Master 2 Apprentissage machine pour la science des données

PROJET DE DEEP LEARNING

Unsupervised representation learning by predicting image rotations

BAYRI Maryeme JELTI Imane Bouraï Assia

Université de paris - ufr des sciences fondamentales et biomédicales

Année universitaire : 2021 - 2022

TABLE DES FIGURES

TABLE DES MATIÈRES

CHAPITRE 1

INTRODUCTION GÉNÉRALE

1.1 Introduction générale

Le présent document constitue une synthèse du travail réalisé dans le cadre du projet de l'UE Deep Learning au sein de l'Université de Paris. Un projet durant lequel nous étions chargés de lire et de comprendre l'article intitulé "Unsupervised representation learning by predecting image rotations" [1] et d'appliquer les méthodes utilisées sur les données MNIST. L'objectif de ce projet de deep learning est de connaître comment utiliser les réseaux convolutifs dans le domaine de l'imagerie, plus précisément, dans la détection des transformations géométriques appliquées sur des images. En dressant tout d'abord un état de l'art des différents algorithmes de reconnaissance d'image, qui nous permettra d'évaluer sainement les méthodes et les réalisations des auteurs de l'article [1].

1.2 Description du projet

Ces dernières années, les réseaux de neurones convolutifs profonds (ConvNets) ont transformé le domaine de l'imagerie grâce à leur capacité inégalée à apprendre des caractéristiques sémantiques de haut niveau des images. Cependant, pour réussir à apprendre ces caractéris-

tiques, ils ont généralement besoin de quantités massives d'images labelisées manuellement, ce qui est à la fois coûteux et peu pratique. Par conséquent, l'apprentissage non supervisé de caractéristiques sémantiques, c'est-à-dire l'apprentissage sans effort de labellisation manuelle, est d'une importance cruciale pour exploiter avec succès la grande quantité de données visuelles disponibles aujourd'hui.

Pour résoudre ce problème S. Gidariset al ont proposé une nouvelle approche d'apprentissage auto-supervisé de caractéristiques qui entraîne un modèle ConvNet à être capable de reconnaître la rotation d'image qui a été appliquée à des images d'entrée. Ils ont démontré que cette tâche apparemment simple fournit en fait un signal de supervision très puissant pour l'apprentissage de caractéristiques sémantiques des images.

Notre travail est donc de comprendre leur travaux et réimplémenter leurs algorithmes en faisant une comparaison avec l'application des ConvNets sur des données MNIST labellisées et de pouvoir implémenter les modèles.

Le projet a été élaboré en quatres grandes étapes, à savoir :

- Compréhension de l'article et des approches utilisées.
- Mise en œuvre d'une baseline de reconnaissance des images MNIST.
- Implémentation du modèle de prédiction des rotations des images sur les données MNIST.
- Application du fine-tuning sur le modèle précédent afin de prédire le contenu des images.

Dans un premier temps, nous découvrirons le projet dans son contexte général. Nous analyserons ensuite le projet de plus près , en nous assurant à expliciter les méthodes et les algorithmes d'apprentissage auto-supervisé et non-supervisé existants. Nous entamerons ensuite l'élaboration des spécifications décrivant les besoins, qui nous permettront d'expliquer les méthodes et les algorithmes de deep learning utilisés par les auteurs de l'article.

Viendra par la suite la partie technique, où nous détaillerons l'implémentation de nos différents modèles.

Pour finir nous présenterons les résultats de nos experimentations et réalisations éffectuées.

CHAPITRE 2

ÉTAT DE L'ART

2.1 Convolutional Neural Networks

Les Convolutional Neural Networks sont des modèles d'inspiration biologique issus d'une recherche de D. H. Hubel et T. N. Wiesel du laboratoire de neurophysiologie, département de pharmacologie de la Harvard Medical School. Ils ont proposé une explication de la façon dont les mammifères perçoivent visuellement le monde qui les entoure en utilisant une architecture en couches de neurones dans le cerveau [2].

Dans leur hypothèse, il existe un ensemble de cellules qui sont des neurones et ces cellules forment des grappes (couches). Ces grappes représentent les différentes caractéristiques qui sont apprises. Ainsi, chaque fois que nous voyons quelque chose, une série de ces groupes est activée. Chacun de ces groupes détectera un ensemble de caractéristiques que possède l'objet détecté. Cet article a inspiré Yann LeCun, chef de l'intelligence artificielle chez Facebook AI Research, et le père fondateur des réseaux convolutifs. Et ce, en utilisant trois caractéristiques principales du cortex visuel des mammifères qui sont :

- Local Connection : comment chaque ensemble de neurones est connecté à un autre en-

semble (cluster) de neurones. [2]

- Layering : La hiérarchie des caractéristiques (features) qui sont apprises.[2] - Spatial Invariance : : un décalage du signal d'entrée entraîne un décalage identique du signal de sortie. La plupart d'entre nous sont capables de reconnaître des visages spécifiques dans diverses conditions parce que nous apprenons l'abstraction. Ces abstractions sont donc invariantes par rapport à la taille, le contraste, la rotation, l'orientation.[2]

2.2 ImgeNet classifcation avec deep CNNs (AlexNet)

AlexNet est le nom d'une architecture de réseau neuronal convolutif (CNN), conçue par Alex Krizhevsky en collaboration avec Ilya Sutskever et Geoffrey Hinton, qui était le directeur de thèse de Krizhevsky.[3]

Dans cet article, Alex Krizhevsky, Ilya Sutskever et Georey E. Hinton ont formé un CNN profond avec cinq couches convolutionnelles, dont certaines avaient des couches de max-pooling, et trois couches entièrement connectées avec un softmax final à 1000 voies pour les 1000 classes différentes. L'ensemble de données ImageNet contient 1,2 million d'images. Le réseau a 60 millions de paramètres et donc, pour accélérer l'apprentissage, ils ont utilisé ReLU qui est une fonction d'activation non linéaire, qui rend la l'apprentissage plus rapide qu'avec d'autres fonctions telles que tanH. Ils ont également réparti le réseau sur deux GPU et les ont laissé communiquer uniquement dans certaines couches pour optimiser le calcul. Afin de réduire l'overfetting, ils ont utilisé l'augmentation des données : transformation horizontales et modification des intensités du RVB, ce qui n'a nécessité aucun calcul puisque la transformation a été effectuée pendant que le modèle formait le batch précédent. Enfin, ils ont utilisé une technique de Dropout pour éviter l'overfetting. [4].

Leur réseau a eu de très bons résultats avec un taux d'erreur de 37,5% et de 17% pour les ensembles de tests les plus importants.

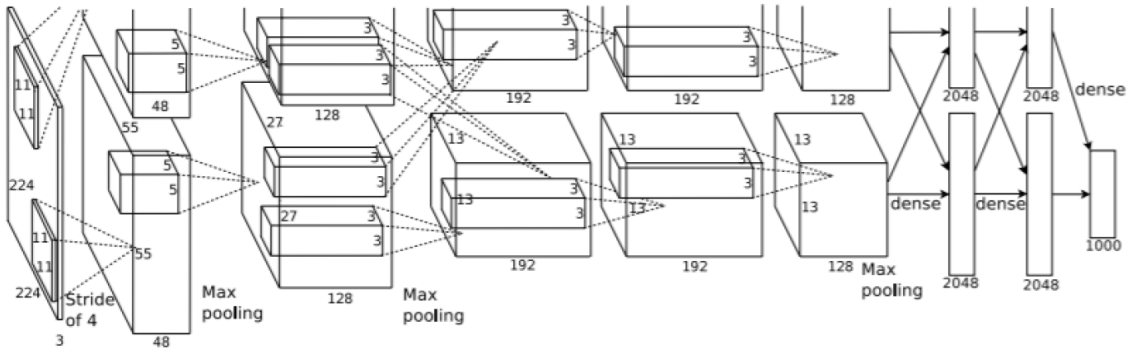


FIGURE 2.1 – Network layers

2.3 Classification de la base de données MNIST à l'aide d'un réseau de neurones

La comparaison de Wan Ahu entre ANN, Autocoder et CNN appliquée à MNIST, le jeu de données des chiffres manuscrits, était assez claire et bien faite. Il a expliqué les différences entre les solutions et a montré comment CNN surpasse les autres méthodes.

Un autoencodeur est un réseau neuronal conçu pour apprendre une représentation de son entrée en essayant de la copier sur sa sortie. Il peut être considéré comme deux fonctions distinctes, un encodeur et un décodeur. Ils ont testé cette technique sur la base de données de chiffres manuscrits MNIST, en cherchant à savoir si elle sera utile ou non. L'architecture utilisée est un réseau à 28 * 28 entrées, plusieurs couches cachées et 28 * 28 sorties, chacun des pixels des 28 par 28 parties de l'image étant une entrée et une sortie uniques.

Quant au CNN, ils ont conçu un réseau de neurones convolutifs inspiré de l'architecture LeNet et l'ont testé sur la base de données de chiffres manuscrits MNIST.

batch_size	learning rate	number_of_epochs	Model	accuracy
100	0.005	6	Original	97.65%
100	0.005	6	Autoencoder	74.38%
100	0.005	6	CNN	98.84%

FIGURE 2.2 – Comparaison des modèles

CHAPITRE 3

DESCRIPTION DES DONNÉES ET MÉTHODES

3.1 Dataset

3.1.1 MNIST

Dans ce projet nous allons travailler sur le jeu de données de chiffres manuscrits MNIST (figure 3.1).

FIGURE 3.1 – MNIST Dataset

MNIST est une grande base de données de petites images carrées de taille 28 X 28 pixels représentant des chiffres uniques écrits à la main entre 0 et 9. Elle se compose d'un total de 70 000 images manuscrites de chiffres, l'ensemble d'entraînement comprenant 60 000 images et l'ensemble de test en comprenant 10 000. Toutes les images sont étiquetées avec le chiffre respectif qu'elles représentent. Il y a un total de 10 classes de chiffres (de 0 à 9).

3.1.2 Rotation des images

L'objectif de notre travail est d'apprendre les caractéristiques sémantiques des images en utilisant les ConvNets d'une manière non supervisée. Pour atteindre cet objectif, S. Gidaris et al proposent d'entraîner un modèle ConvNet pour estimer la transformation géométrique appliquée à une image qui lui est donnée en entrée.

Afin de mettre en œuvre ces transformations géométriques sur les images, nous faisons des rotations de 90, 180 et 270 degrés sur chaque image des données MNIST (le cas 0 degrés est l'image elle-même) et nous utilisons les opérations de transposition. Plus précisément, pour une rotation de 90 degrés, nous transposons d'abord l'image, puis nous la retournons verticalement (upside-down flip), pour une rotation de 180 degrés, nous retournons l'image d'abord verticalement, puis horizontalement (left-right flip), et enfin, pour une rotation de 270 degrés, nous retournons d'abord l'image verticalement, puis nous la transposons.

3.2 Description de l'approche implémentée

3.2.1 Apprentissage auto-supervisé

L'apprentissage auto-supervisé ou self-supervised learning est une sorte d'apprentissage non supervisé où la tâche d'apprentissage supervisée est créée à partir des données d'entrée non étiquetées.

Cette tâche peut être aussi simple que de prédire la moitié supérieure ou inférieure d'une image, la version en niveaux de gris de l'image colorée, les canaux RVB de la même image

ou même prédire la rotation des images comme présenté dans ce projet (figure 3.2).
La principale caractéristique de cette méthode d'apprentissage, est qu'au lieu d'entraîner un modèle à l'aide de données étiquetées, on les fabrique automatiquement.
L'idée est donc de remplacer le travail fastidieux que représente l'étiquetage manuel lorsqu'on dispose d'un gros volume de données par celui consistant à concevoir et mettre en place un mécanisme d'étiquetage automatique.

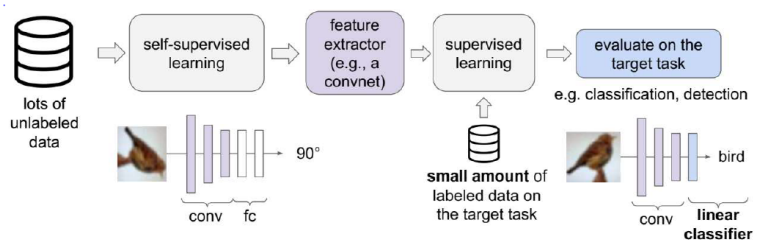


FIGURE 3.2 – Approche auto-supervisée en utilisant la prédiction de la rotation d'images [6].

3.2.2 RotNet

Le travail effectué par les auteurs de l'article se base sur une approche auto-supervisée et propose d'apprendre des représentations d'images en entraînant des ConvNets à reconnaître la transformation géométrique qui est appliquée à l'image donnée en entrée.
Chacune de ces transformations géométriques est appliquée à chaque image du dataset original. Les images transformées obtenues en sortie sont introduites dans le modèle ConvNet qui est formé pour reconnaître la transformation de chaque image.
Dans ce cas, c'est l'ensemble des transformations géométriques qui définit la tâche préalable que le modèle ConvNet doit apprendre. Par conséquent, afin de réaliser l'apprentissage non supervisé de caractéristiques sémantiques, il est important que le modèle ConvNet apprenne les transformations géométriques.
Les auteurs ont proposé de définir les transformations géométriques comme étant les rotations de l'image par 0, 90, 180 et 270 degrés.

Ainsi, le modèle AlexNet utilisé par les auteurs est entraîné sur la tâche de classification

d'image à 4 voies consistant à reconnaître l'une des quatre rotations d'image 3.3.

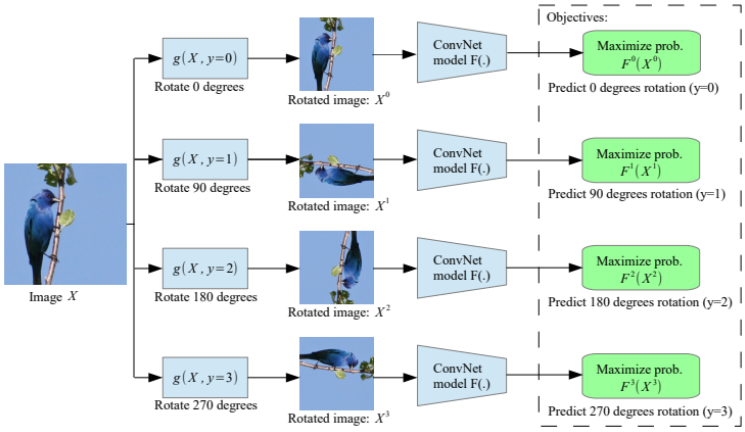


FIGURE 3.3 – Illustration de l'approche auto-supervisée proposée dans [1].

D'après le papier étudié, les avantages de l'utilisation de la rotation d'images comme transformation géométrique sont les suivants :

— Elle oblige le modèle à apprendre des caractéristiques sémantiques : en effet, afin de prédire avec succès la rotation d'une image, le modèle doit nécessairement apprendre à localiser les objets saillants dans l'image, à reconnaître leur orientation et leur type, puis à mettre en relation l'orientation de l'objet avec l'orientation dominante dans laquelle chaque type d'objet tend à être représenté dans les images disponibles.

— Absence de perturbations visuelles : un autre avantage important de l'utilisation des rotations d'images par des multiples de 90 degrés par rapport à d'autres transformations géométriques, est qu'elles peuvent être implémentées par des opérations de retournement et de transposition qui ne laissent pas de parasites visuels facilement détectables qui conduiraient le ConvNet à apprendre des caractéristiques triviales sans valeur pratique pour les tâches de prédictions.

— La tâche de reconnaissance de la rotation est bien définie, c'est-à-dire qu'étant donné une image ayant subi une rotation de 0, 90, 180 ou 270 degrés, il n'y a généralement aucune ambiguïté sur la transformation de la rotation.

— L'implémentation de rotation d'images est plutôt facile.

CHAPITRE 4

ARCHITECTURE DES MODÈLES ET RÉSULTATS

4.1 Baseline

4.1.1 Architecture du modèle

Nous avons entraîné un réseau de neurones en utilisant 100 labels uniquement. Cette méthode supervisée nous aidera à comparer les résultats de cette approche avec les résultats obtenus en entrainant les 100 labels en utilisant le modèle pré-entrainé de RotNet sur des données non labellisées.

La figure 4.1 résume le modèle du réseau CNN que nous avons implémenté et entraîné sur les données MNIST.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320

 max_pooling2d (MaxPooling2D  (None, 13, 13, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 11, 11, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 5, 5, 64)         0
 2D)

 conv2d_2 (Conv2D)           (None, 3, 3, 64)          36928

 flatten (Flatten)           (None, 576)               0

 dense (Dense)               (None, 256)               147712

 dropout (Dropout)           (None, 256)               0

 dense_1 (Dense)             (None, 500)               128500

 dropout_1 (Dropout)         (None, 500)               0

 dense_2 (Dense)             (None, 64)                32064

 dropout_2 (Dropout)         (None, 64)                0

 dense_3 (Dense)             (None, 10)                650

=================================================================
Total params: 364,670
Trainable params: 364,670
Non-trainable params: 0
_________________________________________________________________
```

FIGURE 4.1 – Illustration de l'approche auto-supervisée proposée dans [1].

4.1.2 Résultats

L'entrainement du dataset MNIST avec 100 labels uniquement a donné les résultats suivants

- loss : 0.0333
- accuracy : 0.9800
- val_loss : 1.4985
- val_accuracy : 0.7911

Nous constatons que notre modèle donne de très bons résultats de classification sur MNIST. La figure 4.2 illuste les courbes obtenues lors des étapes d'entrainement et de test du modèle implémenté sur la baseline.

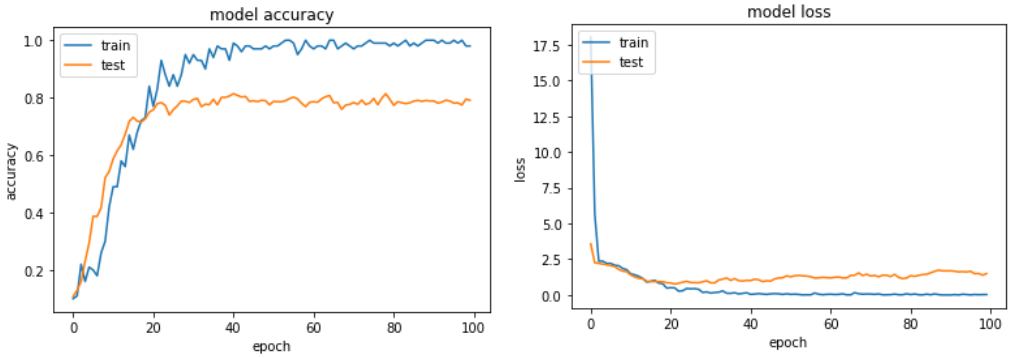


FIGURE 4.2 – Evolution de l'accuracy et de la loss lors de l'entrainement et du test du modèle .

4.2 RotNet sur MNIST

4.2.1 Architecture du modèle

Concernant l'architecture du modèle RotNet, nous avons essayé d'utiliser une architecture ConvNet nous permmettant de pédire la rotation de l'image avec un taux d'erreur sur la prédiction de la rotation atteignant les 26.97%.

La figure 4.3 ci dessous présente l'architecture de notre modèle.

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv_1 (Conv2D)             (None, 26, 26, 20)        200

 max_pooling2d_4 (MaxPooling  (None, 13, 13, 20)       0
 2D)

 conv_2 (Conv2D)             (None, 11, 11, 50)        9050

 max_pooling2d_5 (MaxPooling  (None, 5, 5, 50)         0
 2D)

 permute (Permute)           (None, 5, 5, 50)          0

 flatten_1 (Flatten)         (None, 1250)              0

 dense_1 (Dense)             (None, 500)               625500

 dense_2 (Dense)             (None, 360)               180360

=================================================================
Total params: 815,110
Trainable params: 815,110
Non-trainable params: 0
_________________________________________________________________
```

FIGURE 4.3 – Architecture du modèle RotNet

La figure 4.4 représente les résultats donnés par le modèle chargé de prédire la rotation des

images.

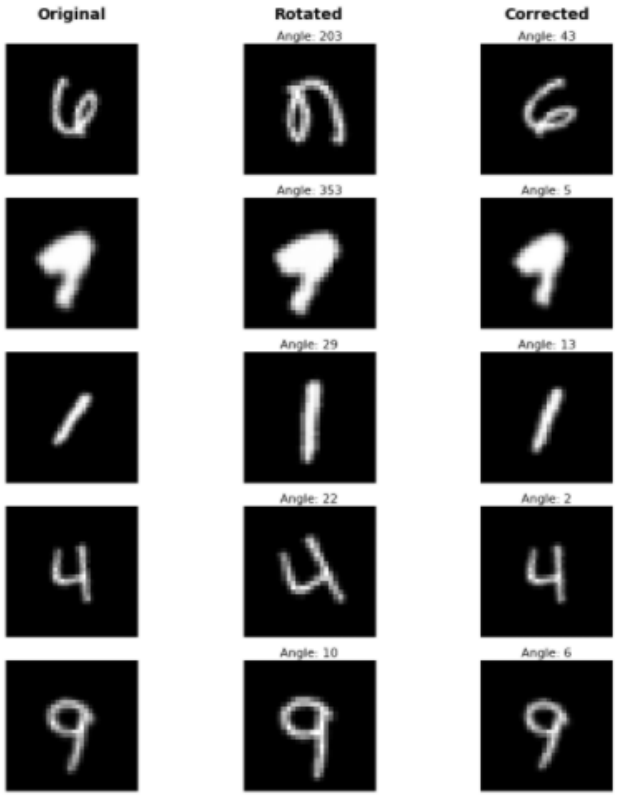


FIGURE 4.4 – Résultats du modèle RotNet

4.2.2 Résultats

Après avoir pré-entrainé notre modèle auto-supervisé RotNet en utilisant les données non labellisées, nous avons mené une étape de finetuning sur ce modèle dans le but de prédire les classes en utilisant les 100 données labellisées de MNIST.

Les figure 4.5 ci dessous montre les résultats obtenus lors de cette étape.

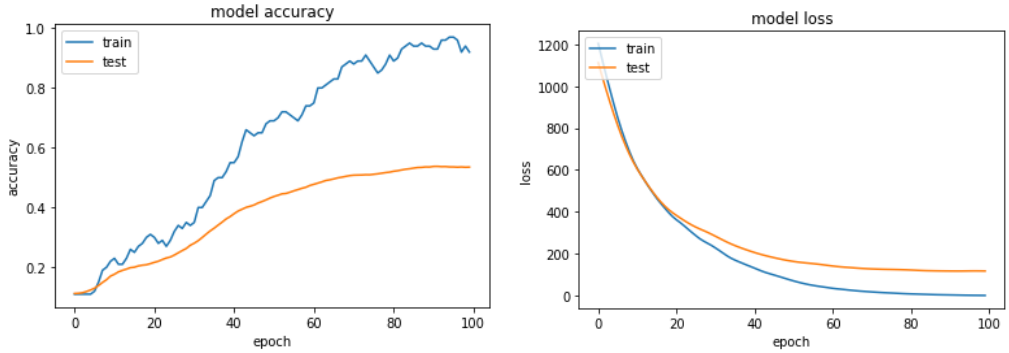


FIGURE 4.5 – Evolution de l'accuracy et de la loss lors de l'entrainement et du test du modèle .

CHAPITRE 5

CONCLUSION

Dans ce travail, nous nous sommes initiées aux méthodes d'apprentissage profond auto-supervisées, où nous avons implémenté une méthode inspirée d'un article de S.Gidaris et al se basant sur la prédiction de la rotation d'images sur les données MNIST.

Nous avons pu comparer les résultats obtenus à l'issue de l'entrainement du modèle inspiré de l'approche auto-supervisée avec les résultats donnés par un modèle implémenté sur une baseline de 100 labels et nous ne sommes pas parvenus à obtenir des résultats meilleurs en termes d'Accuracy et de Loss. Ceci est probablement dû à notre choix d'architecture du modèle ConvNet utilisé sur les données MNIST. Une autre raison pourrait etre les différence des datasets utilisés, étant donné que les auteurs du papier ont travaillé sur ImageNet et CIFAR-10

Nous mettons en Annexe le notebook contenant le code python.

BIBLIOGRAPHIE

[1] Spyros Gidaris, Praveer Singh, Nikos Komodakis, UNSUPERVISED REPRESENTATION LEARNING BY PREDICTING IMAGE ROTATIONS, 2018

[2] Siraj Raval. Convolutional neural network. Github.com, 2017.

[3] Gershgorn, Dave. "The data that transformed AI research—and possibly the world". Quartz.

[4] Ilya Sutskever Alex Krizhevsky and Georey E. Hinton. Imagenet classication with deep convolutional neural networks. University of Toronto.

[5] Wan Zhu. Classication of mnist handwritten digit database using neural network. Research School of Computer Science, Australian National University, Acton, ACT 2601, Australia.

[6] Cours de Depp Learning M2 AMSD, Transfer learning Self-supervised learning, Blaise Hanczar, Novembre 2021.

ANNEXE

Projet_Deep_Learning_Mnist

December 10, 2021

1 1- Importing packages

[ ]:
```
import os
import sys
import math
import cv2
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.layers import Dense, Conv2D, MaxPooling2D, Flatten,␣
↪Activation,BatchNormalization
from tensorflow.keras.models import Sequential,load_model
from tensorflow.keras.utils import to_categorical
from tensorflow.keras import backend, layers

from __future__ import division


from keras.preprocessing.image import Iterator
from keras.utils.np_utils import to_categorical
import keras.backend as K
from keras.callbacks import ModelCheckpoint, EarlyStopping, TensorBoard
from keras.datasets import mnist
from keras.layers import Dense, Dropout, Flatten, Input
from keras.layers import Conv2D, MaxPooling2D
from keras.models import Model
```

2 2- Loading and preparing the data

[ ]:
```
(x_train, y_train), (x_test, y_test) = mnist.load_data() #load mnist data
```

[ ]:
```
# sort and get index of each class
a = []
for i in range(0, 10):
    b = []
    for j in range(0, 60000):
        if y_train[j] == i:
            b.append(j) # each b contains indices of i class
    a.append(b)
```

[ ]:
```
import random
rand = []
for i in range(0, 10):
  rand.append(random.choices(a[i], k=10)) # choose randomly 10 sample per class
rand = np.array(rand).flatten().tolist()
```

[ ]:
```
x_train_lab = x_train[rand] # apply mask to get 100 random samples
y_train_lab = y_train[rand] # 10 samples per class
```

[ ]:
```
not_rand = [] # list of indices that dont exist in labeled data
for i in range(0, 60_000):
    if i not in rand:
        not_rand.append(i)
not_rand = np.array(not_rand)

x_train_unlab = x_train[not_rand] # apply the mask to get the unlabled data
y_train_unlab = y_train[not_rand]
```

3 3- Rotation functions

[ ]:
```
#Needed functions for ResNet
def angle_difference(x, y):
    """
    Calculate minimum difference between two angles.
    """
    return 180 - abs(abs(x - y) - 180)


def angle_error(y_true, y_pred):
    """
    Calculate the mean diference between the true angles
    and the predicted angles. Each angle is represented
    as a binary vector.
    """
    diff = angle_difference(K.argmax(y_true), K.argmax(y_pred))
    return K.mean(K.cast(K.abs(diff), K.floatx()))
```

```
def angle_error_regression(y_true, y_pred):
    """
    Calculate the mean diference between the true angles
    and the predicted angles. Each angle is represented
    as a float number between 0 and 1.
    """
    return K.mean(angle_difference(y_true * 360, y_pred * 360))


def binarize_images(x):
    """
    Convert images to range 0-1 and binarize them by making
    0 the values below 0.1 and 1 the values above 0.1.
    """
    x /= 255
    x[x >= 0.1] = 1
    x[x < 0.1] = 0
    return x


def rotate(image, angle):
    """
    Rotates an OpenCV 2 / NumPy image about it's centre by the given angle
    (in degrees). The returned image will be large enough to hold the entire
    new image, with a black background
    Source: http://stackoverflow.com/questions/16702966/
 ↪rotate-image-and-crop-out-black-borders
    """
    # Get the image size
    # No that's not an error - NumPy stores image matricies backwards
    image_size = (image.shape[1], image.shape[0])
    image_center = tuple(np.array(image_size) / 2)

    # Convert the OpenCV 3x2 rotation matrix to 3x3
    rot_mat = np.vstack(
        [cv2.getRotationMatrix2D(image_center, angle, 1.0), [0, 0, 1]]
    )

    rot_mat_notranslate = np.matrix(rot_mat[0:2, 0:2])

    # Shorthand for below calcs
    image_w2 = image_size[0] * 0.5
    image_h2 = image_size[1] * 0.5

    # Obtain the rotated coordinates of the image corners
```

```
    rotated_coords = [
        (np.array([-image_w2,  image_h2]) * rot_mat_notranslate).A[0],
        (np.array([ image_w2,  image_h2]) * rot_mat_notranslate).A[0],
        (np.array([-image_w2, -image_h2]) * rot_mat_notranslate).A[0],
        (np.array([ image_w2, -image_h2]) * rot_mat_notranslate).A[0]
    ]

    # Find the size of the new image
    x_coords = [pt[0] for pt in rotated_coords]
    x_pos = [x for x in x_coords if x > 0]
    x_neg = [x for x in x_coords if x < 0]

    y_coords = [pt[1] for pt in rotated_coords]
    y_pos = [y for y in y_coords if y > 0]
    y_neg = [y for y in y_coords if y < 0]

    right_bound = max(x_pos)
    left_bound = min(x_neg)
    top_bound = max(y_pos)
    bot_bound = min(y_neg)

    new_w = int(abs(right_bound - left_bound))
    new_h = int(abs(top_bound - bot_bound))

    # We require a translation matrix to keep the image centred
    trans_mat = np.matrix([
        [1, 0, int(new_w * 0.5 - image_w2)],
        [0, 1, int(new_h * 0.5 - image_h2)],
        [0, 0, 1]
    ])

    # Compute the tranform for the combined rotation and translation
    affine_mat = (np.matrix(trans_mat) * np.matrix(rot_mat))[0:2, :]

    # Apply the transform
    result = cv2.warpAffine(
        image,
        affine_mat,
        (new_w, new_h),
        flags=cv2.INTER_LINEAR
    )

    return result


def largest_rotated_rect(w, h, angle):
    """
```

```
    Given a rectangle of size wxh that has been rotated by 'angle' (in
    radians), computes the width and height of the largest possible
    axis-aligned rectangle within the rotated rectangle.
    Original JS code by 'Andri' and Magnus Hoff from Stack Overflow
    Converted to Python by Aaron Snoswell
    Source: http://stackoverflow.com/questions/16702966/
↪rotate-image-and-crop-out-black-borders
    """

    quadrant = int(math.floor(angle / (math.pi / 2))) & 3
    sign_alpha = angle if ((quadrant & 1) == 0) else math.pi - angle
    alpha = (sign_alpha % math.pi + math.pi) % math.pi

    bb_w = w * math.cos(alpha) + h * math.sin(alpha)
    bb_h = w * math.sin(alpha) + h * math.cos(alpha)

    gamma = math.atan2(bb_w, bb_w) if (w < h) else math.atan2(bb_w, bb_w)

    delta = math.pi - alpha - gamma

    length = h if (w < h) else w

    d = length * math.cos(alpha)
    a = d * math.sin(alpha) / math.sin(delta)

    y = a * math.cos(gamma)
    x = y * math.tan(gamma)

    return (
        bb_w - 2 * x,
        bb_h - 2 * y
    )


def crop_around_center(image, width, height):
    """
    Given a NumPy / OpenCV 2 image, crops it to the given width and height,
    around it's centre point
    Source: http://stackoverflow.com/questions/16702966/
↪rotate-image-and-crop-out-black-borders
    """

    image_size = (image.shape[1], image.shape[0])
    image_center = (int(image_size[0] * 0.5), int(image_size[1] * 0.5))

    if(width > image_size[0]):
        width = image_size[0]
```

```
    if(height > image_size[1]):
        height = image_size[1]

    x1 = int(image_center[0] - width * 0.5)
    x2 = int(image_center[0] + width * 0.5)
    y1 = int(image_center[1] - height * 0.5)
    y2 = int(image_center[1] + height * 0.5)

    return image[y1:y2, x1:x2]


def crop_largest_rectangle(image, angle, height, width):
    """
    Crop around the center the largest possible rectangle
    found with largest_rotated_rect.
    """
    return crop_around_center(
        image,
        *largest_rotated_rect(
            width,
            height,
            math.radians(angle)
        )
    )


def generate_rotated_image(image, angle, size=None, crop_center=False,
                           crop_largest_rect=False):
    """
    Generate a valid rotated image for the RotNetDataGenerator. If the
    image is rectangular, the crop_center option should be used to make
    it square. To crop out the black borders after rotation, use the
    crop_largest_rect option. To resize the final image, use the size
    option.
    """
    height, width = image.shape[:2]
    if crop_center:
        if width < height:
            height = width
        else:
            width = height

    image = rotate(image, angle)

    if crop_largest_rect:
        image = crop_largest_rectangle(image, angle, height, width)
```

```
    if size:
        image = cv2.resize(image, size)

    return image


class RotNetDataGenerator(Iterator):
    """
    Given a NumPy array of images or a list of image paths,
    generate batches of rotated images and rotation angles on-the-fly.
    """

    def __init__(self, input, input_shape=None, color_mode='rgb', batch_size=64,
                 one_hot=True, preprocess_func=None, rotate=True,␣
 ↪crop_center=False,
                 crop_largest_rect=False, shuffle=False, seed=None):

        self.images = None
        self.filenames = None
        self.input_shape = input_shape
        self.color_mode = color_mode
        self.batch_size = batch_size
        self.one_hot = one_hot
        self.preprocess_func = preprocess_func
        self.rotate = rotate
        self.crop_center = crop_center
        self.crop_largest_rect = crop_largest_rect
        self.shuffle = shuffle

        if self.color_mode not in {'rgb', 'grayscale'}:
            raise ValueError('Invalid color mode:', self.color_mode,
                             '; expected "rgb" or "grayscale".')

        # check whether the input is a NumPy array or a list of paths
        if isinstance(input, (np.ndarray)):
            self.images = input
            N = self.images.shape[0]
            if not self.input_shape:
                self.input_shape = self.images.shape[1:]
                # add dimension if the images are greyscale
                if len(self.input_shape) == 2:
                    self.input_shape = self.input_shape + (1,)
        else:
            self.filenames = input
            N = len(self.filenames)
```

```
        super(RotNetDataGenerator, self).__init__(N, batch_size, shuffle, seed)

    def _get_batches_of_transformed_samples(self, index_array):
        # create array to hold the images
        batch_x = np.zeros((len(index_array),) + self.input_shape,␣
↪dtype='float32')
        # create array to hold the labels
        batch_y = np.zeros(len(index_array), dtype='float32')

        # iterate through the current batch
        for i, j in enumerate(index_array):
            if self.filenames is None:
                image = self.images[j]
            else:
                is_color = int(self.color_mode == 'rgb')
                image = cv2.imread(self.filenames[j], is_color)
                if is_color:
                    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

            if self.rotate:
                # get a random angle
                rotation_angle = np.random.randint(360)
            else:
                rotation_angle = 0

            # generate the rotated image
            rotated_image = generate_rotated_image(
                image,
                rotation_angle,
                size=self.input_shape[:2],
                crop_center=self.crop_center,
                crop_largest_rect=self.crop_largest_rect
            )

            # add dimension to account for the channels if the image is␣
↪greyscale
            if rotated_image.ndim == 2:
                rotated_image = np.expand_dims(rotated_image, axis=2)

            # store the image and label in their corresponding batches
            batch_x[i] = rotated_image
            batch_y[i] = rotation_angle

        if self.one_hot:
            # convert the numerical labels to binary labels
            batch_y = to_categorical(batch_y, 360)
        else:
```

```
            batch_y /= 360

        # preprocess input images
        if self.preprocess_func:
            batch_x = self.preprocess_func(batch_x)

        return batch_x, batch_y

    def next(self):
        with self.lock:
            # get input data index and size of the current batch
            index_array = next(self.index_generator)
        # create array to hold the images
        return self._get_batches_of_transformed_samples(index_array)


def display_examples(model, input, num_images=5, size=None, crop_center=False,
                     crop_largest_rect=False, preprocess_func=None,␣
↪save_path=None):
    """
    Given a model that predicts the rotation angle of an image,
    and a NumPy array of images or a list of image paths, display
    the specified number of example images in three columns:
    Original, Rotated and Corrected.
    """

    if isinstance(input, (np.ndarray)):
        images = input
        N, h, w = images.shape[:3]
        if not size:
            size = (h, w)
        indexes = np.random.choice(N, num_images)
        images = images[indexes, ...]
    else:
        images = []
        filenames = input
        N = len(filenames)
        indexes = np.random.choice(N, num_images)
        for i in indexes:
            image = cv2.imread(filenames[i])
            image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
            images.append(image)
        images = np.asarray(images)

    x = []
    y = []
    for image in images:
```

```
        rotation_angle = np.random.randint(360)
        rotated_image = generate_rotated_image(
            image,
            rotation_angle,
            size=size,
            crop_center=crop_center,
            crop_largest_rect=crop_largest_rect
        )
        x.append(rotated_image)
        y.append(rotation_angle)

    x = np.asarray(x, dtype='float32')
    y = np.asarray(y, dtype='float32')

    if x.ndim == 3:
        x = np.expand_dims(x, axis=3)

    y = to_categorical(y, 360)

    x_rot = np.copy(x)

    if preprocess_func:
        x = preprocess_func(x)

    y = np.argmax(y, axis=1)
    y_pred = np.argmax(model.predict(x), axis=1)

    plt.figure(figsize=(10.0, 2 * num_images))

    title_fontdict = {
        'fontsize': 14,
        'fontweight': 'bold'
    }

    fig_number = 0
    for rotated_image, true_angle, predicted_angle in zip(x_rot, y, y_pred):
        original_image = rotate(rotated_image, -true_angle)
        if crop_largest_rect:
            original_image = crop_largest_rectangle(original_image,␣
↪-true_angle, *size)

        corrected_image = rotate(rotated_image, -predicted_angle)
        if crop_largest_rect:
            corrected_image = crop_largest_rectangle(corrected_image,␣
↪-predicted_angle, *size)

        if x.shape[3] == 1:
```

```
            options = {'cmap': 'gray'}
        else:
            options = {}

        fig_number += 1
        ax = plt.subplot(num_images, 3, fig_number)
        if fig_number == 1:
            plt.title('Original\n', fontdict=title_fontdict)
        plt.imshow(np.squeeze(original_image).astype('uint8'), **options)
        plt.axis('off')

        fig_number += 1
        ax = plt.subplot(num_images, 3, fig_number)
        if fig_number == 2:
            plt.title('Rotated\n', fontdict=title_fontdict)
        ax.text(
            0.5, 1.03, 'Angle: {0}'.format(true_angle),
            horizontalalignment='center',
            transform=ax.transAxes,
            fontsize=11
        )
        plt.imshow(np.squeeze(rotated_image).astype('uint8'), **options)
        plt.axis('off')

        fig_number += 1
        ax = plt.subplot(num_images, 3, fig_number)
        corrected_angle = angle_difference(predicted_angle, true_angle)
        if fig_number == 3:
            plt.title('Corrected\n', fontdict=title_fontdict)
        ax.text(
            0.5, 1.03, 'Angle: {0}'.format(corrected_angle),
            horizontalalignment='center',
            transform=ax.transAxes,
            fontsize=11
        )
        plt.imshow(np.squeeze(corrected_image).astype('uint8'), **options)
        plt.axis('off')

    plt.tight_layout(pad=0.4, w_pad=0.5, h_pad=1.0)

    if save_path:
        plt.savefig(save_path)
```

4 4- Self-supervised training of unlabeled data with Rotation Network for rotation classification

[ ]:
```
###### RotNet MODEL

model_name = 'rotnet_mnist'

# number of convolutional filters to use
nb_filters = 64
# size of pooling area for max pooling
pool_size = (2, 2)
# convolution kernel size
kernel_size = (3, 3)
# number of classes
nb_classes = 360

nb_train_samples, img_rows, img_cols = x_train_unlab.shape
img_channels = 1
input_shape = (img_rows, img_cols, img_channels)
nb_test_samples = x_test.shape[0]

input = Input(shape=(img_rows, img_cols, img_channels))
model = Sequential()
model.add(layers.Conv2D(20,
                  [3, 3],
                  input_shape=[28, 28, 1],
                  activation='relu',
                  name='conv_1'))
model.add(layers.MaxPool2D())
model.add(layers.Conv2D(50, [3, 3], activation='relu', name='conv_2'))
model.add(layers.MaxPool2D())
model.add(layers.Permute((2, 1, 3)))
model.add(layers.Flatten())
model.add(layers.Dense(500, activation='relu', name='dense_1'))
model.add(layers.Dense(360, activation='softmax', name='dense_2'))

model.summary()
```

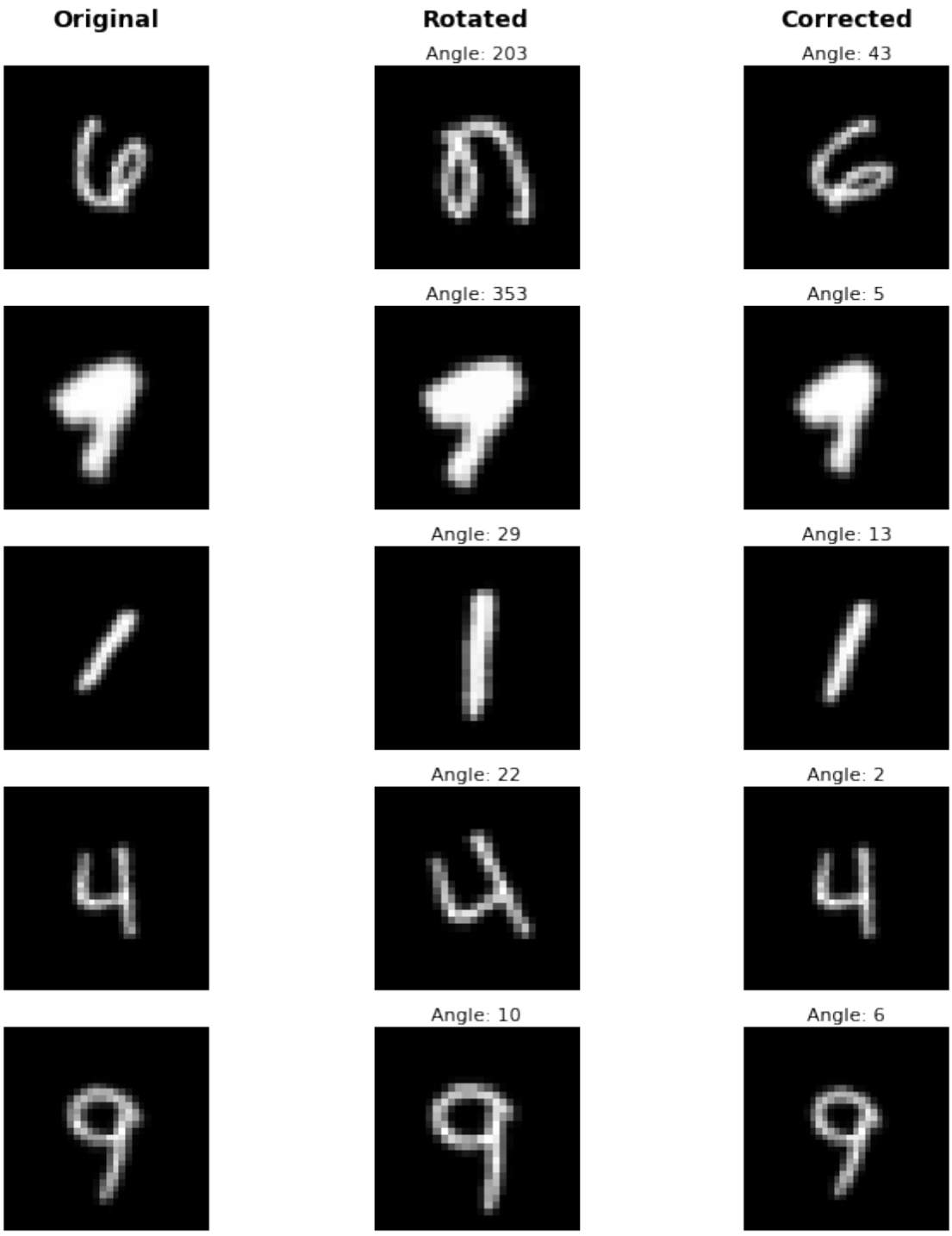
[ ]:
```
# model compilation
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=[angle_error])
```

[ ]:
```
model_name = 'rotnet_mnist'

# number of convolutional filters to use
```

```
nb_filters = 64
# size of pooling area for max pooling
pool_size = (2, 2)
# convolution kernel size
kernel_size = (3, 3)
# number of classes
nb_classes = 360
# training parameters
batch_size = 128
nb_epoch = 50
nb_train_samples, img_rows, img_cols = x_train_unlab.shape
img_channels = 1
input_shape = (img_rows, img_cols, img_channels)
nb_test_samples = x_test.shape[0]
output_folder = 'models'
if not os.path.exists(output_folder):
    os.makedirs(output_folder)

# callbacks
checkpointer = ModelCheckpoint(
    filepath=os.path.join(output_folder, model_name + '.hdf5'),
    save_best_only=True
)
early_stopping = EarlyStopping(patience=2)
tensorboard = TensorBoard()

# training loop
history= model.fit_generator(
    RotNetDataGenerator(
        x_train_unlab,
        batch_size=batch_size,
        preprocess_func=binarize_images,
        shuffle=True
    ),
    steps_per_epoch=nb_train_samples / batch_size,
    epochs=50,
    validation_data=RotNetDataGenerator(
        x_test,
        batch_size=batch_size,
        preprocess_func=binarize_images
    ),
    validation_steps=nb_test_samples / batch_size,
    verbose=1,
    callbacks=[checkpointer, early_stopping, tensorboard]
)
```

[ ]:
```
#Test results
batch_size = 128
out = model.evaluate_generator(
    RotNetDataGenerator(
        x_test,
        batch_size=batch_size,
        preprocess_func=binarize_images,
        shuffle=True
    ),
    steps=len(y_test) / batch_size
)

print('Test loss:', out[0])
print('Test angle error:', out[1])
```

[ ]:
```
#Displaying outputs of RotNet
num_images = 5

display_examples(
    model,
    x_test,
    num_images=num_images,
    preprocess_func=binarize_images,
)
```


Original
Rotated
Corrected
Angle: 203
Angle: 43
Angle: 353
Angle: 5
Angle: 29
Angle: 13
Angle: 22
Angle: 2
Angle: 10
Angle: 6

5 5- Supervised finetuning of labeled data using pre-trained RotNet for Mnist classification

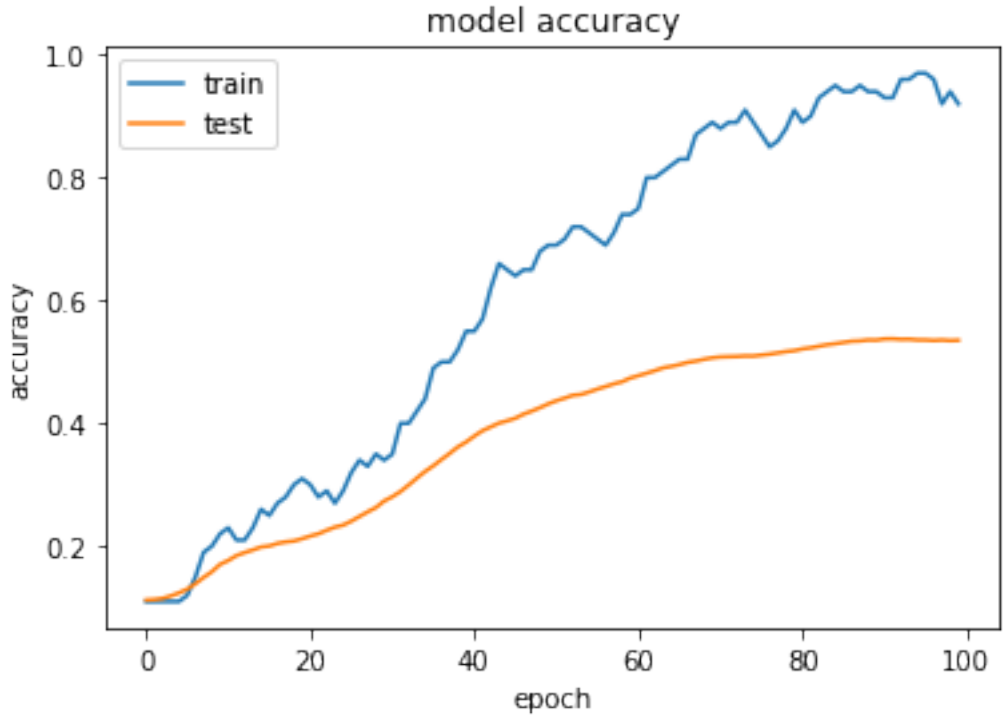
[ ]:
```
#Freezing the convolutional base of our pre-trained Rotnet
model.trainable = False
#Replacing last layer
num_targets = 10
base_output = model.layers[-2]. output
new_output = tf.keras.layers.Dense(activation="softmax",␣
↪units=num_targets)(base_output)
new_model = tf.keras.models.Model(inputs=model.inputs, outputs=new_output)
new_model.summary()
```

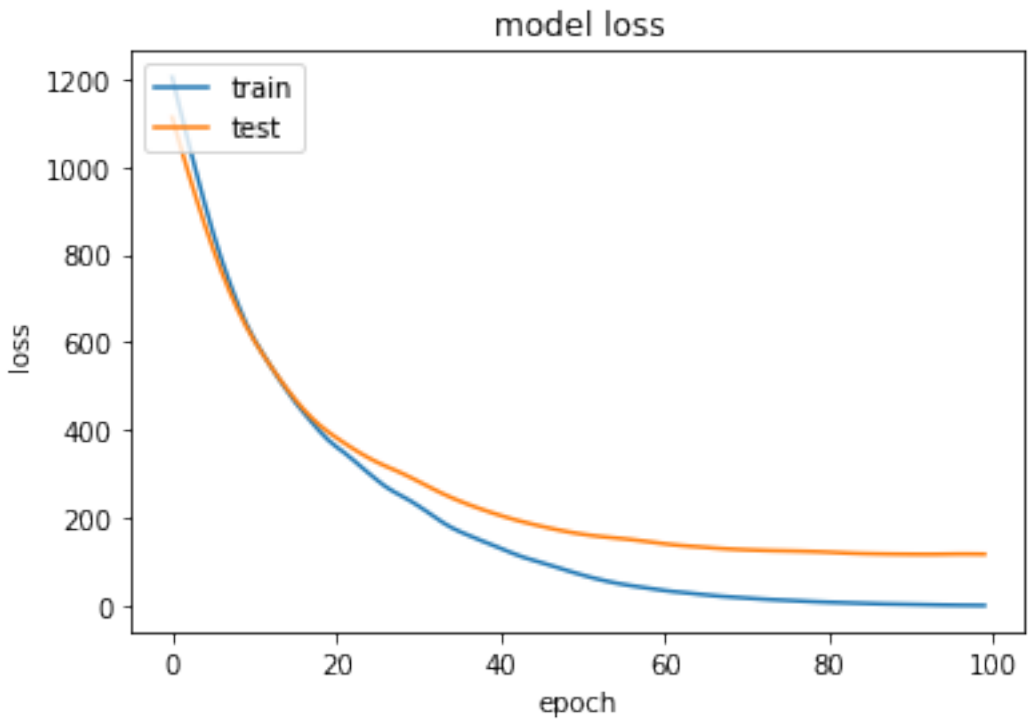
[ ]:
```
# reshaping data to tensors since we're using CNN
x_train = x_train.reshape((60000, 28, 28, 1)).astype('float32')
y_train = to_categorical(y_train)

x_train_lab = x_train_lab.reshape((100, 28, 28, 1))
y_train_lab = to_categorical(y_train_lab)

x_test = x_test.reshape((10000, 28, 28, 1))
y_test = to_categorical(y_test)

print("x_train: ", len(x_train_lab))
print("y_train: ", len(y_train_lab))
print("x_test: ", len(x_test))
print("y_test: ", len(y_test))
```

[ ]:
```
new_model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
history2 = new_model.fit(x_train_lab, y_train_lab, epochs=100,␣
↪batch_size=120,validation_data=(x_test, y_test))
```

[ ]:
```
# summarize history for accuracy
plt.plot(history2.history['accuracy'])
plt.plot(history2.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

# summarize history for loss
plt.plot(history2.history['loss'])
plt.plot(history2.history['val_loss'])
```

```
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```





6 6- Supervised training of labeled data for Mnist classification

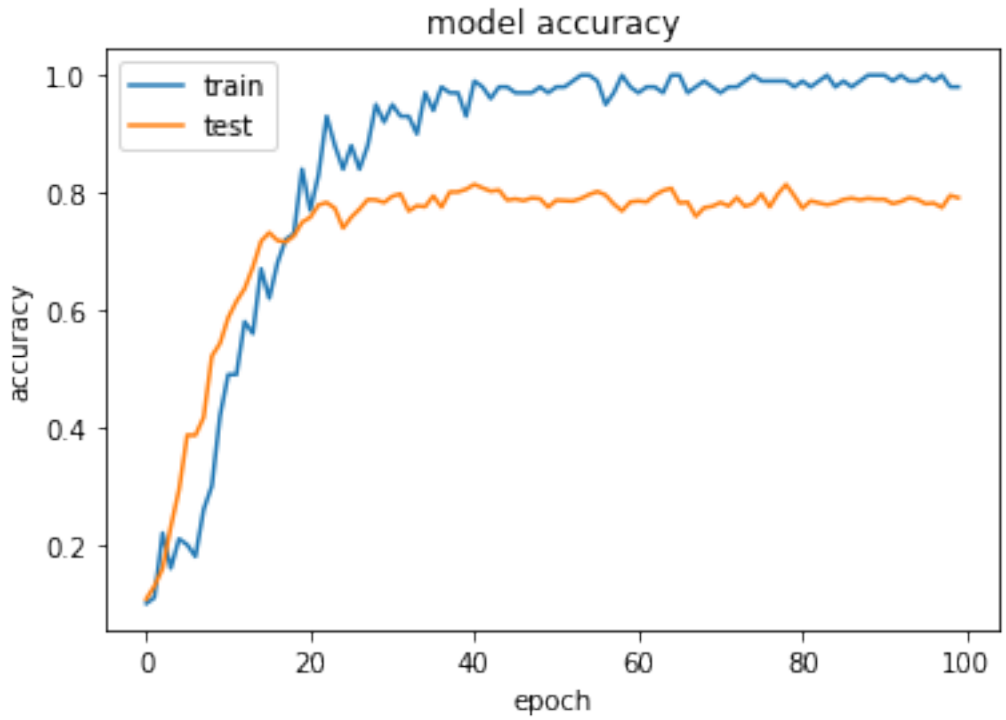
[ ]:
```
#Supervised Model of the 100 labels

model_cnn = Sequential()
# convolution layer with 3x3 filter
model_cnn.add(Conv2D(32, (3,3), activation="relu", input_shape=(28, 28, 1))) #␣
 ↪maxpooling layer
model_cnn.add(MaxPooling2D((2,2)))
# convolution layer with 3x3 filter
model_cnn.add(Conv2D(64, (3,3), activation="relu"))
# maxpooling layer
model_cnn.add(MaxPooling2D((2,2)))
# convolution layer with 3x3 filter
model_cnn.add(Conv2D(64, (3,3), activation="relu"))
# flatten the layers
model_cnn.add(Flatten())
# dense layer with 256 units and relu activation
model_cnn.add(Dense(256, activation= 'relu'))
# dropout layer to keep 60% of units
model_cnn.add(Dropout(0.4))
```

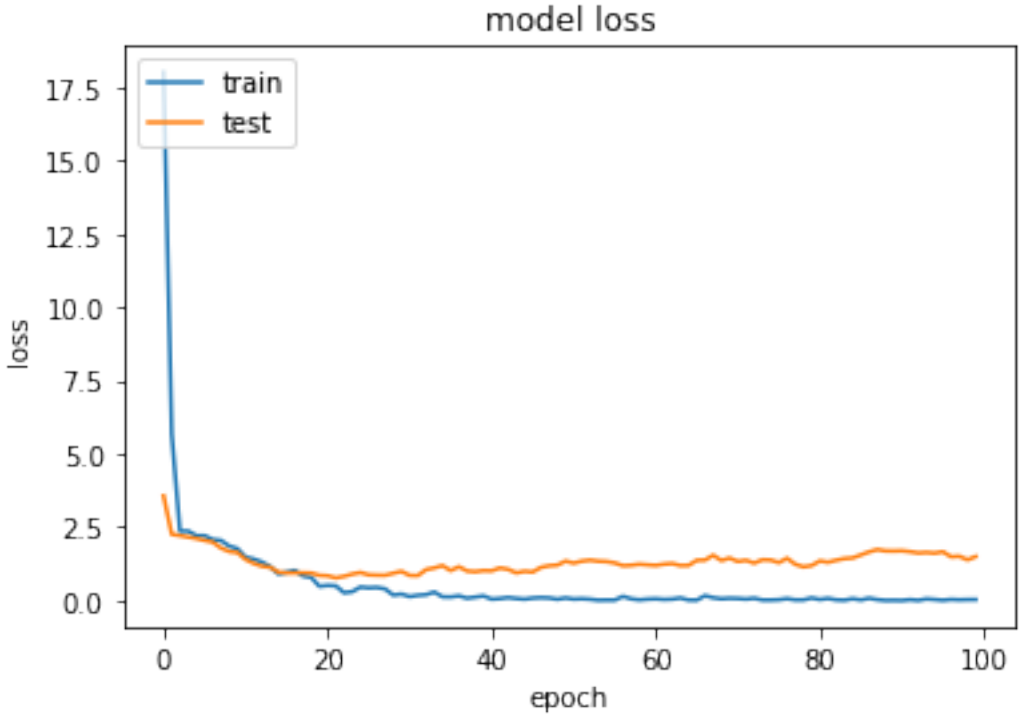
```
# dense layer with 500 units and relu activation
model_cnn.add(Dense(500, activation= 'relu'))
# dropout layer to keep 40% of units
model_cnn.add(Dropout(0.6))
# dense layer with 64 units and relu activation
model_cnn.add(Dense(64, activation= 'relu'))
# dropout layer to keep 80% of units
model_cnn.add(Dropout(0.2))
# Output layer with 10 units and softmax activation
model_cnn.add(Dense(10, activation= 'softmax'))
# get summary of the model
model_cnn.summary()
```

[ ]:
```
# compile model with adam optimizer and categorical loss
model_cnn.compile(optimizer='adam', loss='categorical_crossentropy',␣
↪metrics=['accuracy'])
```

[ ]:
```
# train the model on  training set
history = model_cnn.fit(x_train_lab, y_train_lab, epochs=100,␣
↪batch_size=10,validation_data=(x_test, y_test))
```

[ ]:
```
# summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```


model accuracy
train
test
accuracy
epoch
1.0
0.8
0.6
0.4
0.2
0
20
40
60
80
100


model loss
train
test
loss
epoch
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0
20
40
60
80
100