Deep Learning for Computer Vision

Mazzetti Francesca 3069839, Luijten Michelle xxx, Scarnera Michele 3081696, Vaccarezza Gaia 3073561

❖ Introduction

The goal of the project is to predict the apparel class of each image of the iMaterialist dataset. To reach it, we explored different models.

❖ Dataset

We chose the Product Attribute Recognition task as our project, using the 'iMaterialist Challenge at FGVC 2017' dataset, which is available in Kaggle.

The project code was written in Python, using version 3.10. We used an Azure Virtual Machine, in addition to our personal computers, to run more complex models that needed high computational power in less time.
The code to download the dataset was provided. Nonetheless, we modified it to be sure that we only imported the uncorrupted and still available images. The code works as follows: if some error occurs for any reason when trying to download an image, then this image is not saved but the iteration proceeds directly to the next one. We did this to get the cleanest and most comfortable-to-use dataset possible.

The dataset is made of images of apparel, divided into four main classes: shoes, dresses, pants and outerwear. Each image of the training dataset contains two pieces of information, the first indicating the apparel class to which the image belongs and the second indicating the labels for one or more tasks. Labels indicate a specific characteristic of the image (i.e. the colour of the shoes can be black, white, etc.) while the tasks are groups of labels with the same purpose (i.e. black, and white are part of colour). The tasks are either specific to a given class (i.e., heel type for shoes) or common to multiple classes (age, colour, gender and material). There are a total of 45 tasks, of which 14 belong to shoes, 11 to dresses, 10 to pants and, finally, 10 to outerwear. The tasks, in turn, have 381 different attribute labels. The distribution of the labels for tasks is not equal. For example, the task shoe: colour has 68 possible attribute labels, while gender has two.

In total, there are 50461 images in the given dataset, of which 42029 belong to the training set and 8432 belong to the validation set. After the selection of the images with information over age, colour, gender or material, the images we used were a little more than half of the total, 27642, of which 24642 images are from the training set and 3000 from the validation set. The training set was used to train the models and to teach them how to correctly classify the images in the various classes, by making them observe many examples of the pair image labels. The validation set, on the other hand, was used to verify the prediction performance of the models on data whose correct class was already known but the model had never been seen. Moreover, it was very useful to detect cases of overfitting.
The training set is divided into 7555 images (30.7% of the total) of dress, 7343 (29.8%) of outerwear, 5378 (21.8%) of pants and 4366 (17.7%) of shoes. The validation set is made of 1128 images (37.6% of the total) of dresses, 765 (25.5%) of outerwear, 594 (19.8%) of pants, and 513 (17.1%) of shoes.
It is worth noting that the frequency order of the various apparel classes is the same between the training set and the validation set. There are some changes in the percentages, but, in the end, the two sets have comparable distributions.

The amount of images per label for each task is very unevenly distributed. For example, in the training set colour has labels that are present 24 times while others are 368 times.

One of the main difficulties was using a dataset with so many possible labels per task, some of which are, in practice, very similar and difficult to distinguish, even for humans. Secondly, the uneven distribution of images across labels of the same tasks was a challenge. Finally, there was an obstacle common to all Computer Vision projects, having more images available could have helped our model obtain better performance.

- ❖ Theory

We decided to focus on theory before designing our models because we believe that the structure of a neural network should be grounded on solid theory. Guessing the structure and proceeding only based on results would, in our opinion, lead to a wrong and not generalizable model.

We hypothesised that the information contained in the other tasks could help predict apparel class more accurately. One way of exploiting this information is multi-task learning. Multi-task learning (MTL) is a subfield of machine learning in which multiple tasks are learned jointly, exploiting similarities and differences across tasks. That what is learned for each task may help other tasks be learned better. This can result in improved learning efficiency and prediction accuracy for the task-specific models when compared to the model that was only trained on the main task (apparel class).
We can motivate multi-task learning by taking for example The Karate Kid (1984). In the movie, sensei Mr Miyagi teaches the karate kid seemingly unrelated tasks such as sanding the floor and waxing a car. In hindsight, these, however, turn out to equip him with valuable skills that are relevant to learning karate.
We applied multi-task learning by making 2 or more tasks sharing the hidden layers while keeping several task-specific output layers. The apparel (the main task to predict) is always present while the other tasks are added to the model in turn. The benefit of doing so, and so the reasons why multi-task learning may work, are several. Firstly, this sharing greatly reduces the risk of overfitting. This makes intuitive sense: learning just the apparel bears the risk of overfitting while learning apparel and another task jointly force it to find a representation that captures all of the tasks and the less is our chance of overfitting on our original task. Secondly, if a task is very noisy, it can be difficult for a model to differentiate between relevant and irrelevant features. MTL can help the model focus its attention on those features that matter as other tasks will provide additional evidence for the relevance or irrelevance of those features. Third, some patterns may be easy to learn for some tasks, while being difficult to learn for apparel. This might either be because apparel interacts with the features in a more complex way. Through MTL, we can allow the model to eavesdrop.

- ❖ Preprocessing of data

To reduce the risk of overfitting we decided to use data augmentation. This means that we applied several random transformations to be performed on the images such as rotation (of 45 degrees), width and height shift (of 0.25 of the size), horizontally flipping half of the images and zoom (of 0.2). Moreover, we rescaled the images, already decoded into grids of pixels and converted them into floating point tensors, from [0, 255] to [0,1].

- ❖ Training

Before describing the neural networks, we describe the hyperparameters of the training phase.
We defined the batch size to be 128. At first, we designed training of 60 epochs but after the first run,

we understood that it was not necessary to have so many of them so we chose to use 20 epochs. Still, we used an early stopping method that breaks the training when the validation loss has an absolute change of less than 0.001 for more than $int(max([6,\ 1 + \sqrt{epochs}]))$ epochs. As we used 20 epochs for all training, and the patience is 6 epochs. Yet we decided to apply this more general formula for future use of the model because, in our opinion, it is important to be more patient as the number of epochs increases.

About our models, we decided to use Convolutional Neural Networks because they are specialised in processing data that has a grid-line topology. Therefore, as the scope of the project is to classify images, that can be represented as a 2D grid of pixels, they are the best choice for our purposes.
Each model is trained 5 times, each training from scratch: the first time only with apparel and the others with 2 tasks: apparel + one of the tasks common to all apparel classes (colour, gender, age and material). As we already said, we decided to use multi-task learning which in this case means choosing the tasks that are common across all 4 apparel classes and adding one of these tasks at a time to see if by learning it the neural network becomes more accurate in predicting the apparel class. The common classes are colour, material, size and age.
As colour and material are characterised by many labels, not equally present we decided to group the labels that were present less than 1% of the time in a unique label called "other". It is important to note that as the majority of the images have just 2 labels, one for apparel and one for another task, the labels of the missing tasks are coded as -1.

- The Model of the Private Layers with 1.4 millions parameters

The first model was called "the private" due to its layers that are private for each task. Therefore, Multi-task learning is not applied to it. We built it to make comparisons to models where MLT is applied and understand whether our theory is correct. Indeed, we trained it with the same approach as the other models (5 times with apparel only and apparel + 1 common task). In this way, the difference in outcomes can be only driven by the structure of the neural network.

The model starts with an input layer followed by three unstrided convolution layers with 8,16 and 32 filters and kernel size (4,4). All of them use ReLU as activation function and apply padding to keep as much information as the original input as possible. After each convolution layer, there is a max pooling layer of size (4,4) to reduce the number of parameters and control overfitting. The third max pooling layer outputs a 3D tensor that needs to be flattened to be accepted as input by the dense layer, so we turn it into a vector. Next, the model has a dropout layer with a rate of 0.5, which means that it randomly sets the 50% of the features to 0. The reason is that the model has a lot of parameters and this layer may reduce overfitting. After, there is a common 128-wide fully connected (dense) layer with tanh activation function followed by two dense layers that are specific for each task in the model. The wideness of the latter is proportional to the number of labels that each task has. In particular, they are 16*(number of labels) and 8*(number of labels) wide with a ReLu activation function. The reason for this choice is that we believe that if a task has more labels, it requires more complexity than one with fewer labels. Finally, the model is completed by an n-wide output layer that uses soft-max as an activation function. These n values are the probabilities that the model assigns to each label. We predict the label by taking the one with the highest probability.

- The Model of the Public Layers with 1.4 millions parameters

Its structure is identical to the previous one until the Dropout Layer. After it, the model is composed of two layers with ReLU activation functions, that are shared by all the tasks. They are 160 and 126

wide. After that, there are the same 128-wide and n-wide dense layers of the last model. Therefore, while the “private” or “task-specific” layers disappear, two “public” layers are introduced. The reason for this choice is to have a model that suits better for multi-task learning. Indeed, with this “public” layout, the randomised weights in the NN always converge to what the apparel class tells them to go. Moreover, every other task now has a less-than-random weights setup to tweak every time it gets the chance. With private layers, instead, tasks are left on their own with no help from the apparel class' tweaking.

- The Model of the Public Layer with 3m parameters

In this model, we increased the number of parameters to check if a larger neural network would give more accurate results for the smaller neural network. As the “public layers” neural networks performed better than the “private layers” ones we increased the parameter only in the first. Therefore, the model has the same structure as the previous one, with the only difference being that the two 160 and 126-wide dense layers become both 384-wide.

- The Model of the Public Layer with 3m parameters - Apparel + 3 other tasks

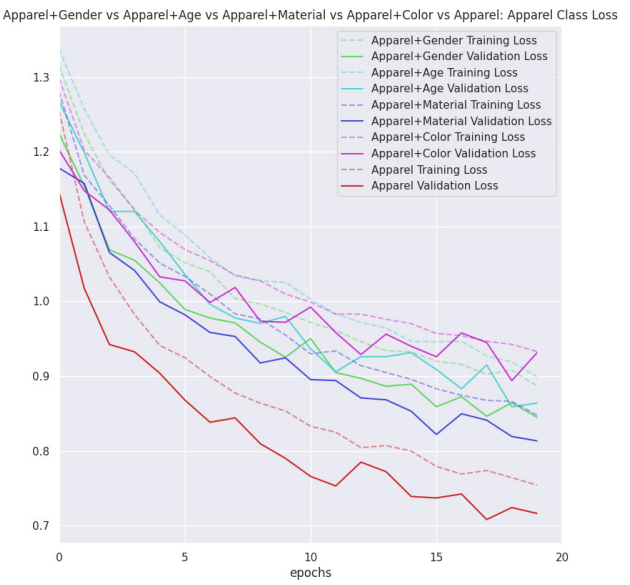
We decided not to train this class of neural networks due to the unsatisfactory results gotten during the training of the previous model. Yet, we would have added two tasks at a time to the main apparel model in the same way we did when adding one task at a time.

❖ Model Evaluation

To decide which model is the best for predicting the apparel class we used different metrics. In this section, we discuss only some comparisons between the models we trained and tested due to space constraints: the “public” 3 millions parameters model with 1 task (apparel) compared to the same model with 2 tasks (apparel + other). We leave out the “private” 1.4 millions parameter models vs the “public” 1.4 millions parameters model and the latter vs the “public” 3 millions parameters models. The “public” 3 millions parameters perform better than the other two. Note that the logic and measures we explain in the next lines are the same ones we used for the comparison we do not discuss.

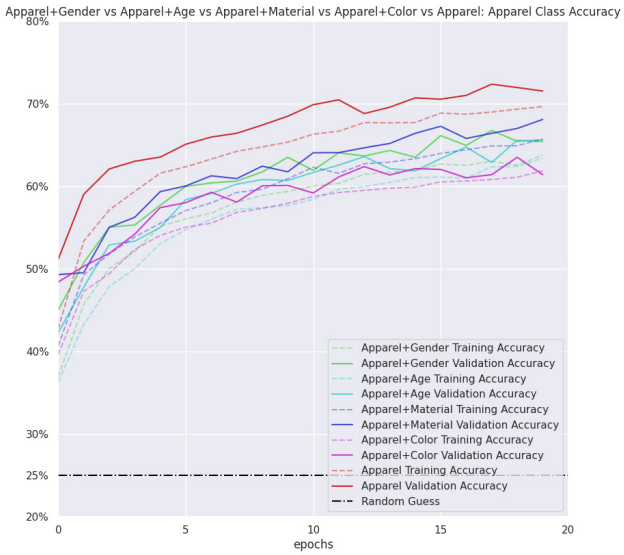
- Validation Loss

The general loss was optimised using the Adam algorithm which is a stochastic gradient descent method that is based on adaptive estimation of first-order and second-order moments as it is computationally efficient, has little memory requirement and is well suited for problems that are large in terms of parameters. We decided to set up weights for the different tasks so that the apparel is the most important (weight=1) while the others have less impact (each subsequent task has 0.95 times less weight). In the context of wanting to predict more than one task, as a loss function, we used a function that can handle more categories at a time, the Sparse Categorical Cross Entropy function. It works in the same way as the Categorical Cross Entropy function, the only difference is in how the targets/labels should be encoded. This function will ignore the label when it has the value -1. Being interested in predicting the apparel class of each image, we took our decision to focus on the loss values of the apparel class.



We used validation loss compared to training loss for two reasons. Firstly, to understand if a model was characterised by overfitting, meaning that it has low bias and high variance and so it is not able to be generalised once it predicts using data different from the training set. Overfitting can be spotted by plotting validation and training loss and checking if the training loss continues to decrease as epochs pass while the validation loss has stopped. Thanks to several methods we used to prevent overfitting, none of our models shows this issue. Secondly, as we aim to minimise the loss, a model is considered better than the others when it has a lower validation loss (but does not overfit). The Apparel Only model is the one with lower validation loss.

	Apparel	Apparel+Material	Apparel+Colour	Apparel+Gender	Apparel+Age
Loss	0.716	0.813	0.931	0.844	0.863



- Validation Accuracy

As we used the Sparse Categorical Cross Entropy as a loss function we used Sparse Accuracy as a measure of accuracy. Accuracy calculates how often predictions match integer labels, focusing on the true positive and true negative. The model with higher validation is the Only Apparel neural network with 0.715, followed by the Apparel and Material one. Yet, accuracy is generally used when the class distribution is similar while the F1-score is a better metric when there are imbalanced classes. Our dataset is quite imbalanced and thus, we also took into account the F1-score to evaluate our model and take the final decision.

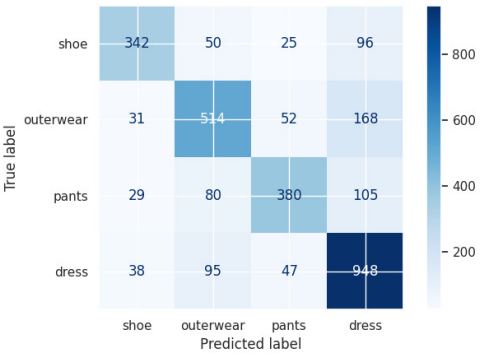
	Apparel	Apparel+Material	Apparel+Colour	Apparel+Gender	Apparel+Age
Accuracy	0.715	0.68	0.614	0.656	0.653

- Precision, Recall, F score

Thanks to the confusion matrix we calculated several important measures: Precision, Recall and F scores for each label of the apparel task for each model, and average and weighted F scores for each model. On the right, there is the confusion matrix of the Only Apparel model.

Precision is the ratio of true positive to the total predicted positive observations (true positive + false positive). The question that this metric answer is of all images that we predicted as dresses, how many actually dresses? For example, the precision of dresses in the apparel-only model is 0.72 meaning that around ¾ of the photos that our predictor classifies as dresses are actually dresses.

Instead, recall is the ratio of true positives to all observations in the actual class (true positive + false negative). The question recall answers are: of all the images that dress, how many did we label as dresses? For example, the recall of dresses in the apparel-only model is 0.84, meaning that this neural network classifies more than 4/5 of the dresses' images as dresses.



Yet, as it is not simple to have a correct decision with so many values we also calculated the F1 score per label of apparel, which is a weighted average between precision and recall. It is weighted because it uses the harmonic mean instead of the arithmetical one, giving more weight to lower values. For example, if a neural network has both precision and recall equal to 0.8, while another one has a precision of 0.6 and recall of 1. Arithmetically, the mean of the precision and recall is the same yet, the F1 for the first is 0.8 while for the second only 0.75, leading to the choice of a model that is both precise and sensitive for that label.
As we want to make decisions based on precision and recall, we calculated the average of the F1 scores. There also exists a weighted F1 score, which is the average weighted by the number of samples from that class. In general, if one is working with an imbalanced dataset where all classes are equally important, using the first F1 score is a good choice as it treats all classes equally, and so we did. The model with a higher average F1 score is the Apparel Only one with a 0.75 score and 0.05 to 0.11 points more than the others.

	Apparel	Apparel+Material	Apparel+Colour	Apparel+Gender	Apparel+Age
Range Precision	0.70-0.78	0.64-0.75	0.54-0.73	0.58-0.75	0.61-0.74
Range Recall	0.64-0.84	0.63-0.78	0.45-0.65	0.48-0.81	0.52-0.83
Average F1 score	0.72	0.67	0.61	0.63	0.64

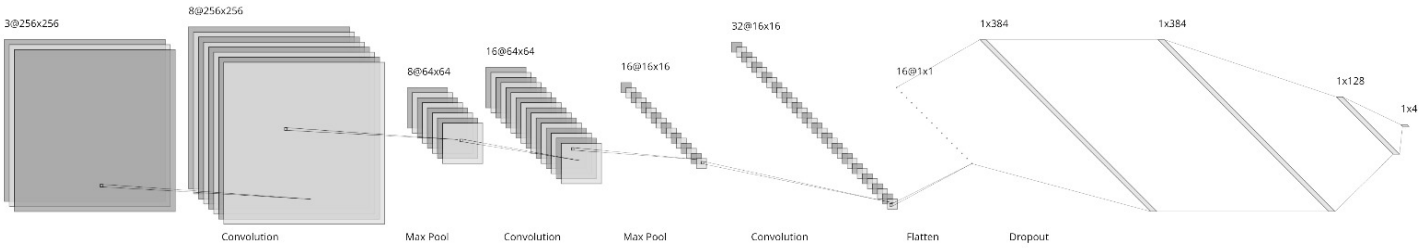
It is important to note that the F1 score gives equal importance to precision and recall. Yet, in real-life problems, this is not the case. For example, if the classifier needs to predict whether a person is healthy or not, it may be more important to detect as many unhealthy people as possible between those that have a disease (recall becomes more important). For our neural network, this is not the case and so we used F1 score.

- Conclusion

The Only Apparel model with “public” layers and 3 millions parameters has both the higher apparel validation accuracy, the higher average F1 score and the lower validation loss. Therefore, we chose this one as the best model. This means that, in this case, Multi-Task Learning has not had a fruitful effect, as learning jointly other tasks together with Apparel does not improve the goodness of the model with respect to learning Apparel only.

❖ Limitation and Future Work

Due to time and search cost constraints, we decided to choose hyperparameters and models using theory, logic and common sense, instead of trying every possible combination. Yet, we would like to present some other ideas that would have been interesting to try. Firstly, the hyperparameter selection may be done using the validation set to optimise both the iteration of early-stopping and the accuracy at early-stopping. Secondly, it may be useful to try other models, regularisations, pruning and pre-trained models. Thirdly, we would like to explore an approach to dealing with tasks with a lot of labels. One approach would be to group similar labels manually. For example, map navy blue, light blue, and other shades of blue into one unique blue label. Something that might be less error-prone, and ultimately more scalable if done right, is to properly set up a feature space where those groupings can happen naturally. For instance, training a variational autoencoder with a feature space that’s expressive enough should be able to achieve this.


3@256x256
8@256x256
8@64x64
16@64x64
16@16x16
32@16x16
16@1x1
1x384
1x384
1x128
1x4
Convolution
Max Pool
Convolution
Max Pool
Convolution
Flatten
Dropout