Reichman University

M.Sc. Machine Learning & Data Science **October 27, 2022**

FASHION COMPATIBILITY TRANSFER-LEARNING WITH EFFICIENT-NET AND BERT

Tomer Cohen
Reichman University, Herzliya, Israel
itomerco@gmail.com

Eliran Gerbi
Reichman University, Herzliya, Israel
elirangerbi@gmail.com

Supervisor: ***Dr. Asi Messica***
Reichman University, Herzliya, Israel
asnat.messica@post.runi.ac.il

ABSTRACT

The ubiquity of online fashion shopping demands an effective recommendation services for customers. In this paper, we study fashion recommendation for two main daily use-cases:

I **Fill-In-The-Blank (FITB)** - suggesting an item that best matches a set of other items to form a stylish outfit (a collection of fashion items)

II **Outfit Compatibility** - predicting the compatibility of an outfit.

For the Fashion Compatibility recommendation we consider **every outfit as a collection of clothing items with a strictly defined ordered sequence based on the item's Main Category ID** - top to bottom and then on to accessories (e.g.shirt, pants, shoes and sunglasses).

Our proposed solution harnesses the Transfer Learning methodology approach to leverage the power of pre-trained Image Recognition CNN and NLP Language models for the down-task of Fashion Compatibility. We do so by seperately learning fashion items' visual and semantic embedding (using their images and descriptions respectively) and then combining them together to find strong fashion compatibility relationships.

Our solution achieved a substantial **+11.8%** performance improvement for the FITB use-case (#1) over the reference SOTA solution.

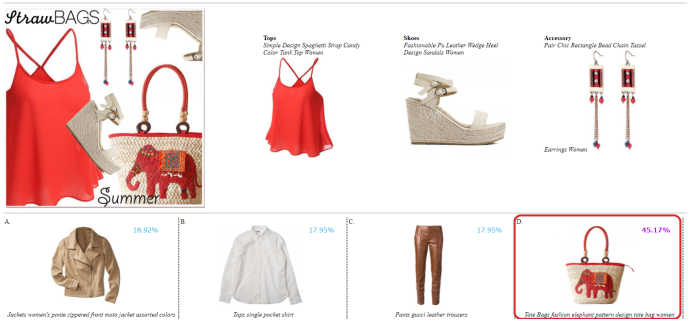

Fig. 1. Usecase I: Find the missing item in a listing (FITB)

Fig. 2. Usecase II: Outfit Compatibility Score

Keywords Transfer-Learning · Transformers · Deep Neural Netrowks [DNN] · EfficientNet · BERT · Long Short-term memory [LSTM] · Embedding · Ensemble · XGBoost

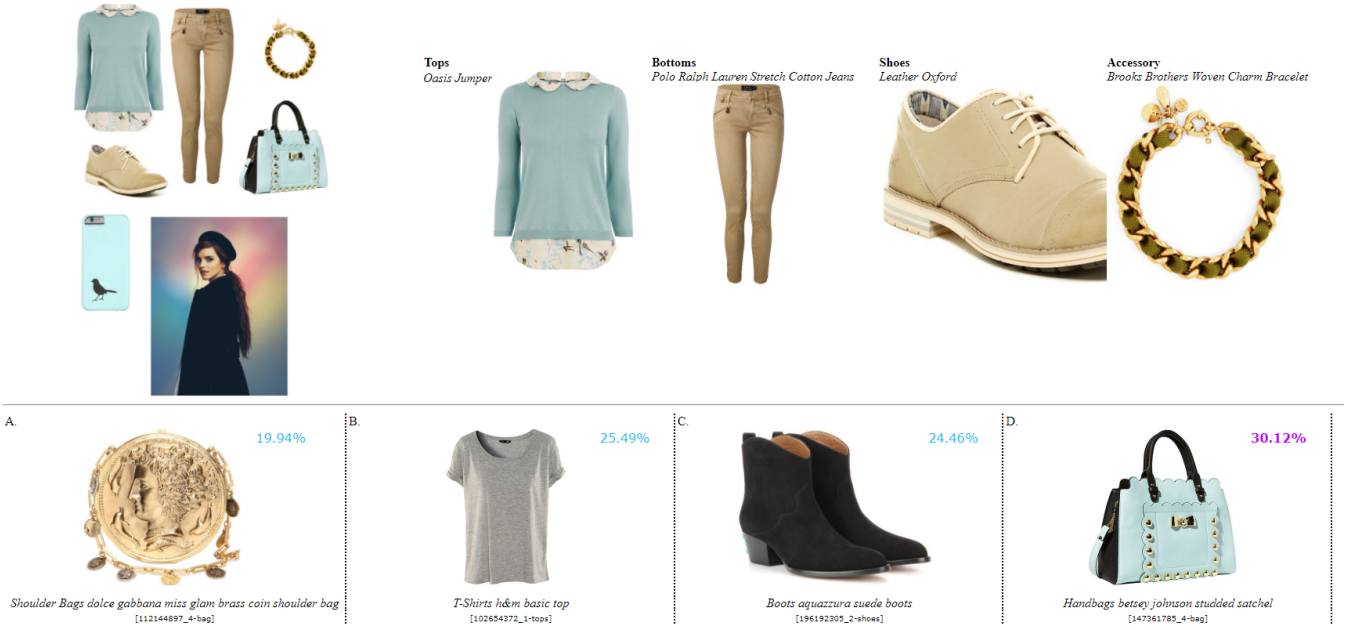

Fig. 3. Outfit #147361785: A fashion look from January 2015 featuring preppy shirts, beige jeans and leather oxfords. Browse and shop related looks. Fill in the blank (FITB) - missing handbag.

1. INTRODUCTION

With the proliferation of social networks, people share almost everything in their daily life online nowadays. They share the movies they watched, the places they visited, and also, the outfits they've created and or wore.
Polyvore.com was one such example of Fashion-focused online communities, were people showcased their personal styles and connected to others that share similar fashion taste. Unfortunately, the company was shut down in April 2018.

There are two kinds of fashion compatibility problems. One is ranking a complete outfit composed out of multiple fashion items that people may be interested in. The other is recommending fashion items that make good matches with some given items in a given outfit. For example, recommending a business bag for a given business outfit already containing a shirt (top item), trousers (bottoms) and shoes.

Our approach to finding the best automated solution for these tasks is to utilize highly capable Image Recognition and NLP pre-trained models by which to project the fashion items' visual and textual embedded features vectors in two separate spaces accordingly, estimate compatibility via similarities of these vectors in each space separately and then combine these estimations into a final fashion compatibility scoring and best item recommendation.

We speculate that BERT's highly capable modeling engine, is capable of pinpoint the items descriptions' fine properties while also considering their position with-in the ordered outfits they belong to. This capability enables us to find semantic similarities between an outfit and a suggested candidate for a missing item and thus to establish an effective Fashion Compatibility matching mechanism based on text only.
More specifically, given a rich set of pre-ordered outfits, we fine-tune the BERT (HuggingFace) model to learn item compatibility, conditioned on the other items in the outfit and their ordered compatibility relationships.

For the outfit's visual styling learning, we train a Bidirectional-LSTM (Bi-LSTM [1]) model on top of an EfficientNet_B4 (EN_B4) CNN model. This way, the EN_B4 model creates an embedded vector that encapsulates the item's visual features, which serves as an input to the Bi-LSTM model to learn the in-outfit items' positional relationship.

The trained Bi-LSTM-EN_B4 network was initially trained on both Image-Net [2] and Fashion-Mnist [3] databases (see appendix 10.2), and then fine-tuned on the Polyvore item's dataset. This highly capable final model can not only perform the aforementioned recommendations effectively but also predict the compatibility of a given outfit [See chapter *Bi-LSTM with EfficientNet System Implementation* 6.1.5 for

more details].

This Bi-LSTM-EN_B4 solution improves the previous *BiLSTM_InceptionV3* solution by **1.1%** on the Outfit Compatibility use-case (#2).

As for the top layer of our suggested solution for the FITB use-case, we introduce an XGBoost Classifier ensemble layer, combining the NLP based language classification (BERT) model's results with the Image Recognition CNN classification (BiLSTM-EfficientNet) model's results and a few additional metadata features.

This suggested solution architecture enables multi-modal (images+text) fashion-compatibility learning & inference approach achieving 76.7% accuracy on the FITB use-case.

The main contributions of this work are summarized as follows:

- We successfully employed Transfer Learning methodology for the Fashion Items' Compatibility downstream task.
- We've adapted BERT's *Next Sentence Prediction* capability for the task of Fill In The Blank missing item.
- We've incorporated an Ensemble of Transform Learning models creating a Hybrid multi-modal Fashion Compatibility Solution.
- Our work outperforms current SOTA method by 11.8% improvement [see Table_3].

2. RELATED WORK

In previous work described in the original paper ***"Learning Fashion Compatibility with Bidirectional LSTMs"*** [4] the authors' solution jointly learned a visual-semantic embedding and used this representation to infer the compatibility relationships among fashion items. Learning the transitions between the ordered items with-in outfits serves as a proxy for identifying the compatibility relationships of fashion items.
Though this sound quite promising, it falls short in enabling (1.) meaningful outfit's items descriptions semantics (achieved 29.2% accuracy on the FITB use-case) and (2.) combining images & text to achieve finer Fashion Compatibility learning.

In their work they've managed to achieve 66.7% prediction accuracy on the FITB use-case (selecting the correct missing item from 4 optional choices) using a Bi-LSTM model on top of an Inception-V3 CNN model.
However, adding the Outfits & Items descriptions' semantic learning, minimizing the items images and descriptions cosine distance at the embedding space, contributed only 2% to the final integrated Bi-LSTM + VSE model prediction, with overall 68.6% accuracy on the FITB use-case and 0.90 on the Outfit Compatibility use-case.

Items Categories				
Main CID	Main Category	Train Count	Test Count	Notes
0	Outer	6,890	1,145	
1	Tops	12,429	1,638	
2	Full	4,334	957	
3	Bottoms	11,408	1,492	
4	Shoes	16,622	2,635	
5	Leg-wear	947	121	
6	Head-wear	2,520	312	
7	Bag	13,307	2,166	
8	Accessory	26,299	3,660	
9	Cosmetics	10,697	724	
11	_other_fashion	2,372	639	Discarded
12	_other	6,981	3,115	Discarded

Table 1. Items Distribution within Main Categories in the Polyvore dataset [5]

3. POLYVORE DATASET

Polyvore was a community-powered social commerce website hosting virtual fashion styling board function which allowed community members to add products into a shared product index, and use them to create image collages called "Outfits" (/"Sets"). They could also browse other users' sets for inspiration, share sets with friends and interact with people through comments and likes.

The Polyvore Dataset contains 21,889 outfits, 142,478 items with images and descriptions (text). There are 167 Categories (AKA CategoryIDs/ Sub-Categories) distributing between 10 Main-Categories as detailed in Table_1.

In order to train the model, and then test it for both use-cases (FITB and Outfit Compatibility), the dataset was initially split into Train set (17,316 outfits and 114,806 items), Evaluation set (1,497 outfits and 9,068 items), and Test set (3,076 outfits and 18,604 items), keeping the same distributions of main and sub categories in all 3 datasets.

4. DATA PREPARATION

We've conducted the following data preparation actions on the Polyvore data before model training stage:

4.1. Data Cleaning:

- Items form Sub Category "_other_fashion" from "_other" Sub Category where removed from the Train-

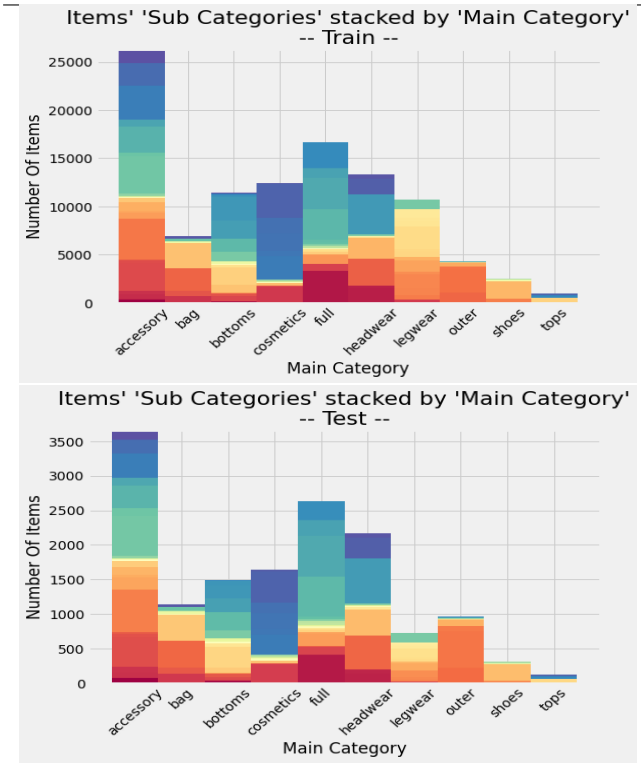

Fig. 4. Polyvore's outfits items' sub-categories Distribution within Train and Test Data Sets

ing and the Test set (see numbers in Table_1) .

- 1,369 items with name shorter than 3 characters were removed from the Train Set, for the NLP task with BERT.
- Outfits that has less than 3 items or contains more than 3 items from the same Main Category (e.g. an outfit with more than 3 accessories items) where removed from the dataset. This resulted in removing 974 oufits from the training data and 485 outfits from the test data.
- In every outfit, items are sorted by their Main Category in order to reflect the correct items ordering with in all outfits.

Finally we've ended up using 16,342 outfits (105,453 items), out of the 17,316 outfits in the given Train data, and 1,225 outfits (7,145 items) in the Evaluation dataset.

4.2. Prepare Train, Eval and Test sets for the FITB use-case:

Training BERT model for the FITB use-case required a Fill-In-The-Blank (FITB) variation of the Train and Eval data sets.

In the originally given FITB formatted Test dataset, created by the original paper's authors, the correct answer was always the first choice and the 3 other (wrong) choices for the missing item were randomly selected from the whole items dataset (with replacement).

This resulted with the following anomalies in the FITB Test dataset:

- There were incidents where the same choice repeated more than once. In some cases this duplication occurred for the correct item itself.
- Selecting the most appropriate missing item ended up to be just selecting the first choice of the four options. This obviously makes the task of selecting the true missing item much easier.
- In most cases all the 3 "wrong choice" items were not selected from the same main category as the true missing item. So the model can simply select the item belongs to the missing item's main category type (according to the outfits' ordered "top to bottom" item's main category rule) without examining the item's image visual features and items description's semantic properties.

In order to avoid these drawbacks in this project, we've applied the following guidelines when creating the FITB Train, Eval, and Custom-Test datasets:

- 2 out of the 3 other (wrong) choices were selected from the same main category to which the true missing item belongs to.
- All 4 choices are unique (i.e no replacements)
- The correct answer was uniformly distributed among the 4 optional choices' indexes.

The final performance comparison reported for the FITB use-case (#1) on the Originally Given FITB Test dataset, is detailed in section 7 "Performance Comparison And Analysis".

5. APPROACH

Here we overview our suggested solution leveraging the strong classification capabilities and Transfer-Learning opportunity of both EfficientNet-B4 and HuggingFace's BertForMultipleChoice (based on BERT's "Next Sentence Prediction training" mode) models.

5.1. Usecase I: FIll In The Blank [FITB]

Our proposed solution for the FITB task has two main separately trained layers [see Fig. 5]:

5.1.1. *1st (Lower) Layer - Image & Text Classification models with Transfer Learning*

This layer is built of two models trained separately on the Polyvore dataset after being prepared and cleaned as described in the "Data Preparation" section.

- **BiLSTM over EfficientNet_B4 (BiLSTM_ENB4) model** for the Image Classification & Compatibility task: learn Polyvore's items visual properties along with their visual relationships to construct a visually compatible items sets for fashion outfits.
- **HuggingFace's BertForMultipleChoice model** for the Item's description Classification & Compatibility task: based on BERT's "Next Sentence Prediction training" mode, we fine-tune BERT model to capture Polyvore's items' description semantic similarity and their relationships to construct a semantically compatible fashion outfits.

Our approach creates 2 separate high dimensional visual and textual embedding representations of all items. In these vector spaces, products with similar & related styles and attributes will be closer than unrelated ones.

5.1.2. *2nd (Top) Layer - XGBoost Classifier*

Utilizing the separately pre-trained models in the 1st layer, we've trained an XGBoost [6] Classifier model to combine these results, in to a final single FITB recommendation for each outfit. [see section_6.3 *"Top layer ensemble for FITB use-case"*]

5.2. Usecase II: Outfit Compatibility

The outfit compatibility use-case suggested in the paper measures the cosine similarity between the outfit's item's embedded vectors created by the pre-trained EffcientNet model, to those computed by a trained BiLSTM_ENB4 model executed on all the outfit's items.
The average cosine similarity results for all the outfit's items is considered as the overall Outfit Compatibility score.

That is, if the BiLSTM_ENB4 model suggests a specific "bottom" item as the most compatible one to the "top" item (see "Item Categories" in Table_1), the Outfit Compatibility use-case estimates how close it's embedded vector to the one belong to the true "bottom" item of this outfit.

Computing this kind of similarity for all items in a suggested outfit, the Fashion Compatibility model produces an overall mean score for the compatibility level of all items in the outfit with the ones suggested by the pre-trained model in the same positions.

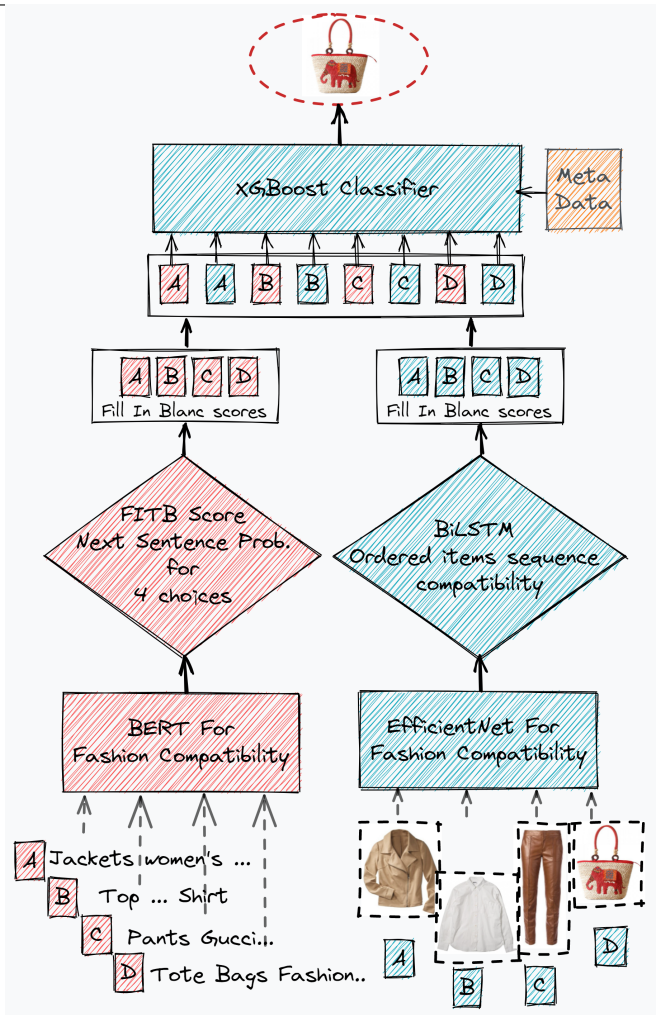

Fig. 5. Suggested Solution - High level Architecture Schema

The dataset prepared to evaluate a Fashion Compatibility model for this use-case contains 7,076 records, each specifies a set of items belongs to a specific outfit and a label 1 for a valid outfit and 0 for non valid outfit.

The authors created this dataset based on the following sources:

1. All the 3,076 outfits from the Test dataframe, with label = '1' (AKA 'compatible' outfits)
2. 4,000 fake outfits, built specifically for this task by random sampling of items from the Test dataframe, with label = '0' (AKA not compatible outfits)

5.2.1. *Outfit Compatibility use-case with BertForFashionCompatibility model - an alternative approach*

The Outfit Compatibility approach described above is not applicable by the BERT model in general, and more specifically

Fig. 6. Example 1: Low outfit compatibility score

Fig. 7. Example 1: High outfit compatibility score

by the "BertForMultipleChoice" extension we've used for the FITB use-case.
Therefore, our suggested solution for the Outfit Compatibility use-case, as described and implemented in the paper, was implemented solely by the Bi-LSTM_ENB4 model.

However, we suggest two other slightly different approaches to consider for the outfit compatibility use-case using the BertForFassionCompatibility model:

*** Approach I:** Compute outfit's compatibility score based on the overall visual /semantic fashion compatibility relations between every item and the following one in the outfit, when ordered by the items' main category

*** Approach II.:** Compute the overall visual /semantic fashion compatibility relations between every item and the other items in the outfit.

We think that both of these two suggestions could be valid, reasonable an even more intuitive for this use-case.
Therefore, we've implemented both of these approaches based on the items descriptions' 512d semantic embeddings extracted from the BertForFahsionCompatibility model (section 6.2.4).

6. EXPERIMENT

6.1. Image Classification via Transfer Learning For Fashion Compatibility

In this section we describe how we've designed, configured and implemented the Image Classification & Fashion Compatibility Relations learning sub-system, as part of Layer-I, to effectively learn the Polyvore's outfits images for the Fashion Compatibility task.

6.1.1. EfficientNet model selection

The ***EfficientNet_B4*** model was selected after a thorough work we've conducted to find the best configuration our system can execute, bounded by it's limited compute power and memory, in order to leverage the most out of the BiLSTM+EficientNet models' joint capabilities on our task.

Still, in order to avoid reducing the batch size from 10 outfits to 2 outfits we've used a reduced image size of 300 pixels (compared to the 380pixels originally defined for EfficientNet_B4) and still were forced to set the batch size to 8 outfits and clear the cuda (GPU) cache at the start of every epoch and every evaluation in order not to exceed the 24GB GPU memory space.

6.1.2. Pre-Train Image Classification model

Google's EfficientNet_B4 model comes pre-trained on the **ImageNet** dataset (containing 1,281,167 pictures in the Train dataset and additional 50K images in the validation set)

In order to improve this pre-trained model with Fashion Clothing visual features we've further fine-tuned it with the **Fashion Mnist** dataset (60,000 pictures in Train set & +10K in Test set as described above).

The Fashion Mnist items' categories are very relevant for our use-case as they include quite similar items to those in the Polyvore data.
Fashion-mnist categories:

* T-Shirt/Top	* Trouser	* Pullover
* Dress	* Coat	* Sandal
* Shirt	* Sneaker	* Bag
* Ankle Boot		

6.1.3. Learn item's Fashion Compatibility Relations with Bi-LSTM

In order for our model to learn the outfits images' Fashion-Compatibility **Relations** we've added a Bidirectional Long short-term memory (Bi-LSTM) model that accepts the EfficientNet's output.

That is, we've replaced the EfficientNet model's original top layer with a new 1792X512 Dense layer (512d output), in order to have the output of the EfficientNet's model to match the Bi-LSTM's input size.

This way, the LSTM was trained to predict the next item found in the outfit, sorted by the items' main category-types (see **Table_[1]**). Selecting the Bi-Directional flavor of the

LSTM model enables the learning of Fashion Compatibility Relation of each item and its' previous and next items in the outfit.

The final Bi-LSTM model selected has a single LSTM layer, no Dropouts and 512 dimensions input, hidden & state vectors.

6.1.4. Polyvore Images Pre-processing

First we've resized Polyvore's various sizes pictures to a uniform shape of 300x300 pixels and normalized all pictures' pixels values to the range of [0-1].
Then, in order to improve the ability of the EfficientNet model to generalize it's learning on the Polyvore's items images we've used multiple image augmentation techniques to enrich the variations of the images such as random image re-scaling, horizontal flips, rotation and cropping.

6.1.5. Bi-LSTM with EfficientNet - Model Implementation

Our code is based on the Learning Fashion Compatibility with Bidirectional LSTMs - PyTorch version in-which the author have implemented the solution suggested in the original paper.

As our solution handles the outfit's items visual and textual correlations separately and for the image classification we've replaced the Inception_V3 originally used with the Efficient_B4, pretraineld on the Fashion-Mnist Dataset (as described above), most of the model's configurations and settings have been changed.

Additionally, in our work we've changed the code and fixed a few weaknesses (/critical bugs) in the original implementation:

- We've improved the mechanism by which the system loop through the item with in a outfit so that it can handle outfits with missing items. For example: Given an outfit with only 4 items from main categories - "outer", "top", "leg-wear" and "bag". It is basically missing both bottoms and shoes items. While, the original code would have required this outfit to be removed, our solution does enables "sparse" outfits with multiple items missing, capable of fitting an item with the highest score from any of the missing main categories.

- In the paper, they've stated that they "stop the training process when the loss on the validation set stabilizes." This is quite an amorphous statement for how they objectively decided on ending the learning. In our work we've added a clear evaluation step, performed with-in the training process after every epoch, as usually done when training ML models, and stopped the training on the best results achieved on the Evaluation set.

- EfficientNet_B4, Bi-LSTM and training hyper-parameters: As our system has already been finetuned on Fashion-Mnist, much less epochs were required to fintune the EfficientNet_B4 model. Therefore we've changed most of the system components and learning hypter parameters as follows:

  1. Learning rate was changed from 0.2 to 0.01. The original learning rate used is very high compared to the usually used for fine-tuning a large pre-trained model such as Inception, EfficientNet, BERT etc. In our case the normal value of 0.01 worked just fine.
  2. Weight decay was increased by 200% from 1e-4 to 3e-4 to enforce stronger regularization and reduce model over fitting.
  3. Bi-LSTM: Single layer with no dropouts. In the original work a very high Dropout rate of 0.7 was used as there was an intense learning required for the InceptionV3 model on the Polyvore dataset.
  4. In the original code 100 epochs are specified to be performed with no evaluation with-in the learning process to capture the model at it's best state (checkpoints) evaluated on the evaluation set.
     Our training archived great results almost from the very first epochs and converged after 39 epochs, whle still improving the paper's results in all aspects.

6.1.6. Model Training

Training this system on the Polyvore outfits' train dataset took anywhere between 10 hours to 3 days depending on the training configuration such as the EfficientNet's version used (B0, B3, B4 and B5 were tested), items image sizes, number of outfits in every batch, learning rate, the number of layers and Dropout rates configured for the BiLSTM's model, optimizer and scheduler properties, etc.

The selected best model was saved via checkpoint during the training when highest accuracy was recorded on the Eval FITB dataset.

6.2. Transfer Learning with BERT for Fashion Compatibility

*6.2.1. **Constructing Outfit Query Sentence as input for BERT***

In order to train BERT to learn the right structure of the outfit's items, just like it knows to learn the relevancy (attention) between words in a standard English sentence, we've carefully

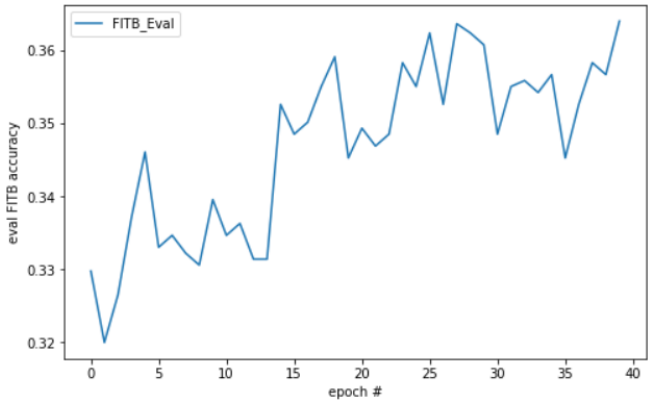

Fig. 8. Bi-LSTM_ENB4 model training - accuracy progress on Eval FITB dataset

crafted the 'Context' sentence to give BERT the required outfit's internal items order.

Building the 'Context' (the first sentence in the query) for BERT is systematically done as follows:

I. Order all outfit's items according to their Main Category ID (Ascending)

II. Randomly select an element from the Outfit to be the one we're trying to train BERT to predict it's high relevancy to the outfit.

III. For every item left in the outfit's items list concatenate the following properties:

   a. [optional] The missing item's Sub Category (available in the given data)

   b. Item Name (description)

In order to avoid information leakage from the query data to the predicted missing item, we've discarded the outfit's name and description just for the case that same/similar words are shared between the outfit's description and it's internal item's description. For example the item's Brand or famous designer.
Additionally, we didn't include the items' main categories as we wanted BERT to find out the right item missing in the position of the missing item considering it's implicit category, as can be understood from the items' descriptions.

An example for such [Context][Choices] created for an outfit (illustrated in Figure_[9]):

- **Context**: [color block fashions A fashion look from .. Coats roland .. Tank tops colour block .. Knee Length Skirts Jonathan .. Sandals topshop block .. Accessories auesome fashion ..]
- **Choice_1**: [Clutches proenza schouler ..]
- **Choice_2**: [Clutches burberry petal ..]
- **Choice_3**: [Tote Bags miss selfridge pink ..]
- **Choice_4**: [Floral Decor mother pearl ..]

6.2.2. Training Vanilla BERT - Initial performance analysis

Due to quite strict GPU memory limitations we've encountered running BERT model in the relatively modest 'bert-base-cased' flavour, on both google Colab-Pro online environment and on a new appliance which boasts NVIDIA's top new 3090 GPU card with 24GB memory on card (+ 64GB memory and AMD newest R9 chip), we've struggled to fully utilize BERT's potential to fine-tune BERT model to our specific downstream task. We've managed to run a successfully training only after limiting our queries to 205 word ids in total with batch size of 16 outfits.

A single, 3 epochs long, model train on a GPU, utilizing all available GPU memory, takes roughly 2hrs thus testing and fine-tuning the models tested in this paper took quite a lot of time overall.

6.2.3. Training Vanilla BERT model

We first conducted training test on a vanilla BERT model with various features combinations (like the outfit's name, outfit's description and item Sub Category) in order to analyze their overall contribution for a trained model's performance.

We've analyzed BERT's learning and prediction performance based on the following combinations in order to find the importance of each of those feature for best predictions.

Outfits & Items information in Training DataSets for Vanilla BERT model			
Combination Code	Outfit Name & Desc.	CID (Sub Cat.)	Main Cat.Type
nOFnCIDnCT	-	-	-
nOFwCIDnCT	-	+	-
wOFnCIDnCT	+	-	-
wOFwCIDnCT	+	+	-
wOFwCIDwCT	+	+	+

Table 2. Training Data Features Combinations.
Training the final ***BertForFashionCompatibility*** model was done with combination **"nOFwCIDnCT"** (config #2)

Our findings from the vanilla BERT model's performance - Figure_[10]:

- **nOFnCIDnCT**:The vanilla BERT model is capable to infer correctly with 52.7% accuracy with no outfit information nor any item's category features information.

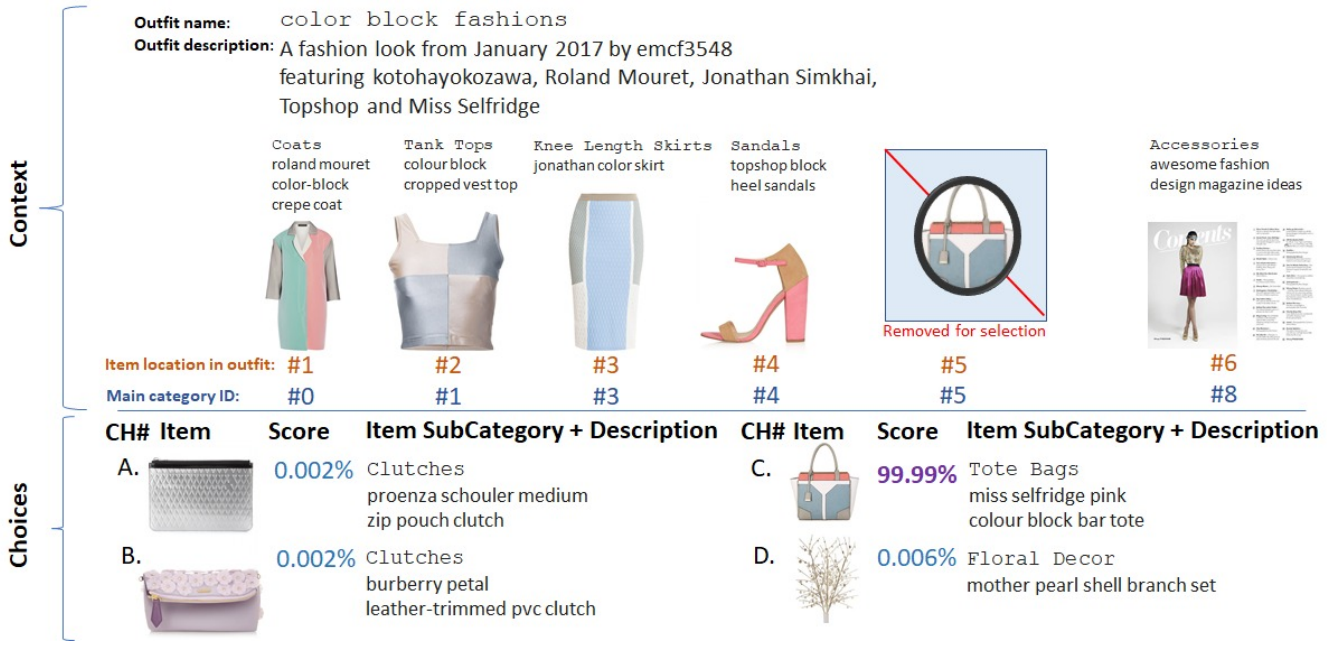

Fig. 9. Outfit & Items text information integrated in the [Context + Choice] format for the *BertForFashionCompatibility* model in the FITB usecase

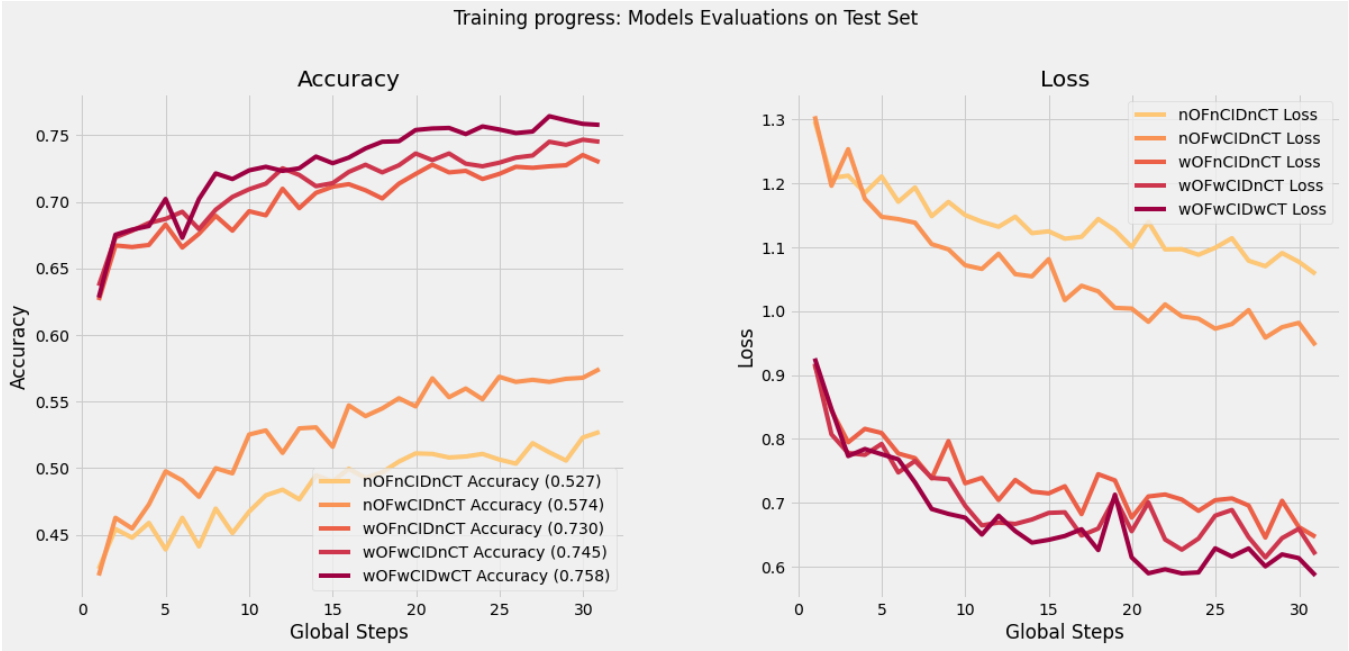

Fig. 10. Vanilla BERT Training Progress with various FITB query parameters included.

This by itself is +80% improvement over previously achieved by VSE model (method #1 in Table_3)

- **nOFwCIDnCT, wOFwCIDnCT**: Item's Sub Category contributes roughly 9% to the model's performance accuracy when is the only data added to the training over the items' names. When combined with the outfit's information the contribution further falls to 2% only.
- **wOFnCIDnCT**: Outfit's Name and Description as a single features has the highest impact on BERT's performance improvement over the most basic, items' names only option.
- **wOFwCIDwCT**: Item Sub Category's Contribution to the model's accuracy performance when combined with other information is 1.3%.

As speculated, keeping the outfits' name and description and the missing item's main category in the query text enabled the model to showcase much stronger learning and inferring capabilities consider the accuracy scored recorded on the Eval and the Test sets. That is, for substantial number of outfits, this information have disclosed some meta information correlated to many of the item's description.

Therefore, in order to reflect the model's real textual and semantic learning capabilities, we've discarded the outfits' name and description, along with their main category name, from the query string, keeping only the items' sub-category (category ID), when training our **BertForFahsionCompatibility** custom model.

6.2.4. *BertForFashionCompatibility*

Model Architecture

Our system is built on top BERT model and leveraging HuggingFace's "BertForMultipleChoice" extension to BERT's vanilla second use case "Next Sentence Prediction".

HuggingFace's ***BertPreTrainedModel*** is "an abstract class to handle weights initialization and a simple interface for downloading and loading pre-trained models", and wraps a bare Bert Model transformer without any specific head on top thus outputs raw hidden-states.

The ***BertForFashionCompatibility*** [BFFC] module extends ***BertPreTrainedModel*** while adding the following elements:

I. 2 PyTorch Linear layers that reduces BERT's output values gradually from 768 (BERT's hidden layer size) -> 64 - > 1 (last classification layer with CrossEntropyLoss)

II. Dropout is performed on the first two linear layers. Dropout probability is 0.1 on the first layer and 0.5 on the second one (in order to avoid over-fitting our learning on the Training Data, as we're making a deeper NN head).

III. Learning Rate = 3e-5, close to the min value recommended for BERT training.

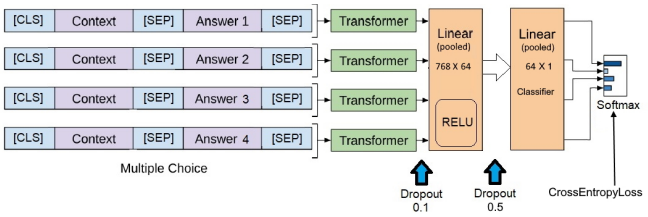

Fig. 11. Custom BertForFashionCompatibility model - A [CLS] token is inserted at the beginning of the first sentence and a [SEP] token is inserted at the end of each sentence.

Training & Results Analysis

BertForFashionCompatibility model on the Eval and Custom FITB Test datasets.

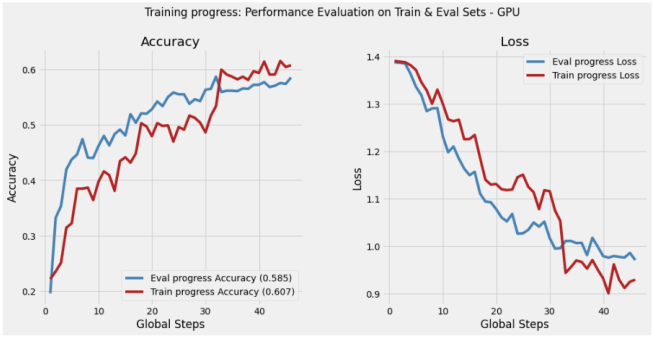

Fig. 12. Bert For Fashion Compatibility model training - accuracy and loss progress on FITB Eval dataset

Model evaluation with **Custom FITB Test**: In order to properly evaluate BERT's real learning capabilities for our FITB downtask use-case, we've created a new Custom FITB Test dataframe, built from the same original test dataset used by the authors. In this Custom FITB Test dataframe, we've used the same approach used to construct the Train and Eval FITB to create a harder but much more indicative FITB Test dataframe (see "**Data Preparation**" - section 4).

6.3. Top layer ensemble for FITB use-case

As briefly described in section 5.1.2, we've used an XGBoost Classifier ML model in order to aggregate the FITB predictions computed for each of the 4 choices suggested

Classification Report:

	precision	recall	f1-score	support
0	0.5362	0.5978	0.5653	644
1	0.5063	0.5778	0.5397	623
2	0.5839	0.5427	0.5625	667
3	0.6734	0.5556	0.6088	657
accuracy			0.5681	2591
macro avg	0.5750	0.5685	0.5691	2591
weighted avg	0.5761	0.5681	0.5695	2591

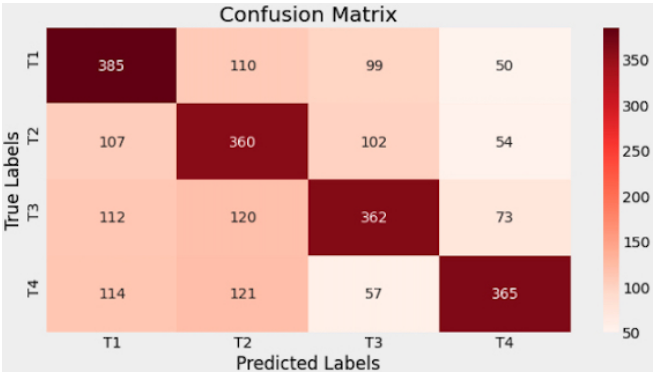

Fig. 13. Bert For Fashion Compatibility model confusion matrix on **CUSTOM** FITB Test dataset

as the missing item in every outfit, along with some extra metadata properties found in the original Train, Eval and Test dataframes.

We've decided to use an ensemble boosting tree based model as opposed to a neural network due to the relatively low number of features and samples available for the model to train.

At this stage we already have the following computed 10 parameters for each outfit in Train, Eval and Test datasets:

- 8 probability parameters (float) - Choices' fitting probability as computed by both Bi-LSTM+EN_B4 and BertForFashionCompatibility models.
- 2 Indexes parameters (int) - The index of the choice with the highest probability from both models.

The XGBoost Classifier was trained on these 10 features along with the following additional metadata properties for each choice as found in the originally given Train, Eval & Test datasets (jsons):

- Item's number in the outfit's sequence.
- Item's Price.
- Number of likes given by the Polyvore's community.
- Number of views by the Polyvore's community.
- Item's sub-category id

Finally, the following additional features were created (engineered), computed and added to the XGBoost datasets:

1. Items' price rank relative to other items within the outfit.
2. Items' likes rank relative to other items within the outfit.
3. Outfit's parameters:
    - Total Price
    - Total Number of views
    - Total Number of likes
    - Total number of days since the outfit was created (till the time the dataframe data was collected)

The final XGBoost was trained with learning rate of 0.2 and 5,000 estimators.

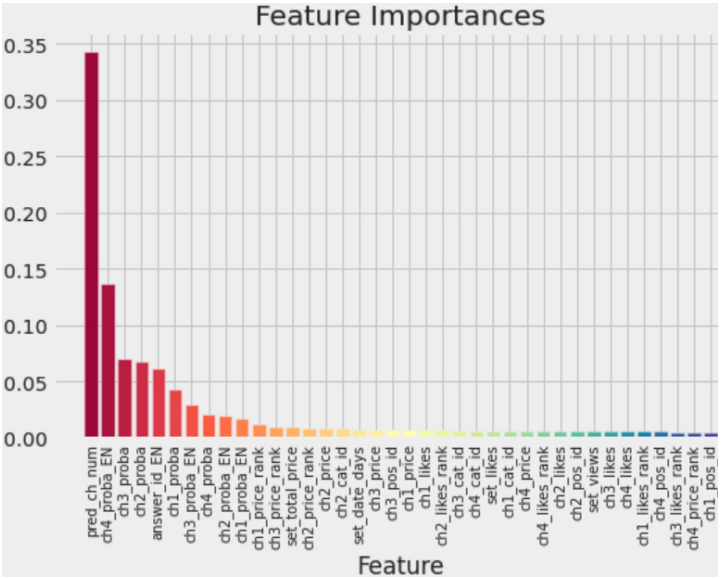

Fig. 14. XGBoost Classifier - Top layer Ensemble model - Feature importance

7. RESULTS & ANALYSIS

In this work we've utilized the Transform Learning ML approach and successfully improved the original SOTA solution we know for the downstream task of Fashion Compatibility.

We've focused on two main use-cases specified in the "Learning Fashion Compatibility with Bidirectional LSTMs" paper, for which concrete numerical performance values were specified, so that we could compare the performance achieved implementing our approach with the work described in the original paper.

7.1. Bi-LSTM_ENB4 Model - Performance & Results Analysis

As mentioned above, training and fine-tuning this model's hyper parameters was very demanding in terms of CPU, GPU and memory utilization, and thus took quite some time to complete.
The final results indicated a strong learning capability compared to the results reported by the original paper.

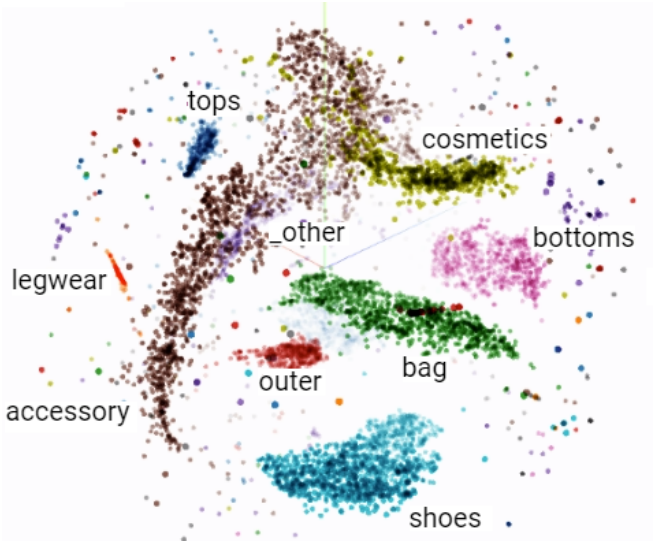

Fig. 15. Bi-LSTM+EfficientNet_B4 For Fashion Compatibility 142K Items Embeddings' Main Categories - tSNE 512dim reduction to 3D-Visualization (colors represent original categories as indicated in table 1)

Figure_15 clearly showcases the strong learning capability of the fine-tuned Bi-LSTM-ENB4 model by figuring out the outfits items' main categories' separation just by learning each item's visual properties and it's location in the ordered outfits it belongs to.

We've managed to utilize EfficientNet_B4 with a Bi-LSTM model on top, pre-trained on FashionMnist, in order to achieve 67.33% accuracy on the FITB use-case, 0.9% improvement over the BiLSTM_InceptionV3 model.

This model achieved 0.90 AUC score on the Outfit Compatibility task which just as high as the best result reported for the BiLSTM_InceptionV3+VSE (based on image & text).
All results are outlined in Table_3

7.2. BertForFashionCompatibility [BFFC] Results and Analysis

Here are the BertForFashionCompatibility model's Confusion Matrix and ROC-AUC on the Original Test dataframe.

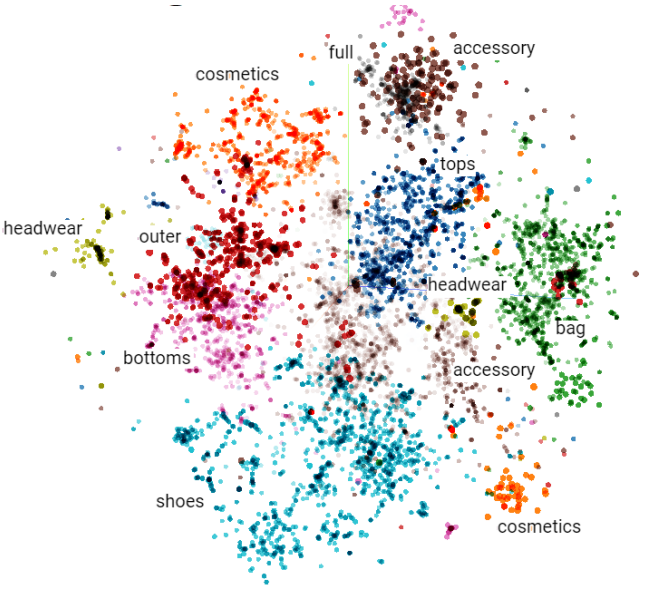

Fig. 16. BERT For Fashion Compatibility 105K Items Embeddings' Main Categories - tSNE 512dim reduction to 3D-Visualization (colors represent original categories as indicated in table 1)

Classification Report:

	precision	recall	f1-score	support
0	0.6601	0.6447	0.6523	774
1	0.6779	0.6808	0.6794	708
2	0.6748	0.6774	0.6761	775
3	0.6799	0.6899	0.6848	819
accuracy			0.6733	3076
macro avg	0.6732	0.6732	0.6731	3076
weighted avg	0.6732	0.6733	0.6732	3076

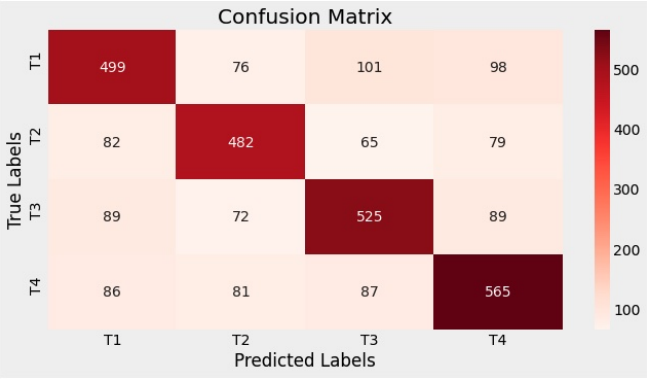

Fig. 17. Bert For Fashion Compatibility model confusion matrix on **ORIGINAL given FITB test**

Figure_16 is a tSNE items' embeddings spatial representation of our model's strong capability to automatically infer on the item's Main Category when not included in

train data. The Items' embedding vectors are computed as the average embedding of the words in the items' name feature.

We've harnessed BERT's strong word semantic learning & embedding capabilities and achieved 67.3% accuracy (ROC-AUC=0.8834), about 130% improvement on the FITB use-case, compared to the performance of the VSE model, based on text only, reported in "Learning Fashion Compatibility with Bidirectional LSTMs" [4].

On the (alternative) Outfit Compatibility use-case we've suggested and implemented with the BertForFashionCompatibility model, we've achieved 0.6 AUC result. Though it is not the same Compatibility mechanism described in the original paper, we consider this outfit compatibility approach to be at least as relevant as the original one, thus these results reflect relatively good performance for the BERT based model as well.

7.3. XGBoost Classifier [Top layer]

As described above, each of the models achieved 67.33% accuracy which 2,071 correct missing item selections.
Comparing the model's successful selections we've found the following:

- There were 1,446 outfits for which both models selected the correct missing item (i.e. same outfits).
- There were other 1,250 outfits for which only one of the models managed to recommend the correct missing item.
- Combining these two outfits groups, the maximum accuracy that could theoretically be achieved utilizing both models is 2,696 outfits translated to 87.6% accuracy.

Using the XGBoost-Classifier model as an aggregator layer of the ensemble model, we've managed to successfully achieve additional 913 correct missing items recommendations, out of the 1,250. This is translated to 2,359 successful missing items selections which is 76.7% from the Test set. That is, the ensemble model managed to exceed each of the individual models' accuracy performance by 13.9% (288 outfits).

7.4. Overall Performance Comparison

Finally, our overall solution, having an XGBoost Classifier model on top of the BiLSTM_ENB4 and BertForFashionCompatibility Models (#6 in Table_3) achieved 12% improvement over BiLSTM_InceptionV3+VSE model (#5 in Table_3) suggested in the original paper.

```
Classification Report:
              precision    recall  f1-score   support

           1       0.788     0.757     0.772       781
           2       0.817     0.708     0.759       764
           3       0.805     0.794     0.799       786
           4       0.677     0.808     0.737       745

    accuracy                          0.767      3076
   macro avg       0.772     0.767     0.767      3076
weighted avg       0.773     0.767     0.767      3076
```

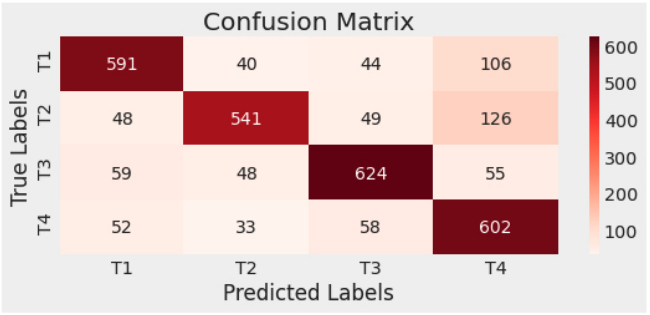

Fig. 18. XGBoost Classifier - Top layer Ensemble model - Confusion Matrix results

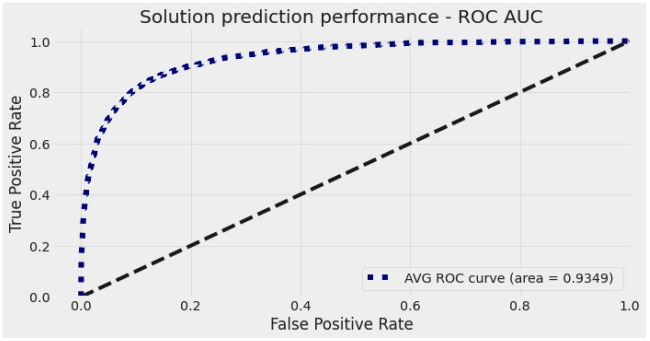

Fig. 19. XGBoost Classifier - Top layer Ensemble model - ROC curve

Performance Comparison on Test Data			
#	Method	FITB Accuracy	Comp. AUC
1	VSE	29.2%	0.56
2	BiLSTM_InceptionV3	66.7%	0.89
3	BertForFashionComp.	67.3%	0.60*, 0.59**
4	BiLSTM_ENB4	67.3%	0.90
5	BiLSTM_InceptionV3+VSE	68.6%	0.90
6	XGBoost + BiLSTM_ENB4 + BFFC	76.7%	[Not applicable]

Table 3. Results Comparison (yellow rows = our work).
*, ** - Our suggested alternative Outfit Compatibility approach. See section 5.2.1 for detailed description.

8. DISCUSSION

This interesting project was our first work utilizing the Transform Learning ML approach. Though it wasn't a simple task as one may expect, we've learned a lot about the amazing capabilities of the BERT and EfficientNet strong NN models.

Even more fascinating is the versatile learning that can be performed combining the EfficientNet image classification model, or the BERT language model, with an LSTM model to accomplish a robust Deep Neural Network based Time Series down-tasks in the image classification or language semantic domains.

Yet, among the more important takeaways from this work is the importance of having enough good clean data as a prerequisite for any successful ML pipeline.

9. REFERENCES

[1] Sepp Hochreiter; Jurgen Schmidhuber, "Long short-term memory," *https://www.researchgate.net/profile/Sepp-Hochreiter/publication/13853244_Long_Short-term_Memory/links/5700e75608aea6b7746a0624/Long-Short-term-Memory.pdf*, 1997.

[2] Olga Russakovsky; Jia Deng, "Imagenet large scale visual recognition challenge," *https://arxiv.org/abs/1409.0575*, 2015.

[3] Han Xiao; Kashif Rasul, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *https://arxiv.org/pdf/1708.07747*, 2017.

[4] Han et al. Xintong Han; Zuxuan Wu; Yu-Gang Jiang and Larry S Davis., "Learning fashion compatibility with bidirectional lstms [https://arxiv.org/abs/1707.05691v1]," *ACM*, vol. In Proceedings of the 2017 ACM on Multimedia Conference, pp. 1078–1086, 2017.

[5] Xintong Han; Zuxuan Wu; Yu-Gang Jiang; Larry S Davis, "Learning fashion compatibility with bidirectional lstms," *https://github.com/xthan/polyvore-dataset#polyvore-dataset*, 2017.

[6] Tianqi Chen; Carlos Guestrin, "Xgboost: A scalable tree boosting system," *https://arxiv.org/pdf/1603.02754*, 2016.

[7] Quoc V. Le Mingxing Tan, "Efficientnet: Rethinking model scaling for convolutional neural networks [https://arxiv.org/abs/1905.11946]," *International Conference on Machine Learning*, 2019.

[8] Vaswani; Ashish; Noam Shazeer; Niki Parmar; Jakob Uszkoreit; Llion Jones; Aidan N. Gomez; Lukasz Kaiser and Illia Polosukhin., "Attention is all you need. [https://arxiv.org/abs/1706.03762]," *In Advances in Neural Information Processing Systems*, pp. 5998–6008, 2017.

[9] Kristina Devlin Jacob; Chang Ming-Wei; Lee Kenton; Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *https://arxiv.org/abs/1810.04805v2*, 2018.

10. APPENDIX

10.1. EfficientNet

Convolutional neural networks (CNNs) are commonly developed at a fixed resource cost, and then scaled up in order to achieve better accuracy when more resources are made available.

In the EfficientNet image classification model, Google showcased a novel model scaling method using a simple yet highly effective compound coefficient to scale up CNNs in a more structured manner. Unlike conventional approaches that arbitrarily scale network dimensions, such as width, depth and resolution, our method uniformly scales each dimension with a fixed set of scaling coefficients.

In their paper "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks [7]" focuses on how to efficiently scale Convolutional Neural Networks (ConvNets) to improve performance. The typical way of scaling ConvNets for improved performance is to increase one of the following: network depth, width or image resolution. Though it is possible to scale two or more of these together it is currently a tedious process and generally leads to models with sub-optimal accuracy and efficiency.

The authors take this one step further and use an algorithm called neural architecture search (NAS) to find an optimum baseline network. The NAS algorithm at a high level uses reinforcement learning to determine optimum structures, given an objective function. Using NAS a new family of models was created called EfficientNets. The performance of these models is compared against ConvNets on the ImageNet dataset and the results are shown in **Figure_[20] Model Size vs. Accuracy Comparison**

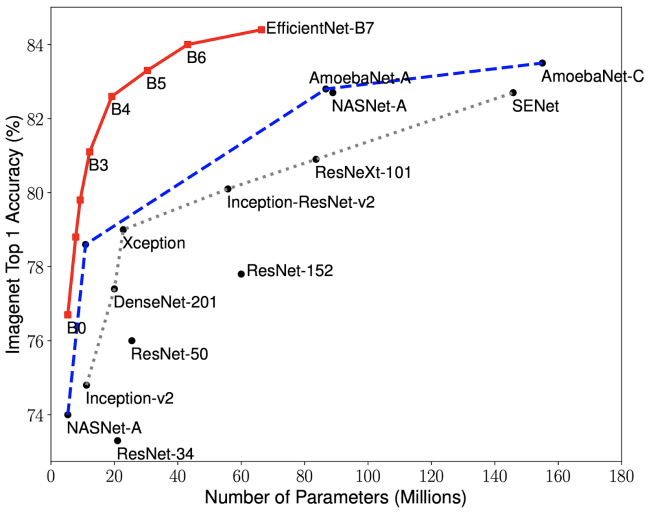

Fig. 20. Google EfficientNet Vs Others - Model Size vs. Accuracy Comparison. EfficientNet-B0 is the baseline network developed by AutoML MNAS, while Efficient-B1 to B7 are obtained by scaling up the baseline network

10.2. Image Datasets used for Transfer Learning

10.2.1. Image-Net

The ImageNet project is a large visual database designed for use in visual object recognition software research.
ImageNet contains more than 14 million images have been hand-annotated to indicate what concepts/objects are pictured and is organized according to the WordNet hierarchy.

Each meaningful concept in WordNet, possibly described by multiple words or word phrases, is called a "synonym set" or "synset" and there are more than 100,000 synsets in WordNet.

ImageNet's creators aim was to provide on average 1000 images to illustrate each synset. It includes more than 20,000 categories, with a typical category, such as "balloon" or "strawberry", consisting of several hundred images.

10.2.2. Fashion Mnist

An open source images dataset consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image, associated with a label from 10 classes.
It was intended to serve as a direct drop-in replacement for the original MNIST dataset for benchmarking machine learning algorithms. It shares the same image size and structure of training and testing splits.

10.3. Transformers

The Transformer [8] model is a type of neural network that accepts a sequence as input (e.g. a sequence of words) and produces some output (e.g. sentiment prediction).

Unlike traditional recurrent networks such as LSTMs, which process each sequence element in turn, the Transformer encoder reads the entire sequence of words at once and processes all elements simultaneously by forming direct connections between individual elements through an attention mechanism.
This characteristic allows the model to learn the context of a word based on all of its surroundings (left and right of the word), enables greater parallelization and, most importantly, results in higher accuracy across a range of tasks.

10.4. BERT

Giving machines the ability to understand natural language has been an aspiration of Artificial Intelligence since it's early stages. The Turing Test — one of the earliest suggested measurement of machine intelligence — is based on a machine's ability to converse in natural language.
In recent years a huge leap forward was achieved in the field of Natural Language Processing (NLP) in the shape of new deep learning models that have dramatically improved the ability of machines to understand language, resulting in large performance increases across NLP tasks ranging from sentiment analysis to question answering.

Among the most famous language understanding deep neural network models is BERT [9] (Bidirectional Encoder Representations from Transformers), which is a model pre-trained over a large corpus namely all of English Wikipedia and 11,000 books.

BERT is built on two major concepts in the field of NLP - **Transfer Learning** and the **Transformer model** and was trained for two main related tasks:

I. Masked Language Modeling - predicting a missing word in a sentence.
Before feeding word sequences into BERT, 15% of the words in each sequence are replaced with a [MASK] token. The model then attempts to predict the original value of the masked words, based on the context provided by the other, non-masked, words in the sequence.

II. Next sentence prediction - predicting if one sentence naturally follows another. In this use-case, the model receives pairs of sentences as input and learns to predict if the second sentence in the pair is the subsequent sentence in the original document.

To help the model distinguish between the two sentences in training, the input is processed in the following way before entering the model:

1. A [CLS] token is inserted at the beginning of the first sentence and a [SEP] token is inserted at the end of each sentence.
2. A sentence embedding indicating Sentence A or Sentence B is added to each token. Sentence embeddings are similar in concept to token embeddings with a vocabulary of 2.
3. A positional embedding is added to each token to indicate its position in the sequence.

To predict if the second sentence is indeed connected to the first, the following steps are performed:

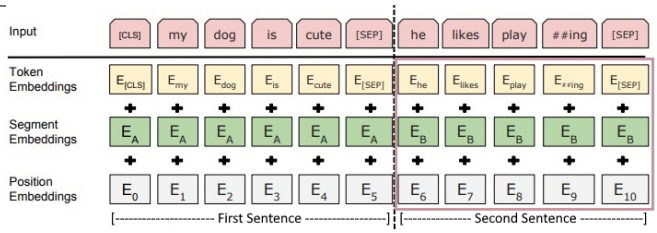

Fig. 21. Source: BERT [Devlin et al., 2018][9], with modifications

1. The entire input sequence goes through the Transformer model.
2. The output of the [CLS] token is transformed into a 2×1 shaped vector, using a simple classification layer (learned matrices of weights and biases).
3. Calculating the probability of IsNextSequence with softmax.

As such, BERT, as a rich pre-trained model, comes to any downstream task with a profound understanding of the workings of the English language.

10.5. Transfer Learning

An efficient approach for utilizing a large pre-trained model for prediction on a downstream task. That is, train a model in one domain, and then with minor adjustment and fine-tuning (if at all), infer in another domain leveraging the pre-acquired knowledge.