Introduction

“Build a deep learning model in a few minutes? It’ll take hours to train! I don’t even have a good enough machine.” I’ve heard this countless times from aspiring data scientists who shy away from building deep learning models on their own machines.

You don’t need to be working for Google or other big tech firms to work on deep learning datasets! It is entirely possible to build your own neural network from the ground up in a matter of minutes without needing to lease out Google’s servers. Fast.ai’s students designed a model on the Imagenet dataset in 18 minutes – and I will showcase something similar in this article.

Setting Up the Structure of Our Image Data

Our data needs to be in a particular format in order to solve an image classification problem. We will see this in action in a couple of sections but just keep these pointers in mind till we get there.

You should have 2 folders, one for the train set and the other for the test set. In the training set, you will have a .csv file and an image folder:

The .csv file contains the names of all the training images and their corresponding true labels

The image folder has all the training images.

The .csv file in our test set is different from the one present in the training set. This test set .csv file contains the names of all the test images, but they do not have any corresponding labels. Can you guess why? Our model will be trained on the images present in the training set and the label predictions will happen on the testing set images

If your data is not in the format described above, you will need to convert it accordingly (otherwise the predictions will be awry and fairly useless).

Breaking Down the Process of Model Building

Before we deep dive into the Python code, let’s take a moment to understand how an image classification model is typically designed. We can divide this process broadly into 4 stages. Each stage requires a certain amount of time to execute:

Loading and pre-processing Data – 30% time

Defining Model architecture – 10% time

Training the model – 50% time

Estimation of performance – 10% time

Let me explain each of the above steps in a bit more detail. This section is crucial because not every model is built in the first go. You will need to go back after each iteration, fine-tune your steps, and run it again. Having a solid understanding of the underlying concepts will go a long way in accelerating the entire process.

Stage 1: Loading and pre-processing the data

Data is gold as far as deep learning models are concerned. Your image classification model has a far better chance of performing well if you have a good amount of images in the training set. Also, the shape of the data varies according to the architecture/framework that we use.

Hence, the critical data pre-processing step (the eternally important step in any project). I highly recommend going through the 'Basics of Image Processing in Python' to understand more about how pre-processing works with image data.

But we are not quite there yet. In order to see how our model performs on unseen data (and before exposing it to the test set), we need to create a validation set. This is done by partitioning the training set data.

In short, we train the model on the training data and validate it on the validation data. Once we are satisfied with the model's performance on the validation set, we can use it for making predictions on the test data.

Time required for this step: We require around 2-3 minutes for this task.

Stage 2: Defining the model's architecture

This is another crucial step in our deep learning model building process. We have to define how our

model will look and that requires answering questions like:

How many convolutional layers do we want?

What should be the activation function for each layer?

How many hidden units should each layer have?

And many more. These are essentially the hyperparameters of the model which play a MASSIVE part in deciding how good the predictions will be.

How do we decide these values? Excellent question! A good idea is to pick these values based on existing research/studies. Another idea is to keep experimenting with the values until you find the best match but this can be quite a time consuming process.

Time required for this step: It should take around 1 minute to define the architecture of the model.

Stage 3: Training the model

For training the model, we require:

Training images and their corresponding true labels

Validation images and their corresponding true labels (we use these labels only to validate the model and not during the training phase)

We also define the number of epochs in this step. For starters, we will run the model for 10 epochs (you can change the number of epochs later).

Time required for this step: Since training requires the model to learn structures, we need around 5 minutes to go through this step.

And now time to make predictions!

Stage 4: Estimating the model's performance

Finally, we load the test data (images) and go through the pre-processing step here as well. We then predict the classes for these images using the trained model.

Time required for this step: ~ 1 minute.

Setting Up the Problem Statement and Understanding the Data

We will be picking up a really cool challenge to understand image classification. We have to build a model that can classify a given set of images according to the apparel (shirt, trousers, shoes, socks, etc.). It's actually a problem faced by many e-commerce retailers which makes it an even more interesting computer vision problem.

This challenge is called 'Identify the Apparels' and is one of the practice problems we have on our DataHack platform. You will have to register and download the dataset from the above link.

Steps to Build Our Model

Time to fire up your Python skills and get your hands dirty. We are finally at the implementation part of our learning!

Setting up Google Colab

Importing Libraries

Loading and Preprocessing Data – (3 mins)

Creating a validation set

Defining the model structure – (1 min)

Training the model – (5 min)

Making predictions – (1 min)

Let's look at each step in detail.

Step 1: Setting up Google Colab

Since we're importing our data from a Google Drive link, we'll need to add a few lines of code in our Google Colab notebook. Create a new Python 3 notebook and write the following code blocks:

```
!pip install PyDrive
```

This will install PyDrive. Now we will import a few required libraries:

```
import os
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials
```

Next, we will create a drive variable to access Google Drive:

```
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)
```

To download the dataset, we will use the ID of the file uploaded on Google Drive:

```
download = drive.CreateFile({'id': '1BZOv422XJvxFUnGh-0xVeSvgFgqVY45q'})
```

Replace the 'id' in the above code with the ID of your file. Now we will download this file and unzip it:

```
download.GetContentFile('train_LbELtWX.zip')
!unzip train_LbELtWX.zip
```

You have to run these code blocks every time you start your notebook.

Step 2 : Import the libraries we'll need during our model building phase.

```
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras.utils import to_categorical
from keras.preprocessing import image
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from keras.utils import to_categorical
from tqdm import tqdm
```

Step 3: Recall the pre-processing steps we discussed earlier. We'll be using them here after loading the data.

```
train = pd.read_csv('train.csv')
```

Next, we will read all the training images, store them in a list, and finally convert that list into a numpy array.

```
# We have grayscale images, so while loading the images we will keep grayscale=True, if you have RGB images, you should set grayscale as False
train_image = []
for i in tqdm(range(train.shape[0])):
    img = image.load_img('train/'+train['id'][i].astype('str')+'.png', target_size=(28,28,1), grayscale=True)
```

```
    img = image.img_to_array(img)
    img = img/255
    train_image.append(img)
X = np.array(train_image)
```

As it is a multi-class classification problem (10 classes), we will one-hot encode the target variable.

```
y=train['label'].values
y = to_categorical(y)
```

Step 4: Creating a validation set from the training data.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42, test_size=0.2)
```

Step 5: Define the model structure.

We will create a simple architecture with 2 convolutional layers, one dense hidden layer and an output layer.

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),activation='relu',input_shape=(28,28,1)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
```

Next, we will compile the model we've created.

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

Step 6: Training the model.

In this step, we will train the model on the training set images and validate it using, you guessed it, the validation set.

```
model.fit(X_train, y_train, epochs=10, validation_data=(X_test, y_test))
```

Step 7: Making predictions!

We'll initially follow the steps we performed when dealing with the training data. Load the test images and predict their classes using the model.predict_classes() function.

```
download = drive.CreateFile({'id': '1KuyWGFEpj7Fr2DgBsW8qsWvjqEzfoJBY'})

download.GetContentFile('test_ScVgIM0.zip')

!unzip test_ScVgIM0.zip
```

Let's import the test file:

```
test = pd.read_csv('test.csv')
```

Now, we will read and store all the test images:

```
test_image = []

for i in tqdm(range(test.shape[0])):

    img = image.load_img('test/'+test['id'][i].astype('str')+'.png', target_size=(28,28,1), grayscale=True)

    img = image.img_to_array(img)

    img = img/255
```

```
    test_image.append(img)

test = np.array(test_image)

# making predictions

prediction = model.predict_classes(test)
```

We will also create a submission file to upload on the DataHack platform page (to see how our results fare on the leaderboard).

```
download = drive.CreateFile({'id': '1z4QXy7WravpSj-S4Cs9Fk8ZNaX-qh5HF'})

download.GetContentFile('sample_submission_I5njJSF.csv')

# creating submission file

sample = pd.read_csv('sample_submission_I5njJSF.csv')

sample['label'] = prediction

sample.to_csv('sample_cnn.csv', header=True, index=False)
```

Download this sample_cnn.csv file and upload it on the contest page to generate your results and check your ranking on the leaderboard. This will give you a benchmark solution to get you started with any Image Classification problem!

You can try hyperparameter tuning and regularization techniques to improve your model's performance further. I ecnourage you to check out this article to understand this fine-tuning step in much more detail – 'A Comprehensive Tutorial to learn Convolutional Neural Networks from Scratch'.