Individual Project Report

Fashion Finder: Classify Clothing Categories with AI

Group-04

Meet Daxini

DATS-6303

Deep-Learning

Dr. Amir Jafari

05/03/2024

Contents

1 Introduction

"Fashion Finder" is a project that aims to apply computer vision to the fashion industry by using machine learning models to classify images of people wearing various outfits. The goal is to enhance consumer experience and provide valuable insights to retailers and designers by understanding current trends and customer preferences.

2 Individual Work

1. Creating the final dataset from the kaggle dataset and created shell script for downloading the dataset
2. Doing basic eda of the data like creating correlation plot and distribution of the datset
3. Scripts to train and evaluate CNN, Resnet, and ViT models
4. Using Class weights for imbalanced dataset, freezing and unfreezning layers
5. Created the app using streamlit
6. The percentage of the code that I found from the internet is around 50%

3 Individual Work Description

3.1 CNN

First, I designed a custom convolutional neural network (CNN) for this task as CNN are most popular for image classification tasks. CNN architecture comprises three convolutional layers each followed by batch normalization and a ReLU activation. Each convolutional layer is followed by a max pooling layer to reduce the spatial dimensions of the feature maps. The final feature maps are passed through a global average pooling layer, reducing each feature map to a single value. These values are then fed into a fully connected layer that outputs the predictions for the 46 classes.

3.2 Resnet

Then I then tried pretrained models. First I tried was Resnet, The key innovation of ResNet is the introduction of residual connections, also known as skip connections. These connections allow the network to learn residual functions instead of learning unreferenced functions. The idea is that it is easier to optimize the residual mapping than to optimize the original, unreferenced mapping. The residual connections enable the gradients to flow directly through the network, mitigating the vanishing gradient problem and allowing for much deeper networks to be trained effectively. I found that Resnet 101 was working best for the dataset based on F1 score. ResNet-101 as the name suggests consists of 101 layers.

3.3 Vision Transformer (ViT)

After experimenting with CNN and ResNet architectures, I explored the Vision Transformer (ViT) model for multi-label image classification task. ViT is a relatively new architecture that adapts the transformer model, which has been highly successful in natural language processing, to the domain of computer vision. Initially, I attempted to use a pre-trained ViT model called vit-base-patch16-224 from the Hugging Face library. However, I encountered challenges with this model, as the loss remained high during training, and I was unable to resolve the issue. As an alternative, I turned to the ViT implementation provided by PyTorch in the torchvision library. The ViT model I used, specifically the vit_b_16 model from PyTorch, is pre-trained on the ImageNet dataset.

The architecture of ViT differs from traditional CNNs. Instead of using convolutional layers, ViT divides the input image into fixed-size patches of 16x16 pixels. These patches are linearly embedded and combined with positional embeddings to preserve spatial information. The resulting sequence of embedded patches is then fed into a transformer encoder, which learns to attend to different patches and capture their relationships.

The transformer encoder consists of multiple layers of self-attention and feed-forward networks. The self-attention mechanism allows the model to weigh the importance of different patches and learn their dependencies. After passing through the transformer layers, the encoded representation is fed into a linear classification head, which predicts the presence or absence of each class label.

To adapt the pre-trained ViT model to multi-label classification task, I modified the classification head by replacing it with a linear layer that outputs the desired number of classes (46 in this case). I also applied a sigmoid activation function to the output of the classification head to obtain probability scores for each class.

The ViT model demonstrated impressive performance on the test set, achieving high F1 scores and accuracy. The self-attention mechanism of the transformer architecture allowed the model to effectively capture the relationships between different regions of the image and make accurate predictions for multiple labels simultaneously.

3.3.1 Class Weight Calculation

Given the imbalanced nature of the dataset I implemented a strategy to manage the uneven distribution of classes effectively. This strategy involves calculating class weights to ensure that the model does not become biased toward more frequently occurring classes.

To calculate class weights, I first converted the categorical labels into a binary matrix where each row represents an image and each column represents a class. This matrix was achieved using the Pandas get_dummies function on the Category column of training data:

```
label_matrix = train_data["Category"].str.get_dummies(",")
```

Next, I computed the frequency of each class across all samples:

```
class_frequencies = label_matrix.sum()
total_samples = class_frequencies.sum()
```

Using these frequencies, I determined the weight for each class by dividing the total number of samples by the product of the frequency of the class and the number of classes, ensuring that under-represented classes are given higher importance during model training:

```
class_weights = total_samples / (class_frequencies * len(class_frequencies))
```

To apply these weights during the training phase, I calculated sample weights for each image in the dataset. As it is Multi label, I were doing the sum of the weights of each class present as the weight of the image. But it was taking a lot of time so I got the same operation done faster multiplying the binary label matrix with the class weight matrix:

```
sample_weights = label_matrix.dot(class_weight_tensor).values
```

I then used these sample weights to create a sampler for the inbuilt DataLoader module in pytorch, ensuring that each batch of data is representative of the overall dataset, despite the class imbalance:

```
sampler = WeightedRandomSampler(
sample_weights, num_samples=len(train_dataset), replacement=True
)
```

Finally, the DataLoader was configured to use this sampler along with a specified batch size and number of worker threads for loading data:

```
train_loader = DataLoader(train_dataset, batch_size=16, sampler=sampler)
```

After using class weights on models fairly improved across all classes, and its ability to generalize also got better which reduces the risk of bias towards more frequent classes.

3.3.2 Freeze-Unfreeze technique

The freeze-unfreeze technique, is a method for accelerating the training of deep neural networks by progressively freezing layers. The motivation behind this technique is that early layers in deep architectures tend to converge to simple configurations (e.g., edge detectors) and may not require as much fine-tuning as later layers, which contain most of the parameters.

In the script, I employed a variant of the freeze unfreeze technique. I start by freezing all layers of the pre-trained models except for the last fully connected layer. During training, I gradually unfreeze more layers as the number of epochs progresses. This allows the model to adapt its weights to the specific task while leveraging the pre-trained features from earlier layers.

The freeze-unfreeze technique helped speed up the training process but it did not improved the F1 scores or Accuracy that much.

4 Results

To evaluate the effectiveness of my models, I analyzed their performance using a variety of metrics, including Accuracy, F1 Score, Precision, and Recall. These metrics provide insights into how well each model predicts the correct clothing categories.

Model	**Accuracy**	**F1 Score**	**Precision**	**Recall**
ViT_B_16	15.03%	74.84%	78.50%	71.51%
ResNet_101	7.43%	65.01%	69.05%	61.41%
CNN	4.74%	55.63%	71.75%	45.43%

Table 1: Overall performance metrics of the models.

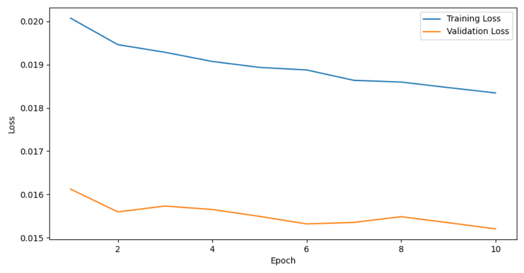

Figure 1: Training and Validation Loss Trends for CNN Model Across Epochs

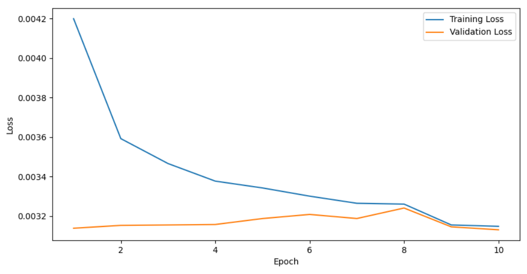

Figure 2: Training and Validation Loss Trends for ViT Model Across Epochs

5 Conclusion

The project successfully tackled the complex task of multi-label classification of fashion apparel using state-of-the-art machine learning models such as CNNs, ResNets, InceptionNet and transformers. Among these, the Vision Transformer (ViT) model stood out, demonstrating superior accuracy and generalization capabilities across various clothing categories.

Furthermore, the integration of these models into a Streamlit-based application provided a seamless and user-friendly platform for real-time fashion classification. This application allows users to effortlessly upload images and receive immediate, reliable classification results, making it an excellent tool for both fashion enthusiasts and industry professionals.

6 Future Enhancements

I focused on multi-label classification of fashion images using CNN, ResNet, and transformer-based models. In the future, I aim to:

1. Implement unsupervised learning techniques, such as K-means clustering, to pre-screen uploaded images and identify non-apparel images. This will improve the efficiency and accuracy of the application by preventing the processing of irrelevant images.

2. Gain a deeper understanding of each machine learning models used in the project. This knowledge will enable me to conduct more effective post-training analysis and optimize the models for better accuracy in future iterations.

7 References

1. Pytorch documentation.
2. google/vit-base-patch16-224
3. Fine-Tuning Vision Transformer with Hugging Face and PyTorch
4. Training data-efficient image transformers & distillation through attention
5. FREEZEOUT: ACCELERATE TRAINING BY PROGRESSIVELY FREEZING LAYERS.
6. Pytorch Multi Label Classifier github example
7. Enhancing Multi-Class Classification with Focal Loss in PyTorch