Aesthetic Classification of Outfits Using Machine Learning

Maia Nkonabang Nitya Kashyap

Abstract

This project uses machine learning to classify outfit images into aesthetic categories (e.g., Y2K, academia, punk). We scraped over 14,000 Pinterest-sourced images via Google Images, preprocessed and labeled them across 10 style categories, and tested multiple classification approaches. Our baseline method used multiclass Support Vector Machines (SVMs) with handcrafted features such as color histograms and Local Binary Patterns (LBP). We trained a convolutional neural network (CNN) from scratch using PyTorch for our main models. Also, we experimented with transfer learning via a pre-trained ResNet backbone connected to an MLP head. Evaluation across training, development, and test splits shows that the pretrained CNN outperforms the SVM baselines and the basic CNN in accuracy and generalization. We also identified key challenges around overlapping aesthetics and lack of computer resources. All code and data used in this project are publicly available. GitHub repository: https://github.com/nityakashyap7/outfit-aesthetic-classifier.git. Dataset DOI: https://doi.org/10.5281/zenodo.15164901. Commands to reproduce results:

python3 baseline.py Runs SVM baseline

python3 cnn.py cnn Runs standard CNN

python3 pretrained_plus_mlp.py Runs pretrained model with MLP

1. Introduction

Our project is focused on developing a machine-learning model that labels images of outfits into aesthetic categories such as bohemian, Y2K, punk, and academia. The model will take an image of a person wearing an outfit or with their clothes laid flat and produce the most similar aesthetic label. Initially, we considered building an outfit generator, but because of the scope of our coursework, we decided to classify using a neural network as a more feasible approach. We automated a scraping script and collected a vast data set of over 14,000 images from Pinterest.com, divided into 10 fashion categories. We preprocessed the photos and experimented with a few different classification approaches, including a baseline using multiclass support vector machines (SVMs) and a PyTorch convolutional neural network (CNN). We experimented with different feature vectors for the SVMs, e.g., local binary patterns and image color histograms, and tried both One-vs-One and One-vs-All configurations. Our final CNN outperforms most baselines in development accuracy, and we hope to continue to improve its architecture.

This project addresses the real-world challenge of fashion aesthetic identification and organization, enabling users to recognize their style better and match similar looks online. It has direct applications in trend analysis, personal styling, and fashion search functions. Throughout this project, we have preprocessed the dataset, improved our CNN architecture, and transferred learning from pre-trained vision models to enhance its performance.

2. Related Work

There has been a lot of machine learning research in this field. One notable study by Ziegler et al. (1) used the DeepFashion dataset with an elastic warping rotation-invariant model to improve clothing category classification and fashion landmark detection. However, a key limitation was that elastic warping—where an image is stretched and distorted—could negatively impact results due to the dataset's limited size. Our project does not augment images this way; instead, we scraped over a thousand images for almost every category so our dataset would not be as limited.

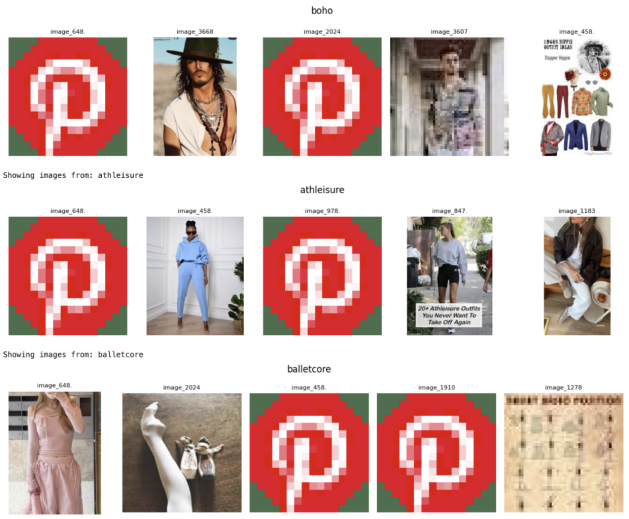
Similarly, Jain and Kumar (1) leveraged the DeepFashion dataset with multiple models, including Decision Trees, Naïve Bayes, Random Forest, and Bayesian Forest, to build a classification system for clothing categories and subcategories. Their approach, however, was limited by the labor-intensive nature of manually labeling internet-sourced data. Our approach had similar limitations as scraping takes hefty computer resources, and we were limited by the machines we had.

Another project by Jo et al. (1) developed a vector-based

user-preferred fashion recommendation system using a CNN model and a private dataset. However, despite its accuracy and precision, reliance on text-based search methods posed challenges since fashion recommendation heavily depends on clothing design. In our case, we categorized clothing aesthetics into different labels under different folders to better utilize the supervised learning method of CNNs without heavily relying on text.

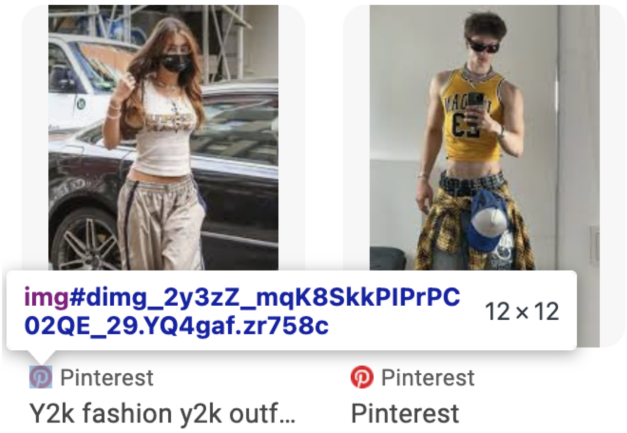
3. Dataset and Evaluation

After doing a Google search for pre-existing datasets, we found none that corresponded to our specific classification task of fashion aesthetics. Thus, we compiled our custom dataset by scraping Google Images using Selenium. Initially, we intended to scrape off more fashion-centered websites like Pinterest directly, but this would have added the issue of automated authentication before searching and scraping. Google is generally known to be more bot-friendly. We got around this issue by specifying *site:pinterest* within the search query. In the first version of the dataset, we had over 31,000 images. Here is a visual sample of what we found:

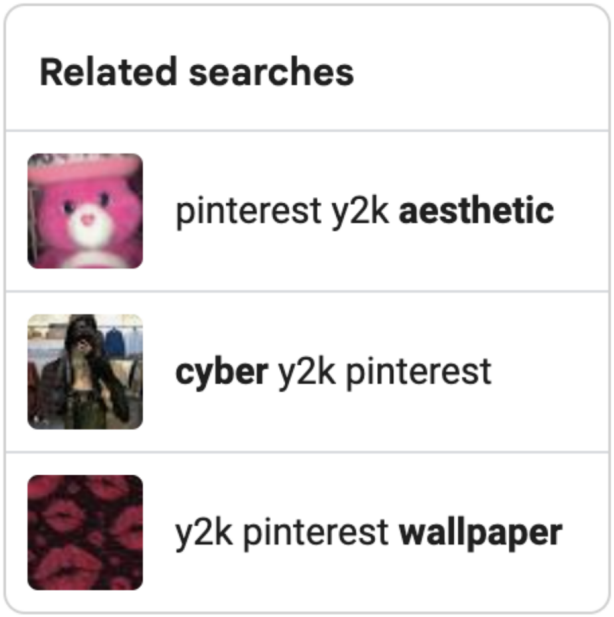


This reveals two glaring issues:

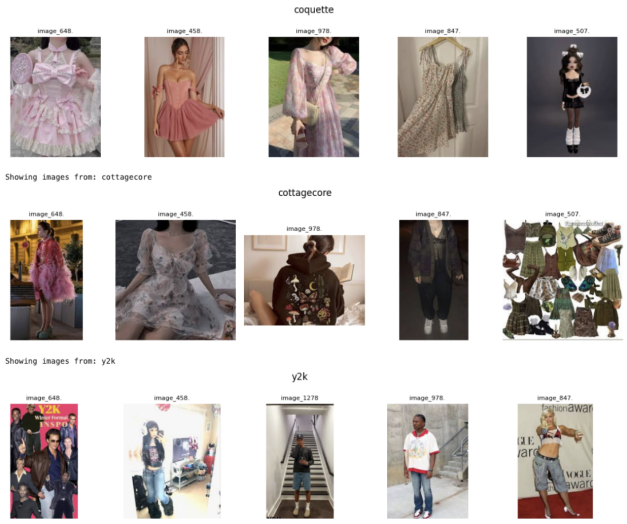
1. Many scraped images were just the Pinterest logo (6 alone in this random sample of 15). This indicates that the scraping script was not specific enough to filter out the correct images for download. Each search result had a tiny Pinterest logo below the image we were looking to scrape.



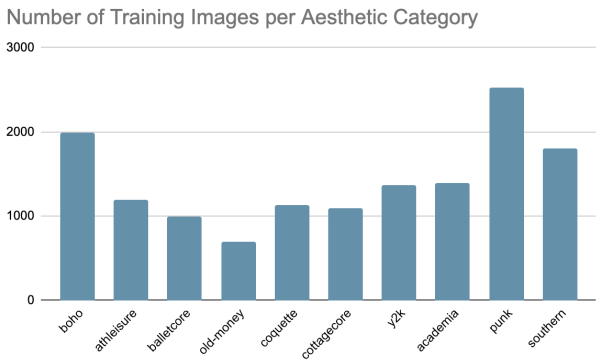
2. 2 of the 15 image samples were too blurry even for us to make out the outfit. We hypothesized that these came from other thumbnail images that were not part of the search results like the ones below:



We modified the scraping script to only download images greater than 100x100 pixels to fix this. This seemed to fix the issue; upon previewing the contents of version 2 of the dataset, we found clearer images without the Pinterest logos:



This refined dataset has 14,212 images, only about half of what we had previously. This makes sense because, for every image, the original scraping file included a corresponding Pinterest logo and a few extraneous thumbnail images. Here is the final dataset split of all our categories, which is slightly imbalanced:



We split the images into 8,527 for the training set, 2,842 for the development set, and 2,843 for the testing set. We used macro and weighted F1 scores for the baseline and accuracy for the CNN to evaluate the model's performance.

4. Methods

To solve the task of fashion aesthetic classification, we implemented two main approaches: a traditional machine learning baseline using SVMs, and a neural network-based classifier using a convolutional architecture in PyTorch. Both methods take as input `.npy` images scraped from Pinterest, each labeled by aesthetic (e.g., *southern*, *Y2K*, *punk*). All images were resized to 224×224 to maintain a uniform input dimension across methods.

4.1. Baseline: Support Vector Machine (SVM) with Handcrafted Features

Our baseline uses a multi-class Support Vector Machine (SVM) implemented via scikit-learn's `SVC` function. We evaluated both One-vs-One (OvO) and One-vs-All (OvA) classification schemes using an RBF kernel with regularization parameter $C = 0.5$. We split the dataset as previously stated in the last section. Checkpointing was implemented using `pickle` to save trained models and avoid retraining on each run.

The features used for the SVMs were handcrafted color histograms. Each `.npy` image was first resized to 64×64 using anti-aliasing to reduce computation while preserving detail. Images were checked for color format and converted to RGB if necessary. We computed 32-bin histograms for feature extraction on each color channel (Red, Green, and Blue) using `numpy.histogram`. The histograms were normalized independently and concatenated into a single 96-dimensional feature vector representing the image's color distribution. This flattened histogram vector served as the input to the SVM.

We also experimented with other features:

- **Local Binary Patterns (LBP)**: Encodes texture by thresholding surrounding pixel values, capturing edge-like structures.
- **Histogram of Oriented Gradients (HOG)**: Extracts gradient-based features useful for shape and texture representation.
- **Raw Byte Vectors**: Converted raw image bytes into flat vectors, which performed poorly due to encoding noise.

Ultimately, color histograms yielded the best F1 scores in our baseline experiments, likely due to aesthetics being strongly correlated with color schemes (e.g., pastel for *coquette*, black-heavy for *punk/goth*).

4.2. Main Method: Convolutional Neural Network (CNN) in PyTorch

We implemented a custom CNN using `torch.nn.Module` for our main method. Each `.npy` image was resized to 224×224 and flattened before being reshaped back into a 2D input for the network. The CNN architecture consists of:

- A `Conv2D` layer with 5 filters and a kernel size of 3
- A `ReLU` activation function
- Two `MaxPool2D` layers for spatial downsampling

- A fully connected (`Linear`) hidden layer with dimension 200
- A `Dropout` layer (initial probability = 0.05)
- A final output `Linear` layer producing logits over 10 classes

The image flows through this network as follows:

```
Conv2D → ReLU → MaxPool → MaxPool →
Flatten → Linear → ReLU → Dropout →
            Output Layer
```

After the two pooling layers, the spatial dimension is reduced to 55×55, yielding a final flattened feature size of $55 \times 55 \times 5 = 15{,}125$. This is passed into the hidden layer. The loss function used was `CrossEntropyLoss`, and the optimizer was stochastic gradient descent (SGD) with a learning rate of 0.1. We trained the model for 30 epochs with a batch size of 32. To prevent retraining, checkpointing was added at the end of every epoch, saving the model's state dictionary and optimizer state.

This CNN is relatively simple, but its performance already surpassed the baseline. It captures spatial relationships and higher-order visual patterns that the handcrafted SVM features will likely miss. In the next section, we improve this for our final method.

4.3. Improved Main Method: Pretrained ResNet with MLP Head

To improve upon our earlier CNN architecture, we implemented a more advanced pipeline using a pretrained ResNet50 model as a fixed feature extractor, followed by a custom multilayer perceptron (MLP) head. This approach leverages transfer learning to better encode complex visual patterns associated with fashion aesthetics, which the simpler CNN struggled to generalize across.

We designed a preprocessing pipeline tailored for the pretrained model. Each image saved as a `.npy` array was checked for proper channel formatting (ensuring 3-channel RGB). We then applied a `torchvision` transform that resized the shorter image side to 256 pixels, center-cropped to 224×224, converted the image to a tensor, and normalized using ImageNet statistics (mean = [0.485, 0.456, 0.406], std = [0.229, 0.224, 0.225]).

A custom PyTorch `Dataset` class was used to apply these transforms on-the-fly. Images were passed through a pretrained ResNet50 model with the classification head (`fc`) replaced by an `Identity()` layer, producing 2048-dimensional feature vectors. These were saved and used to train an MLP classifier.

The final classifier, implemented in PyTorch, has the following architecture:

- A `Linear` layer from 2048 to 512, followed by `ReLU`
- A `Dropout` layer with probability 0.2
- A second `Linear` layer from 512 to 128, followed by `ReLU`
- Another `Dropout` layer (0.2)
- A final `Linear` layer from 128 to 10 for class prediction

We trained the model using cross-entropy loss and stochastic gradient descent (SGD) with a learning rate of 10^{-2}, batch size of 32, and 50 epochs. Early stopping was guided by the development set's macro F1 score, with the best performing model checkpoint saved and reloaded after training.

This improved method directly addresses the limitations of our earlier CNN. By using ResNet50's robust, pretrained encoder, we eliminated the need for our model to learn low-level visual features from scratch—allowing it to focus on more domain-specific patterns. Training was also faster and more stable due to using fixed features. While we kept the ResNet50 layers frozen to manage memory and runtime constraints, future work could explore unfreezing later layers for fine-tuning, given sufficient compute resources.

5. Experiments

Our evaluations compared performance across the baseline SVM classifiers and the CNNs using standard classification metrics such as macro and weighted F1 scores and accuracy on the training, development, and test splits.

5.1. Baseline Experiments

Color histograms consistently performed best among the features tested for the SVMs. These features are likely effective because aesthetics such as *coquette*, *punk*, and *cottagecore* often correlate with dominant color palettes, which histograms can capture well. Histogram of Oriented Gradients (HOG) performed second best, likely due to its robustness to lighting and its ability to distinguish textures and shapes. Local Binary Patterns (LBP), which focus purely on texture, performed the worst, indicating that color information plays a more important role in aesthetic classification than texture alone.

We also tested a combination of color + texture features (concatenating histogram and LBP vectors), but the results were lower than color alone. This may suggest redundancy or interference in the combined representation. The SVM classifier's One-vs-One (OvO) and One-vs-All (OvA) variants

yielded nearly identical scores, indicating that the classification strategy did not significantly impact class separability.

Hyperparameter Tuning. We experimented with several values of the regularization parameter $C \in \{0.1, 0.5, 1.0\}$ and gamma values for the RBF kernel. The best-performing configuration was $C = 0.5$ with `gamma` set to `auto`.

Method	Macro F1	Weighted F1	Vectors
LBP	0.1018	0.1331	8470
Color Histogram	**0.2320**	**0.2744**	8101
Color + Texture	0.1429	0.1808	8383
HOG	0.1910	0.2333	8479

5.2. CNN Experiments

The first convolutional neural network (CNN) somewhat outperformed the SVM baselines regarding the development macro F1 score. While the quality of features limits the SVM, the CNN learns directly from the raw pixel data, allowing it to capture abstract visual patterns that are hard to hand-engineer. This includes subtle cues like outfit layering, silhouette shape, and fabric detail — aspects important to fashion that histogram-based features cannot capture. However, it fell short because this CNN was not complex enough for fashion-based classification.

During training, we experimented with several hyperparameters:

- **Learning rate:** Best result with 10^{-2}.
- **Dropout:** An initial dropout of 0.01.
- **Hidden layer dimension:** Fixed at 200.
- **Epochs and batch size:** 10 and 30 epochs with batch size of 32.

Increasing dropout to 0.1 and 0.2 did not improve the accuracy for the CNN much– development accuracy only increased by 2%. Increasing the learning rate from 10^{-3} to 10^{-2} improved the second CNN's development accuracy from 0.16 to 0.65 for 10 epochs. Below is the final comparison between the CNNs.

Run	Train Acc	Dev Acc	Dev F1
CNN 1.0 (10 ep)	0.167	0.178	0.030
CNN 1.0 (30 ep)	0.883	0.267	0.235
CNN 2.0 (10 ep)	0.684	0.655	0.630
CNN 2.0 (30 ep)	0.795	**0.670**	**0.645**
CNN 2.0 (50 ep)	0.871	0.687	0.663

The results clearly show that the second CNN model—ResNet50 with a custom MLP head—outperformed both the baseline and the original CNN across all metrics. This performance gain is likely because ResNet50 was pretrained on a large and diverse dataset (ImageNet), allowing it to extract high-level visual features that are useful for identifying fashion elements associated with different aesthetics (e.g., denim for Y2K or flowy white skirts for cottagecore). Additionally, incorporating a tailored 3-layer MLP head helped adapt the pre-trained features to our specific dataset, leading to further accuracy and F1 score improvements.

6. Discussion

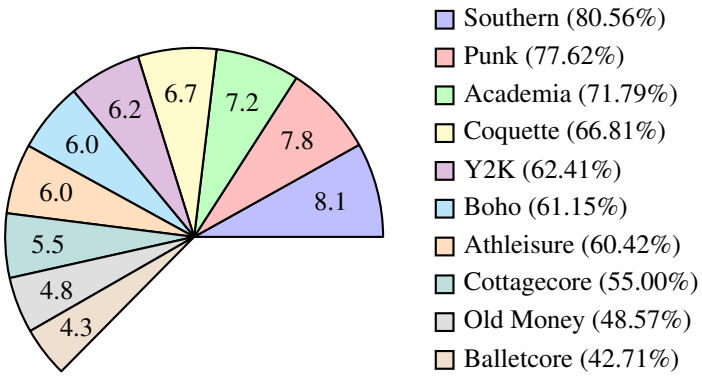


Figure 1. Test accuracy by aesthetic category.

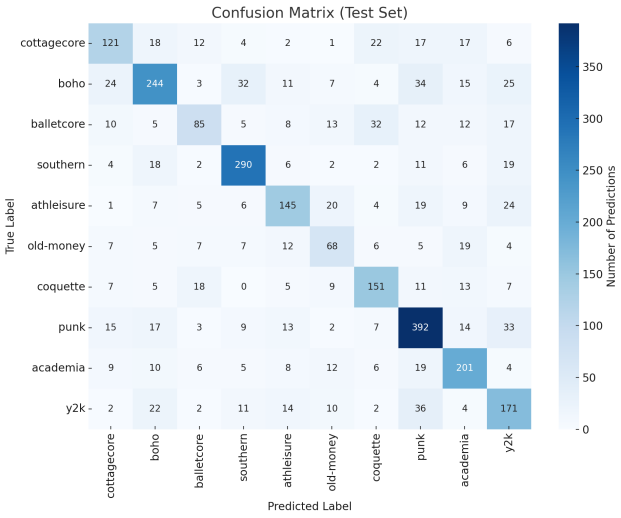


Figure 2. Confusion matrix of the test set predictions. Each row represents the true class, and each column the predicted class.

Figure 1 shows the breakdown of test accuracy across all ten aesthetic categories. The model performs best on classes with highly distinct visual signals—such as *southern* and *punk*—which often have consistent background settings

or color schemes. On the other hand, aesthetics like *old money* and *balletcore* had the lowest accuracy, likely due to their subtle or overlapping features with adjacent styles such as *academia* and *athleisure*. Figure 2 also shows most confusion occurs between semantically or visually similar categories such as *athleisure* vs. *boho*, and *old-money* vs. *academia*, supporting our hypothesis about misclassification due to overlapping aesthetics because fashion doesn't have set boundaries. This highlights the importance of class clarity and dataset consistency in achieving high classification performance.

Beyond model performance, we also encountered substantial engineering and computational constraints. Training the CNN for 30 epochs typically took 30–60 minutes, which limited how many experiments we could conduct in a reasonable timeframe. With access to more dedicated GPU or CPU resources, we could scale the dataset further, experiment with more complex architectures, or implement data augmentation strategies to enhance generalization.

We developed a custom preprocessing pipeline for our pretrained ResNet model that included normalization and resizing. Initially, we used `np.resize`, but discovered that it distorted image quality—so we revised our approach to preserve visual integrity. However, the improved pipeline introduced its bottlenecks: it took over 20 minutes to preprocess the current dataset and consumed enough memory to crash our systems occasionally. As a result, while data augmentation would have been a valuable strategy to give our MLP more training data, it was ultimately not feasible to scale preprocessing further under our resource limitations.

We also considered unfreezing some of the later layers in ResNet50 to enable fine-tuning, which could have improved model adaptability. However, we left the pre-trained layers frozen due to the already significant computing cost of running the CNN end-to-end. Fine-tuning would have increased training time and memory usage beyond what our hardware could handle. With improved infrastructure, this step could enable stronger feature learning and better overall performance in future iterations.

7. Conclusion

Through this project, we set out to classify fashion images into aesthetic categories using machine learning. We began by collecting our own dataset via web scraping, as no publicly available dataset existed for our task. This process alone taught us a lot about dataset design, visual noise, and real-world data limitations.

We evaluated multiple baseline methods using Support Vector Machines with different features (LBP, color histograms, and HOG), and compared them to a convolutional neural network trained from scratch and a pretrained ResNet CNN. Ultimately, the second CNN outperformed all baselines in development accuracy and demonstrated better generalization to diverse outfit images.

One of our key takeaways was the importance of visual consistency in the dataset. Categories with clearer defining features (e.g., *punk*, *southern*) had significantly higher performance than more subjective or overlapping aesthetics (e.g., *old money*, *balletcore*). We also learned that design choices, such as learning rate and hidden layers, could greatly influence performance.

One direction for future improvement could involve adding attention mechanisms, allowing the model to focus more directly on outfit-specific regions of an image rather than relying on background context or general textures.

One of the main lessons we learned about this task is that fashion is inherently subjective and often resists rigid categorization—many outfits simply don't fit neatly into a single aesthetic. The confusion matrix bears this out: every class shows substantial misclassifications into other categories. In practice, outfit ensembles frequently blend elements from multiple styles, producing looks that straddle—or even merge—aesthetic boundaries. As a result, it can be very difficult to assign such hybrid outfits to one "neat box."

This project has given us practical experience in model evaluation, error analysis, and the challenges of applying ML to subjective, real-world visual tasks.

References

[1] Shushi, A., & Abdulazeez, A. M. (2024). Fashion design classification based on machine learning and deep learning algorithms: A review. *Indonesian Journal of Computer Science, 13*(3). https://doi.org/10.33022/ijcs.v13i3.3980

[2] Krukowski, I. (2024, May 13). Web scraping tutorial using Selenium & Python (+ examples). *ScrapingBee*. https://www.scrapingbee.com/blog/selenium-python/

[3] Scrapy. (2024, November 19). AutoThrottle extension (Version 2.12.0) [Documentation]. *Scrapy Project Documentation*. https://docs.scrapy.org/en/latest/topics/autothrottle.html

[4] TexAu. (n.d.). Proxy rotation: Definition, importance & best practices. *TexAu Glossary*. https://www.texau.com/glossary/proxy-rotation

[5] PyTorch. (n.d.). *ResNet — PyTorch vision hub*. Retrieved May 8, 2025, from https://pytorch.org/hub/pytorch_vision_resnet/