IAM Writer Recognition

This notebook is pretty much a translation of the "handwriting_recognition" notebook by Priyanka Dwivedi. I have chosen to rewrite it differently here as to make it easier to follow, for my own better understanding, and for others who wish to learn from it.

The goal of the notebook is to use the method explained in the paper DeepWriter: A Multi-Stream Deep CNN for Text-independent Writer Identification to identify the writer (author) of a text based on their writing styles. To do so, we'll use the IAM Handwriting Database. Please make sure the dataset has been correctly set up before executing the notebook as outlined here.

Reading The Dataset

The first step is to create a dictionary which will map each form ID (sentence) to a writer. This information is available in the `forms.txt` file, where each line (except for the first 16 lines, which are documentation) defines the form ID at index `0`, and its writer at index `1`.

```
# Create a dictionary to store each form ID and its writer
import os
from itertools import islice

form_writer = {}
forms_file_path = "../data/forms.txt"
with open(forms_file_path) as f:
    for line in islice(f, 16, None):
        line_list = line.split(' ')
        form_id = line_list[0]
        writer = line_list[1]
        form_writer[form_id] = writer
```

Visualize dictionary (as array for simplicity):

```
list(form_writer.items())[0:5]
print("Number of form-writer pairs:", len(form_writer))
print(list(form_writer.items())[0:5])
print("Sample form-writer mappings:", list(form_writer.items())[:5])
```

```
Number of form-writer pairs: 1539
[('a01-000u', '000'), ('a01-000x', '001'), ('a01-003', '002'), ('a01-003u',
'000'), ('a01-003x', '003')]
Sample form-writer mappings: [('a01-000u', '000'), ('a01-000x', '001'), ('a01-
003', '002'), ('a01-003u', '000'), ('a01-003x', '003')]
```

For efficiency reasons, we'll select the 50 most common writers from the dictionary we have created, and the rest of the notebook will only focus on them (as opposed to using the 221 authors present in the dataset).

```
# Select the 50 most common writer

from collections import Counter

top_writers = []
num_writers = 50
writers_counter = Counter(form_writer.values())
for writer_id,_ in writers_counter.most_common(num_writers):
    top_writers.append(writer_id)
```

Visualize the writer id of the top 50 writers:

```
print("Top writer IDs:", top_writers[0:5])
print(top_writers[0:5])
```

```
Top writer IDs: ['000', '150', '151', '152', '153']
['000', '150', '151', '152', '153']
```

From the 50 most common writers we have selected, we'll now need to select the forms (sentences) they have written:

```
top_forms = []
for form_id, author_id in form_writer.items():
    if author_id in top_writers:
        top_forms.append(form_id)
```

Visualize the form id of the top 50 writers:

```
print("Number of top forms:", len(top_forms))
print("Sample form IDs:", top_forms[:5])
print(top_forms[0:5])
```

```
Number of top forms: 452
Sample form IDs: ['a01-000u', 'a01-003u', 'a01-007u', 'a01-011u', 'a01-014u']
['a01-000u', 'a01-003u', 'a01-007u', 'a01-011u', 'a01-014u']
```

Create a temp directory which contains only the sentences of the forms selected above:

```python
import os
import glob
import shutil

# Create temp directory to save writers' forms in (assumes files have already been copied if the directory exists)
temp_sentences_path = "../data/temp_sentences"
if not os.path.exists(temp_sentences_path):
    os.makedirs(temp_sentences_path)

# Debugging Line 4: Check if 'top_forms' is correctly set
print(f"Top Forms: {top_forms}")

original_sentences_path = os.path.join("..", "data", "sentences", "*", "*", "*.png")

# Debugging Line 5: Verify the Paths
print("Files found:", glob.glob(original_sentences_path)[:5])

for file_path in glob.glob(original_sentences_path):
    image_name = file_path.split(os.path.sep)[-1]  # Use os.path.sep for cross-platform compatibility
    form_id = image_name.split('-')[0] + '-' + image_name.split('-')[1]

    if form_id in top_forms:
        # Debugging Line 6: Check if Files are Copied
        print(f"Copying file {file_path} to {temp_sentences_path}/{image_name}")
        try:
            shutil.copy(file_path, os.path.join(temp_sentences_path, image_name))
        except Exception as e:
            print(f"Failed to copy {file_path}. Error: {e}")
```

```
Top Forms: ['a01-000u', 'a01-003u', 'a01-007u', 'a01-011u', 'a01-014u', 'a01-020u', 'a01-026u', 'a01-030u', 'a01-043u', 'a01-049u', 'a01-049x', 'a01-053u', 'a01-058u', 'a01-063u', 'a01-068u', 'a01-072u', 'a01-077u', 'a01-082u', 'a01-087u', 'a01-091u', 'a01-096u', 'a01-102u', 'a01-107u', 'a01-113u', 'a01-117u', 'a01-122u', 'a01-128u', 'a01-132u', 'a01-132x', 'a02-017', 'a02-020', 'a02-024', 'a02-027', 'a02-032', 'a02-037', 'a02-042', 'a02-090', 'a02-093', 'a02-098', 'a02-102', 'a02-106', 'a02-111', 'a02-124', 'a03-047', 'a03-050', 'a03-071', 'a03-073', 'a03-080', 'a03-089', 'a05-000', 'a05-013', 'a05-017', 'a05-022', 'a05-025', 'a05-029', 'a05-039', 'a05-044', 'a05-048', 'a05-053', 'a05-058', 'a05-062', 'a05-069', 'a05-073', 'a05-080', 'a05-084', 'a05-089', 'a05-094', 'a05-099', 'a05-104', 'a05-108', 'a05-113', 'a05-116', 'a05-121', 'a05-125', 'a06-124', 'a06-134', 'a06-141', 'a06-147', 'a06-157', 'b05-055', 'b05-058', 'b05-062', 'b05-067', 'b05-071', 'b06-000', 'b06-008', 'b06-012', 'b06-019', 'b06-023', 'b06-032', 'b06-036', 'b06-056', 'b06-059', 'b06-064', 'b06-071', 'b06-079', 'b06-093', 'b06-097', 'b06-100', 'b06-110', 'c03-000a', 'c03-000b', 'c03-000c', 'c03-000d', 'c03-000e', 'c03-000f', 'c03-003a', 'c03-003b', 'c03-003c', 'c03-003d', 'c03-003e', 'c03-003f', 'c03-
```

```
007a', 'c03-007b', 'c03-007c', 'c03-007d', 'c03-007e', 'c03-007f', 'c03-016a',
'c03-016b', 'c03-016c', 'c03-016d', 'c03-016e', 'c03-021a', 'c03-021b', 'c03-
021c', 'c03-021d', 'c03-021e', 'c03-021f', 'c03-081a', 'c03-081b', 'c03-081c',
'c03-081d', 'c03-081e', 'c03-081f', 'c03-084a', 'c03-084b', 'c03-084c', 'c03-
084d', 'c03-084e', 'c03-084f', 'c03-087a', 'c03-087b', 'c03-087c', 'c03-087d',
'c03-087e', 'c03-087f', 'c03-094a', 'c03-094b', 'c03-094c', 'c03-094d', 'c03-
094e', 'c03-094f', 'c03-096a', 'c03-096b', 'c03-096c', 'c03-096d', 'c03-096e',
'c03-096f', 'c06-000', 'c06-005', 'c06-011', 'c06-020', 'c06-027', 'c06-031',
'c06-039', 'c06-043', 'c06-052', 'c06-076', 'c06-080', 'c06-083', 'c06-087', 'c06-
100', 'c06-116', 'c06-128', 'd06-008', 'd06-015', 'd06-020', 'd06-030', 'd06-046',
'd06-050', 'd06-063', 'd06-082', 'd07-082', 'd07-085', 'd07-089', 'd07-093', 'd07-
096', 'd07-100', 'd07-102', 'e07-000', 'g03-049', 'g05-098', 'g06-011a', 'g06-
011b', 'g06-011c', 'g06-011e', 'g06-011f', 'g06-011g', 'g06-011h', 'g06-011i',
'g06-011j', 'g06-011k', 'g06-011l', 'g06-011m', 'g06-011n', 'g06-011o', 'g06-
011p', 'g06-011r', 'g06-018a', 'g06-018b', 'g06-018c', 'g06-018d', 'g06-018e',
'g06-018f', 'g06-018g', 'g06-018h', 'g06-018i', 'g06-018j', 'g06-018k', 'g06-
018l', 'g06-018m', 'g06-018n', 'g06-018o', 'g06-018p', 'g06-018r', 'g06-026a',
'g06-026b', 'g06-026c', 'g06-026d', 'g06-026e', 'g06-026f', 'g06-026g', 'g06-
026h', 'g06-026i', 'g06-026j', 'g06-026k', 'g06-026l', 'g06-026m', 'g06-026n',
'g06-026o', 'g06-026p', 'g06-026r', 'g06-031a', 'g06-031b', 'g06-031c', 'g06-
031d', 'g06-031e', 'g06-031f', 'g06-031g', 'g06-031h', 'g06-031i', 'g06-031j',
'g06-031k', 'g06-031l', 'g06-031m', 'g06-031n', 'g06-031o', 'g06-031p', 'g06-
031r', 'g06-037b', 'g06-037c', 'g06-037d', 'g06-037e', 'g06-037f', 'g06-037g',
'g06-037h', 'g06-037i', 'g06-037j', 'g06-037k', 'g06-037l', 'g06-037m', 'g06-
037n', 'g06-037o', 'g06-037p', 'g06-037r', 'g06-042a', 'g06-042b', 'g06-042c',
'g06-042d', 'g06-042e', 'g06-042f', 'g06-042g', 'g06-042h', 'g06-042i', 'g06-
042j', 'g06-042k', 'g06-042l', 'g06-042m', 'g06-042n', 'g06-042o', 'g06-042p',
'g06-042r', 'g06-045a', 'g06-045b', 'g06-045c', 'g06-045d', 'g06-045e', 'g06-
045f', 'g06-045g', 'g06-045h', 'g06-045i', 'g06-045j', 'g06-045k', 'g06-045l',
'g06-045m', 'g06-045n', 'g06-045o', 'g06-045p', 'g06-045r', 'g06-047a', 'g06-
047b', 'g06-047c', 'g06-047d', 'g06-047e', 'g06-047f', 'g06-047g', 'g06-047h',
'g06-047i', 'g06-047j', 'g06-047k', 'g06-047l', 'g06-047m', 'g06-047n', 'g06-
047o', 'g06-047p', 'g06-047r', 'g06-050a', 'g06-050b', 'g06-050c', 'g06-050d',
'g06-050e', 'g06-050f', 'g06-050g', 'g06-050h', 'g06-050i', 'g06-050j', 'g06-
050k', 'g06-050l', 'g06-050m', 'g06-050n', 'g06-050o', 'g06-050p', 'g06-050r',
'g06-089', 'g06-093', 'g06-096', 'g06-101', 'g06-105', 'g06-109', 'g06-115', 'h05-
012', 'h06-000', 'h06-003', 'h06-079', 'h06-082', 'h06-085', 'h06-089', 'h06-092',
'h06-096', 'j06-000', 'j06-005', 'j06-008', 'j06-014', 'j06-018', 'j06-022', 'j06-
026', 'j06-030', 'j06-034', 'j06-051', 'j06-056', 'm06-019', 'm06-031', 'm06-042',
'm06-048', 'm06-056', 'm06-067', 'm06-076', 'm06-083', 'm06-091', 'm06-098', 'm06-
106', 'n02-098', 'n02-104', 'n02-109', 'n02-114', 'n02-120', 'n02-127', 'n06-074',
'n06-082', 'n06-092', 'n06-100', 'n06-111', 'n06-119', 'n06-123', 'n06-128', 'n06-
133', 'n06-140', 'n06-148', 'n06-156', 'n06-163', 'n06-169', 'n06-175', 'n06-182',
'n06-186', 'n06-194', 'n06-201', 'p03-057', 'p03-087', 'p03-096', 'p03-103', 'p03-
112', 'p06-030', 'p06-042', 'p06-047', 'p06-052', 'p06-058', 'p06-069', 'p06-088',
'p06-096', 'p06-104', 'p06-242', 'p06-248', 'r03-053', 'r06-000', 'r06-003', 'r06-
007', 'r06-011', 'r06-018', 'r06-022', 'r06-027', 'r06-035', 'r06-041', 'r06-044',
'r06-049', 'r06-053', 'r06-057', 'r06-062', 'r06-066', 'r06-070', 'r06-076', 'r06-
090', 'r06-097', 'r06-103', 'r06-106', 'r06-111', 'r06-115', 'r06-121', 'r06-126',
'r06-130', 'r06-137', 'r06-143']
Files found: ['..\\data\\sentences\\a01\\a01-000u\\a01-000u-s00-00.png',
'..\\data\\sentences\\a01\\a01-000u\\a01-000u-s00-01.png',
'..\\data\\sentences\\a01\\a01-000u\\a01-000u-s00-02.png',
'..\\data\\sentences\\a01\\a01-000u\\a01-000u-s00-03.png',
'..\\data\\sentences\\a01\\a01-000u\\a01-000u-s01-00.png']
Copying file ..\data\sentences\a01\a01-000u\a01-000u-s00-00.png to
../data/temp_sentences/a01-000u-s00-00.png
Copying file ..\data\sentences\a01\a01-000u\a01-000u-s00-01.png to
../data/temp_sentences/a01-000u-s00-01.png
Copying file ..\data\sentences\a01\a01-000u\a01-000u-s00-02.png to
../data/temp_sentences/a01-000u-s00-02.png
Copying file ..\data\sentences\r06\r06-143\r06-143-s03-00.png to
../data/temp_sentences/r06-143-s03-00.png
Copying file ..\data\sentences\r06\r06-143\r06-143-s04-00.png to
../data/temp_sentences/r06-143-s04-00.png
Copying file ..\data\sentences\r06\r06-143\r06-143-s04-01.png to
../data/temp_sentences/r06-143-s04-01.png
```

Create arrays of file inputs (a form) and their respective targets (a writer id):

```
import os
import glob
import shutil
import numpy as np

img_files = np.zeros((0), dtype=str)
img_targets = []

path_to_files = os.path.join(temp_sentences_path, '*')
for file_path in glob.glob(path_to_files):
    img_files = np.append(img_files, file_path)
    file_name, _ = os.path.splitext(file_path.split(os.path.sep)[-1])
    form_id = '-'.join(file_name.split('-')[0:2])
    if form_id in form_writer:
        img_targets.append(form_writer[form_id])

# Convert img_targets to a NumPy array
img_targets = np.array(img_targets)

# Debugging Line 7: Validate Array Populations
print("Array lengths:", len(img_files), len(img_targets))
```

```
Array lengths: 4909 4909
```

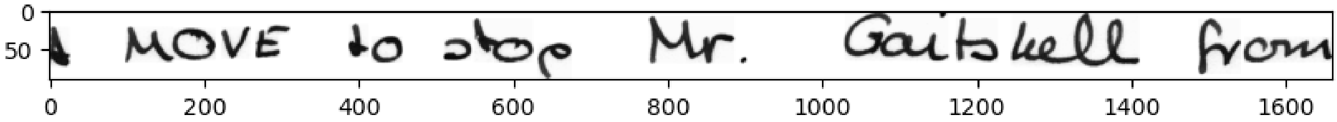
Visualize the form -> writer id arrays:

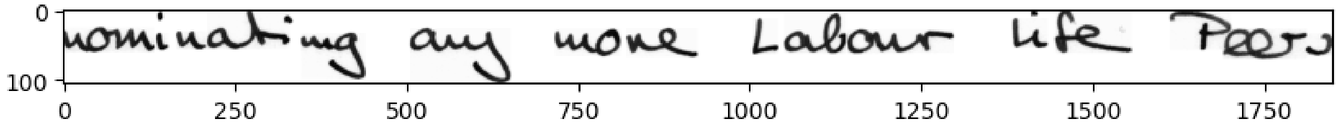
```
print(f"Checking path: {path_to_files}")
files_found = glob.glob(path_to_files)
print(f"Found {len(files_found)} files.")

print(img_files[0:5])
print(img_targets[0:5])
```

```
Checking path: ../data/temp_sentences\*
Found 4909 files.
['../data/temp_sentences\\a01-000u-s00-00.png'
 '../data/temp_sentences\\a01-000u-s00-01.png'
 '../data/temp_sentences\\a01-000u-s00-02.png'
 '../data/temp_sentences\\a01-000u-s00-03.png'
 '../data/temp_sentences\\a01-000u-s01-00.png']
['000' '000' '000' '000' '000']
```

Visualize dataset's images:

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
%matplotlib inline

for file_name in img_files[:2]:
    img = mpimg.imread(file_name)
    plt.figure(figsize = (10,10))
    plt.imshow(img, cmap ='gray')
```





Encode writers with a value between 0 and `n_classes-1`:

```
from sklearn.preprocessing import LabelEncoder

encoder = LabelEncoder()
encoded_img_targets = encoder.fit_transform(img_targets)

print("Writer ID         : ", img_targets[:2])
print("Encoded writer ID: ", encoded_img_targets[:2])
```

```
Writer ID        :  ['000' '000']
Encoded writer ID:  [0 0]
```

Split dataset into train, validation, and tests sets:

```
from sklearn.model_selection import train_test_split

# Split dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(img_files,
encoded_img_targets, test_size=0.2, shuffle = True)

# Further split training set into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2,
shuffle = True)

print(X_train.shape, X_val.shape, X_test.shape)
print(y_train.shape, y_val.shape, y_test.shape)
```

```
(3141,) (786,) (982,)
(3141,) (786,) (982,)
```

Define a couple of constants that will be used throughout the model:

```
CROP_SIZE = 113
NUM_LABELS = 50
BATCH_SIZE = 16
```

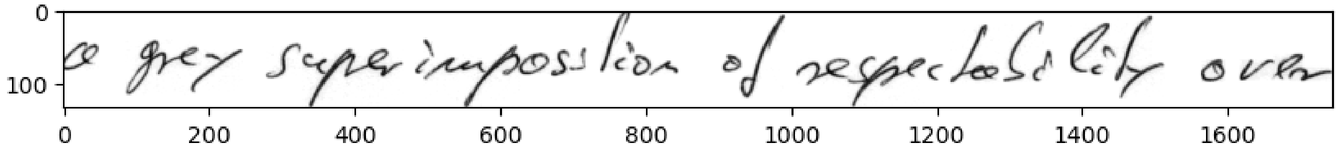
As suggested in the paper, the input to the model are not unique sentences but rather random patches cropped from each sentence. The `get_augmented_sample` method is in charge of doing so by resizing each sentence's height to `113` pixels, and its width such that original aspect ratio is maintained. Finally, from the resized image, patches of `113x113` are randomly cropped.

```
from sklearn.utils import shuffle
from PIL import Image
import random

def get_augmented_sample(sample, label, sample_ratio):
    # Get current image details
    img = Image.open(sample)
    img_width = img.size[0]
    img_height = img.size[1]
    # Compute resize dimensions such that aspect ratio is maintained
    height_fac = CROP_SIZE / img_height
    size = (int(img_width * height_fac), CROP_SIZE)

    # Resize image
    new_img = img.resize((size), Image.ANTIALIAS)
    new_img_width = new_img.size[0]
    new_img_height = new_img.size[1]

    # Generate a random number of crops of size 113x113 from the resized image
    x_coord = list(range(0, new_img_width - CROP_SIZE))
    num_crops = int(len(x_coord) * sample_ratio)
    random_x_coord = random.sample(x_coord, num_crops)

    # Create augmented images (cropped forms) and map them to a label (writer)
    images = []
    labels = []
    for x in random_x_coord:
        img_crop = new_img.crop((x, 0, x + CROP_SIZE, CROP_SIZE))
        # Transform image to an array of numbers
        images.append(np.asarray(img_crop))
        labels.append(label)

    return (images, labels)
```

Let's visualize what the `get_augmented_sample` method does by augmenting one sample from the training set. Let's first take a look at how the original image looks like:

```
sample, label = X_train[0], y_train[0]
img = mpimg.imread(sample)
plt.figure(figsize = (10,10))
plt.imshow(img, cmap ='gray')
print("Label: ", label)
```

```
Label:  21
```



A now, let's augment it and see the result:

```
images, labels = get_augmented_sample(sample, label, 0.1)
```

```
C:\Users\User\AppData\Local\Temp\ipykernel_46608\3032259505.py:16:
DeprecationWarning: ANTIALIAS is deprecated and will be removed in Pillow 10
(2023-07-01). Use Resampling.LANCZOS instead.
  new_img = img.resize((size), Image.ANTIALIAS)
```

The `labels` returned by the `get_augmented_sample` is simply the label of the original image for each cropped patch:

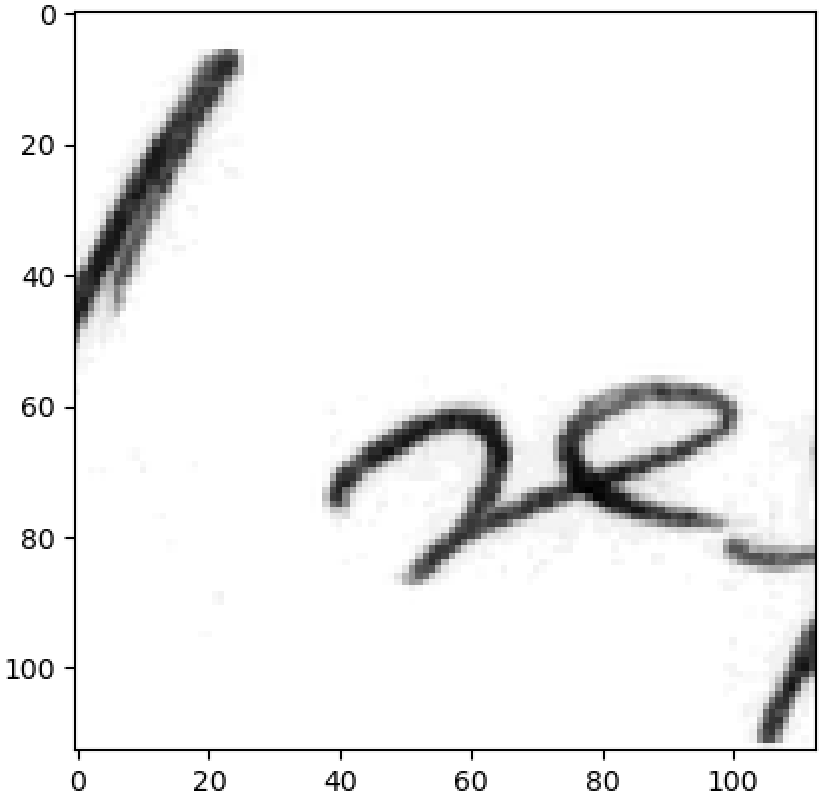
```
print(labels)
print("Num of labels: ", len(labels))
```

```
[21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21,
21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21,
21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21,
21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21,
21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21,
21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21,
21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21]
Num of labels:  138
```

And the `images` returned by it are the random patches created from the original image (only two samples shown for simplicity):

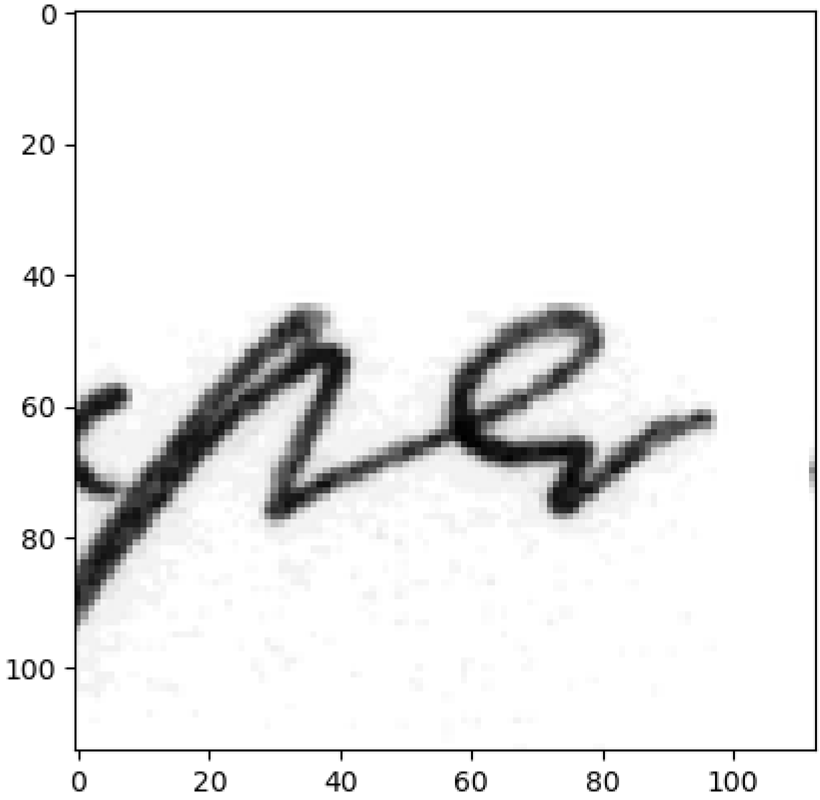
```
print(len(images))
plt.imshow(images[0], cmap ='gray')
```

```
138




<matplotlib.image.AxesImage at 0x279b253ddf0>
```



```
plt.imshow(images[1], cmap ='gray')
```

```
<matplotlib.image.AxesImage at 0x279b2165b80>
```



The model uses a generator in order to be able to call `get_augmented_sample` when training the model:

```
import operator
from functools import reduce
from keras.utils import to_categorical

def generate_data(samples, labels, batch_size, sample_ratio):
    while 1:
        for offset in range(0, len(samples), batch_size):
            batch_samples = samples[offset:(offset + batch_size)]
            batch_labels = labels[offset:(offset + batch_size)]

            # Augment each sample in batch
            augmented_batch_samples = []
            augmented_batch_labels = []
            for i in range(len(batch_samples)):
                sample = batch_samples[i]
                label = batch_labels[i]
                augmented_samples, augmented_labels = get_augmented_sample(sample,
label, sample_ratio)
                augmented_batch_samples.append(augmented_samples)
                augmented_batch_labels.append(augmented_labels)

            # Flatten out samples and labels
            augmented_batch_samples = reduce(operator.add,
augmented_batch_samples)
            augmented_batch_labels = reduce(operator.add, augmented_batch_labels)

            # Reshape input format
            X_train = np.array(augmented_batch_samples)
            X_train = X_train.reshape(X_train.shape[0], CROP_SIZE, CROP_SIZE, 1)

            # Transform input to float and normalize
            X_train = X_train.astype('float32')
            X_train /= 255

            # Encode y
            y_train = np.array(augmented_batch_labels)
            y_train = to_categorical(y_train, NUM_LABELS)

            yield X_train, y_train
```

Create training, validation, and test generators:

```
train_generator = generate_data(X_train, y_train, BATCH_SIZE, 0.3)
validation_generator = generate_data(X_val, y_val, BATCH_SIZE, 0.3)
test_generator = generate_data(X_test, y_test, BATCH_SIZE, 0.1)
```

```
import tensorflow as tf
gpus = tf.config.experimental.list_physical_devices('GPU')
if gpus:
    try:
        for gpu in gpus:
            tf.config.experimental.set_memory_growth(gpu, True)
    except RuntimeError as e:
        print(e)
```

```
def resize_image(img):
    size = round(CROP_SIZE/2)
    return tf.image.resize(img, [size, size])
```

The model used is exactly the same as the one in the "handwriting_recognition" notebook by Priyanka Dwivedi:

```
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Lambda, Activation
from keras.layers.convolutional import Convolution2D, ZeroPadding2D, MaxPooling2D
from keras.optimizers import Adam
from keras import metrics

model = Sequential()

# Define network input shape
model.add(ZeroPadding2D((1, 1), input_shape=(CROP_SIZE, CROP_SIZE, 1)))
# Resize images to allow for easy computation
model.add(Lambda(resize_image))

# CNN model - Building the model suggested in paper
model.add(Convolution2D(filters= 32, kernel_size =(5,5), strides= (2, 2),
padding='same', name='conv1'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2), name='pool1'))

model.add(Convolution2D(filters= 64, kernel_size =(3, 3), strides= (1, 1),
padding='same', name='conv2'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2), name='pool2'))

model.add(Convolution2D(filters= 128, kernel_size =(3, 3), strides= (1, 1),
padding='same', name='conv3'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2), name='pool3'))


model.add(Flatten())
model.add(Dropout(0.5))
model.add(Dense(512, name='dense1'))
model.add(Activation('relu'))
model.add(Dropout(0.5))

model.add(Dense(256, name='dense2'))
model.add(Activation('relu'))
model.add(Dropout(0.5))

model.add(Dense(NUM_LABELS, name='output'))
model.add(Activation('softmax'))

model.compile(loss='categorical_crossentropy', optimizer=Adam(), metrics=['acc'])

print(model.summary())
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 zero_padding2d (ZeroPadding  (None, 115, 115, 1)      0         
 2D)                                                             
                                                                 
 lambda (Lambda)             (None, 56, 56, 1)         0         
                                                                 
 conv1 (Conv2D)              (None, 28, 28, 32)        832       
                                                                 
 activation (Activation)     (None, 28, 28, 32)        0         
                                                                 
 pool1 (MaxPooling2D)        (None, 14, 14, 32)        0         
                                                                 
 conv2 (Conv2D)              (None, 14, 14, 64)        18496     
                                                                 
 activation_1 (Activation)   (None, 14, 14, 64)        0         
                                                                 
 pool2 (MaxPooling2D)        (None, 7, 7, 64)          0         
                                                                 
 conv3 (Conv2D)              (None, 7, 7, 128)         73856     
                                                                 
 activation_2 (Activation)   (None, 7, 7, 128)         0         
                                                                 
 pool3 (MaxPooling2D)        (None, 3, 3, 128)         0         
                                                                 
 flatten (Flatten)           (None, 1152)              0         
                                                                 
 dropout (Dropout)           (None, 1152)              0         
                                                                 
 dense1 (Dense)              (None, 512)               590336    
                                                                 
 activation_3 (Activation)   (None, 512)               0         
 dropout_1 (Dropout)         (None, 512)               0

 dense2 (Dense)              (None, 256)               131328

 activation_4 (Activation)   (None, 256)               0

 dropout_2 (Dropout)         (None, 256)               0

 output (Dense)              (None, 50)                12850

 activation_5 (Activation)   (None, 50)                0

=================================================================
Total params: 827,698
Trainable params: 827,698
Non-trainable params: 0
_________________________________________________________________
None
```

Next, the model is trained for 20 epochs and the models obtained after each epoch are saved to the `./model_checkpoints` directory

```
from keras.callbacks import ModelCheckpoint

# Create directory to save checkpoints at
model_checkpoints_path = "./model_checkpoints"
if not os.path.exists(model_checkpoints_path):
    os.makedirs(model_checkpoints_path)

# Save model after every epoch using checkpoints
create_checkpoint = ModelCheckpoint(
    filepath = "./model_checkpoints/check_{epoch:02d}_{val_loss:.4f}.hdf5",
    verbose = 1,
    save_best_only = False
)

# Fit model using generators
history_object = model.fit_generator(
    train_generator,
    steps_per_epoch = round(len(X_train) / BATCH_SIZE),
    validation_data = validation_generator,
    validation_steps = round(len(X_val) / BATCH_SIZE),
    epochs = 20,
    verbose = 1,
    callbacks = [create_checkpoint]
)
```

```
C:\Users\User\AppData\Local\Temp\ipykernel_46608\3366608101.py:16: UserWarning:
`Model.fit_generator` is deprecated and will be removed in a future version.
Please use `Model.fit`, which supports generators.
  history_object = model.fit_generator(
C:\Users\User\AppData\Local\Temp\ipykernel_46608\3032259505.py:16:
DeprecationWarning: ANTIALIAS is deprecated and will be removed in Pillow 10
(2023-07-01). Use Resampling.LANCZOS instead.
  new_img = img.resize((size), Image.ANTIALIAS)


Epoch 1/20
196/196 [==============================] - ETA: 0s - loss: 3.4459 - acc: 0.2137
Epoch 1: saving model to ./model_checkpoints\check_01_2.9294.hdf5
196/196 [==============================] - 334s 2s/step - loss: 3.4459 - acc:
0.2137 - val_loss: 2.9294 - val_acc: 0.2694
Epoch 2/20
196/196 [==============================] - ETA: 0s - loss: 2.7850 - acc: 0.2656
Epoch 2: saving model to ./model_checkpoints\check_02_2.3100.hdf5
196/196 [==============================] - 388s 2s/step - loss: 2.7850 - acc:
0.2656 - val_loss: 2.3100 - val_acc: 0.3527
Epoch 3/20
196/196 [==============================] - ETA: 0s - loss: 2.3540 - acc: 0.3343
Epoch 3: saving model to ./model_checkpoints\check_03_2.1663.hdf5
196/196 [==============================] - 393s 2s/step - loss: 2.3540 - acc:
0.3343 - val_loss: 2.1663 - val_acc: 0.3764
Epoch 4/20
196/196 [==============================] - ETA: 0s - loss: 2.0742 - acc: 0.3940
Epoch 4: saving model to ./model_checkpoints\check_04_1.9487.hdf5
196/196 [==============================] - 393s 2s/step - loss: 2.0742 - acc:
0.3940 - val_loss: 1.9487 - val_acc: 0.4229
Epoch 5/20
196/196 [==============================] - ETA: 0s - loss: 1.8527 - acc: 0.4480
Epoch 5: saving model to ./model_checkpoints\check_05_1.5315.hdf5
196/196 [==============================] - 375s 2s/step - loss: 1.8527 - acc:
0.4480 - val_loss: 1.5315 - val_acc: 0.5368
Epoch 6/20
196/196 [==============================] - ETA: 0s - loss: 1.6651 - acc: 0.4953
Epoch 6: saving model to ./model_checkpoints\check_06_1.3693.hdf5
196/196 [==============================] - 387s 2s/step - loss: 1.6651 - acc:
0.4953 - val_loss: 1.3693 - val_acc: 0.5806
Epoch 7/20
196/196 [==============================] - ETA: 0s - loss: 1.5082 - acc: 0.5398
Epoch 7: saving model to ./model_checkpoints\check_07_1.2957.hdf5
196/196 [==============================] - 368s 2s/step - loss: 1.5082 - acc:
0.5398 - val_loss: 1.2957 - val_acc: 0.6016
Epoch 8/20
196/196 [==============================] - ETA: 0s - loss: 1.4060 - acc: 0.5687
Epoch 8: saving model to ./model_checkpoints\check_08_1.2402.hdf5
196/196 [==============================] - 365s 2s/step - loss: 1.4060 - acc:
0.5687 - val_loss: 1.2402 - val_acc: 0.6223
```

```
Epoch 9/20
196/196 [==============================] - ETA: 0s - loss: 1.3226 - acc: 0.5937
Epoch 9: saving model to ./model_checkpoints\check_09_1.1062.hdf5
196/196 [==============================] - 358s 2s/step - loss: 1.3226 - acc:
0.5937 - val_loss: 1.1062 - val_acc: 0.6613
Epoch 10/20
196/196 [==============================] - ETA: 0s - loss: 1.2491 - acc: 0.6168
Epoch 10: saving model to ./model_checkpoints\check_10_1.0222.hdf5
196/196 [==============================] - 356s 2s/step - loss: 1.2491 - acc:
0.6168 - val_loss: 1.0222 - val_acc: 0.6847
Epoch 11/20
196/196 [==============================] - ETA: 0s - loss: 1.1783 - acc: 0.6375
Epoch 11: saving model to ./model_checkpoints\check_11_0.9577.hdf5
196/196 [==============================] - 366s 2s/step - loss: 1.1783 - acc:
0.6375 - val_loss: 0.9577 - val_acc: 0.7061
Epoch 12/20
196/196 [==============================] - ETA: 0s - loss: 1.1203 - acc: 0.6556
Epoch 12: saving model to ./model_checkpoints\check_12_0.9189.hdf5
196/196 [==============================] - 361s 2s/step - loss: 1.1203 - acc:
0.6556 - val_loss: 0.9189 - val_acc: 0.7194
Epoch 13/20
196/196 [==============================] - ETA: 0s - loss: 1.0756 - acc: 0.6696
Epoch 13: saving model to ./model_checkpoints\check_13_0.8940.hdf5
196/196 [==============================] - 360s 2s/step - loss: 1.0756 - acc:
0.6696 - val_loss: 0.8940 - val_acc: 0.7260
Epoch 14/20
196/196 [==============================] - ETA: 0s - loss: 1.0372 - acc: 0.6811
Epoch 14: saving model to ./model_checkpoints\check_14_0.9065.hdf5
196/196 [==============================] - 347s 2s/step - loss: 1.0372 - acc:
0.6811 - val_loss: 0.9065 - val_acc: 0.7184
Epoch 15/20
196/196 [==============================] - ETA: 0s - loss: 0.9920 - acc: 0.6955
Epoch 15: saving model to ./model_checkpoints\check_15_0.8637.hdf5
196/196 [==============================] - 358s 2s/step - loss: 0.9920 - acc:
0.6955 - val_loss: 0.8637 - val_acc: 0.7336
Epoch 16/20
196/196 [==============================] - ETA: 0s - loss: 0.9524 - acc: 0.7072
Epoch 16: saving model to ./model_checkpoints\check_16_0.7901.hdf5
196/196 [==============================] - 390s 2s/step - loss: 0.9524 - acc:
0.7072 - val_loss: 0.7901 - val_acc: 0.7580
Epoch 17/20
196/196 [==============================] - ETA: 0s - loss: 0.9241 - acc: 0.7159
Epoch 17: saving model to ./model_checkpoints\check_17_0.7386.hdf5
196/196 [==============================] - 355s 2s/step - loss: 0.9241 - acc:
0.7159 - val_loss: 0.7386 - val_acc: 0.7745
Epoch 18/20
196/196 [==============================] - ETA: 0s - loss: 0.8996 - acc: 0.7240
Epoch 18: saving model to ./model_checkpoints\check_18_0.7276.hdf5
196/196 [==============================] - 360s 2s/step - loss: 0.8996 - acc:
0.7240 - val_loss: 0.7276 - val_acc: 0.7780
Epoch 19/20
```

```
196/196 [==============================] - ETA: 0s - loss: 0.8938 - acc: 0.7259
Epoch 19: saving model to ./model_checkpoints\check_19_0.8167.hdf5
196/196 [==============================] - 353s 2s/step - loss: 0.8938 - acc:
0.7259 - val_loss: 0.8167 - val_acc: 0.7484
Epoch 20/20
196/196 [==============================] - ETA: 0s - loss: 0.8745 - acc: 0.7315
Epoch 20: saving model to ./model_checkpoints\check_20_0.7311.hdf5
196/196 [==============================] - 359s 2s/step - loss: 0.8745 - acc:
0.7315 - val_loss: 0.7311 - val_acc: 0.7771
```

Load a saved model weights and use them to predict labels in the test set:

```
model_weights_path = "./model_checkpoints/model_weights.hdf5"
if model_weights_path:
    model.load_weights(model_weights_path)
    scores = model.evaluate_generator(test_generator,
steps=round(len(X_test)/BATCH_SIZE))
    print("Accuracy: ", scores[1])
else:
    print("Set model weights file to load in the 'model_weights_path' variable")
```

```
C:\Users\User\AppData\Local\Temp\ipykernel_46608\1622692593.py:4: UserWarning:
`Model.evaluate_generator` is deprecated and will be removed in a future version.
Please use `Model.evaluate`, which supports generators.
  scores = model.evaluate_generator(test_generator,
steps=round(len(X_test)/BATCH_SIZE))
C:\Users\User\AppData\Local\Temp\ipykernel_46608\3032259505.py:16:
DeprecationWarning: ANTIALIAS is deprecated and will be removed in Pillow 10
(2023-07-01). Use Resampling.LANCZOS instead.
  new_img = img.resize((size), Image.ANTIALIAS)


Accuracy:  0.704222559928894
```