EE 467 Lab 2: Breaking CAPTCHAs with Tensorflow / Keras

In lab 2, we are going to work on automatically breaking CAPTCHAs with deep learning! Originally, CAPTCHA stands for "Completely Automated Public Turing test to tell Computers and Humans Apart", and traditional CAPCTHAs serve as a great tool to stop spam bots and malicious crawlers. Today, as we have made huge progress in computer vision and deep learning, these CAPCTHAs are no longer unbreakable by computers. Now let's prove the correctness of the claim by ourselves!

As usual, please check if the helper library, `lab_2_helpers.py` and the extracted dataset directory, `captcha-images` exist under the same directory. Then, please install all libraries used for this lab:

```
%pip install matplotlib scikit-learn "opencv-python>4" imutils

# Install Tensorflow with CPU support only, or ...
%pip install "tensorflow>2"
# Install Tensorflow with GPU support:
#%pip install tensorflow-gpu>2
```

```
Requirement already satisfied: matplotlib in /usr/local/lib/python3.12/dist-packages (3.10.0)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.12/dist-packages (1.6.1)
Requirement already satisfied: opencv-python>4 in /usr/local/lib/python3.12/dist-packages (4.12.0.88)
Requirement already satisfied: imutils in /usr/local/lib/python3.12/dist-packages (0.5.4)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (1.3.3)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (4.61.1
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (1.4.9)
Requirement already satisfied: numpy>=1.23 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (2.0.2)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (25.0)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (11.3.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (3.3.1)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (2.9
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn) (1.16.3)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn) (1.5.3)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn) (3
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from python-dateutil>=2.7->matplo
Requirement already satisfied: tensorflow>2 in /usr/local/lib/python3.12/dist-packages (2.19.0)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow>2) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow>2) (1.6.
Requirement already satisfied: flatbuffers>=24.3.25 in /usr/local/lib/python3.12/dist-packages (from tensorflow>2) (2
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/local/lib/python3.12/dist-packages (from t
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.12/dist-packages (from tensorflow>2) (0.
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow>2) (18.1.
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.12/dist-packages (from tensorflow>2) (3.4.
Requirement already satisfied: packaging in /usr/local/lib/python3.12/dist-packages (from tensorflow>2) (25.0)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<6.0.0dev,>=3.20.3 in /u
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow>2) (2.
Requirement already satisfied: setuptools in /usr/local/lib/python3.12/dist-packages (from tensorflow>2) (75.2.0)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow>2) (1.17.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow>2) (3.3.0
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.12/dist-packages (from tensorflow>2
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow>2) (2.0.1)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.12/dist-packages (from tensorflow>2) (1.
Requirement already satisfied: tensorboard~=2.19.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow>2) (2.
Requirement already satisfied: keras>=3.5.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow>2) (3.10.0)
Requirement already satisfied: numpy<2.2.0,>=1.26.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow>2) (2
Requirement already satisfied: h5py>=3.11.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow>2) (3.15.1)
Requirement already satisfied: ml-dtypes<1.0.0,>=0.5.1 in /usr/local/lib/python3.12/dist-packages (from tensorflow>2)
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.12/dist-packages (from astunparse>=1.6.0-
Requirement already satisfied: rich in /usr/local/lib/python3.12/dist-packages (from keras>=3.5.0->tensorflow>2) (13.
Requirement already satisfied: namex in /usr/local/lib/python3.12/dist-packages (from keras>=3.5.0->tensorflow>2) (0.
Requirement already satisfied: optree in /usr/local/lib/python3.12/dist-packages (from keras>=3.5.0->tensorflow>2) (0
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from requests<3,>
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests<3,>=2.21.0->ten
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requests<3,>=2.21.
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages (from requests<3,>=2.21.
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.12/dist-packages (from tensorboard~=2.19.0->
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3.12/dist-packages (from
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.12/dist-packages (from tensorboard~=2.19.0->
Requirement already satisfied: markupsafe>=2.1.1 in /usr/local/lib/python3.12/dist-packages (from werkzeug>=1.0.1->te
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.12/dist-packages (from rich->keras>=3.
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.12/dist-packages (from rich->keras>=
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.12/dist-packages (from markdown-it-py>=2.2.0->ric
```

Next, we import all tools needed before starting:

```
import os, pickle, glob, math
from pprint import pprint

import cv2
import numpy as np
import imutils
from imutils import paths
from tensorflow.keras import Sequential, layers
from matplotlib import pyplot as plt
from matplotlib.gridspec import GridSpec
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split

from lab_2_helpers import *
```

Preprocessing

Ground Truth Characters Extraction

As usual, we will start pre-processing stage by loading CAPTCHA images into the memory:

```
!tar -xJf captcha-images.tar.xz
```

```
# Dataset images folder
CAPTCHA_IMAGE_FOLDER = "./captcha-images"

# List of all the captcha images we need to process
captcha_image_paths = list(paths.list_images(CAPTCHA_IMAGE_FOLDER))
# Review image paths
pprint(captcha_image_paths[:10])
```

```
['./captcha-images/3F2L.png',
 './captcha-images/J3Q3.png',
 './captcha-images/2HKY.png',
 './captcha-images/3G9T.png',
 './captcha-images/B6RD.png',
 './captcha-images/6RTM.png',
 './captcha-images/2E8X.png',
 './captcha-images/2SBQ.png',
 './captcha-images/2NNC.png',
 './captcha-images/4NGH.png']
```

Note that for each image, its file name (without extension) happens to be its corresponding CAPTCHA text. Thus, we extract file names for all CAPTCHA images and save them as labels for future use:

```
def extract_captcha_text(image_path):
    """ Extract correct CAPTCHA texts from file name of images. """
    # Extract file name of image from its path
    # e.g. "./captcha-images/2A2X.png" -> "2A2X.png"
    image_file_name = os.path.basename(image_path)
    # Extract base name of image, omitting file extension
    # e.g. "2A2X.png" -> "2A2X"
    return os.path.splitext(image_file_name)[0]

captcha_texts = [extract_captcha_text(image_path) for image_path in captcha_image_paths]
# Review extraction results
pprint(captcha_texts[:10])
```

```
['3F2L', 'J3Q3', '2HKY', '3G9T', 'B6RD', '6RTM', '2E8X', '2SBQ', '2NNC', '4NGH']
```

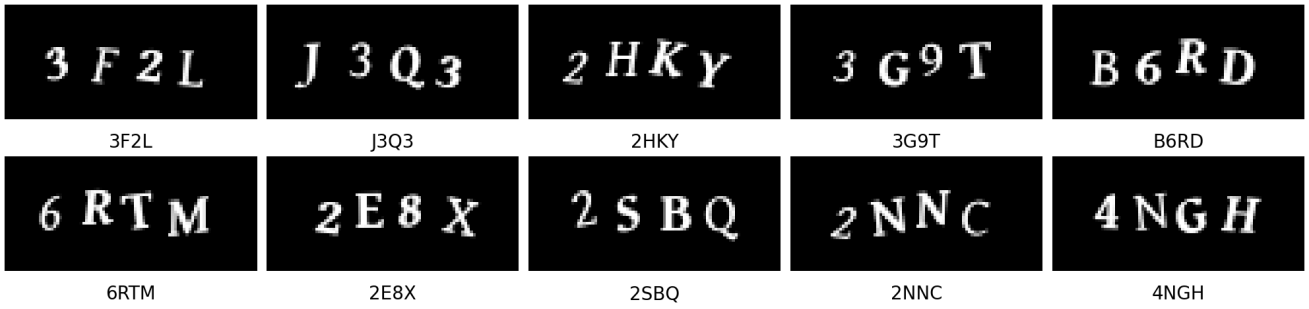
Loading and Transforming Images

For the feature extraction stage, we are going to extract individual characters from these CAPTCHAs. This is done by looking for contours (bounding boxes) around characters, then cropping the CAPTCHAs such as only the contour areas are preserved. We begin feature extraction by loading and transforming images:

```
import cv2

def load_transform_image(image_path):
```

```
    """ Load and transform image into grayscale. """
    ## [ TODO ]
    # 1) Load image with OpenCV
    image = cv2.imread(image_path)

    # 2) Convert image to grayscale
    image_gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    # 3) Add extra padding (8px) around the image
    image_padded = cv2.copyMakeBorder(image_gray, 8, 8, 8, 8, cv2.BORDER_REPLICATE)

    return image_padded

captcha_images = [load_transform_image(image_path) for image_path in captcha_image_paths]
# Review loaded CAPTCHAs
print_images(
    captcha_images[:10], n_rows=2, texts=captcha_texts[:10]
)
```

```
Figure(2000x500)
```



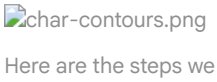
Next, we will split our dataset into train-validation set and test set. The former set will be used for training and validation in deep character classification model, while the latter will be used for testing our CAPTCHA recognition pipline end-to-end:

```
# Train-validation-test split seed
TVT_SPLIT_SEED = 31528476

# Perform split on CAPTCHA images as well as labels
captcha_images_tv, captcha_images_test, captcha_texts_tv, captcha_texts_test = train_test_split(
    captcha_images, captcha_texts, test_size=0.2, random_state=TVT_SPLIT_SEED
)

print("Train-validation:", len(captcha_texts_tv))
print("Test:", len(captcha_texts_test))
```

```
Train-validation: 908
Test: 228
```

Bounding Box Extraction

It's now time to perform the most important feature extraction step: finding contours and extracting characters. Contours can be explained simply as a curve joining all the continuous points (along the boundary), having same color or intensity. It is useful for shape analysis and object detection and recognition. For our task however, we are **more interested in the bounding boxes around characters**, since these are the part of images we will be used for character classification:



Here are the steps we are going to perform:

- Find contours around characters in CAPTCHAs.
- Get the position and size for each corresponding bounding box.
- If a bounding box is too wide (width-to-height ratio larger than 1.25), chances are that we have bounded two letters in a single bounding box. In this case, split the bounding box vertically from the center into two.

- Store all bounding boxes for the CAPTCHA.
- If there aren't 4 bounding boxes for the CAPTCHA, ignore it since our character extraction process must have run into problems in this case.
- Sort the bounding boxes by their X coordinates, so that they match the order corresponding letters occur.
- For each bounding box, extract corresponding region of the image, and store it as an instance of corresponding character at `${CHAR_IMAGE_FOLDER}/{letter}/{count}.png`.

After these steps, we have transformed CAPTCHA images into images of single character. This simplifies our task since now our model only needs to deal with classification (from character image to character itself) rather than also dealing with detection (finding and extracting charatcers).

```
# Character images folder template
CHAR_IMAGE_FOLDER = f"./char-images-{TVT_SPLIT_SEED}"

def extract_chars(image):
    """ Find contours and extract characters inside each CAPTCHA. """
    # Threshold image and convert it to black-white
    image_bw = cv2.threshold(image, 0, 255, cv2.THRESH_BINARY_INV | cv2.THRESH_OTSU)[1]
    # Find contours (continuous blobs of pixels) the image
    contours = cv2.findContours(image_bw, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)[0]

    char_regions = []
    # Loop through each contour
    for contour in contours:
        # Get the rectangle that contains the contour
        x, y, w, h = cv2.boundingRect(contour)

        # Compare the width and height of the bounding box,
        # detect if there are letters conjoined into one chunk
        if w / h > 1.25:
            # Bounding box is too wide for a single character
            # Split it in half into two letter regions
            half_width = int(w / 2)
            char_regions.append((x, y, half_width, h))
            char_regions.append((x + half_width, y, half_width, h))
        else:
            # Only a single letter in contour
            char_regions.append((x, y, w, h))

    # Ignore image if less or more than 4 regions detected
    if len(char_regions)!=4:
        return None
    # Sort regions by their X coordinates
    char_regions.sort(key=lambda x: x[0])

    # Character images
    char_images = []
    # Save each character as a single image
    for x, y, w, h in char_regions:
        # Extract character from image with 2px margin
        char_image = image[y - 2:y + h + 2, x - 2:x + w + 2]
        # Save character images
        char_images.append(char_image)

    # Return character images
    return char_images

def save_chars(char_images, captcha_text, save_dir, char_counts):
    """ Save character images to directory. """
    for char_image, char in zip(char_images, captcha_text):
        # Get the folder to save the image in
        save_path = os.path.join(save_dir, char)
        os.makedirs(save_path, exist_ok=True)

        # Write letter image to file
        char_count = char_counts.get(char, 1)
        char_image_path = os.path.join(save_path, f"{char_count}.png")
        cv2.imwrite(char_image_path, char_image)

        # Update count
        char_counts[char] = char_count+1

# Force character extraction even if results are already available
FORCE_EXTRACT_CHAR = False
```

```
char_counts = {}
# Extract and save images for characters
if FORCE_EXTRACT_CHAR or not os.path.exists(CHAR_IMAGE_FOLDER):
    for captcha_image, captcha_text in zip(captcha_images_tv, captcha_texts_tv):
        # Extract character images
        char_images = extract_chars(captcha_image)
        # Skip if extraction failed
        if char_images is None:
            continue
        # Save character images
        save_chars(char_images, captcha_text, CHAR_IMAGE_FOLDER, char_counts)
```

Label Encoding

During the training stage, we are going to load character images from previous stages as features and generate corresponding labels from their path. We will then rescale features, one-hot encode labels (occurred characters) and save labels to an external file.

```
# Path of occurred characters (labels)
LABELS_PATH = "./labels.pkl"

def make_feature(image):
    """ Process character image and turn it into feature. """
    # Resize letter to 20*20
    image_resized = resize_to_fit(image, 20, 20)
    # Add extra dimension as the only channel
    feature = image_resized[..., None]

    return feature

def make_feature_label(image_path):
    """ Load character image and make feature-label pair from image path. """
    # Load image and make feature
    feature = make_feature(cv2.imread(image_path, cv2.COLOR_BGR2GRAY))
    # Extract label based on the directory the image is in
    label = image_path.split(os.path.sep)[-2]

    return feature, label

# Make features and labels from character image paths
features_tv, labels_tv = unzip((
    make_feature_label(image_path) for image_path in paths.list_images(CHAR_IMAGE_FOLDER)
))

# Scale raw pixel values into range [0, 1]
features_tv = np.array(features_tv, dtype="float")/255
# Convert labels into one-hot encodings
lb = LabelBinarizer()
labels_one_hot_tv = lb.fit_transform(labels_tv)
# Number of classes
n_classes = len(lb.classes_)

# Further split the training data into training and validation set
X_train, X_vali, y_train, y_vali = train_test_split(
    features_tv, labels_one_hot_tv, test_size=0.25, random_state=955996
)
# Save mapping from labels to one-hot encoding
with open(LABELS_PATH, "wb") as f:
    pickle.dump(lb, f)
```

Training

Next, we build a Convolutional Neural Network (CNN) as our classification model with Tensorflow / Keras. The structure of the neural network is shown below:

After building the neural network, we train it and save weights.

```
# Batch size
BATCH_SIZE = 32
```

```
# Number of epochs
N_EPOCHS = 10

# Path of model weights file
MODEL_WEIGHTS_PATH = "./captcha-model.weights.h5"
# Force training even if weights are already available
FORCE_TRAINING = True

# Build a feed-forward neural network
model = Sequential()

## [ TODO ]
# Implement the neural network shown above

# First convolution block: (*, 20, 20, 1) -Conv2D+ReLU-> (*, 20, 20, 20) -MaxPooling2D-> (*, 10, 10, 20)
# (Remember to include input shape for the first layer!)
# 1) Convolution layer: 20 channels, 5*5 kernel, ReLU activation, padded to maintain same shape
# 2) Max pooling layer: 2*2 kernel
model.add(layers.Conv2D(filters=20, kernel_size=(5,5), activation = "relu", padding = "same", input_shape = (20, 20,
model.add(layers.MaxPooling2D(pool_size=(2, 2)))

# Second convolution block: (*, 10, 10, 20) -Conv2D+ReLU-> (*, 10, 10, 50) -MaxPooling2D-> (*, 5, 5, 50)
# Convolution layer: same as above, but use 50 channels
model.add(layers.Conv2D(filters=50, kernel_size=(5,5), activation = "relu", padding = "same",))
model.add(layers.MaxPooling2D(pool_size=(2, 2)))

# Flatten layer reshape features to 1D: (*, 5, 5, 50) -Flatten-> (*, 1250)
model.add(layers.Flatten())

# First fully-connected (linear) layer: (*, 1250) -FC+ReLU -> (*, 500)
model.add(layers.Dense(500, activation = "relu"))
# Last fully-connected (linear) layer: (*, 500) -FC+Softmax-> (*, n_classes)
# Softmax outputs a categorical distribution representing probability for each character
model.add(layers.Dense(n_classes, activation = "softmax"))

# Build classification model with Adam optimizer and cross entropy loss
model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])

if FORCE_TRAINING or not os.path.exists(MODEL_WEIGHTS_PATH):
    # Train the neural network model
    model.fit(
        X_train, y_train, validation_data=(X_vali, y_vali),
        batch_size=BATCH_SIZE, epochs=N_EPOCHS, verbose=1
    )
    # Save model weights to disk
    model.save_weights(MODEL_WEIGHTS_PATH)
else:
    model.load_weights(MODEL_WEIGHTS_PATH)

# Show model summary
print(model.summary())
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/convolutional/base_conv.py:113: UserWarning: Do not pass an
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/10
84/84 ━━━━━━━━━━━━━━━━━━━━ 11s 94ms/step - accuracy: 0.2206 - loss: 2.9069 - val_accuracy: 0.8833 - val_loss: 0.4406
Epoch 2/10
84/84 ━━━━━━━━━━━━━━━━━━━━ 9s 84ms/step - accuracy: 0.9562 - loss: 0.2305 - val_accuracy: 0.9787 - val_loss: 0.1434
Epoch 3/10
84/84 ━━━━━━━━━━━━━━━━━━━━ 7s 80ms/step - accuracy: 0.9876 - loss: 0.0836 - val_accuracy: 0.9787 - val_loss: 0.1324
Epoch 4/10
84/84 ━━━━━━━━━━━━━━━━━━━━ 8s 90ms/step - accuracy: 0.9941 - loss: 0.0358 - val_accuracy: 0.9854 - val_loss: 0.0997
Epoch 5/10
84/84 ━━━━━━━━━━━━━━━━━━━━ 8s 90ms/step - accuracy: 0.9903 - loss: 0.0477 - val_accuracy: 0.9843 - val_loss: 0.0834
Epoch 6/10
84/84 ━━━━━━━━━━━━━━━━━━━━ 8s 98ms/step - accuracy: 0.9980 - loss: 0.0125 - val_accuracy: 0.9798 - val_loss: 0.0998
Epoch 7/10
84/84 ━━━━━━━━━━━━━━━━━━━━ 7s 85ms/step - accuracy: 0.9989 - loss: 0.0061 - val_accuracy: 0.9843 - val_loss: 0.1044
Epoch 8/10
84/84 ━━━━━━━━━━━━━━━━━━━━ 9s 71ms/step - accuracy: 0.9969 - loss: 0.0087 - val_accuracy: 0.9843 - val_loss: 0.0837
Epoch 9/10
84/84 ━━━━━━━━━━━━━━━━━━━━ 4s 43ms/step - accuracy: 0.9996 - loss: 0.0028 - val_accuracy: 0.9865 - val_loss: 0.0995
Epoch 10/10
84/84 ━━━━━━━━━━━━━━━━━━━━ 6s 59ms/step - accuracy: 1.0000 - loss: 7.1486e-04 - val_accuracy: 0.9888 - val_loss: 0.10
Model: "sequential_1"
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 20, 20, 20)	520
max_pooling2d (MaxPooling2D)	(None, 10, 10, 20)	0
conv2d_1 (Conv2D)	(None, 10, 10, 50)	25,050
max_pooling2d_1 (MaxPooling2D)	(None, 5, 5, 50)	0
flatten (Flatten)	(None, 1250)	0
dense (Dense)	(None, 500)	625,500
dense_1 (Dense)	(None, 32)	16,032

```
 Total params: 2,001,308 (7.63 MB)
 Trainable params: 667,102 (2.54 MB)
 Non-trainable params: 0 (0.00 B)
 Optimizer params: 1,334,206 (5.09 MB)
None
```

Evaluation

During the training part, we have validated the performance of our neural network model on images of single characters. Now it's time to test and evaluate CAPTCHAs from the beginning to the end. First, we will need to build the pipeline for CAPTCHA character prediction:

```
# Load labels from file (so we can translate model predictions to actual letters)
with open(LABELS_PATH, "rb") as f:
    lb = pickle.load(f)

# Test our pipeline (and model) with the test set.
# However, you'd want to replace this with some random CAPTCHAs in the real world.

# Dummy character images
DUMMY_CHAR_IMAGES = np.zeros((4, 20, 20, 1))

# Indices of CAPTCHAs on which extractions failed
extract_failed_indices = []
# Extracted character images
char_images_test = []

# Extract character images and make features
for i, captcha_image in enumerate(captcha_images_test):
    # Extract character images
    char_images = extract_chars(captcha_image)

    if char_images:
        char_images_test.extend(char_images)
    # Use dummy character images as placeholder if extraction failed
    else:
        extract_failed_indices.append(i)
        char_images_test.extend(DUMMY_CHAR_IMAGES)
```

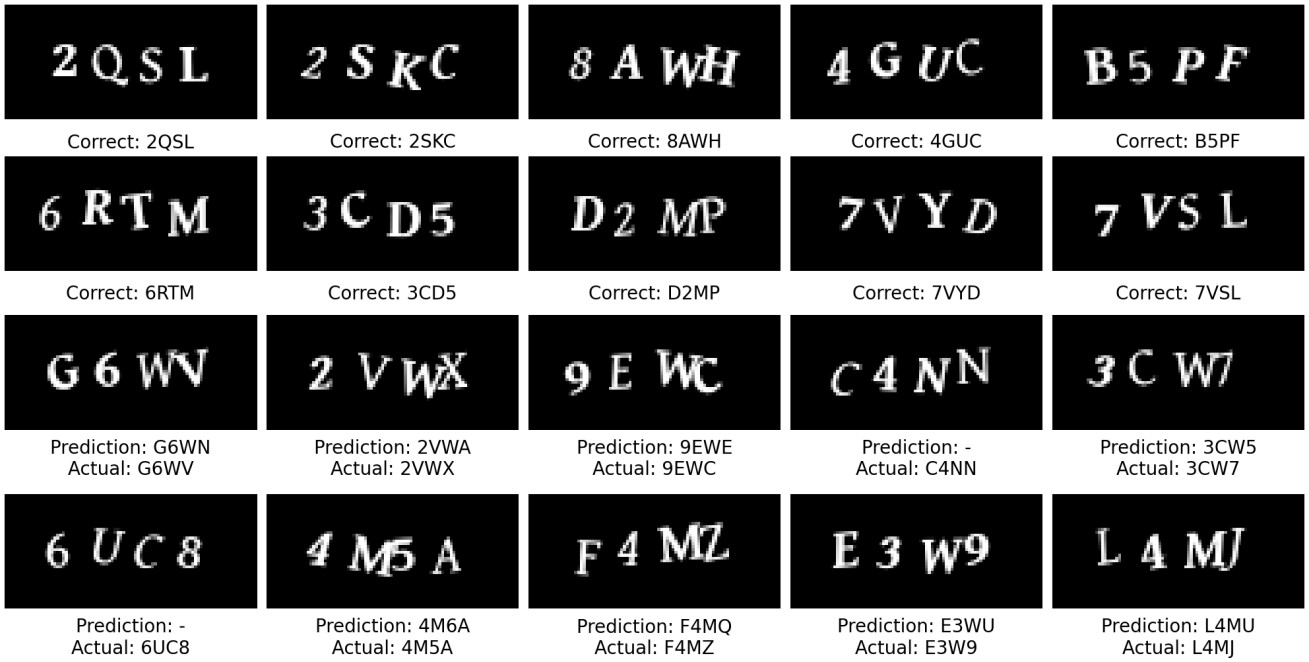
```
# Make features for character images
features_test = [make_feature(char_image) for char_image in char_images_test]
# Scale raw pixel values into range [0, 1]
features_test = np.array(features_test, dtype="float")/255

# Make features for character images
features_test = [make_feature(char_image) for char_image in char_images_test]
# Scale raw pixel values into range [0, 1]
features_test = np.array(features_test, dtype="float")/255
# Predict labels with neural network
preds_test = model.predict(features_test)
# Convert one-hot encoded characters back
preds_test = lb.inverse_transform(preds_test)

# Group all 4 characters for the same CAPTCHA
preds_test = ["".join(chars) for chars in group_every(preds_test, 4)]
# Update result for CAPTCHAs on which extractions failed
for i in extract_failed_indices:
    preds_test[i] = "-"
```

```
29/29 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step
```

Now, we can compute the accuracy of our pipeline, as well as taking a look at correct and incorrect CAPTCHA text predictions:

```
# Number of CAPTCHAs to display
N_DISPLAY_SAMPLES = 10

# Number of test CAPTCHAs
n_test = len(captcha_texts_test)
# Number of correct predictions
n_correct = 0

# Indices of correct predictions
correct_indices = []
# Indices of incorrect predictions
incorrect_indices = []

for i, (pred_text, actual_text) in enumerate(zip(preds_test, captcha_texts_test)):
    if pred_text==actual_text:
        ## [ TODO ]
        # 1) Update number of correct predictions
        n_correct += 1
        ## [ TODO ]
        # 2) Collect index of correct prediction
        if len(correct_indices)<N_DISPLAY_SAMPLES:
            correct_indices.append(i)
    else:
        # 3) Collect index of incorrect prediction
        if len(incorrect_indices)<N_DISPLAY_SAMPLES:
            incorrect_indices.append(i)

# Show number of total / correct predictions and accuracy
print("# of test CAPTCHAs:", n_test)
print("# correctly recognized:", n_correct)
print("Accuracy:", n_correct/n_test, "\n")

# Show all correct predictions
print_images(
    [captcha_images_test[i] for i in correct_indices],
    texts=[f"Correct: {captcha_texts_test[i]}" for i in correct_indices],
    n_rows=2
)

# Show all incorrect predictions
print_images(
    [captcha_images_test[i] for i in incorrect_indices],
    texts=[
        f"Prediction: {preds_test[i]}\nActual: {captcha_texts_test[i]}" \
        for i in incorrect_indices
    ],
    n_rows=2,
    fig_size=(20, 6),
    text_center=(0.5, -0.25)
)
```

```
# of test CAPTCHAs: 228
# correctly recognized: 217
Accuracy: 0.9517543859649122

Figure(2000x500)
Figure(2000x600)
```



Open-Ended Extension:

TensorFlow is one popular framework for implementing Deep Neural Networks, and another widely used framework is **PyTorch**. After you successfully complete this lab:

- Use your favorite language model and/or coding assistant (e.g., **GPT**, **Copilot**, **Gemini**, **Cursor**, etc.) to **convert your TensorFlow CNN implementation into PyTorch**.
- Repeat the same training + evaluation experiments in PyTorch.
- You should observe **similar results** (minor differences are expected due to randomness and implementation details).

✅ **Submission Requirement:**
Include this as a **separate notebook** in your lab-submission GitHub repository named:

`Breaking-CAPTCHAS-Pytorch.ipynb`

References

1. How to break a CAPTCHA system in 15 minutes with Machine Learning: https://medium.com/@ageitgey/how-to-break-a-captcha-system-in-15-minutes-with-machine-learning-dbebb035a710
2. CaptchaSolver Jupyter Notebook: https://github.com/BenjaminWegener/CaptchaSolver