

Capstone Project Report

On

**Image classification Of MNIST Fashion
dataset**

By

Akpojicheko Eyekpegaha

Table of Contents

● Project Overview	3
● Problem statement	3
● Evaluation metrics	4
● Datasets and inputs	4
● Solution statement	5
● Project design:	6
Loading the dataset	6
Model training	7
Model evaluation, profiling and debugging	8
● Conclusion	13
● Reference	14

Project Overview

The capstone project, is basically how to understand and use perfectly aws sagemaker resources together with pytorch to perform a computer vision task. This project makes use of the famous mnist fashion dataset on image classification model to create an efficient model with a convolution neural network. The benefit of computer vision is to enable computers to see the world or objects as humans do. And with the help of pretrained models, a machine learning framework such as pytorch and AWS sagemaker, this process can be a lot simplified when using a convolutional neural network algorithm. The model created during this project can serve as a useful tool to fashion brands, their customers would get a good experience when using this tool to select a particular fashion item of their choice.

The aim of this project is to master the use of creating image classification models with AWS sagemaker because it is a core part of computer vision and machine learning engineering in general

Problem statement

To achieve my aim, I used the following steps in solving an image classification problem with PyTorch:

1. Load the dataset from MNIST fashion website
2. Creating a validation dataset from train data, creating separate folders for train, valid and test dataset
3. Loading the datasets to s3 and performing job training on a preprocessed dataset with pytorch on AWS sagemaker
4. Performing debugging and profiling
5. Creating inference and prediction

Different pretrained models were tried with different hyperparameter tuning to achieve a good result based on accuracy.

Evaluation metrics

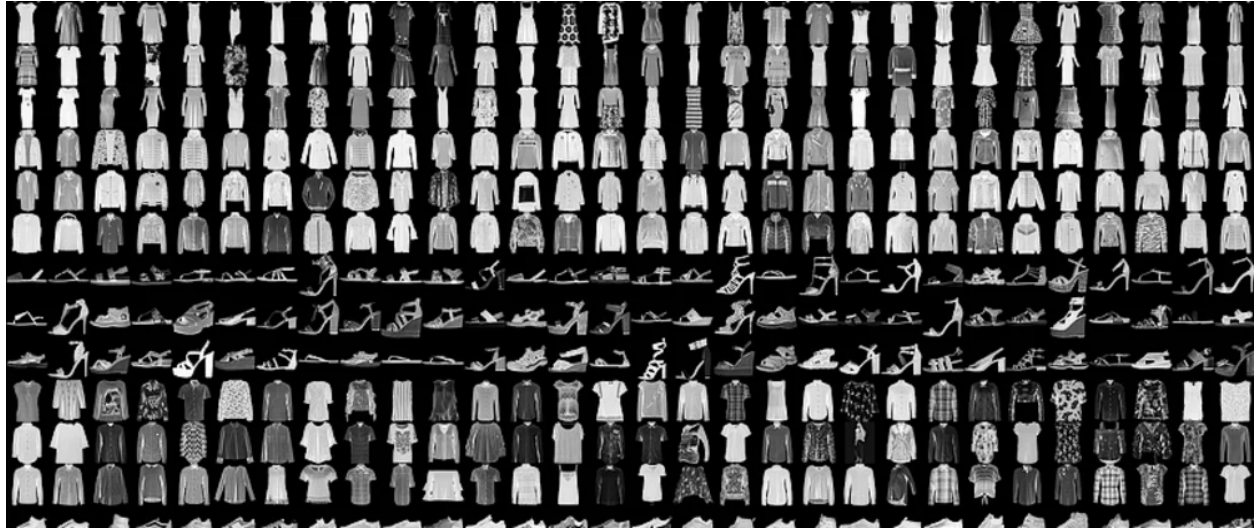
The accuracy of the model, one of the common metrics used to judge a model, is what I made use of for this project. My aim to achieve an high accuracy after completing the project was met. Here is a screenshot of the percentage accuracy of the different classes represented in the dataset.

```
cls_correct = [0 for _ in range(10)]
for k in range(correct_preds.shape[0]):
    n = correct_preds['label'][k]
    if correct_preds['prediction'][k] == n:
        cls_correct[n] += 1
cls_accu = []
for n in range(10):
    cls_accu.append(cls_correct[n]/1000)
    print(f'The accuracy of label {n} is:{cls_accu[n]: .04f} or {cls_accu[n]*100: .0f}%')
```

```
The accuracy of label 0 is: 0.8580 or 86%
The accuracy of label 1 is: 0.9780 or 98%
The accuracy of label 2 is: 0.8610 or 86%
The accuracy of label 3 is: 0.9060 or 91%
The accuracy of label 4 is: 0.8450 or 84%
The accuracy of label 5 is: 0.9590 or 96%
The accuracy of label 6 is: 0.7040 or 70%
The accuracy of label 7 is: 0.9550 or 96%
The accuracy of label 8 is: 0.9780 or 98%
The accuracy of label 9 is: 0.9540 or 95%
```

Datasets and inputs

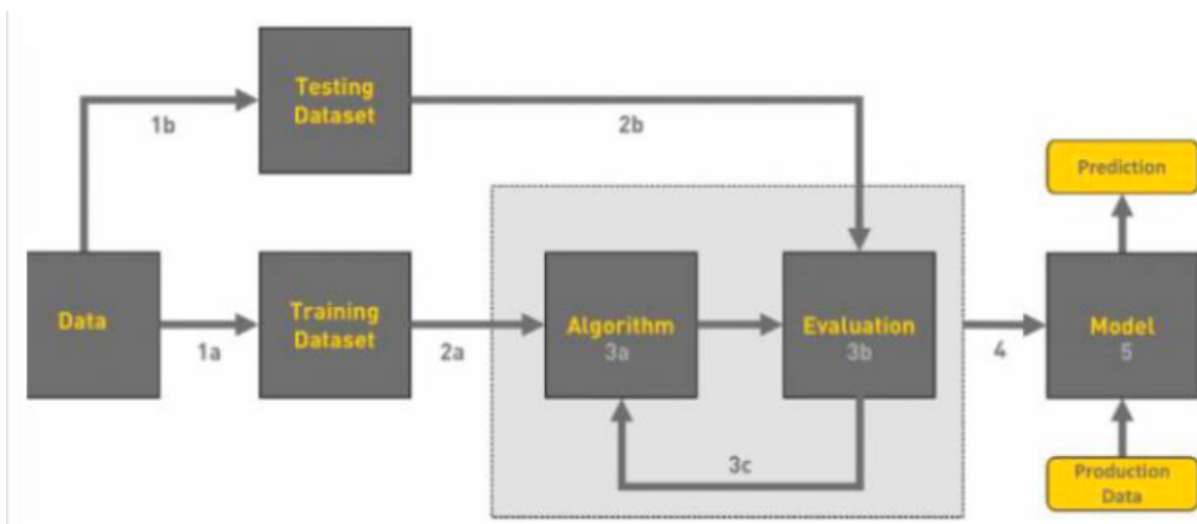
The dataset used for this project is the MNIST dataset, made publicly available. It consists of about 60 images grouped in 10 classes of fashion wears. This dataset contains grey scale images of sizes 28 by 28. For easy usage and processing, I split the data into train, valid and test folders, containing subfolders subsequently. In training the model, I used the ResNet34 pretrained model to perform training and evaluation as well. After preparing the data, the data is moved to AWS S3 where training was carried out with pyTorch.



fashion MNIST dataset

Solution statement

My idea was to create a deep learning model that is able to identify accurately the different classes of the MNIST fashion dataset (basically 10 classes) with a high percentage accuracy by using amazon sagemaker. As a result of hyperparameter tuning, data preprocessing and the use of a good pretrained model, the final architecture and hyperparameters were chosen because they performed the best among some other trials. The different steps taken to achieve this solution is as follows:



Project design

The workflow to approaching the solution is as follows:

Loading the dataset

The dataset was downloaded and saved with matplotlib's `plt.imsave` code as follows:

```
training_data = datasets.FashionMNIST(
    root="fmnist",
    train=True,
    download=True,
    transform=ToTensor()
)

test_data = datasets.FashionMNIST(
    root="fmnist",
    train=False,
    download=True,
    transform=ToTensor()
)
```

```
labels_names = {
    0: "T-Shirt",
    1: "Trouser",
    2: "Pullover",
    3: "Dress",
    4: "Coat",
    5: "Sandal",
    6: "Shirt",
    7: "Sneaker",
    8: "Bag",
    9: "Ankle-Boot",
}

# Create a function to show an image and try to save an image
def create_image(data_image):
    """
    data_image: an image with label from torch dataset
    """
    img, label = data_image
    plt.imshow(img.squeeze(), cmap="gray")
    # We use jpg format.
    filename = labels_names[label]+' .jpg'
    plt.imsave(filename, np.array(img.squeeze()), cmap='gray')
    print('Image file name: ',filename)
```

The total number of images was known to be 60000 and steps to split these among train, test and valid folders were done by creating a function that separates the files in a folder during iteration.

```
print(len(test_df), len(train_df), len(valid_df))

10000 48000 12000
```

Further steps were taken to upload the data to S3

```

]: os.environ["DEFAULT_S3_BUCKET"] = bucket
os.environ['SM_CHANNEL_TRAIN']= s3_data_dir
os.environ['SM_OUTPUT_DATA_DIR']= s3_output_dir
os.environ['SM_MODEL_DIR']= s3_model_dir

print(os.environ['SM_CHANNEL_TRAIN'])

s3://sagemaker-us-east-1-697405440952/images/

]: # Upload data to S3 bucket.
inputs = sagemaker_session.upload_data(path=datapath, bucket=data_bucket, key_prefix=datapath)
print(inputs)

```

Setting up parameters and estimator for job training

The learning rate and batch size as well as the estimator and tuner were set up resulting in fitting:

```

begin = time()
tuner.fit({"train": s3_data_dir})
tunning_time = time() - begin
mins = tunning_time // 60
secs = tunning_time % 60
print(f'Tunning time: {mins: .0f}m {secs: .0f}s')

```

```

.....!
Tunning time: 12m 28s

```

The entry_point for fitting was the fm_hpo.py script which contains code that transforms the data by flipping, normalising and transforming it to tensor during preprocessing. Resizing was not taken into consideration since image size was considerably small and good enough for training. Other function in the script was the train, test, create data loader using arguments such as an optimiser, criterion, model, scheduler, data loader etc.

Model training

The best parameters were used to perform training. With the use of fm_model.py script as entry_point, 20 epochs, resNet 34 was used on the best parameters

Describe the tuning results

Prepare to perform Training on Best Estimator

```
[10]: best_estimator.hyperparameters()

[10]: {'_tuning_objective_metric': 'average test loss',
      'batch_size': '64',
      'learning_rate': '0.000988624968369079',
      'sagemaker_container_log_level': '20',
      'sagemaker_estimator_class_name': 'PyTorch',
      'sagemaker_estimator_module': 'sagemaker.pytorch.estimator',
      'sagemaker_job_name': 'pytorch_fmnist-2022-02-07-08-44-33-486',
      'sagemaker_program': 'fm_hpo.py',
      'sagemaker_region': 'us-east-1',
      'sagemaker_submit_directory': 's3://sagemaker-us-east-1-697405440952/pytorch_fmnist-2022-02-07-08-44-33-486/source/sourcedir.tar.gz'}

[11]: hyperparameters = {"batch_size": int(best_estimator.hyperparameters()['batch_size'].replace("'", '')), \
      "lr": best_estimator.hyperparameters()['learning_rate']}
      hyperparameters

[11]: {'batch_size': 64, 'lr': '0.000988624968369079'}
```

Model evaluation, profiling and debugging.

```
In [21]: #Plot a debugging output.
         plot_tensor(trial, tensor_name)
```

loaded TRAIN data
loaded EVAL data
completed TRAIN plot
completed EVAL plot



The loss function which computes a value that estimates how far away the output is from the target was calculated during this project as cross entropy loss. The graph above simply shows the reduction in cross-entropy output which is the loss function of the model. The profiling and debugging done was to show the training performance. With smdebug, the analysis of the training job was provided and the profiler report was made available with this code.

```
profiler_report_name = [  
    rule["RuleConfigurationName"]  
    for rule in estimator.latest_training_job.rule_job_summary()  
    if "Profiler" in rule["RuleConfigurationName"]  
][0]  
print(profiler_report_name)
```

ProfilerReport

```
! tar czf ProfilerReport.tgz ProfilerReport/
```

After successfully conducting a training job with the best hyperparameter available, I easily called for deployment with `pytorch_model.deploy()`, with the use of `fm_inference.py` script as `entry_point`.

```

import sagemaker
import boto3
from sagemaker.tuner import CategoricalParameter, ContinuousParameter, HyperparameterTuner
from sagemaker.pytorch import PyTorch
from sagemaker import get_execution_role
from sagemaker.debugger import Rule, DebuggerHookConfig, TensorBoardOutputConfig, CollectionConfig, ProfilerRule, rule_configs
from sagemaker.debugger import ProfilerConfig, FrameworkProfile

from sagemaker.pytorch import PyTorchModel
from sagemaker.predictor import Predictor

```

```

jpeg_serializer = sagemaker.serializers.IdentitySerializer("image/jpeg")
json_deserializer = sagemaker.deserializers.JSONDeserializer()

```

```

class ImagePredictor(Predictor):
    def __init__(self, endpoint_name, sagemaker_session):
        super(ImagePredictor, self).__init__(
            endpoint_name,
            sagemaker_session=sagemaker_session,
            serializer=jpeg_serializer,
            deserializer=json_deserializer,
        )

```

```

pytorch_model = PyTorchModel(model_data=model_location,
                              role=role,
                              entry_point='fm_inference.py',
                              py_version='py36',
                              framework_version='1.8',
                              predictor_cls=ImagePredictor)

```

```

begin = time()
predictor = pytorch_model.deploy(initial_instance_count=1, instance_type='ml.m5.large')
c_time = time()-begin
print('Creating Endpoint Time: {:.01f}s'.format(c_time))

```

```

-----!Creating Endpoint Time: 182.6s

```

Inference was made possible by transforming the image data, and calling the model for prediction. The code is as follows:

```

101
102 # inference
103 def predict_fn(input_object, model):
104     logger.info('In predict fn')
105
106     mean = [0.2860819389520745]
107     std = [0.3529394901810539]
108
109     test_transform = transforms.Compose([
110         transforms.Grayscale(num_output_channels=1),
111         transforms.ToTensor(),
112         transforms.Normalize(mean, std),
113     ])
114
115     logger.info("transforming input")
116     input_object=test_transform(input_object)
117     with torch.no_grad():
118         logger.info("Calling model")
119         prediction = model(input_object.unsqueeze(0))
120
121     return prediction.cpu().numpy().tolist()

```

The model was able to predict accurately the different classes represented in the dataset, therefore the aim of this project was achieved more or less. Although it got some predictions wrong when randomly picking an image, the time in which it requires to make a prediction is also impressive, a useful asset in developing a computer vision model.

```
In [148...  
output = predictor.predict(input)  
output
```

```
Out[148...  
[[0.5479651689529419,  
  1.8651485443115234,  
  -2.5988097190856934,  
  5.487130641937256,  
  1.931625485420227,  
  -6.758813381195068,  
  3.092844247817993,  
  -2.8216121196746826,  
  -1.2999895811080933,  
  -0.9873822927474976]]
```

```
In [149...  
import numpy as np  
prediction = np.argmax(output)  
print('prediction:', prediction, ', ', filename, 'label: ', label)  
if prediction == label:  
    print('Prediction is correct')  
else:  
    print('Prediction is not correct')
```

```
prediction: 3 , images/test/6/Shirt_586.jpg label: 2  
Prediction is not correct
```

```
In [108...  
begin = time()  
preds = []  
for k in range(testset.shape[0]):  
    filename = testset['fullname'][k]  
    with open(filename, 'rb') as f:  
        input = f.read()  
        output = predictor.predict(input)  
        preds.append(np.argmax(output))  
  
seconds = time() - begin  
mins = seconds // 60  
print(f'Testing time: {seconds: 0.1f}s')
```

```
Testing time: 121.9s
```

```
In [43]: correct_preds = testset[equal_indexes]
correct_preds.reset_index(inplace=True, drop=True)
correct_preds.head()
```

```
Out[43]:
```

	fullname	label	prediction
0	images/test/9/Ankle-Boot_1.jpg	9	9
1	images/test/2/Pullover_1.jpg	2	2
2	images/test/1/Trouser_1.jpg	1	1
3	images/test/1/Trouser_2.jpg	1	1
4	images/test/6/Shirt_1.jpg	6	6

```
In [44]: accu = sum(equal_indexes)/testset.shape[0]
print(f'The accuracy of all test images is: {accu:0.4f} or{accu*100: .0f}%')

The accuracy of all test images is: 0.8998 or 90%
```

```
In [45]: cls_names = list(pd.read_csv('classes.csv')['class_name'])
cls_names
```

```
Out[45]: ['T-Shirt',
'Trouser',
'Pullover',
'Dress',
'Coatadd_label',
'Sandal',
'Shirt',
'Sneaker',
'Bag',
'Ankle-Boot']
```

Conclusion

I successfully completed this project with good accuracy, though there is always room for improvement. By concluding this project, I learnt how to use pytorch for image classification. With so many trials I made to choose the right parameter, I ended up using resnet34 pretrained model with 20 epochs.to conduct training and eventually making an inference

Reference

<https://dev.to/sandeepbalachandran/machine-learning-dense-layer-k2i>

<https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>

<https://docs.microsoft.com/en-us/windows/ai/windows-ml/tutorials/pytorch-train-model>

<https://towardsdatascience.com/how-to-train-an-image-classifier-in-pytorch-and-use-it-to-perform-basic-inference-on-single-images-99465a1e9bf5>