CAPTCHA kapituluje przed AI

Systemy CAPTCHA (*Completely Automated Public Turing test to tell Computers and Humans Apart*)[1], które mają na celu odróżnianie ludzi od botów, odgrywają istotną rolę w zapewnianiu bezpieczeństwa w Internecie. Jednak obecne sieci neuronowe osiągają coraz lepsze wyniki w dziedzinie Computer Vision, co stawia pod znakiem zapytania skuteczność tych systemów jako narzędzia ochrony przed botami. Wraz z rozwojem technik głębokiego uczenia, takich jak konwolucyjne sieci neuronowe (CNN), modele rekurencyjne (RNN) oraz ich hybrydy, rośnie również zdolność sztucznej inteligencji do przechodzenia przez coraz bardziej skomplikowane zabezpieczenia. Czy jest więc możliwe odpowiednie wytrenowanie sieci do obejścia tego typu zabezpieczeń?

Sebastian Tyralski

absolwent studiów magisterskich na Uniwersytecie Ekonomicznym w Krakowie oraz Associate Software Engineer w firmie Bayer. Pasjonat programowania i sztucznej inteligencji. Swoje doświadczenie w *deep learningu* poszerza poprzez autorskie projekty na platformie Kaggle, skupiając się na praktycznym zastosowaniu sieci konwolucyjnych. Prywatnie zajmuje się również projektowaniem i wdrażaniem nowoczesnych aplikacji oraz stron internetowych.

1 K. Kaur, S. Behal *Captcha and Its Techniques: A Review*, 2014, https://www.researchgate.net/profile/Sunny-Behal/publication/285110169_Captcha_and_Its_Techniques_A_Review/links/565bede108ae4988a7bb0d0c/Captcha-and-Its-Techniques-A-Review.pdf, s. 1 [dostęp: 10.06.2025].

Zrealizowany przeze mnie projekt „CaptchaPass" (w ramach mojej pracy magisterskiej) przyniósł odpowiedź na to pytanie.

Moja eksperymentalna aplikacja generuje dwa rodzaje CAPTCHY:

- klasyczną obrazkową, imitującą popularny mechanizm reCAPTCHA od firmy Google (developers.google.com/recaptcha?hl=pl);
- kody CAPTCHA, czyli zdeformowane ciągi znaków używane dalej na nielicznych stronach, które jeszcze nie przeszły na rozwiązanie Google'a.

CAPTCHA następnie jest wysyłana modelowi do oceny, a ocena analizowana przez aplikację, która decyduje, czy predykcja modelu była dobra czy zła.

Architektura

W projekcie „CaptchaPass" wykorzystałem trójwarstwowy podział architektury (*three-tier architecture*), co zapewnia separację odpowiedzialności, ułatwia rozwój, testowanie i skalowanie poszczególnych komponentów (www.ibm.com/think/topics/three-tier-architecture). Frontend to właściwie webowa aplikacja bazująca na React, wyświetlająca CAPTCHA i analizująca wyniki modelu (react.dev). Projektując backend, postawiłem na język Python oraz framework Flask, dzięki któremu mogłem w łatwy sposób zintegrować frontend aplikacji z pythonowym API, modelami sieci oraz bazą danych (flask.palletsprojects.com/en/stable; devstockacademy.pl/blog/narzedzia-i-automatyzacja/rest-api-co-to-jest-i-jak-dziala-przyklady-zastosowan). MongoDB w projekcie pełni funkcję przechowalni dla obrazów reCAPTCHA, obrazów ze zniekształconym kodem CAPTCHA oraz etykietami, które pozwalają Reactowi na weryfikację predykcji modelu.

W tego typu eksperymentach – z różnymi modelami sieci, zestawami metryk oraz dodatkowym *fine-tuningiem* modelu – przechowywane dane często ulegały nawet znacznym zmianom. Użycie MongoDB zamiast bazy relacyjnej pozwoliło w łatwy sposób dopasować bazę do tych zmian. Ale to nie jedyny argument na rzecz użycia Mongo. Ważniejsze było uniknięcie zbędnego skomplikowania kodu przez narzut mapowania obiektowo-relacyjnego (ORM), jakie występuje w bazach SQL-owych. Dzięki zastosowaniu w MongoDB dokumentów BSON, dane przychodzące jako JSON mogą w niemal niezmienionej formie trafić do bazy (www.json.org/json-en.html). I to tak naprawdę dużo ułatwia, bo w projekcie typu *Proof of Concept* ważniejszy jest efekt niż zbędne komplikacje architektury.

Modele sieci

Tworząc projekt „CaptchaPass", postanowiłem postawić na dwa modele sieci:

- model konwolucyjnej sieci neuronowej (CNN),
- model OCR, do optycznej detekcji znaków.

Model konwolucyjny wykorzystuje uczenie transferowe (*transfer learning*), którego bazą jest stworzony przez firmę Google model InceptionV3 (research.google/pubs/inception-v4-inception-resnet-and-the-impact-of-residual-connections-on-learning). Model ten należy do rodziny sieci GoogLeNet i wyróżnia się zastosowaniem tzw. *Inception Modules*, które umożliwiają równoległe przetwarzanie obrazu przez wiele filtrów o różnej wielkości (1x1, 3x3, 5x5), co pozwala sieci automatycznie dopasowywać zakres analizy przestrzennej (paperswithcode.com/method/inception-module). I co najciekawsze, nie wymagał on dużego dostrajania (*fine-tunning*), bo już w pierwszych 15 epokach osiągnął wynik bliski 70 proc.

Przy modelu OCR postanowiłem postawić na gotowe rozwiązanie, a mianowicie model z rodziny PaddleOCR (paddlepaddle.github.io/PaddleOCR/main/en/index.html). Architektura PaddleOCR to rozbudowany system przetwarzania obrazu, zaprojektowany w formie modularnego *pipeline'u*, który automatyzuje cały proces rozpoznawania tekstu od obrazu wejściowego po końcowy wynik w postaci ciągu znaków. Każdy z wielu komponentów realizujących kolejne etapy przetwarzania wykorzystuje różne, wyspecjalizowane modele głębokiego uczenia. Sama detekcja tekstu w PaddleOCR wykorzystuje modele DB (*Differentiable Binarization*), EAST (*Efficient and Accurate Scene Text Detector*) oraz SAST (*Segmentation-based Scene Text Detector*). Ich zadaniem jest utworzenie masek obszarów tekstowych na obrazie (*bounding box*). Następnie pora na zastosowanie klasyfikatora kierunku, którego zadaniem jest normalizacja orientacji tekstu przed rozpoznaniem. Pomaga to w sytuacjach, gdy tekst jest obrócony. Gdy mamy już zastosowany powyższy klasyfikator oraz wykryte *bounding boxy*, wchodzimy w etap faktycznego rozpoznawania tekstu. Tu również zastosowano kilka modeli uczenia głębokiego:

- CRNN (*Convolutional Recurrent Neural Network*),
- Rosetta,
- RARE (*Recurrent Attentive Reader*),
- SRN (*Sequence Recognition Network*),
- StarNet.

Na tym etapie wyodrębniamy strukturę *Backbone – Neck – Head*, czyli modularny schemat głębokiego uczenia:

- *Backbone* – to głęboka sieć konwolucyjna, której zadaniem jest ekstrakcja cech z obrazu;

- *Neck* – to warstwy transformujące cechy do postaci sekwencyjnej, przystosowanej do odczytu przez dekoder (najczęściej BiLSTM);
- *Head* – to dekoder odpowiedzialny za generowanie tekstu.

Dekoder jest tak naprawdę kluczowym elementem architektury rozpoznawania znaków, ponieważ to on odpowiada za ostateczną interpretację wysokopoziomowych cech wyodrębnionych przez sieć splotową i przekucie ich na zrozumiały dla człowieka tekst. W nowoczesnych systemach OCR, takich jak Paddle, precyzję zawdzięczamy mechanizmom atencji (*attention*) oraz algorytmowi CTC (*Connectionist Temporal Classification*) (https://docs.pytorch.org/docs/stable/generated/torch.nn.CTCLoss.html).

Podejście wykorzystujące mechanizm uwagi pozwala modelowi – podczas przewidywania kolejnych znaków – skupić się na konkretnych fragmentach obrazu, co świetnie sprawdza się przy nieregularnym tekście, takim jak kody CAPTCHA. Jeśli jednak kluczowa jest szybkość reakcji, to wygrywa CTC. Eliminuje on konieczność kosztownego wyrównywania sekwencji i pozwala na efektywne przetwarzanie potokowe. Co ciekawe, niektóre systemy idą o krok dalej, stosując strategię CML (*Collaborative Mutual Learning*). To zaawansowana technika, w której dwa niezależne modele uczą się od siebie nawzajem w trakcie procesu treningowego. Jeden z nich (zazwyczaj mniejszy i szybszy, bazujący na CTC) czerpie wiedzę od modelu bardziej złożonego, co pozwala na uzyskanie lekkości algorytmu przy zachowaniu precyzji typowej dla znacznie cięższych architektur. Dla projektu takiego jak mój oznaczałoby to możliwość uruchomienia skutecznego ataku na zabezpieczenia nawet przy ograniczonych zasobach sprzętowych.

Przebieg treningu

Odpowiednie wytrenowanie obu modeli wymagało nie tylko odpowiednio dobranego podejścia, lecz także dokładnych, złożonych zbiorów danych. Do trenowania modelu konwolucyjnego odpowiedzialnego za klasyfikację obrazów wykorzystałem zbiór „Google reCAPTCHA Image Dataset" zawierający obrazy przyporządkowane do jednej z 12 kategorii tematycznych, tak jak podczas weryfikacji reCAPTCHA (www.kaggle.com/datasets/mikhailma/test-dataset). Zbiór ten pierwotnie występował w formacie zgodnym z YOLO (*You Only Look Once*), czyli popularnym formatem używanym do detekcji obiektów, w którym każdemu obrazowi towarzyszy plik tekstowy zawierający współrzędne prostokąta otaczającego obiekt oraz jego klasę (docs.ultralytics.com/datasets/detect/#ultralytics-yolo-format).

Tu napotkałem pierwszy problem, bo pobrany zbiór danych był niepełny. Z 12 klas obrazów jedynie 3 były opisane poprawnie w formacie YOLO. Dla pozostałych 9 klas trzeba było opracować niestandardowy algorytm korzystający z modelu YOLOv8, aby uzupełnić brakujące etykiety (docs.ultralytics.com/models/yolov8/).

Na początku algorytm pobiera listę wszystkich klas obrazów i wykrywa brakujące etykiety, co pozwala na uniknięcie duplikowania plików dla etykiet, które już istnieją. Następnie ustala ścieżkę do pliku i etykiety (zamienia rozszerzenie pliku z .png na .txt), po czym uruchamia detekcje modelu YOLO na obrazie. Model zwraca wykryte obiekty w formacie: klasa, x_center, y_center, szerokość, wysokość, a algorytm zapisuje wyniki do pliku tekstowego. Wygenerowane w taki sposób brakujące etykiety pozwalają wykorzystać w pełni zbiór danych do uczenia modelu wykrywania obiektów na obrazach.

W przypadku modelu PaddleOCR dobór odpowiedniego zbioru danych był trudniejszym zadaniem. W początkowym etapie tworzenia sieci trening bazował na zbiorze Synth90K, zawierającym ok. 9 mln sztucznie wygenerowanych obrazów z tekstem zaprojektowanym i używanym do trenowania zaawansowanych modeli OCR, w tym wykorzystującym architektury takie jak CRNN oraz Rosetta (ta innowacyjna architektura opracowana przez korporację Meta charakteryzuje się wysoką dokładnością w rozpoznawaniu tekstu o złożonym układzie i różnorodnych stylach pisma). Jednak dla mojego zadania zbiór ten okazał się zbyt duży i skomplikowany. Ostatecznie w związku z rosnącym problemem przeuczenia (*overfittingu*) modelu oraz problemami w rozumieniu specyficznego zbioru danych przez Paddle, zdecydowałem się na mniejszy i prostszy w użyciu zbiór „CAPTCHA Dataset for Machine Learning", który zawiera ok. tysiąca próbek tekstowych zaprojektowanych pod kody CAPTCHA oraz łączy etykietę na obrazku z nazwą pliku, co pozwala na łatwe i kompatybilne z Paddle użycie zbioru danych w celu przetestowania efektywności modelu w rozpoznawaniu tego typu kodów (https://www.kaggle.com/datasets/mrigaankjaswal/capcha-images-to-training-data).

Trening obu modeli bazował na zasadzie *transfer learningu*, czyli wykorzystaniu wiedzy nabytej przez model bazowy w jednym zadaniu do przyspieszenia nauki i poprawy działania modelu w innym, pokrewnym zadaniu. Pierwszym etapem treningu była inicjalizacja modelu bazowego, a następnie zamrożenie jego warstw. Oznacza to, że wagi modelu podczas nauki nie są aktualizowane, co pozwala na zachowanie wiedzy zdobytej podczas wcześniejszego treningu na dużym zbiorze danych. Następnie na wyjściu zamrożonego modelu dodałem niestandardową architekturę klasyfikacyjną, zaprojektowaną już do konkretnego zadania. Model CNN został skompilowany z optymalizatorem Adam i *learning rate* ustawionym na 0.001. Adam w tym przypadku dobrze się sprawdził ze względu na swoją adaptacyjność i efektywność w zadaniach głębokiego uczenia. Jako funkcję straty wybrałem categorical_crossentropy, odpowiednią dla zadań klasyfikacji wieloklasowej.

W takiej konfiguracji trenowałem model przez 20 epok. Podczas tego etapu skupiłem się na nauce mapowania cech ekstrahowanych przez zamrożony model bazowy na konkretne kategorie CAPTCHA. Następnie przyszedł etap *fine-tunningu*, czyli odmrażania warstw modelu bazowego InceptionV3 w celu jego dostrojenia (https://matlab1.com/shop/python-code/the-google-inception-v3-model/). Aby uniknąć zniszczenia wcześniej nauczonych cech ogólnych odmroziłem jedynie 10 ostatnich warstw modelu. Podczas kolejnych 15 epok dostrajania utrzymałem optymalizator oraz *learning rate* na tym samym poziomie, co pozwoliło uzyskać stabilność treningu oraz płynność kroków w przestrzeni wag. Tak wykonany trening sprawił, że końcowy wynik dokładności modelu oscylował w granicach 80 proc. dokładności na zbiorze walidacyjnym i 95 proc. – na zbiorze treningowym.

Wykresy dokładności zarówno na zbiorze walidacyjnym, jak i treningowym były bardzo obiecujące mimo utrzymującej się niewielkiej straty, która mogła sugerować lekkie problemy z generalizacją i przeuczeniem. Jest to niestety bardzo częste w dzisiejszych systemach sztucznej inteligencji, które nie mogąc odróżnić istotnych cech obiektu od przypadkowego szumu informacyjnego, zaczynają chodzić na skróty i zamiast uczyć się semantycznego kształtu znaku, sieć może nadmiernie skupiać się na specyficznych układach pikseli, artefaktach kompresji czy charakterystycznym dla danego datasetu zniekształceniu tła. W kontekście reCAPTCHA zjawisko to jest szczególnie dotkliwe. Systemy zabezpieczeń celowo wprowadzają tzw. szum adwersarialny (*adversarial noise*), czyli subtelne modyfikacje, które są niemal niewidoczne dla ludzkiego oka, a dla modelu o słabej zdolności do generalizacji stanowią barierę nie do przejścia.

W przypadku PaddleOCR sytuacja była nieco bardziej skomplikowana. Ponieważ niestandardowe zbiory danych napotkały problemy związane z formatowaniem i strukturą danych (zbiory danych były zapisane w plikach .mat, a Paddle oczekiwał lmdb), zdecydowałem się na ocenę skuteczności PaddleOCR zamiast wymuszać dostrojenie modelu (www.ibm.com/docs/en/scdli/1.1.0?topic=dataset-lmdbs).

Testy aplikacji i ocena rozwiązania

Testy projektu „CaptchaPass" polegają na interaktywnej symulacji działania realnych mechanizmów, takich jak Google reCAPTCHA oraz klasyczne CAPTCHA tekstowe. Użytkownik, po załadowaniu aplikacji, może przetestować skuteczność modelu AI w warunkach przypominających codzienne interakcje z internetowymi formularzami. W przypadku symulacji mechanizmu reCAPTCHA *frontend* aplikacji losuje jedną z 12 kategorii obrazów, a następnie 3 obrazy odpowiadające wylosowanej kategorii oraz 6 innych losowych obrazów. Każdy obraz wyciągnięty z bazy danych zawiera również informacje o prawidłowej kategorii, stąd *frontend* wie, czy model wybrał dobrze czy źle. Tak jak w prawdziwym mechanizmie, wszystkie 3 obrazy muszą być wybrane prawidłowo, aby model poprawnie obszedł zabezpieczenie. Obrazy, które przepływają między *frontendem* a *backendem,* są przekazywane w formie tablicy, więc *frontend* dostaje konkretną odpowiedź, które indeksy obrazów odpowiadają predykcji modelu. I dokładnie tak, jak w realnym użyciu, obrazy są pobierane przez wtyczkę i w formie tablicy przesyłane przez API do modelu, następnie model wykonuje predykcję i zwraca indeksy, dzięki czemu wtyczka wie, które obrazy należy wybrać.

Aplikacja działa podobnie również w przypadku CAPTCHY tekstowej – *frontend* wyciąga z bazy danych obraz ze zniekształconym tekstem i prawdziwą etykietą, a następnie porównuje odpowiedź modelu z etykietą obrazu.

Model konwolucyjny wykorzystujący InceptionV3 radzi sobie wyraźnie lepiej – wytrenowany i dostrojony osiąga dokładność na poziomie ok. 80 proc., co jest już bardzo dobrym wynikiem. Dokładna analiza wykresów dokładności i strat tego modelu może sugerować lekkie przeuczenie, co wskazuje, że mimo dobrej trafności predykcji, model nie zawsze skutecznie generalizuje dane, których wcześniej nie widział. Jest to natomiast dobra podstawa do wdrożenia uczenia aktywnego z *feedbackiem*[2], gdzie model ma dokładnie zaznaczone błędy w danych wejściowych przez zwiększenie ich wag, co daje możliwość ich usunięcia. Takie podejście pozwoliłoby w przyszłości wyeliminować napotkane przeuczenie modelu oraz podnieść jego dokładność przy jednoczesnym zachowaniu dobrego poziomu generalizacji nowych danych.

Drugi model wdrażałem bez dodatkowego *fine-tuningu* ze względu na specyfikację modeli PaddleOCR, stworzonych do rozpoznawania trudnego tekstu. Niestety, dokładność tego modelu była zdecydowanie gorsza. Na ok. 1070 obrazów w zbiorze, jedynie 525 model przewidział poprawnie, co daje dość słaby wynik ok. 50 proc. dokładności. Różnice w statystykach ze względu na styl i klasy znaków kodów CAPTCHA pozwalają przypuszczać, że odpowiednio przygotowane dane pod Paddle

2 P. Hu, Z. C. Lipton, A. Anandkumar, D. Ramanan *Active Learning with Partial Feedback*, 2019, https://arxiv.org/pdf/1802.07427, s. 3, [dostęp: 8.06.2025).

lub inne dostępne modele oraz efektywne przeprowadzenie dostrajania mogłyby dać wyniki porównywalne z modelem konwolucyjnym.

> *Obydwa modele wykazują znaczące ograniczenia, które mogą wpływać na ich skuteczność w środowisku produkcyjnym. Największy problem jest z generalizacją nowych danych.*

Pierwszy model wykazuje tendencję do bardzo dobrego rozpoznawania klas obecnych w zbiorze treningowym, jednak jego skuteczność zauważalnie spada w przypadku prób klasyfikacji nieco zmodyfikowanych lub nowych wzorców, które nie pojawiły się w procesie treningu.

W drugim modelu PaddleOCR został użyty w wersji surowej, skonfigurowanej do rozpoznawania tekstu w bardziej uporządkowanych warunkach, takich jak dokumenty, zdjęcia szyldów czy oznaczeń w przestrzeni publicznej. Obrazy z kodami CAPTCHA zawierają natomiast zniekształcenia, nieregularne odstępy między znakami, losowe zakłócenia tła oraz celowe deformacje liter. Zastosowanie modelu w jego domyślnej konfiguracji, bez wcześniejszej adaptacji do tego typu danych, siłą rzeczy ograniczyło jego możliwości rozpoznawcze. Chociaż wynik na poziomie 49,07 proc. może wydawać się słaby w kontekście klasycznych zadań OCR, to w obliczu trudności charakterystycznych dla CAPTCHA oraz braku jakiegokolwiek strojenia, należy go postrzegać raczej jako punkt wyjścia niż rezultat końcowy. Wynik ten pokazuje, że model już na starcie wykazuje pewien stopień generalizacji, co czyni go solidną bazą do dalszego rozwoju.

Jest też i druga strona medalu. Jeśli jest możliwe wytrenowanie AI do automatycznego obchodzenia zabezpieczeń typu CAPTCHA, to dlaczego nie wykorzystać takiego potencjału szerzej? Dzisiaj model konwolucyjny przechodzi CAPTCHA automatycznie, jutro ktoś wykorzysta lokalnie postawionego LLM-a do łamania haseł i uzyskiwania nieautoryzowanego dostępu, o ile już tego nie robi. Pomimo implementacji zabezpieczeń w samych modelach furtka wciąż istnieje. Wystarczy postawić model lokalnie na maszynie atakującego, następnie odpiąć mu hamulce i model zmienia się z pomocnego asystenta w bezlitosną maszynę do automatyzacji ataków, działającą w całkowitym odizolowaniu od jakichkolwiek etycznych granic. To prowadzi do niebezpiecznej demokratyzacji zaawansowanej cyberprzestępczości: narzędzia, które kiedyś wymagały wsparcia całych grup hakerskich, dziś stają się dostępne dla każdego, kto dysponuje odpowiednią mocą obliczeniową i odrobiną determinacji. Gdzie więc leży granica? Czy to naprawdę realne zagrożenie i jutro zostaniemy zalani atakami prowadzonymi przez modele AI? Zanim zaczniemy budować bunkry, należy spojrzeć na sprawę nieco bardziej realistycznie.

Dzisiejsze modele mimo swojej mocy mają wciąż dużo ograniczeń i problemów. Jednym z przykładów jest halucynacja dużych modeli językowych, która sprawia, że jeśli dany model nie widział w zbiorze danych danego problemu i jego rozwiązania, zaczyna sam wymyślać nieistniejące rzeczy, działając na zasadzie predykcji, czyli przewidywania prawdopodobieństwa wystąpienia kolejnego słowa. W skrócie znaczy to, że taki model prędzej usunie nam system Windows, niż przeprowadzi zaawansowany atak na świeżo zaktualizowane firewalle. Chyba, że ktoś dysponuje eksperymentalną wersją LLM, uczącą się na błędach na podstawie sprzężeń zwrotnych z odpiętymi hamulcami.

Projekt „CaptchaPass" w obecnej wersji, mimo że spełnia założenia jako *Proof-of-Concept* oraz środowisko eksperymentalne, oferuje wiele możliwości dalszego rozwoju, zarówno w zakresie poprawy skuteczności działania modeli, jak i rozszerzenia funkcjonalności całego systemu. Jednym z bardziej ambitnych, ale bardzo realnych kierunków rozwoju jest przeniesienie aplikacji z obecnej postaci lokalnej aplikacji testowej do formy wtyczki przeglądarkowej, która mogłaby działać automatycznie. W najbardziej optymistycznym scenariuszu takie eksperymenty mogłyby rzucić nowe światło na zabezpieczenia przed atakami oraz botami i wpłynąć na rozwój nowych form zabezpieczeń.