Student: Bahram Khodabandehlouee

Cs 614 - 900

Assignment: 1 (Computer Vison)

Contents

Pitch

Recommending preferred contents to TV enthusiasts for media companies such as Netflix, YouTube TV or Amazon Prime is crucial. There is a need to automatically tag metadata for content on those platforms to help with data analysis and organization across their users and personalize recommendations. The goal therefore is to train a neural network on a convolutional neural network (CNN) architecture to be able to predict celebrity given an image as an input.

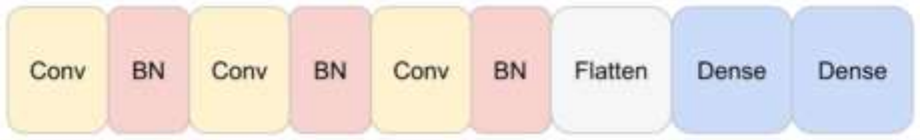
Data Source

The dataset is Pins and is hosted on Kaggle, lick here. The dataset contains about 17534 images of around 105 celebrities. Images are in different sizes and colored. I have written my own code to convert images of 10 celebrities (1006 images in 10 classes) to 35x35 grayscale images. To see the code of the used approach, please click here. In this link, python file 00.py is only for preprocessing and converting 10 classes. You do NOT have to run this code as the results are already saved.

Google drive link:
https://drive.google.com/drive/folders/1sDYLxJmuyh96Byq3b7sxMe2HpYKMfZAI?usp=sharing

Model and data justification

I started with a simple CNN network where I didn't get a high accuracy even after changing hyper parameters. My last and main model is inspired by VGG network, where I used multiple Convolutional and Batch-Normalization layers followed by Flatten and two Dense layers. I included some data augmentation such as using batch_size=32 and took advantage of ImageDataGenerator () method to consider rotation, shifting and other image structures.



- train_acc = 91.62%
- test_acc = 71.52%
- loss = 1.41

VGG network:

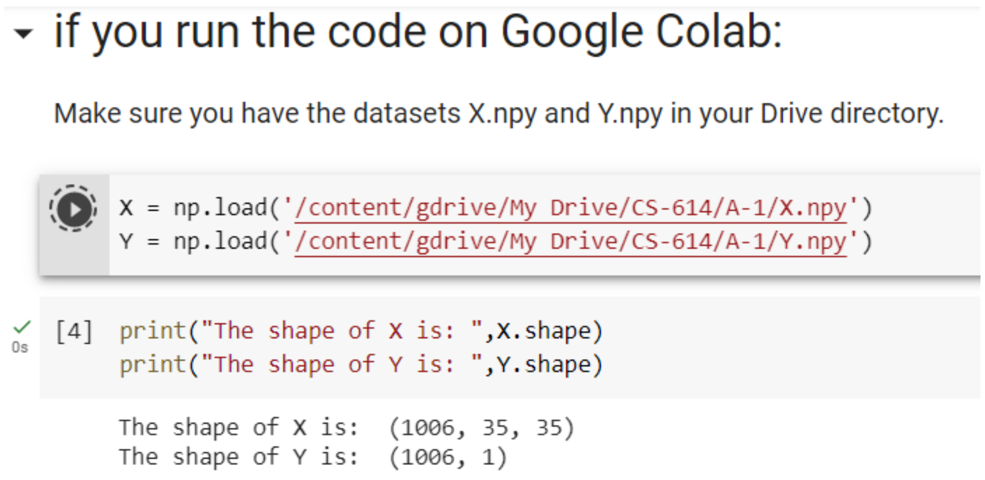
https://viso.ai/deep-learning/vgg-very-deep-convolutional-networks/

My code on google Colab:
https://drive.google.com/drive/folders/1sDYLxJmuyh96Byq3b7sxMe2HpYKMfZAI?usp=sharing

Commented examples

Here are few screen shots of my code:

Loading the data in **Google Colab**:



if you run the code on Google Colab:

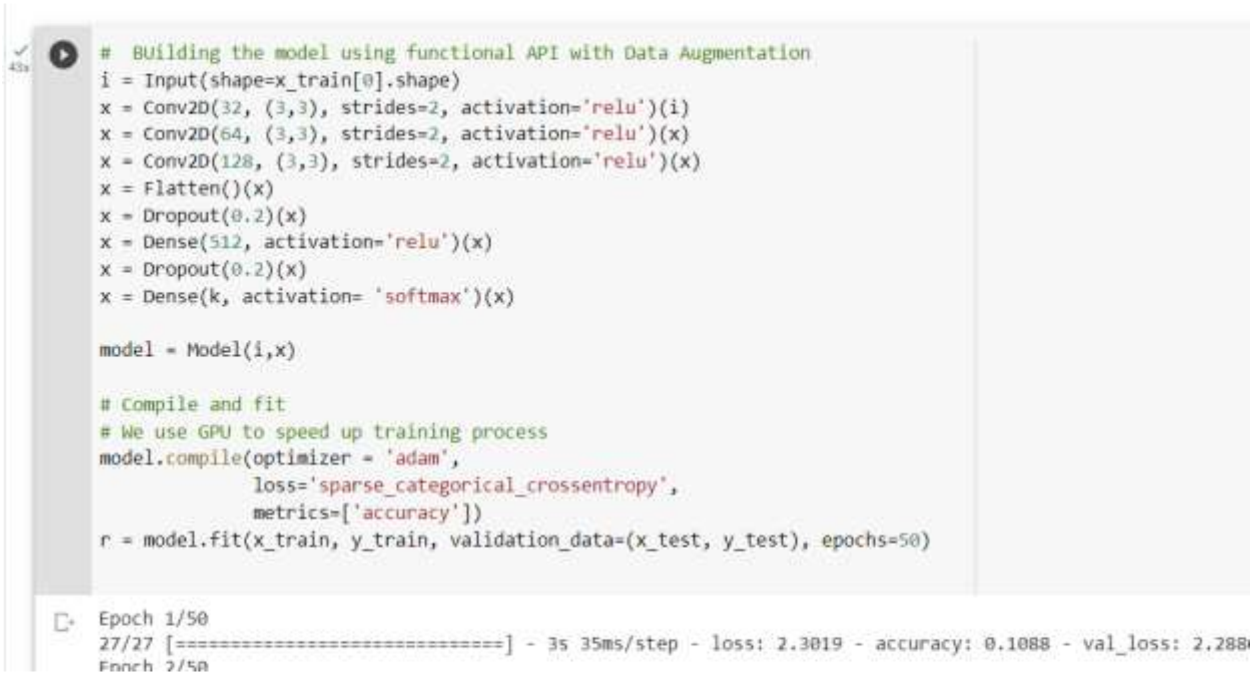
Make sure you have the datasets X.npy and Y.npy in your Drive directory.

```
X = np.load('/content/gdrive/My Drive/CS-614/A-1/X.npy')
Y = np.load('/content/gdrive/My Drive/CS-614/A-1/Y.npy')
```

```
[4] print("The shape of X is: ",X.shape)
    print("The shape of Y is: ",Y.shape)
```

```
The shape of X is:  (1006, 35, 35)
The shape of Y is:  (1006, 1)
```

My first and Simplest CNN model:



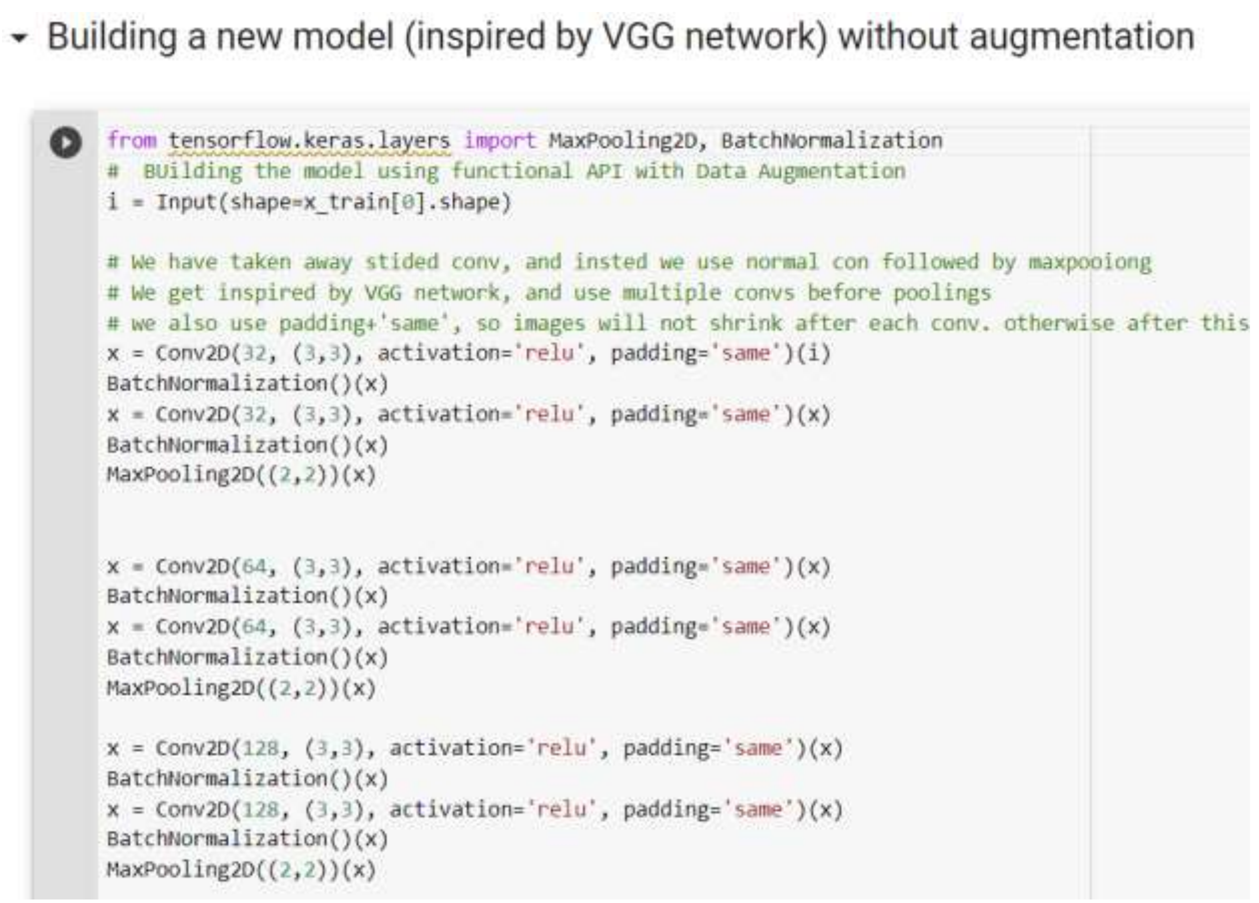
```
#  Building the model using functional API with Data Augmentation
i = Input(shape=x_train[0].shape)
x = Conv2D(32, (3,3), strides=2, activation='relu')(i)
x = Conv2D(64, (3,3), strides=2, activation='relu')(x)
x = Conv2D(128, (3,3), strides=2, activation='relu')(x)
x = Flatten()(x)
x = Dropout(0.2)(x)
x = Dense(512, activation='relu')(x)
x = Dropout(0.2)(x)
x = Dense(k, activation= 'softmax')(x)

model = Model(i,x)

# Compile and fit
# We use GPU to speed up training process
model.compile(optimizer = 'adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
r = model.fit(x_train, y_train, validation_data=(x_test, y_test), epochs=50)
```

```
Epoch 1/50
27/27 [==============================] - 3s 35ms/step - loss: 2.3019 - accuracy: 0.1088 - val_loss: 2.288
Epoch 2/50
```

And my last model; inspired by VGG network:



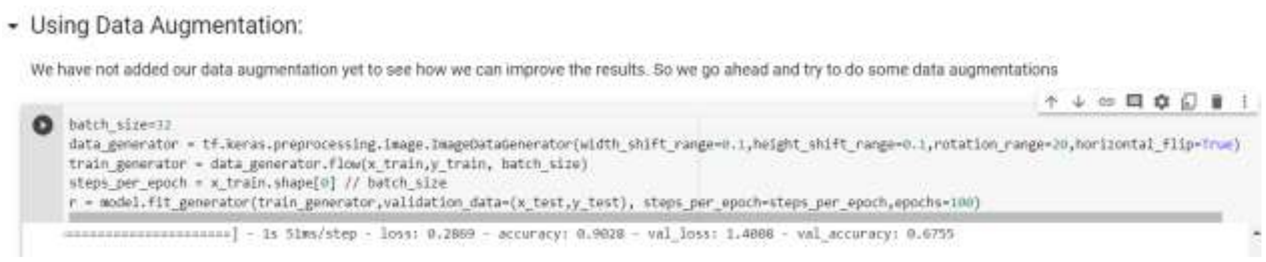
Building a new model (inspired by VGG network) without augmentation

```
from tensorflow.keras.layers import MaxPooling2D, BatchNormalization
#  BUilding the model using functional API with Data Augmentation
i = Input(shape=x_train[0].shape)

# We have taken away stided conv, and insted we use normal con followed by maxpooiong
# We get inspired by VGG network, and use multiple convs before poolings
# we also use padding+'same', so images will not shrink after each conv. otherwise after this
x = Conv2D(32, (3,3), activation='relu', padding='same')(i)
BatchNormalization()(x)
x = Conv2D(32, (3,3), activation='relu', padding='same')(x)
BatchNormalization()(x)
MaxPooling2D((2,2))(x)


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

x = Conv2D(128, (3,3), activation='relu', padding='same')(x)
BatchNormalization()(x)
x = Conv2D(128, (3,3), activation='relu', padding='same')(x)
BatchNormalization()(x)
MaxPooling2D((2,2))(x)
```



Using Data Augmentation:

We have not added our data augmentation yet to see how we can improve the results. So we go ahead and try to do some data augmentations

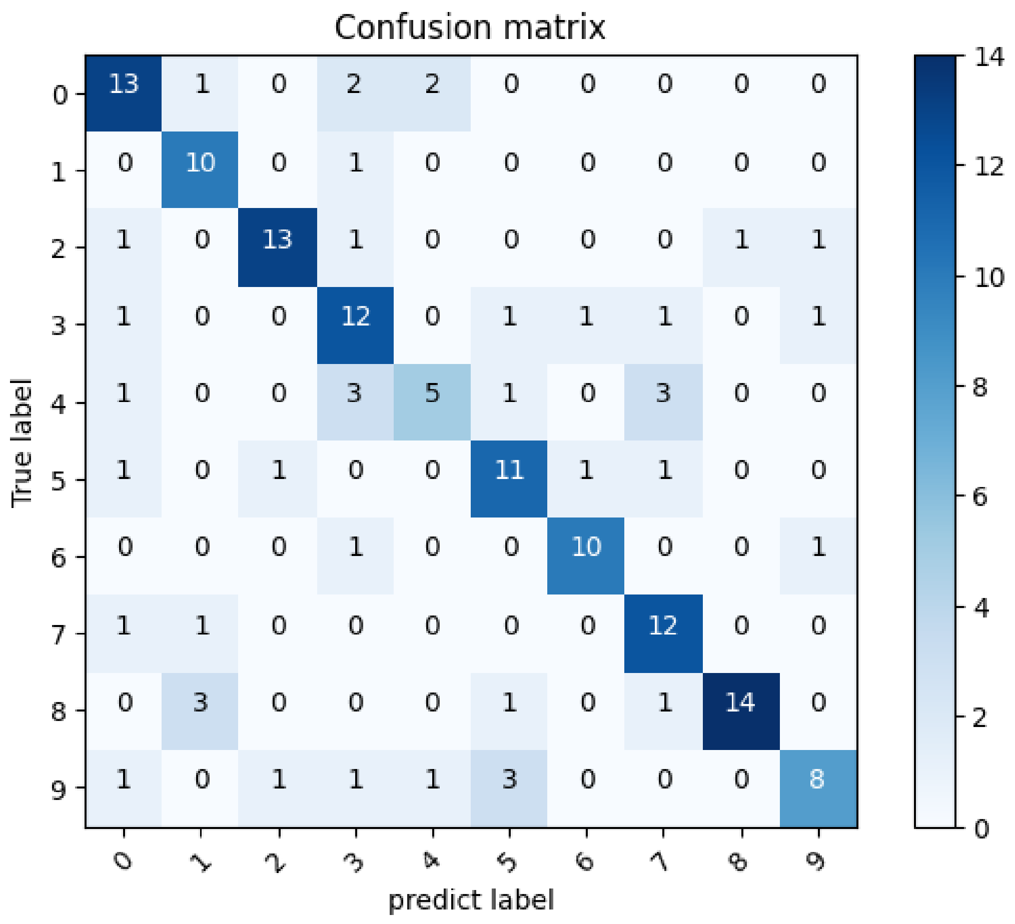
```
batch_size=32
data_generator = tf.keras.preprocessing.image.ImageDataGenerator(width_shift_range=0.1,height_shift_range=0.1,rotation_range=20,horizontal_flip=True)
train_generator = data_generator.flow(x_train,y_train, batch_size)
steps_per_epoch = x_train.shape[0] // batch_size
r = model.fit_generator(train_generator,validation_data=(x_test,y_test), steps_per_epoch=steps_per_epoch,epochs=100)
```

```
=====================] - 1s 51ms/step - loss: 0.2869 - accuracy: 0.9028 - val_loss: 1.4088 - val_accuracy: 0.6755
```

Results got much better,

- train_acc = 91.62%
- test_acc = 71.52%
- loss = 1.41

Testing



	0	1	2	3	4	5	6	7	8	9
0	13	1	0	2	2	0	0	0	0	0
1	0	10	0	1	0	0	0	0	0	0
2	1	0	13	1	0	0	0	0	1	1
3	1	0	0	12	0	1	1	1	0	1
4	1	0	0	3	5	1	0	3	0	0
5	1	0	1	0	0	11	1	1	0	0
6	0	0	0	1	0	0	10	0	0	1
7	1	1	0	0	0	0	0	12	0	0
8	0	3	0	0	0	1	0	1	14	0
9	1	0	1	1	1	3	0	0	0	8

Based on the Confusion Matrix we have promising Results. Some of the wrong predictions are as follow:

- Class 1 has been predicted as class 8 for 3 times
- Class 7 has been predicted for class 4 for 3 times
- Class 5 has been predicted as class 9 for 3 times

Other classes have pretty good numbers.

Code and instructions to run it

Provide a link to the code and any required instructions to run it. Please include some testing examples so we can quickly experience what you experienced with the model.

To see the code, please click here. In this link you have access to the code and data in google colab. To run it in your own machine, please copy

1. code (A-1-Computer vision.ipynb)
2. Data (X.npy and Y.npy)

3. When you run it in your google colab, when it is time to load the data, you should have the data in your own google drive. Also, it will ask you to give permission to connect google drive to google colab.
4. You don’t have to run 00.py as its results are already saved as X.npy and Y.npy

Link to the code:

https://colab.research.google.com/drive/1m7FngITCnWZ5nvNUK42fhj0QMtZYSDmz?usp=sharing