human and automated access to websites. This particular challenge required us to decode CAPTCHAs , each representing a strings of characters from the Windows-1252 character set. Our goal was to build an automated system to decode these images accurately, solving 100 CAPTCHA with 100 seconds.

Given the nature of the challenge, a Convolutional Neural Network (CNN) was chosen due to its effectiveness in image recognition tasks. We generated a large dataset by adding random noise to clean character templates, resulting in a robust training set. This allowed us to train a CNN model capable of decoding the noisy CAPTCHA images efficiently.

In the following sections, we will look into the details of the CNN architecture, the data preprocessing steps, and the model training process that enabled us to tackle this CAPTCHA challenge successfully.

First Look

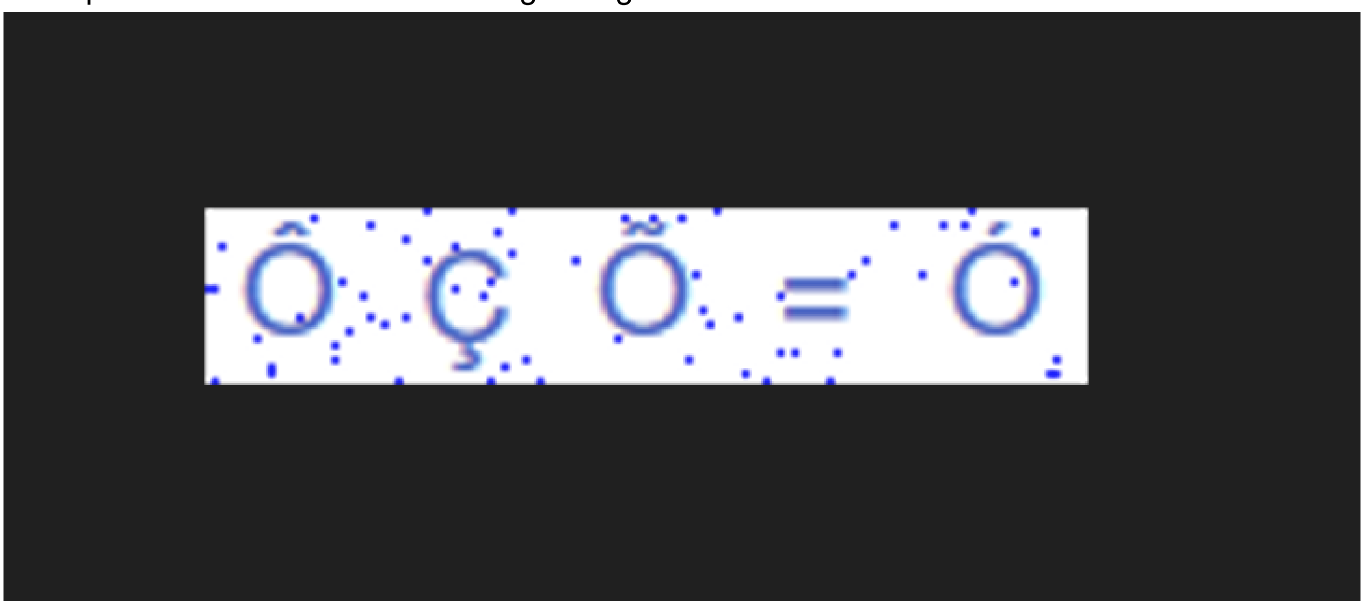
Gaining some more info by connecting with nc we can see 2 options. Request sample data or Start the challenge.

An example of the challenge being started.

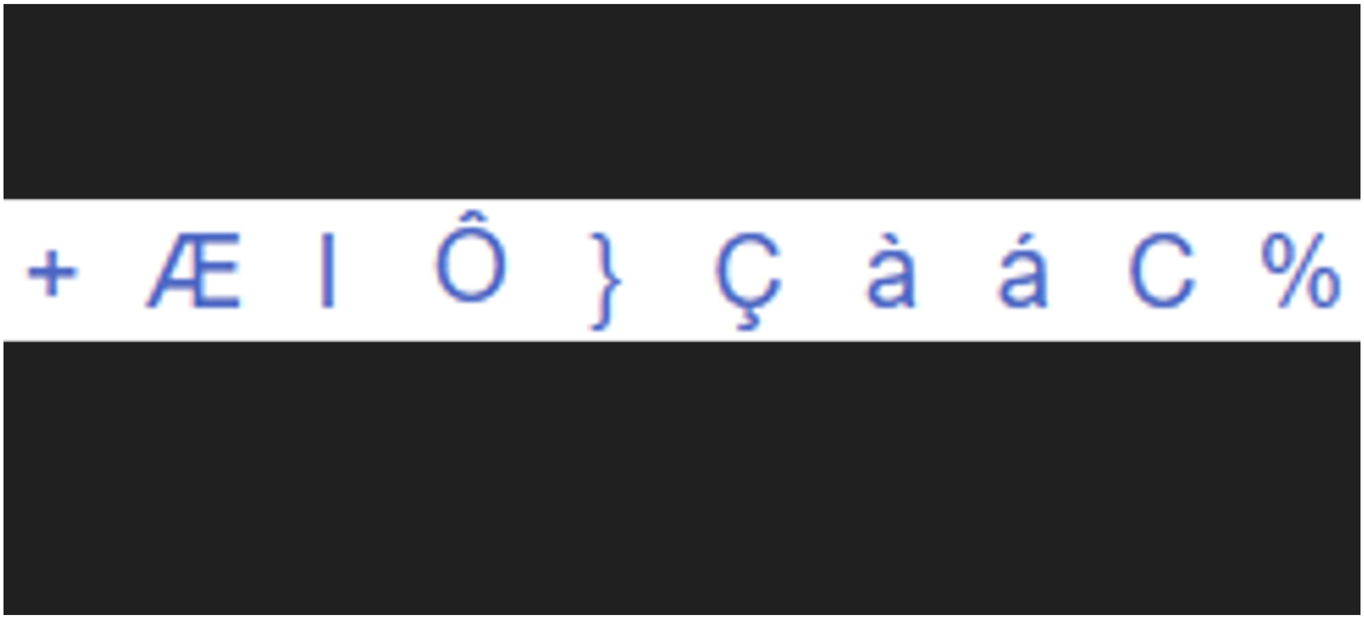
```
┌──(kali㉿kali)-[~/Desktop]
└─$ nc 35.229.44.203 6666
Can you solve 100 captchas in 100.0 seconds? You'll receive base64 encoded images and you'll have to reply with the hex
 string of the symbols shown! E.g. if the image shows "L3AK", you'll need to respond with "4c33414b".
Options:
[0] Request sample data.
[1] Start the challenge.
[*] Enter anything else to exit immediately.
> 1
Captcha #1: iVBORw0KGgoAAAANSUhEUgAAAH0AAAAZCAIAAACTh7Y+AAAHk0lEQVRoBe3BcUxUhx0H8O/vptIB1Sc16o1cTg26F3rWXG+SmcawNNcsRBY
T/nixCTsX4x2im4JbIKag6cPEQlZluo4am8uULjVnauJCwzaIqyENCfZ2Md4IE6x3eXmjGkTEQ3dz3ncgiFY4PRSs2fx8hCSeDyIg8a0TAYknIwISqRCSeO
EBIiDxxERA4rGEJJITAYnpJQISz4YISDw9EZCYRgKQxDARkHiWREDif4kISIiAxKMJSdwjAhIvPJoISDwlIYkZJgISyYiAROpEQOKZEQGJpycCEuOEJB4gA
hLjREAiFSIg8Qi9p0LaZ3FkLwzsWWbFkzF1n9HisLVtz8YTMHo21fTF1uYEfroA44yeTTV93fjOhrLV23IxLtx4trTN4q12eWxIRTQQLG6Ft9rlQc+mmj64
Vb+mIDkhiRSIgEQqREBighvH3/37OWuW2E3xVKz25uCJmLrPaHHY2rZn40n0fbCz53jmgo/1HDvG9DeF1jfFkcByt+rXFIwZOllz/sA1pWG/6sAIEZAYJwI
SD4oGgsWt8Fa7POjZVNMHt+rXFCQnJPEM9Fworuv/3rpFc5ovB5325hIrpkgEpKn7jBaHrW17Nh4gAhLjREBiUu0fdVR0pFXWrypMx13xprpQbTxrA/qPY9
Gp6qVZuCthVG0xg3lLmjcvxswQkpjcrb+duvjb1lh3HJnzM7xbs2NHLnz+murXFKRABCTGhRvPlrbNKt+XO+f3odrujMr9KwvTMU4EJFJg6j6jxWH9zeLB9
04P9SaQtXjuz0tWvJU9Cylq07+28VZRaV65EyMSpr7ViLyp7sHF4tZZlfWrCtMxItRZ0DC4ZmPe7jcsmEAEJJ6SkMQkEuE/hErP3LYuy9qwJi0WunKkC/b0
03N+qPo1BVOV6K3dEW1au0hU9dKsL8Jrj8Yc615tWP8ypszUfUaLBcDswoKFr2Po5F8Hwv+a7X3H5bEhJVcv7dh1+YZb9WsKhoU6CxqGCstWb0OXVj/g2Ji
3+w0LgGggWNw6q7J+VWE6ppcISAwTkpjo6qUduy5fyM0+UWbLxLBE9/HQptO3l7tVv6bgcURA4r4vwmuPxtZozjp3GhKmvtVosS4M7FlmxVSZus9osXy3sn
```

We can see we receive a base64 encoded .png image.

Example of base64 decoded challenge image.



Below is a snippet of the decoded request sample data.



Now that we have an idea of what we are working with lets gather some info and jump into python and create a script to solve it.

Windows -1252 character set. This will give us the possible characters we will need to decode.

https://en.wikipedia.org/wiki/Windows-1252

Windows-1252 (CP1252)[18][19][20][21][22]

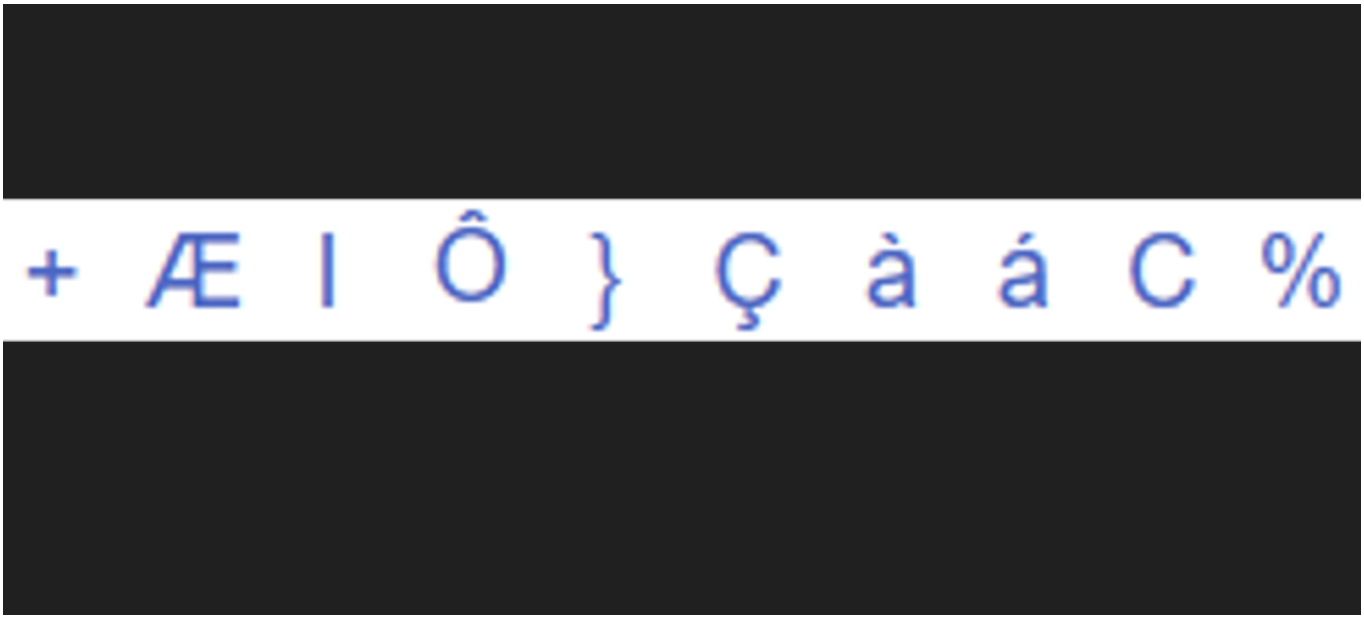
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0_	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1_	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2_	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3_	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4_	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5_	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6_	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7_	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL
8_	€ 20AC		‚ 201A	ƒ 0192	„ 201E	… 2026	† 2020	‡ 2021	ˆ 02C6	‰ 2030	Š 0160	‹ 2039	Œ 0152		Ž 017D	
9_		‘ 2018	’ 2019	“ 201C	” 201D	• 2022	– 2013	— 2014	˜ 02DC	™ 2122	š 0161	› 203A	œ 0153		ž 017E	Ÿ 0178
A_	NBSP	¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	SHY	®	¯
B_	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
C_	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
D_	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
E_	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
F_	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

To complete this challenge wee will need a few things.

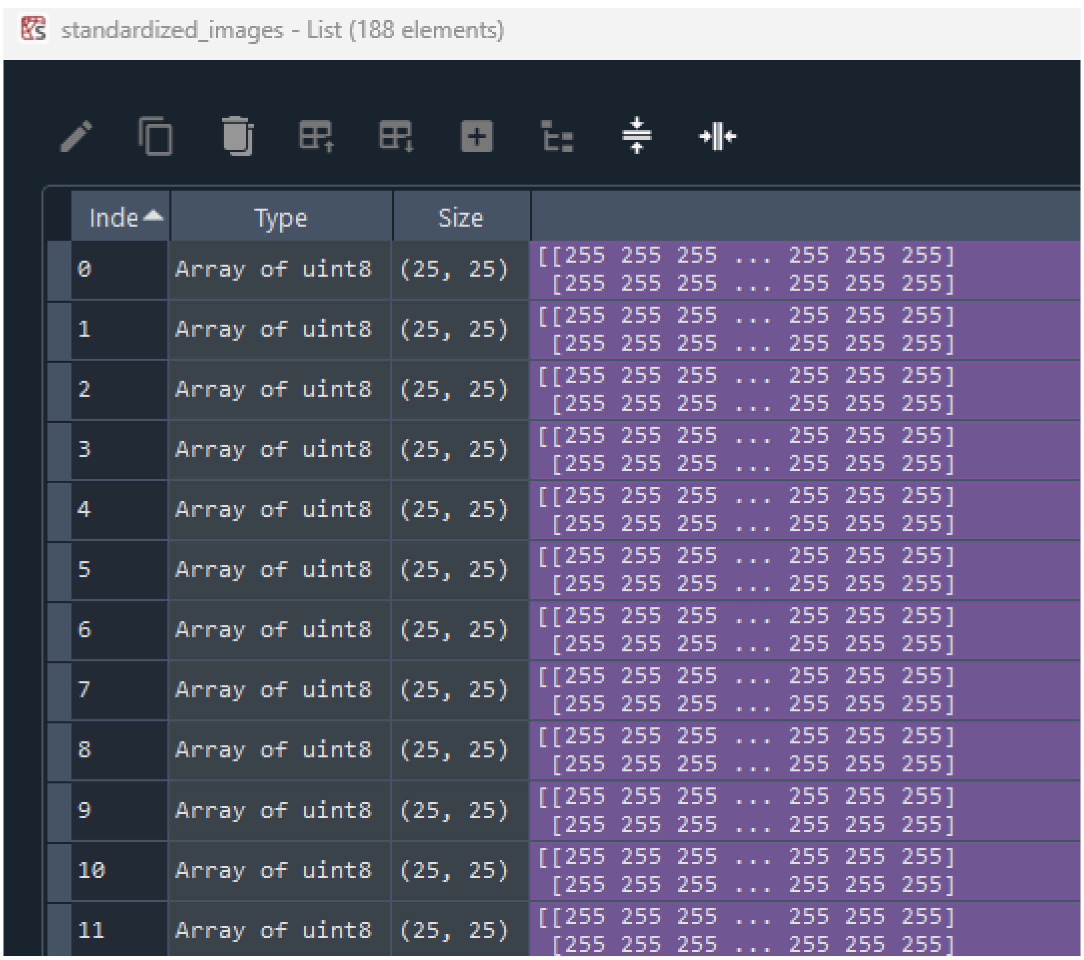
1. A CNN model
2. Training data
3. A script to have the model interact with the nc connection. Convert + preprocess incoming data(images in base64). Convert output to the expected hex format.

Converting the images to more useable data

Slicing up the sample data yields 188 characters, each 25x25.



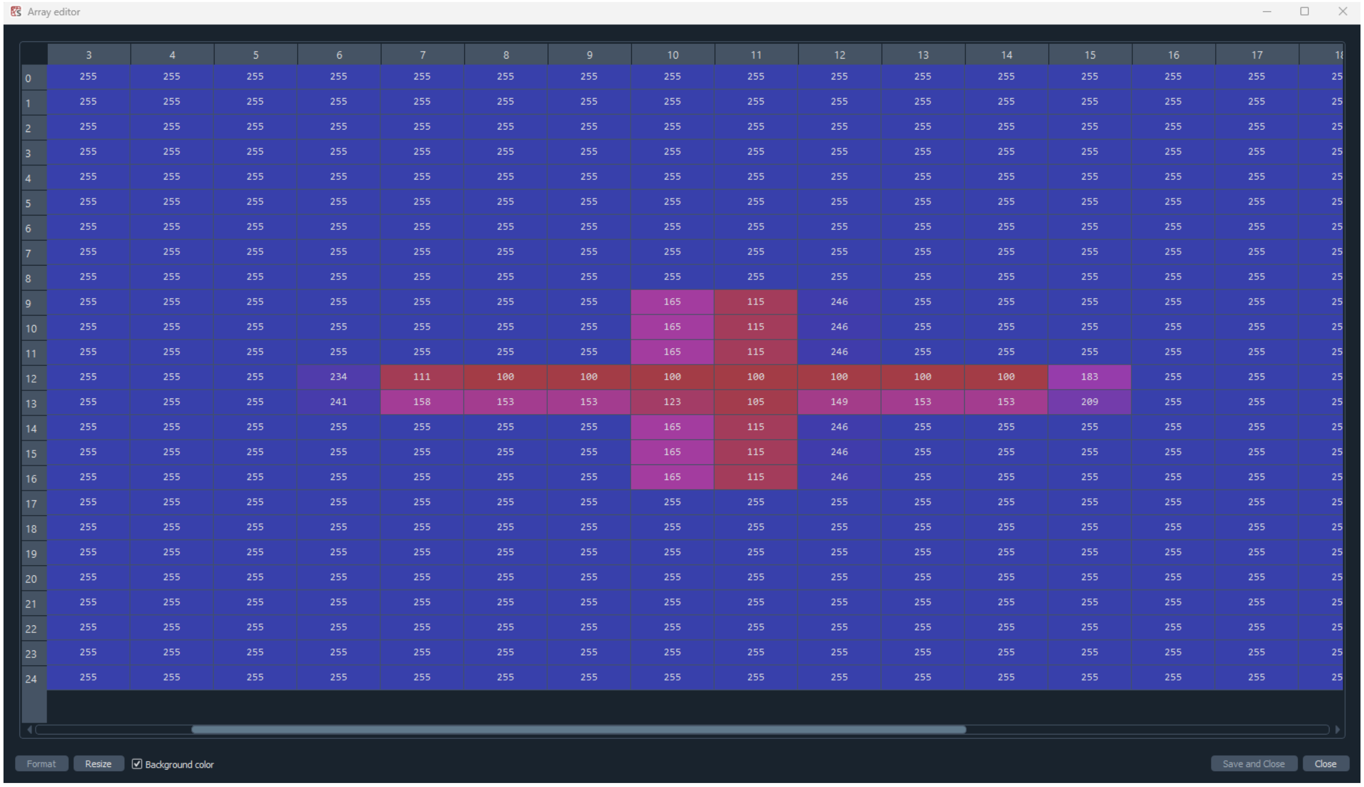
A list of all the characters from the sample image, split up into arrays.



standardized_images - List (188 elements)

Inde	Type	Size	
0	Array of uint8	(25, 25)	[[255 255 255 ... 255 255 255] [255 255 255 ... 255 255 255]
1	Array of uint8	(25, 25)	[[255 255 255 ... 255 255 255] [255 255 255 ... 255 255 255]
2	Array of uint8	(25, 25)	[[255 255 255 ... 255 255 255] [255 255 255 ... 255 255 255]
3	Array of uint8	(25, 25)	[[255 255 255 ... 255 255 255] [255 255 255 ... 255 255 255]
4	Array of uint8	(25, 25)	[[255 255 255 ... 255 255 255] [255 255 255 ... 255 255 255]
5	Array of uint8	(25, 25)	[[255 255 255 ... 255 255 255] [255 255 255 ... 255 255 255]
6	Array of uint8	(25, 25)	[[255 255 255 ... 255 255 255] [255 255 255 ... 255 255 255]
7	Array of uint8	(25, 25)	[[255 255 255 ... 255 255 255] [255 255 255 ... 255 255 255]
8	Array of uint8	(25, 25)	[[255 255 255 ... 255 255 255] [255 255 255 ... 255 255 255]
9	Array of uint8	(25, 25)	[[255 255 255 ... 255 255 255] [255 255 255 ... 255 255 255]
10	Array of uint8	(25, 25)	[[255 255 255 ... 255 255 255] [255 255 255 ... 255 255 255]
11	Array of uint8	(25, 25)	[[255 255 255 ... 255 255 255] [255 255 255 ... 255 255 255]

An example of the first array, the + sign.



	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	1
0	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	25
1	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	25
2	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	25
3	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	25
4	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	25
5	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	25
6	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	25
7	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	25
8	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	25
9	255	255	255	255	255	255	255	165	115	246	255	255	255	255	255	25
10	255	255	255	255	255	255	255	165	115	246	255	255	255	255	255	25
11	255	255	255	255	255	255	255	165	115	246	255	255	255	255	255	25
12	255	255	255	234	111	100	100	100	100	100	100	100	183	255	255	25
13	255	255	255	241	158	153	153	123	105	149	153	153	209	255	255	25
14	255	255	255	255	255	255	255	165	115	246	255	255	255	255	255	25
15	255	255	255	255	255	255	255	165	115	246	255	255	255	255	255	25
16	255	255	255	255	255	255	255	165	115	246	255	255	255	255	255	25
17	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	25
18	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	25
19	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	25
20	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	25
21	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	25
22	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	25
23	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	25
24	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	25

Now we have preprocessed the template data we can think about creating the training data for the model.

Convolutional Neural Networks (CNNs) for CAPTCHA Solving

A Convolutional Neural Network (CNN) is a type of deep learning model particularly well-suited for image processing tasks, making it an excellent choice for solving CAPTCHA challenges.

Data Preparation

To train a CNN, a substantial amount of labeled training data is required. In the case of the CAPTCHA challenge, gathering this data involved several steps:

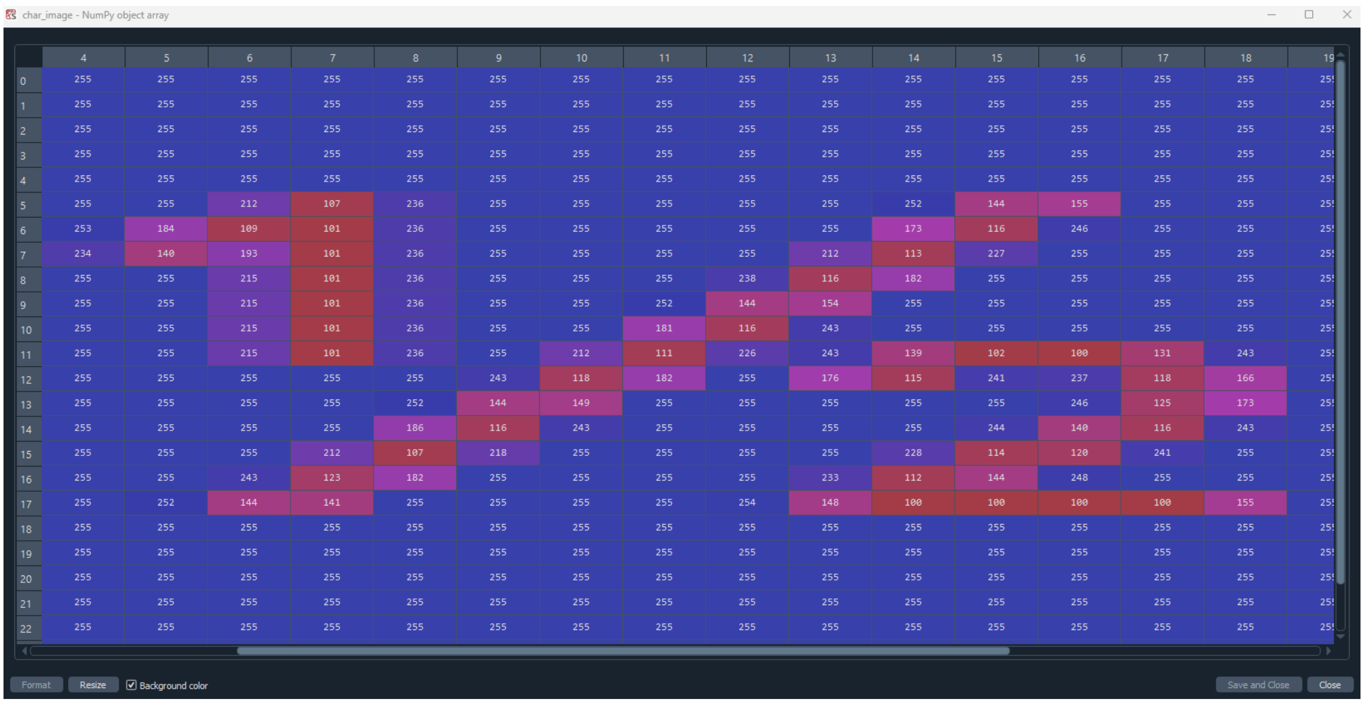
1. **Data Collection**:

- First option to collect data- We could get training data, by trying the challenge, failing and saving the images. We could do this hundreds of times, save them, parse them then label them. But... that sounds like too much work.
- Instead, a more efficient approach was chosen: generating synthetic training data. This involved taking a clean template of each character and adding random noise to create variations similar to those in the challenge. This method ensured a large dataset was created quickly.

Synthetic Data Set Creation

Take for instance this 1/2 character.

Clean template converted to array.



	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	25
1	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	25
2	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	25
3	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	25
4	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	25
5	255	255	212	107	236	255	255	255	255	255	252	144	155	255	255	25
6	253	184	109	101	236	255	255	255	255	255	173	116	246	255	255	25
7	234	140	193	101	236	255	255	255	255	212	113	227	255	255	255	25
8	255	255	215	101	236	255	255	255	238	116	182	255	255	255	255	25
9	255	255	215	101	236	255	255	252	144	154	255	255	255	255	255	25
10	255	255	215	101	236	255	255	181	116	243	255	255	255	255	255	25
11	255	255	215	101	236	255	212	111	226	243	139	102	100	131	243	25
12	255	255	255	255	255	243	118	182	255	176	115	241	237	118	166	25
13	255	255	255	255	252	144	149	255	255	255	255	255	246	125	173	25
14	255	255	255	255	186	116	243	255	255	255	255	244	140	116	243	25
15	255	255	255	212	107	218	255	255	255	255	228	114	120	241	255	25
16	255	255	243	123	182	255	255	255	255	233	112	144	248	255	255	25
17	255	252	144	141	255	255	255	255	254	148	100	100	100	100	155	25
18	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	25
19	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	25
20	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	25
21	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	25
22	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	25

Same template with random noise added.

noisy_img - NumPy object array

	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	255	255	255	255	255	255	255	255	33	255	255	255	255	255	255	255
1	255	255	26	255	255	255	255	255	255	255	255	255	255	255	255	255
2	255	255	255	30	255	29	255	255	255	255	255	255	255	255	255	255
3	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
4	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
5	255	255	212	107	236	255	255	255	37	36	252	144	155	255	255	255
6	253	184	109	101	236	255	255	255	255	255	173	116	246	255	255	255
7	20	140	193	101	236	255	255	255	255	212	113	227	255	255	255	255
8	255	255	215	101	236	255	255	255	238	116	25	255	27	255	255	255
9	255	255	215	40	236	255	255	252	144	154	255	255	255	255	255	255
10	255	255	215	101	236	255	255	181	116	243	35	255	255	255	255	255
11	255	255	39	101	236	255	212	111	226	243	139	102	100	131	243	255
12	255	255	255	255	255	40	118	23	255	176	115	241	237	118	166	255
13	255	255	255	255	252	144	149	255	255	36	255	255	246	125	173	255
14	255	255	255	255	186	116	243	255	255	255	255	244	140	116	243	255
15	255	255	255	212	107	218	255	255	255	255	228	114	120	241	255	255
16	255	255	243	123	182	255	255	255	255	22	112	144	248	255	255	255
17	255	252	144	141	255	255	255	255	254	148	100	100	100	100	155	255
18	255	255	255	255	255	255	255	255	255	255	255	255	28	255	255	28
19	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
20	255	255	255	255	255	255	255	255	255	255	255	26	255	255	255	255
21	255	255	255	255	30	255	255	255	255	255	255	255	255	255	255	255
22	255	21	255	255	255	255	255	255	255	255	255	255	255	255	255	255
23	255	255	255	255	24	255	255	255	255	255	255	255	255	255	255	255
24	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255

Format Resize ☑ Background color Save and Close Close

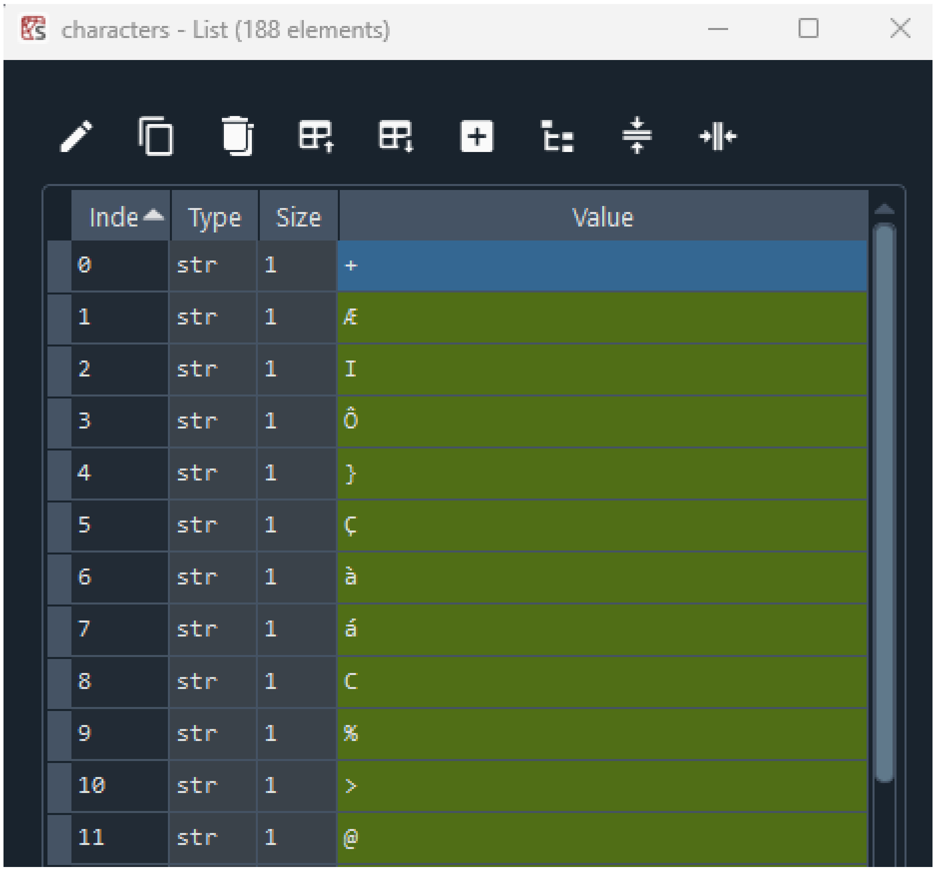
Using this process I created ~100,000 noisy images per character image, resulting in ~1.8 million total images for training. Shown below as noisy images.

noisy_images	list	1888752	[Numpy array, Numpy array, Numpy array, Numpy array, Numpy array, Nump ...

Convert noisy images to X and y(targets). Split X and y into training data and validation data.

X	Array of float32	(1888752, 25, 25)	[[[1. 1. 1. ... 1. 1. 1. ...
X_train	Array of float32	(1511001, 25, 25)	[[[1. 1. 1. ... 1. 1. 1. ...
X_val	Array of float32	(377751, 25, 25)	[[[1. 1. 1. ... 1. 1. 1. ...
y	Array of float64	(1888752, 188)	[[1. 0. 0. ... 0. 0. 0.] [1. 0. 0. ... 0. 0. 0.]
y_train	Array of float64	(1511001, 188)	[[0. 0. 0. ... 0. 0. 0.] [0. 0. 0. ... 0. 0. 0.]
y_val	Array of float64	(377751, 188)	[[0. 1. 0. ... 0. 0. 0.] [0. 0. 0. ... 0. 0. 0.]

The y/targets array way built during the noisy images creation. Using the wikipage I created a list which corresponds to the characters in the sample image in order.



characters - List (188 elements)

Index	Type	Size	Value
0	str	1	+
1	str	1	Æ
2	str	1	I
3	str	1	Ô
4	str	1	}
5	str	1	Ç
6	str	1	à
7	str	1	á
8	str	1	C
9	str	1	%
10	str	1	>
11	str	1	@

As the loop created a noisy image, it would also append the character to the targets list. Resulting in 2 lists of equal length of ~1.8 items. In list X every item is noisy image. Each item in y being the corresponding correct character.

Building the CNN Model

The CNN model was designed to classify the noisy character images into one of the 188 possible characters. Here's a high-level overview of the model architecture and training process:

1. **Model Architecture**:
    - **Convolutional Layers**: These layers apply convolutional filters to the input image, extracting key features such as edges, textures, and patterns.
    - **Activation Functions**: Non-linear activation functions (e.g., ReLU) are applied to introduce non-linearity into the model, enabling it to learn more complex patterns.
    - **Pooling Layers**: Max pooling layers reduce the spatial dimensions of the feature maps, retaining the most important features while reducing the computational load.
    - **Fully Connected Layers**: These layers take the flattened output from the convolutional layers and map it to the final output classes (the 188 characters).

```python
# Define the model
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(25, 25, 1)),
    BatchNormalization(), # Normalizes the inputs to a layer for each mini-batch, stabilizing and speeding up the training process.
    MaxPooling2D((2, 2)), # Reduces the spatial dimensions of the input, making the network more efficient and robust to translations.

    Conv2D(64, (3, 3), activation='relu'),
    BatchNormalization(),
    MaxPooling2D((2, 2)),

    Flatten(), # Flatten layer converts the 2D feature maps to a 1D feature vector.
    Dense(128, activation='relu'),
    Dropout(0.5), # dropout for regularization.
    Dense(num_classes, activation='softmax')
])
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
conv2d_2 (Conv2D)	(None, 23, 23, 32)	320
batch_normalization_2 (BatchNormalization)	(None, 23, 23, 32)	128
max_pooling2d_2 (MaxPooling2D)	(None, 11, 11, 32)	0
conv2d_3 (Conv2D)	(None, 9, 9, 64)	18,496
batch_normalization_3 (BatchNormalization)	(None, 9, 9, 64)	256
max_pooling2d_3 (MaxPooling2D)	(None, 4, 4, 64)	0
flatten_1 (Flatten)	(None, 1024)	0
dense_2 (Dense)	(None, 128)	131,200
dropout_1 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 188)	24,252

Total params: 174,652 (682.23 KB)
Trainable params: 174,460 (681.48 KB)
Non-trainable params: 192 (768.00 B)

Convolutional layer.

Filters (Kernels):

- A convolutional layer contains a set of learnable filters (also known as kernels). Each filter is a small matrix (e.g., 3x3 or 5x5) that slides over the input image and performs element-wise multiplication and summation. This operation is known as convolution.
- In our case we have 32 3x3 kernels in the first layer and 64 3x3 kernels in the second Conv2d layer.

Feature Maps:

- As the filter slides (or convolves) across the image, it produces a two-dimensional array called a feature map. Each value in the feature map corresponds to a specific region of the

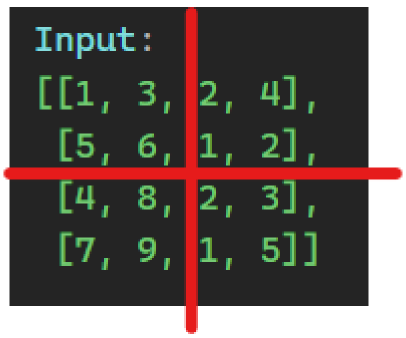
input image.
- Multiple filters are used in a single convolutional layer, each generating a different feature map. These feature maps capture various aspects of the input image, such as edges, textures, and patterns.

Max Pooling layer

Simple Max Pooling example.

```
Input:
[[1, 3, 2, 4],
 [5, 6, 1, 2],
 [4, 8, 2, 3],
 [7, 9, 1, 5]]
```

Break the array into quadrants. (In reality these 'splits' are done by the kernel size)



Output is the max from each quadrant.

```
Output after 2x2 Max Pooling:
[[6, 4],
 [9, 5]]
```

Training the Model

The generated noisy images were used as training data for the CNN. The training process involved:

1. **Training**:
   - The model was trained on the synthetic dataset, learning to classify each noisy image into the correct character class.
2. **Validation**:
   - The model's performance was validated using a separate set of images to ensure it generalized well to unseen data.

```
# Compile the model
model.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy', metrics=['accuracy'])


# Fit(train) the model
history = model.fit(X_train, y_train, batch_size=32, epochs=5, validation_data=(X_val, y_val))
```

Optimizer (ADAM):

- Adam (Adaptive Moment Estimation) is one of the most popular optimization algorithms used in training deep learning models. It combines the advantages of two other extensions of stochastic gradient descent. Specifically, it keeps an exponentially decaying average of past gradients (like momentum) and past squared gradients (like RMSProp).
- The learning rate is a hyperparameter that controls how much to change the model in response to the estimated error each time the model weights are updated. A lower learning rate means that the weights of the model are updated slowly, while a higher learning rate means that the model is updated more quickly.

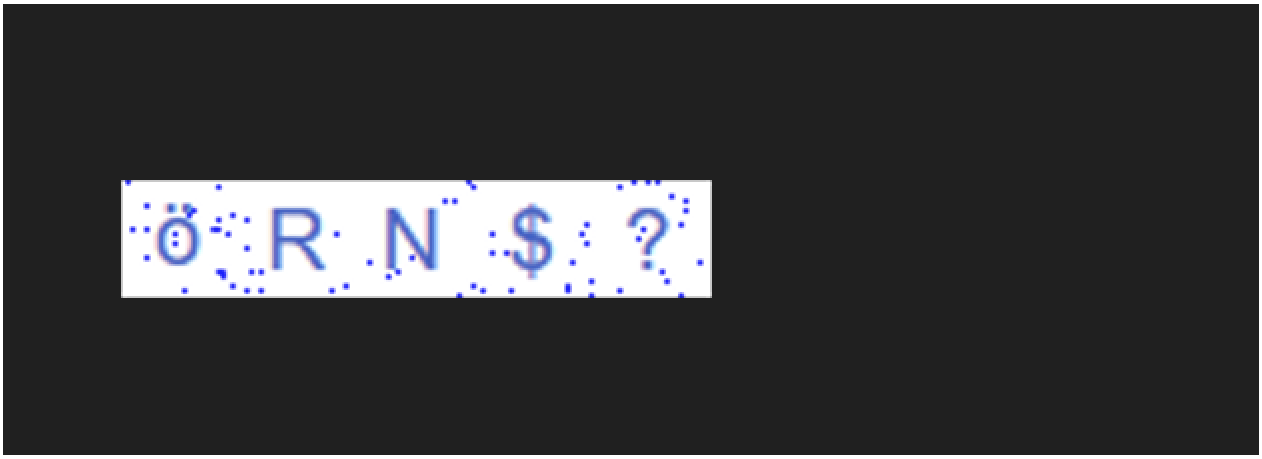
Loss(Categorical Crossentropy):

- The loss function (or cost function) is a method of evaluating how well the neural network models the training data. It takes the predicted output and the actual output and returns a score that indicates the performance of the model. The goal of training is to minimize this loss.
- Categorical crossentropy is used as a loss function for multi-class classification problems, where each sample belongs to one out of multiple classes. It compares the predicted probability distribution over classes with the true distribution (which is typically one-hot encoded).

```
Epoch 1/5
47219/47219 ━━━━━━━━━━━━━━━━━━━━ 346s 7ms/step - accuracy: 0.9373 - loss: 0.2535 - val_accuracy: 0.9990 - val_loss: 0.0086
Epoch 2/5
47219/47219 ━━━━━━━━━━━━━━━━━━━━ 334s 7ms/step - accuracy: 0.9967 - loss: 0.0155 - val_accuracy: 0.9992 - val_loss: 0.0075
Epoch 3/5
47219/47219 ━━━━━━━━━━━━━━━━━━━━ 336s 7ms/step - accuracy: 0.9981 - loss: 0.0095 - val_accuracy: 0.9995 - val_loss: 0.0039
Epoch 4/5
47219/47219 ━━━━━━━━━━━━━━━━━━━━ 342s 7ms/step - accuracy: 0.9983 - loss: 0.0084 - val_accuracy: 0.9994 - val_loss: 0.0043
Epoch 5/5
47219/47219 ━━━━━━━━━━━━━━━━━━━━ 342s 7ms/step - accuracy: 0.9983 - loss: 0.0087 - val_accuracy: 0.9995 - val_loss: 0.0050
11805/11805 ━━━━━━━━━━━━━━━━━━━━ 24s 2ms/step - accuracy: 0.9995 - loss: 0.0049
Validation Loss: 0.004968297667801380
Validation Accuracy: 0.9995287656784058
```

After 5 epochs of training the Validation Accuracy was 99.9%.

Example running the model on a challenge image locally.




```
In [42]: runcell(2, 'C:/Users/Daniel/.spyder-py3/windows-cap.py')
1/1 [==============================] - 0s 13ms/step
Recognized characters:  öRN$?
```

This looks like it is working as intended. Lets try it on the challenge server.

Running the Model

After training, the model was deployed to interact with the remote server, where it processed incoming CAPTCHA images, decoded them, and returned the results in the expected hex format.

```
Sent: 35d6664f77
5ÖfOw
Received: > Captcha #100: iVBORw0KGgoAAAANSUhEUgAAAH0AAAAZCAIAAACTh7Y+AAAGpElEQVRoBe3BDUyTiR0H4N//
Va8RUEnHENa8chJmDFdDep2XI67BEMlijszLLSEYvZp01xpiAowtGHKtzkrmZHconkaZyRL8SEwvW6LpnclKzAhhXKK1MVZ2kwbpuq4HqfhBj1uF9Tf1hoeA
YIt6eu55hCSSIQIS/
zdHQhJPhwhIfIeJgERqhCQeTQQkXkQiIPFEiIDEkyUk8RSIgMTDxnpPXbKcGy2uKGpatxAvOBGQmEoEJGYlJJEMEZBIVaLX5bP8VdP4G31JGl5mQhLfHhGQ+
C5IfOk5eMXpl8pa47ZCBeNEQGIqIYnZRNy+ijMJq8NoVjEn3p71rbdjmsVNLYXFCr4mAhLfuti5y5v6v3/
akoPUfdVx4LI9qj3hLMjDN0RAYhIhiZkFA5Y90d7EAqvDaFYxB4nu1gv1f4N2JLGq0tBYqhEBiedD3P2h7zODobFUgydHBCSmJSQxg8TQ0R1X3TFlaGSe1WE
0q5hKBCRmNxKy14Wvrlladmng2KJs1878XDw3RkL2uptvNq8qT0NKxoIdgeZPb1+8kYCi5KmLt/x8RVmOgkcTknikrzoOXLYPav/
wk39bjsetDqNZRcpi7ZfWu+LvVL1R1nO+qgPm+tXWAjwggrtIzEAEJICw0xbqL9Wt7Y0cDSUyvqf93Z4VejxEBCRmIAISd8Khfa2RP3+RuANAozG/
t8JalI6HiYDEjBK9Lp+lfVSry9y4Nv2VgVvuc7FeLLA2GMx5CsaJ4C4SXxOAJKYV6/JvOh7/WYPB/
M8eU1vc6jCaVSRLBCSA4VO7rhyKLm76qLA4FLDsjkaMeWe35iIVYact5FGgVbVbfjw/
MpyxrTwbKQgGLHui11Vt3UbdHfflg33K0AjeqfrRLwwKkjLYV2Uf7C/
UfVyrZuC+G8H6hkj3D3Vnf6lmYHpCEtMavFazYwDlr7WUL0KX39QWtzqMZhUpCgUsu6MxU4Hr3Szg5jH750dvLW5qKSxWkLyw0xbyZGS2fLDydQUTiYDE4xk
+tevKoeHMw00r9Ur0UF1wUc0PIvtCXsMy15YcACIg8TiG3L4NZ8Yqa1dvK8S4RHfrhXqvZvv+ovI0TEtIYqrE0LFfX/
1jlnqyWpcBoMtvaotbHUazitT4T56v6oC5frW1AHdFTvsqPomXVBoaSzV4DCIgMS7stIU8erWzWocJREBCBCRmIAISuH6tpmEgYipwvZuFUMCyb6yueaUe/
yMCEiIgMaugy7u5ff72/UXlaXgg6PJubofVYTSrmJaQxGQJ/3FvlXdh4y59yRLc0+U3tcWtDuOWZSCRtERkb03QrdEe/mCFHvcN9lXZB/26bNfO/
FwkK+y0hTx6tbNah5SFApbd0awKQ9M6zZDbt+mL3LPv5SAlQZd3c/v87fuLytPwQNDl3dwOq8NoVjEtIYnJwk5byINpZTT+Xl8CiIDE4/
L2rG+9HcNU88z1q60FSFLYaQt59GpntQ4pCwUsu6NZFYamdXA3+S6ueWPHGgUpGXL7NpwZq6xdva0Q4xLdrRfqvQt3HFlVpmBaQhKTDV/suHkdE/
RHnV1jZetzirUafUl2LpISd3/o2/v3eWVv5RRn4hv/
iDo74xnGvLNbc5GcsNMW8ujVzmodphABidldv1bTMHC9dMWJn35prxt6s7moPC3ubvIdGl165P3lebhHBCRmN9hXZR/
sL9R9XKtm4L4bwfqGSLe69PT7y7WYnpDEI4iAxD1dflNb3OowmlUkbSRYXxvp1mW7dubnYoJEZG9N0D2avr15VXkakhF22kIevdpZrUPqhk/
tunJoOLNlw1jNX9JPO5ZrbwXrt0f+ZSo4sSkLyUn0unyW9lGtLnPj2vRXBm65z8V6scDaYDDnKQBEQGISIYlZdflNbXGrw2hWMZEISMwsctpX8Um8pNLQWKr
Bw/wnz1d1/Ef/
1mtH3l5E4rGFnbaQR692VuswF76eDYdvDynIK371gGn0aGvYHc9o3KUvWYLkjQU7As2f3r54IwFFycvPtJoLSnIUjBMBiYmEJGbV5Te1xa0Oo1lFkqJHfxU4
Fl/c1FJYrGCywb4q+6Bfl+3amZ+Le0RAYjZhpy3k0aud1TrMTcz/
efWBm70KACUvX1u3teD1JXiqREDiLiGJORABiRdVOGD57R3rR4XFeNaEJF5WsfZL63uyOqt1eOaEJF5WHQc/+9Orq1rK0/
HMCUm8gERAYm6GjtmD2GrYsgwknjEhiUcQAYmpREDi2RABiSdCBCSeE0ISzzcRkHgiREDiefBf0DH87UNl07sAAAAASUVORK5CYII=
1/1 [==============================] - 0s 17ms/step
34c072a2f2
Sent: 34c072a2f2
4Àr¢ò
Received: > Congratulations, you've done it. Here is your flag: L3AK{0pT!cal_chaR4Ct3R_R3cOgN!tIon_i5_fUN}
```

The challenge gave us 100 seconds to finish, we were able to do it in 10 seconds.

```
[*] Closed connection to 35.229.44.203 port 6666
Connection closed
Elapsed time: 10.201952695846558 seconds
```

Complete python code @ windows-1252-CNN-solve.py