Few Shot Learning with Visual Cues for Fine Grained Fabric Classification

Danish Ebadulla, Rahul Raman, Hridhay Kiran Shetty, Chitra G.M. and Shylaja S.S.

Department of Computer Science Engineering, PES University, Bangalore, Karnataka, India.

*Corresponding author(s). E-mail(s): danish.ebadulla@gmail.com;
Contributing authors: rahulraman@gmail.com;
hridhayks@gmail.com; chitragm@pesu.edu;
shylaja.sharath@pes.edu;

Abstract

Fabrics are composed of textile fibers. Each fabric can be categorized by a multitude of attributes such as weave pattern, density, source, reflectance and composition. Images of fabrics for classification are usually taken in controlled conditions using optimal lighting and microscopic imaging or are supplemented by additional details such as tactile and surface characterization data. This makes scaling of models for unseen classes near impossible as it requires large quantities of additional data and re-training for each new class. This work presents an easily scalable pipeline using few-shot learning to classify fabric images using only visual data from images. We prove the viability of Siamese networks for classification and empirically evaluate various popular classification techniques. Our best siamese network produces an accuracy of 75.25% when trained on 60% of the total fabric classes and evaluated on 100% of the classes, using only few-shot representatives for every class.

Keywords: Fabric classification, autoencoders, triplet loss, few shot learning, siamese networks, t-SNE

1 Introduction

Humans identify fabrics using tactile information. Owing to the similarity in weaves and material among various fabric types, relying solely on visual cues to differentiate fabrics is often challenging for humans without the requisite experience. This task is made even harder by the existence of blended fabrics, combining several materials to produce high quality textiles with characteristics unlike their constituent fibers. This is where neural networks come in.

The task of fabric classification using deep learning has been tackled using several approaches. Most research utilises convolutional neural classifiers and relies on variation and enhancement of input data to gain an edge in accuracy. But these techniques are not capable of being applied in the wild as the specific conditions under which the images are captured or the additional tools that are used to supplement fabric images are often not available in real world conditions. Even if the models produced are capable of identifying fabric images in the wild, adaptability is still a significant shortcoming. New fabrics are continuously developed and regular classifiers or systems that rely on additional data besides visual cues need to go through data acquisition and re-training in order to incorporate a new class of fabrics into their model.

This paper tackles the task of fabric classification using Siamese Networks and few-shot learning. Siamese networks are trained to output a fixed size encoding for every input image and use a similarity metric to classify images. By training on a generalized classification task in a particular domain, Siamese networks can adapt to previously unseen classes without any re-training. This method also tackles the problem of fabric classes having low inter class variance and high intra class variance. By having k representative images for each class, we can cover all the intra class variances that occur in the fabric class without sacrificing classification accuracy or efficiency.

In this work, we explore the popular techniques of fabric classification and compare their performance against Siamese networks with few-shot learning. The main contributions of this paper are (i) The effectiveness of pretraining on a reconstruction task before classification. (ii) A comparative study on the performance of convolutional neural classifiers and Siamese networks on fabric classification using non-magnified images.

2 Related Work

Chandrakant Sonawane *et al.* [1] perform fabric classification and matching using Siamese Networks. Their data collection method involves using a cellular phone microscope mounted on a mobile's camera to capture fabric images and then use CNN based autoencoders and Siamese networks for classification. Despite the lack of class diversity in their dataset, their work serves as a proof of concept for Siamese networks and their viability in fabric classification settings.

Several works[2][3] utilise autoencoders and pre-train their model on a reconstruction task. The encoding from the latent space of the autoencoder is

Fig. 1 Inter and intra class variance The images above consist of a popular fabric class in each row with cotton (row 1), nylon (row 2) and wool (row 3) highlighting the high intra class variance and low inter class variance involved in fabric classification using only images similar to these as input to the models

then used as an input for their classifiers and Siamese networks. The encoding from the autoencoder was found to solve issues with image corruption during downsampling and help with retaining long term dependencies. Using symmetric skip connections by having residual connections between convolution and deconvolution layers was proven by Mao et al.[4] to overcome upsampling data loss.

Siamese network and triplet loss for classification were popularized by Facenet [5]. The network directly learns a mapping from face images to a compact Euclidean space where distances directly correspond to a measure of face similarity. Once this space has been produced, tasks such as face recognition, verification and clustering can be easily implemented using standard techniques with FaceNet embeddings as feature vectors. Their method uses a deep convolutional network trained to directly optimize the embedding itself, rather than an intermediate bottleneck layer as in previous deep learning approaches. To train, triplets of roughly aligned matching / non-matching face patches are generated using a novel online triplet mining method. This approach results in much greater representational efficiency: achieving state-of-the-art face recognition performance using an encoding of only 128-bytes per face.

Works of Yueqi Zhong *et al.* [6] and Muhammad Ather Iqbal Hussain *et al.* [7] adopt popular deep learning architectures for fabric classification and emphasize the importance of sampling and data augmentation in improving the accuracy and generalizability of their models. These works classify fabrics on dyes, yarn-structure and weave pattern.

Ohi et al. [8] perform image-based fabric classifcation using Ensemble Convolutional Neural Networks on the I-bug dataset. They train on 50 textile classes available in the dataset and experiment with CNN architectures such as Inception, ResNet, VGG and MobileNet among others. Their final Ensemble architecture achieves a classification accuracy of 84%. Their work helps set a baseline for fabric classification by processing only the surface image of the fabric.

3 Method

This section covers the components of our work in detail and explains the structure of all our model pipelines and experiments. All our models are deep convolutional networks. We compare the efficiency of autoencoders and the viability of reconstruction as a pre-training task before classification. Convolutional neural classifiers are trained using the aforementioned technique and on a classification task from scratch. In the end, we compare the performance of our classifiers against Siamese networks trained using triplet loss. Our objective was to extract an image embedding of size that is exponentially lower than the input and still be capable of classifying the image with a high degree of accuracy. This would ensure that our models could be deployed with very little compute and storage constraints, as each class would only need an n-byte embedding for our model to recognize it successful.

3.1 Data

The Fabric Dataset[9] is a collection of over 2000 unique samples of fabrics seperated by the fiber type with images taken in the field using a protable photometric stereo sensor. From this dataset, we utilise upto 10 popular classes in differfent combinations throughout our models. These images were pre-processed and cleaned to remove any redundant samples and only those samples which had a minimum composition of 80% of one fiber were selected. Augmentation was applied to the less popular classes to ensure the volume of data across classes was not skewed. The degree of augmentation was kept low to ensure a balance between preventing overfitting while also feeding the model unique cases that it might have to deal with when working with field data. This way we attempt to reduce the variance and bring some homogeneity to the classes. All our experiments use images from cotton, nylon, polyester, denim, silk, wool, crepe, corduroy, satin or terrycloth as classes in varying combinations. Our models were tested on a hold out set of 20% of the Fabric dataset. The test set, although being drawn from the same distribution was created to ensure disjoint samples.

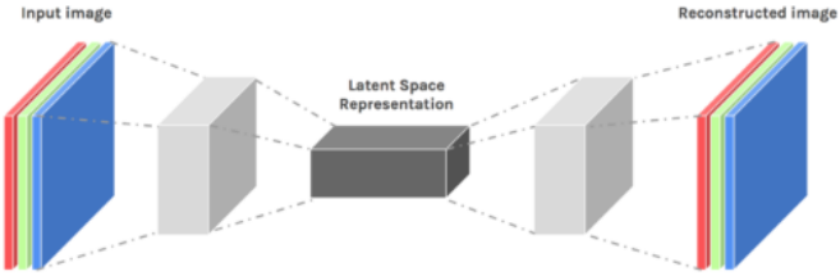

Fig. 2 Representation of an autoencoder

3.2 Autoencoder

The autoencoder is a deep neural network consisting of 2 parts, the encoder and a decoder with a bottleneck in between. We used an autoencoder with a bottleneck to produce a latent space embedding that could serve as a satisfactory encoding of an image. This encoding is then used for classification. Both the input and the output to the autoencoder are the same image and it is trained to recreate the image as accurately as possible. The encoder half of the autoencoder consists of convolutional layers which successively compress the input image. Batch normalization layers are included to stabilize the layers' outputs. The decoder half is almost a mirror of the encoder half, consisting of convolutional layers with up sampling layers to attempt to recreate the image with only a small percentage of the original data. To improve our outputs and reduce loss, we tried using skip connections [4], connecting the output of some of the encoder layers to the corresponding mirror decoder layers, in an attempt to help the decoder layers fine tune their weights to better reconstruct the image after compression. After training the autoencoder, the weights of the encoder are frozen and transferred to a classifier to be used for classification. The details of various models can be found in Table 1. One of the shortcomings of the models in [1] is the bias introduced by the color and patterns of the fabrics. We train on both grayscale and color images to compare their performance. Detailed observations of our experiments will be discussed in section 3.

3.3 Classifier

We use 2 different approaches to train our classifier models. The first approach is similar to the approach taken by [1], using the encoder half of the autoencoder as the convolutional layers of the classifier followed by a few dense layers to flatten and classify. The second approach was based on the assumption pre-training on a reconstruction task is not the optimal approach. The model consists of convolutional layers without a bottleneck followed by dense layers for classification. The classifiers were built to confirm if the pre-training

approach was viable and if the approach could be extended to Siamese Networks for classification tasks. The classifiers were trained on upto 10 classes of fabrics. Results of notable classifiers are documented in Table 2. Our best classifier's architecture is shown in Table A2.

3.4 Triplet Loss and Selection

Our Siamese Network uses triplet loss as the loss function. Triplet loss and triplet mining was first introduced by [5] and works better than contrastive loss for Siamese Networks. It takes in images as triplets instead of pairs. Each triplet consists of an anchor image (x_i^a) of a particular class, a positive image (x_i^p) of the same class and a negative image (x_i^n) of a different class. Triplet loss tries to ensure that the anchor of a particular class is closer to all other images of the same class(positive image) than it is to any image of a different class (negative image). It also enforces a margin between the positive and negative pairs with the parameter α. The overall formula for the loss L that is being minimized is given below:

$$L = \sum_{i}^{N} \left[f(x_i^a) - f(x_i^p)_2^2 - f(x_i^a) - f(x_i^n)_2^2 + \alpha \right]_+ \tag{1}$$

Triplet loss is better than traditional loss functions such as softmax cross entropy loss as it allows us to have a variable number of classes. The addition of a new class to the model only requires a representative image belonging to that class to be passed through the model. The embedding generated for this image can then be used as an anchor for all subsequent images of the new class.

To ensure faster convergence, we need to select hard or semi-hard triplets, i.e triplets where distance (x_i^a, x_i^n) < distance (x_i^a,x_i^p) or where d(a,p) < d(a,n) < d(a,p) + α. These triplets are active and have a higher degree of contribution to the model's improvement.

Another factor that is vital for ensuring fast convergence is the selection of triplets. There are 2 methods to select and generate these triplets that are talked about in [5]:

- Offline mining: Where the triplets are generated offline every n steps and the argmin and argmax is generated on a subset of data
- Online mining: Where the triplets are selected as hard positive/negative samples from within a mini batch.

Models that used online mining performed much better than those that used offline mining. There are 2 different strategies that can be followed when proceeding with online triplet mining [10]:

- Batch all: We first select all the valid triplets, i.e triplets (i,j,k) where i and j belong to the same class but are distinct, while k belongs to a different class but the loss is averaged only on the hard and semi hard triplets

- Batch hard: From our mini batch, we select the hardest positive and the hardest negative for each anchor. This ensures that the selected triplets are hardest among the batch

Our models yield the best performance with batch hard, which is in line with the conclusions drawn in [5].

3.5 Siamese Network

A Siamese neural network is an artificial neural network that uses the same weights while working in tandem on two different input vectors to compute comparable output vectors. Often one of the output vectors is precomputed, thus forming a baseline against which the other output vector is compared. We attempt to obtain an embedding f(x) from an image such that the dimensions of the embedding are much lower than that of the original image and the squared distance between a pair of images belonging to the same class is smaller than the squared distance between a pair of images belonging to different classes. The idea of a Siamese Network was first introduced by [11] and a version of this network is used by [5] to perform facial recognition and verification. We try out a variety of Siamese Network architectures. The main difference in our approach lies in whether or not we use autoencoder weights in the first half of our network. We use triplet loss instead of contrastive loss as triplet loss is better in enforcing a margin between different classes, simultaneously bringing together intra class samples while pushing away inter class samples. We also found that online mining of triplets performs better than offline batch mining. We compare classification accuracy with traditional deep classification networks as we increase the number of input classes to confirm which of the two is a more viable approach for use in real world situations.

3.6 Few Shot Learning

Meta learning is often applied when we have very little data available. We use few shot learning [12] in our Siamese Networks to build a support set having around 5-10 representatives per class which is capable of identifying even unseen classes and sub classes. The idea is cover as many variances as possible in each class so that the model can identify a sample as belonging to a particular class as long as it has an image in the support set that is similar to the sample image. This method ensures our model's long term viability as a suitable fabric classifier as it can adapt to any unseen classes and sub classes as long as it has a representative image in the support set. Our classification function compares the L2 distance of a given input image's embedding with all the representatives in the support set and determines its class as the class of the representative whose distance is closest to the input image. By ensuring our support set covers all the unique types of each class, we obtain high accuracy despite high intra class variance. This gives Siamese Networks an edge over traditional classifiers as traditional classifiers will require additional training and a large dataset to adapt to a new class. Section 3 discusses the results of

testing our Siamese Network with few shot learning and how it performs in comparison to a traditional classifier.

Table 1 Notable autoencoder models

Sl no.	Latent Space	Bottleneck	Compression	Skip Connection	Image Type	Final Loss
1	16x16x64	Yes	4x	No	Grayscale	2.04e-3
2	16x16x64	Yes	4x	Yes	Grayscale	2.20e-5
3	16x16x64	Yes	12x	No	Color	3.48e-3
4	8x8x256	Yes	4x	No	Grayscale	3.84e-3
5	8x8x256	Yes	4x	Yes	Grayscale	7.10e-5
6	16x16x256	No	1x	No	Grayscale	2.32e-3
7	8x8x1024	No	1x	No	Grayscale	4.44e-3
8	8x8x512	Yes	2x	No	Grayscale	4.95e-3
9	4x4x512	Yes	8x	No	Grayscale	7.54e-3

4 Experiments and Results

The following section will cover in detail the experiments, our setups, results and observations. Unless specifically mentioned, all our models use input images of dimension 256x256. All our models are trained on an NVIDIA Tesla K80 GPU.

4.1 Autoencoders

The autoencoders were used to generate a latent space embedding that can be used for classification. Over the course of our experiments, we tried out data compression ratios of 2,4 and 8. We found that a data compression ratio of 4 produced the best results maintaining a balance between accurate image recreation and low embedding size. Shorter rectangular latent spaces were found to perform better when compared to longer flattened latent spaces in the same data compression ratio. Trying to maintain the original ratio of dimensions of the image while compressing it produced better outputs than attempting to flatten it. We also tried using skip connections to help generate a better embedding. This resulted in very fast convergence, low loss and high quality of the image recreation. Despite the high image recreation capability, the skip connection autoencoders produced dismal accuracy during classification. We assume that while the skip connections help in accurate image recreation, it happens so at the cost of a good image encoding. Since the decoder is helped by the skip connections, it is not forced to generate the best image encodings anymore.

We tested our models using RMS and Adam but RMS was found to perform better than Adam. The best autoencoder was trained with a learning rate of

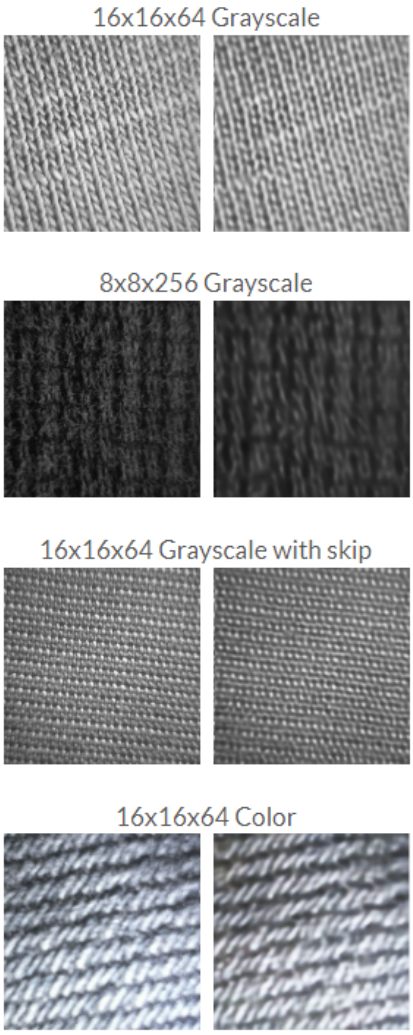

Fig. 3 Comparison of input-output images of various autoencoder models. The image on the left is the input image to the autoencoder and the image on the right is the output image recreated from only a fraction of the original image by the autoencoder

0.01 which is lowered towards the end with MSE for the loss and RMS for the optimizer. All input images are of size 256x256x1 if it is grayscale and 256x256x3 if it is an RGB image. The detailed architecture of various model can be found in Table 1. Sample inputs and outputs to the autoencoders can be seen in Figure 3. Our best bottleneck autoencoder takes in a grayscale image of size 256x256x1 and compresses it to an encoding of size 16x16x64 giving us a data compression ratio of 4 and has a total of 2.7M parameters.

4.2 Classifiers

Table 2 shows the results and accuracy of various classifier models that we trained. There were 2 different types of classifiers used:

- Classifiers that use the bottleneck encoder architecture
- Classifiers that do not use a bottleneck architecture

The models were trained from 5 to 10 different classes. We observed that while both approaches had similar performance with the non-bottleneck models having a slightly higher accuracy. Our experiments also found that classifiers trained without the autoencoder weights pre-loaded produced a higher accuracy than the models that did use the autoencoder weights. The autoencoder architecture that produced the best results also corresponded to the best classifier results, even when trained without the weights. Therefore, we concluded that the architecture itself translates well from autoencoders to classifiers but, contrary to the approach taken by [1], using the autoencoder weights during classification does not actually produce higher accuracy when compared with training the classifier from scratch.
All our classifiers use rectified linear units as the non-linear activation unit at every layer with max pooling and batch normalization. The final layer uses softmax for classification. We tested classification on both color and grayscale images as input and using various combinations of optimizers. Contrary to the observation in autoencoders, Adam was found to perform better than RMS as an optimizer. Binary cross entropy was used as a loss function with a learning rate of 1e-4. The classifiers were trained upto 10 classes of fabrics. Our best classifier was the one trained on grayscale images input at a resolution of 256x256. Results of notable classifiers are documented in Table 2.

Table 2 Prominent classifier models and their accuracies

	Encoder module			Classifier module		
Sl no.	Latent Space	Skip Connection	Compression	Image Type	Classes	Accuracy
1	16x16x64	No	4x	Grayscale	5	80.47%
2	8x8x256	No	4x	Grayscale	5	73.56%
3	16x16x256	No	1x	Grayscale	5	86.37%
3	16x16x64	No	4x	Grayscale	8	66.08%
4	16x16x256	No	1x	Grayscale	8	78.65%
5	16x16x64	Yes	4x	Grayscale	5	49.47%
6	8x8x256	Yes	4x	Grayscale	5	39.12%
7	16x16x64	No	12x	Grayscale	5	68.76%
8	16x16x256	No	3x	Grayscale	5	79.77%

4.3 Siamese Networks

We experimented with various Siamese Network architectures to find the best model for performing classification. Apart from the models that used our encoder architecture and its variants to produce an embedding, we also attempted to incorporate Inception networks [13] and its variants that utilise either max pooling layers or L2 norm pooling layers. The inception models were implemented with pre-trained weights from ImageNet [14] classification. The dense layers were slightly modified to better suit our classification task. The inception networks were trained on color images as the original inception architecture utilised color images as input. Possibly due to the model complexity which caused overfitting or due to the low inter class variance that comes at the cost of utilising color images, the inception models did not perform as well as the models using the encoder architecture. A tabular summary of some of our prominent Siamese Networks can be found in Table 3. We found that while Batch All converged quicker, Batch Hard gave slightly better results with time. Our models were tested separately on both hard and semi-hard triplets. While semi-hard triplets converge faster, we observed that hard triplets give a higher accuracy.

We also experimented with various embedding sizes and found that an embedding of size 128 produced the best tradeoff between computation power and accuracy. Our best model produces an embedding of size 128 and has 2.4M parameters. We use online mining with Batch Hard and use a margin of 0.4. Learning rate is initially set to 1e-4 and set to decay during training. The detailed architecture of the above mentioned model is available in Table A3.

Table 3 Prominent Siamese Networks and the accuracy obtained on each. All models were trained on 5 classes and tested on 8 classes

Sl no.	Architecture Type	Pre-loaded Weights	Accuracy
1	Base Inception	ImageNet	55.30%
2	Inception L2	ImageNet	66.12%
3	Inception Max	ImageNet	61.10%
4	Encoder[1]	Encoder	64.72%
5	Encoder[1]	None	76.72%

[1]Architecture of the best encoder model from Table 1 and 2

4.4 Classifiers vs Siamese Networks

We compare the performance of our best classifier with our best Siamese Network. The classifier trains on all classes while the Siamese Network only trains on percentage of the total classes. Both models are tested on increasing number of classes. While the classifier has higher accuracy initially, the Siamese Network outperforms the classifier as the number of classes increase even though it sees some of the classes for the first time during testing. We found that a seen

to unseen class ratio of 3:2 produced the best tradeoff between accuracy and computation efficiency for the Siamese Network. Table 4 shows the accuracy obtained by both models as we increase the number of classes. The Siamese Network only trains on 4 when being tested on 6 classes, 5 classes when being tested on 8 classes and 6 classes when being tested on all 10. We can observe the sharp drop in accuracy in the classifier as the classes increase while the Siamese Network manages to stay relatively consistent as the classes increase.

Table 4 Comparing the accuracy of Siamese Networks vs. Classifiers. The Siamese Network outperforms the classifier as the number of classes increase, even though it sees only 60% of the total classes during training.

	6 classes	8 classes	10 classes
Siamese Network	78.19%	76.72%	75.25%
Classifier	81.13%	78.65%	72.06%

4.5 Few Shot Learning

The representative images for each class to be used in few-shot learning are chosen through manual inspection. We pick our representatives from a subset of samples from each class that are the most frequent anchors for their particular class. While this approach is not the most efficient, it ensures that the samples are disjoint to a certain extent and serve as suitable representatives for each sub category in a class.

4.6 Visualising with t-SNE

We use t Stochastic Neighbour [15] embedding to visualise our high dimensional data. Since t-SNE only preserves nearest-neighbours, it serves as a valuable tool in visualising the Siamese Network and getting an intuitive idea of whether it has been able to successfully bring together similar classes while increasing the distance between differing classes. Figure 4 shows our best visualisations when t-SNE was applied on

- the data before it was fed into the Siamese Network
- the data after being put through the Siamese Network

Our t-SNE model uses a perplexity of 45 and a learning rate of 200 and is fit until convergence. We can observe the lack of clustering in the embeddings before feeding it through the trained model in Figure 4 and the clusters that have formed after passing it through the trained Siamese Network in Figure 4. Almost all the classes have managed to form clear clusters with some scattering. This proves that the Siamese Network is capable of successfully clustering the fabric classes.

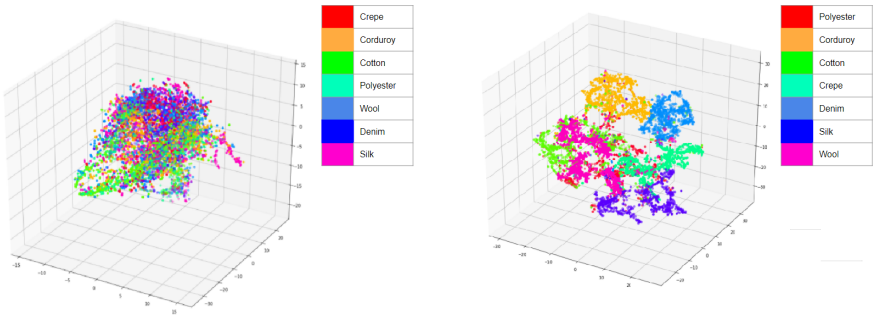

Fig. 4 A 3-dimensional t-SNE of the encodings. The t-SNE plot on the left shows the encodings before being trained on the Siamese Network. The t-SNE plot on the right shows the encodings after training them on the Siamese Network

5 Limitations and Future Work

Our models overcomes the shortcomings stated in [1], namely the bias in classification introduced due to the presence of patterns or colors in fabrics. Initial experiments with color images found that many images of bright colored fabric or lustre got misclassified as silk, while images of fabrics that are blue in color often got misclassified as denim. By switching from color input to grayscale input, we avoid these misclassifications, forcing the model to only utilise the data obtained from the texture and the weave of the fabric for classification. Although we obtained better results on grayscale images, when performing fabric classification in such constrained conditions, the loss of color as a classification feature is still an undeniable shortcoming.

Due to the low intra class variance in the weave pattern of some similar fabrics, the lack of an appropriate representative in the support set of that class will result in the sample being misclassified. This problem is the most common in synthetic classes such as polyester and nylon, which tend to look very similar to natural fabrics if the image is not focused or magnified enough on the texture of the fabric. This is observable in our t-SNE model, Figure 4, where we can notice the absence of a clear large cluster of polyester despite polyester having acceptable classification accuracy. A better method of selecting our support set will resolve this shortcoming.

There are many possible approaches that can be undertaken as future work. Currently, the method of choosing the support set for few shot learning is crude and only intuitively attempts to represent the diversity in a class. Further work can focus on optimizing the method of selecting the representatives for the support set such that each sample of a class in the support set is capable of representing a sub cluster of unique samples in the class. This will greatly improve the accuracy of the model for classification. The Siamese Network can also further be improved. We will also look into the effect of different margins in the Siamese Network and if a dynamic method of varying the margin will help the model to scale better. Additional data will also help to test the scalability

Table 5 Per class classification accuracy obtained on a 3:2 seen to unseen class trained Siamese Network. The table demonstrates the individual accuracy for each positive class (represented as columns) against each negative class(represented as rows).

N\P	Corduroy	Cotton	Crepe	Denim	Nylon	Polyester	Satin	Silk	Terrycloth	Wool
Corduroy	-	80	82	89	70	79	93	76	78	63
Cotton	52	-	61	79	40	60	88	57	90	50
Crepe	78	64	-	76	66	71	69	62	100	72
Denim	80	75	57	-	58	81	82	63	100	56
Nylon	80	74	69	78	-	66	85	65	100	71
Polyester	77	52	65	79	54	-	76	49	100	73
Satin	77	81	72	82	75	69	-	53	100	87
Silk	73	79	69	78	65	76	68	-	100	86
Terrycloth	54	74	61	75	68	72	91	65	-	56
Wool	64	63	49	77	50	63	90	61	92	-

of the model as the Fabric dataset has at most 10 fabric classes with enough unique enough samples to serve as a generalizable training class.

6 Conclusion

In this work, we prove the viability of Siamese Networks for fabric classification at scale. We experimentally prove that the a Siamese Network outperforms a traditional convolutional classifier despite being trained on lesser data than the latter and visualise the embeddings to prove that triplet loss brings together embeddings of the same class while pushing away embeddings of different classes. Our Siamese Network obtains a classification accuracy of 75.25% when trained on 6 unique classes and tested on the original 6 along with 4 additional unseen classes, outperforming a traditional classifier of similar architecture that was trained on all 10 classes by over 3%. We use few shot learning to ensure representation of all the unique variances found in a fabric class and prove its scalability by testing it on unseen classes.

We train a variety of autoencoders with different latent spaces and connections and prove that the latent space generated by an autoencoder does not work well for classification and that training convolutional models from scratch produce better results than transfer learning. We prove that using skip connections for autoencoders do not translate to an effective latent space despite the high quality of image recreation.

Our approach sets us apart from other methods of fabric classification that utilise additional information such as tactile sensor data, albedos, reflectance spheres or semantic data along with the images themselves. Our model is not affected by fabric colors or lighting conditions since it is trained on images very similar to those taken by the average person in the real world. Our model also requires minimal alignment or preprocessing of the images before feeding it into the model, thus further enhancing the simplicity and utility of our model. By improving the selection process of representatives for the support set and fine tuning the model with additional data it is possible to produce a lightweight model capable of classifying fabrics with high accuracy while being easily scalable.

Appendix A Model Architecture Details

Table A1 The architecture of our best autoencoder

Type Of Layer	Filter Size	Filter / Neurons	Output Shape	Params
InputLayer			(256, 256, 1)	0
Conv2D	(7, 7)	16	(256, 256, 16)	800
MaxPooling2D	(2, 2)		(128, 128, 16)	0
Conv2D	(3, 3)	32	(128, 128, 32)	4640
Conv2D	(3, 3)	32	(128, 128, 32)	9248
MaxPooling2D	(2, 2)		(64, 64, 32)	0
Conv2D	(3, 3)	64	(64, 64, 64)	18496
Conv2D	(3, 3)	64	(64, 64, 64)	36928
BatchNormalization			(64, 64, 64)	256
MaxPooling2D	(2, 2)		(32, 32, 64)	0
Conv2D	(3, 3)	128	(32, 32, 128)	73856
BatchNormalization			(32, 32, 128)	512
Conv2D	(3, 3)	128	(32, 32, 128)	147584
BatchNormalization			(32, 32, 128)	512
MaxPooling2D	(2, 2)		(16, 16, 128)	0
Conv2D	(3, 3)	256	(16, 16, 256)	295168
BatchNormalization			(16, 16, 256)	1024
Conv2D	(3, 3)	256	(16, 16, 256)	590080
BatchNormalization			(16, 16, 256)	1024
Conv2D	(3, 3)	512	(16, 16, 512)	11801160
BatchNormalization			(16, 16, 512)	2048
Conv2D	(3, 3)	64	(16, 16, 64)	32832
UpSampling2D	(2, 2)		(32, 32, 64)	0
Conv2D	(3, 3)	128	(32, 32, 128)	73856
BatchNormalization			(32, 32, 128)	512
Conv2D	(3, 3)	128	(32, 32, 128)	147584
BatchNormalization			(32, 32, 128)	512
UpSampling2D	(2, 2)		(64, 64, 128)	0
Conv2D	(3, 3)	64	(64, 64, 64)	73792
BatchNormalization			(64, 64, 64)	256
Conv2D	(3, 3)	64	(64, 64, 64)	36928
UpSampling2D	(2, 2)		(128, 128, 64)	0
Conv2D	(3, 3)	32	(128, 128, 32)	18464
Conv2D	(3, 3)	32	(128, 128, 32)	9248
UpSampling2D	(2, 2)		(256, 256, 32)	0
Conv2D	(3, 3)	16	(256, 256, 16)	4624
Conv2D	(3, 3)	16	(256, 256, 16)	2320
Conv2D	(7, 7)	1	(256, 256, 1)	785

Table A2 The architecture of our best Classifier

Type of Layer	Filter Size	Filters/ Neurons	Output Shape	Params
Input Layer			(256, 256, 1)	0
Conv2D	(3, 3)	16	(256, 256, 16)	160
MaxPooling2D	(2, 2)		(128, 128, 16)	0
Conv2D	(3, 3)	32	(128, 128, 32)	4640
MaxPooling2D	(2, 2)		(64, 64, 32)	0
Conv2D	(3, 3)	64	(64, 64, 64)	18496
BatchNormalization			(64, 64, 64)	256
MaxPooling2D	(2, 2)		(32, 32, 64)	0
Conv2D	(3, 3)	128	(32, 32, 128)	73856
BatchNormalization			(32, 32, 128)	512
MaxPooling2D	(2, 2)		(16, 16, 128)	0
Conv2D	(3, 3)	256	(16, 16, 256)	295168
BatchNormalization			(16, 16, 256)	1024
Flatten			65536	0
Dense		128	128	8388736
Dense		64	64	8256
Dense		32	32	2080
Dense		10	10	198

Table A3 Our best Siamese Network's architecture

Type Of Layer	Filter Size	Filters/ Neurons	Output Shape	Params
InputLayer (image)			(256, 256, 1)	0
Conv2D	(7, 7)	16	(256, 256, 16)	800
MaxPooling2D	(2, 2)		(128, 128, 16)	0
Conv2D	(3, 3)	32	(128, 128, 32)	4640
Conv2D	(3, 3)	32	(128, 128, 32)	9248
MaxPooling2D	(2, 2)		(64, 64, 32)	0
Conv2D	(3, 3)	64	(64, 64, 64)	18496
Conv2D	(3, 3)	64	(64, 64, 64)	36928
BatchNormalization			(64, 64, 64)	256
MaxPooling2D	(2, 2)		(32, 32, 64)	0
Conv2D	(3, 3)	128	(32, 32, 128)	73856
BatchNormalization			(32, 32, 128)	512
Conv2D	(3, 3)	128	(32, 32, 128)	147584
BatchNormalization			(32, 32, 128)	512
MaxPooling2D	(2, 2)		(16, 16, 128)	0
Conv2D	(3, 3)	256	(16, 16, 256)	295168
BatchNormalization			(16, 16, 256)	1024
Conv2D	(3, 3)	256	(16, 16, 256)	590080
BatchNormalization			(16, 16, 256)	1024
Conv2D	(3, 3)	512	(16, 16, 512)	1180160
BatchNormalization			(16, 16, 512)	2048
GlobalAverage-Pooling2D			512	0
Dense		128	128	65664
InputLayer (label)			1	0
Lambda (L2_normalize)			128	0
Concatenate (lambda, label)			129	0

References

[1] Sonawane C, Singh DP, Sharma R, Nigam A, Bhavsar A. Fabric Classification and Matching Using CNN and Siamese Network for E-commerce. In: International Conference on Computer Analysis of Images and Patterns. Springer; 2019. p. 193–205.

[2] Ahrabian K, Babaali B. On Usage of Autoencoders and Siamese Networks for Online Handwritten Signature Verification. CoRR. arXiv preprint arXiv:171202781. 2017;.

[3] Thapar D, Jaswal G, Nigam A, Kanhangad V. PVSNet: Palm vein authentication siamese network trained using triplet loss and adaptive hard mining by learning enforced domain specific features. In: 2019 IEEE 5th International Conference on Identity, Security, and Behavior Analysis (ISBA). IEEE; 2019. p. 1–8.

[4] Mao XJ, Shen C, Yang YB. Image restoration using convolutional auto-encoders with symmetric skip connections. arXiv preprint arXiv:160608921. 2016;.

[5] Schroff F, Kalenichenko D, Philbin J. Facenet: A unified embedding for face recognition and clustering. In: Proceedings of the IEEE conference on computer vision and pattern recognition; 2015. p. 815–823.

[6] Wang X, Wu G, Zhong Y. Fabric identification using convolutional neural network. In: International Conference on Artificial Intelligence on Textile and Apparel. Springer; 2018. p. 93–100.

[7] Iqbal Hussain MA, Khan B, Wang Z, Ding S. Woven Fabric Pattern Recognition and Classification Based on Deep Convolutional Neural Networks. Electronics. 2020;9(6):1048.

[8] Ohi AQ, Mridha MF, Hamid MA, Monowar MM, Kateb FA. FabricNet: A Fiber Recognition Architecture Using Ensemble ConvNets. CoRR. 2021;abs/2101.05564. https://arxiv.org/abs/2101.05564.

[9] Kampouris C, Zafeiriou S, Ghosh A, Malassiotis S. Fine-grained material classification using micro-geometry and reflectance. In: European Conference on Computer Vision. Springer; 2016. p. 778–792.

[10] Hermans A, Beyer L, Leibe B. In defense of the triplet loss for person re-identification. arXiv preprint arXiv:170307737. 2017;.

[11] Taigman Y, Yang M, Ranzato M, Wolf L. Deepface: Closing the gap to human-level performance in face verification. In: Proceedings of the IEEE conference on computer vision and pattern recognition; 2014. p. 1701–1708.

[12] Fei-Fei L, Fergus R, Perona P. One-shot learning of object categories. IEEE transactions on pattern analysis and machine intelligence. 2006;28(4):594–611.

[13] Szegedy C, Liu W, Jia Y, Sermanet P, Reed S, Anguelov D, et al. Going deeper with convolutions. In: Proceedings of the IEEE conference on computer vision and pattern recognition; 2015. p. 1–9.

[14] Krizhevsky A, Sutskever I, Hinton GE. Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems. 2012;25:1097–1105.

[15] Van der Maaten L, Hinton G. Visualizing data using t-SNE. Journal of machine learning research. 2008;9(11).