A comparative study on texture classification using deep learning architecture

Rui-Xuan Yan [1], Yuan-Chen Liu [2,*]
[1, 2]Department of Computer Science
National Taipei University of Education, Taipei City, Taiwan, ROC
liu@tea.ntue.edu.tw

Abstract

Texture classification plays an important role in image analysis. The common method of texture classification is through feature extraction and identification. This method requires manual review and extraction of key features in advance, and then writes a dedicated algorithm, and then evaluates the algorithm with good effect. In recent years, the method of deep learning has been used to replace the feature extraction link through convolution calculation. However, this kind of research only focuses on the texture of a certain object with a special trainer. Therefore, this paper uses the complex texture database and the texture regular database for practice, and uses 12 kinds of trainers for performance evaluation under the transfer learning technology. In addition, the performance differences under different activation functions are also compared. We use the accuracy, precision, recall and F1 score indicators to evaluate the performance. The experimental results show that DenseNet201 has the highest accuracy under two different attribute datasets, while ReLU is the most efficient under five different activation functions. The results of this study clearly analyze the differences in the performance of various common trainers for texture classification, and can provide follow-up researchers to design better performance texture classification trainers.

keyword: texture image, texture classification, feature extraction, deep learning, trainer, activation function, transfer learning

1. Introduction

Textures are uneven grooves on the surface of an object, which are repeated according to certain rules, and can be used to describe the composition of any material. In image analysis, texture is a very important feature, which can represent important information about the surface structure of objects and their relationship with the surrounding environment. Therefore, it is an important method for image segmentation, feature extraction and classification and recognition. Thus, texture analysis has been widely used in many different fields, including X-ray lung texture on medical images [1], muscle or blood vessel texture [2], microscopic image texture [3], telemetry image texture [4], terrain Photo lithological texture [5] and so on. In recent years, the method of deep learning has been used for texture analysis. When the amount of data is larger, the final judgment result will be more accurate. However, the past research only focused on a single trainer, and did not fully analyze the performance of various trainers. Therefore, this paper uses the complex texture database Describable Textures Dataset (DTD) and the

texture regular database University of Illinois at Urbana-Champaign (UIUC) for testing. And under the transfer learning technology, 12 kinds of trainers are used to evaluate the performance to provide reference for those who are interested in texture analysis applications.

This section introduces the motivation and purpose of texture analysis, the second section describes the architectural principles of various neural network models for deep learning, the third section presents the research methods, the fourth section is the experimental results and discussion, and the last is the conclusion.

2. Related work

This section will introduce the architectural principles of in-depth learning of various types of neural network models. There are three subsections in this section. The first subsection will introduce the principle and architecture of the neural network model, the second subsection introduces the classification of activation functions commonly used by deep learning models when performing tasks. Finally, the third subsection introduces the loss functions commonly used by deep learning models for classification tasks.

2.1. Deep learning applied to texture classification

Texture classification is an important field of computer vision. Traditional texture description methods include structural and statistical methods. The former uses spatial arrangement to describe texture features [6], while the latter uses pixel value statistics as features [7]. In recent years, deep learning technology has been widely used in various fields and achieved good results. Therefore, CNN-based texture classification technology has been proposed one after another. Dixit [8] optimized CNN with Whale Optimization Algorithm (WOA) to improve the performance of texture classification. Azadnia [9] used CNN combined with SVM to classify the type of soil texture. Tivive [10] proposed a Convolutional Neural Network (CoNN) for texture classification and proved to be better than other traditional methods for describing textures. Bębas [11] used several classification methods, including SVM, kNN, RF, and deep learning, to analyze PET/MR texture images of two types of lung cancers. Kumar [12] applied deep learning in medical CT and CXR images to classify images infected with COVID-19. Bastidas-Rodrigueza [13] extracted texture features for mechanical failure classification using the deep learning method. Kırbaş [14] used deep learning architectures and transfer learning methods such as ResNet-50, Inception V3, Xception and VGG19 for texture classification and recognition of 12 wood species. Wang [15] analyzed land use and crop classification based on high-resolution deep learning telemetry imagery. Most of the above studies use deep learning to classify textures in different professional fields, including medicine, telemetry, machinery, soil, agriculture, forestry imaging, etc., and combine various mathematical techniques to improve performance. In conclusion, this study explores the performance comparison of well-known deep learning trainers for general standard texture databases. The results will provide a reference for future researchers to use appropriate trainers for texture classification of various topics.

2.2. Deep learning model

Deep learning is an algorithm that builds multi-layer neural networks in a way similar to human neural networks and performs representational learning on data. This subsection will introduce AlexNet, VGG16/VGG19, Resnet, GoogLeNet, Inception v3, Xception, DenseNet201, MobileNetV2, ShuffleNet. The Architecture and principles of these common neural network models.

2.2.1. AlexNet

AlexNet is the name of a convolutional neural network (CNN) architecture, designed by Alex Krizhevsky in collaboration with Ilya Sutskever and Geoffrey Hinton, who was Krizhevsky's Ph.D. advisor. AlexNet model architecture has a total of eight layers, the first five were convolutional layers and max-pooling layers. After the first, second and fifth convolutional layers, we use Maxpooling with a size of 3×3 and a stride of 2, which can better retain important features. And pooling can overlap through stride < size (2 < 3) to form a re-examination of features to avoid important features being discarded and to avoid the problem of overfitting, and the last three are fully Connected Layers using Dropout and Data augmentation to reduce overfitting[6] ◦ AlexNet can input color images of size 224×224. Since the input layer becomes larger, the size of the convolution kernel is set to 11×11 and stride of 4 in the first layer, and also uses a larger step size to extract features; the kernel of the second layer is 5×5, all subsequent kernels use 3×3, and stride is set to 1. AlexNet is considered one of the most influential models in computer vision, and it has spurred more researches to use convolutional neural networks and GPUs to accelerate deep learning, and its architecture is shown in Figure 1.

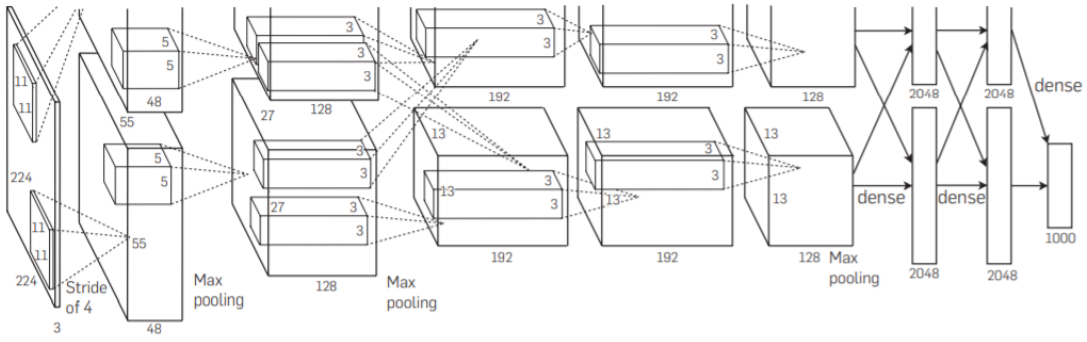

Figure 1. AlexNet architecture [6]

2.2.2. VGG16/VGG19

VGGNet as proposed by University of Oxford visual geometry group, in 2014. ImageNet made its debut in the competition and won the first place in the positioning task and the second place in the classification task of that competition with a low error rate of 7.3%. The architecture of VGGNet is roughly inherited from AlexNet, and several methods are proposed for improvement. The first is that VGGNet uniformly uses a smaller 3×3 convolution filter for convolution operations compared to AlexNet's use of 5×5 and 7×7 convolution filters. By stacking several small convolutional filters to achieve

the same receptive field as the large convolutional filters, this reduces the number of parameters of the model to avoid complicated computations. The second method is to increase the number of filters in each convolutional layer, that is, to increase the number of feature channels to capture more feature information. The third method is to increase the number of layers of the model network. The original AlexNet has a total of eight-layer architecture, and VGGNet increases the network depth to a maximum of 19 layers. It has been confirmed that increasing the number of layers of the neural network can improve the classification accuracy [7]. In addition, the fully connected layer is replaced by 1×1 convolution to facilitate the input of different image sizes and reduce the amount of parameters. At present, the most commonly used models are VGG16 and VGG19, which are 16 layers (13 convolutional layers and 3 fully connected layers) and 19 layers (16 convolutional layers and 3 fully connected layers), and its architecture is shown in Figure 2.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figure 2. VGG16 and VGG19 architecture [7]

2.2.3. ResNet

In 2015, Microsoft proposed the deep residual learning model (ResNet), reducing the classification error rate to 3.5% in the ImageNet competition that year, which is lower than the manual classification error rate (5%). In the past, deep learning models often faced problems such as gradient disappearance and conversion bottlenecks, which worsened the training effect when the number of hidden layers in the neural network increased. The Microsoft team proposes to use residual connections to help deeper networks train more efficiently. The method is to design a shortcut to connect the input data x to the output of the weight layer. At this time, the output feature data is H(x), the work of the weight layer will perform convolution calculation on the input data, output

its features, and finally convert it into the calculation of the residual function F(x)=H(x)-x. The advantage of this is that even if the weight layer cannot output valid feature data due to the problem of gradient disappearance, the residual value is only 0 at most. The input data x of the network layer of this layer is equal to the output feature H(x). That is, the constant value is performed. Identity mapping means that this layer does not extract new feature data and will not affect the performance of the overall network model [8].

Residual connection solves the problem of gradient disappearance caused by too deep network depth. The neural network model can add more network layers to extract more feature information to enhance network performance. The benchmarks of the ResNet network are mainly inspired by the philosophy of the VGG network. The design of the convolutional layer mainly has 3×3 filters, and the convolutional layer directly performs the downsampling with stride 2. The network ends with a global average pooling layer and a 1000-dimensional fully connected layer with softmax. Models for ImageNet classification can use ResNet architectures with different numbers of layers. According to the different network layers, ResNet has derived a variety of structural forms, including Resnet-34, Resnet-50, Resnet-101, Resnet-152, and its architecture is shown in Figure 3.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 3. Resnet architecture [8]

2.2.4. GoogLeNet

The GoogLeNet model was proposed by Google's Szegedy and won the championship in the large-scale image recognition competition ILSVRC in 2014. GoogLeNet reduces the error rate to 6.7% on classification problems. GoogLeNet adopts a concept different from AlexNet or VGG when deepening the network, adding Inception architecture to replace the original separate convolutional layer, and the number of parameters is less than AlexNet, but the network architecture is deeper and more accurate [9]. GoogLeNet uses 9 Inception architectures. Because the deep neural network is prone to the problem of gradient disappearance, in addition to using the network layer of the Inception architecture, GoogLeNet also replaces the original fully connected layer with an average pooling layer. And add two auxiliary classifiers in the middle to avoid gradient

disappearance, improve stability and convergence speed, and its architecture is shown in Figure 4.

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

Figure 4. GoogLeNet architecture [9]

2.2.5. Inception v3

The principles of the Inception v3 network design, including (i) avoid bottlenecks in the network architecture using the initial number of layers; (ii) increase the width and depth of the network to improve performance; (iii) reduce spatial aggregation through low dimensionality without compromising model capability; (iv) High-dimensional features are more suitable for local processing of the network, adding nonlinearity to the network to make training faster [10]. Inception v3 demonstrated the excellent performance of the model on the validation set of the ILSVRC 2012 Classification Task Challenge, achieving 21.2% top-1 and 5.6% top-1 for single-frame evaluation using a model with 5 billion multiply-accumulate operations per inference process 5 error rate, and the total parameters of the model are less than 2.5 million. Inception v3 uses the RMSProp optimizer, factorized 7×7 convolution, BatchNorm and label smoothing for the auxiliary classifier. In addition, it is mentioned that an output layer is added to the network. The auxiliary layer in the middle has little effect in the early stage of training, but it can improve the accuracy. Its architecture is shown in Figure 3.

type	patch size/stride or remarks	input size
conv	3×3/2	299×299×3
conv	3×3/1	149×149×32
conv padded	3×3/1	147×147×32
pool	3×3/2	147×147×64
conv	3×3/1	73×73×64
conv	3×3/2	71×71×80
conv	3×3/1	35×35×192
3×Inception	As in figure 5	35×35×288
5×Inception	As in figure 6	17×17×768
2×Inception	As in figure 7	8×8×1280
pool	8 × 8	8 × 8 × 2048
linear	logits	1 × 1 × 2048
softmax	classifier	1 × 1 × 1000

Figure 5. Inception v3 architecture [10]

2.2.6. Xception

Xception improves the Inception module of Inception v3, and introduces the concept of depth-wise separable convolution: not only does it improve network efficiency, but also better than Inception v3 when using the ImageNet dataset [11]. Xception separates spatial correlation from channel correlation, making more efficient use of parameters, model accuracy is higher, and its architecture is shown in Figure 6.

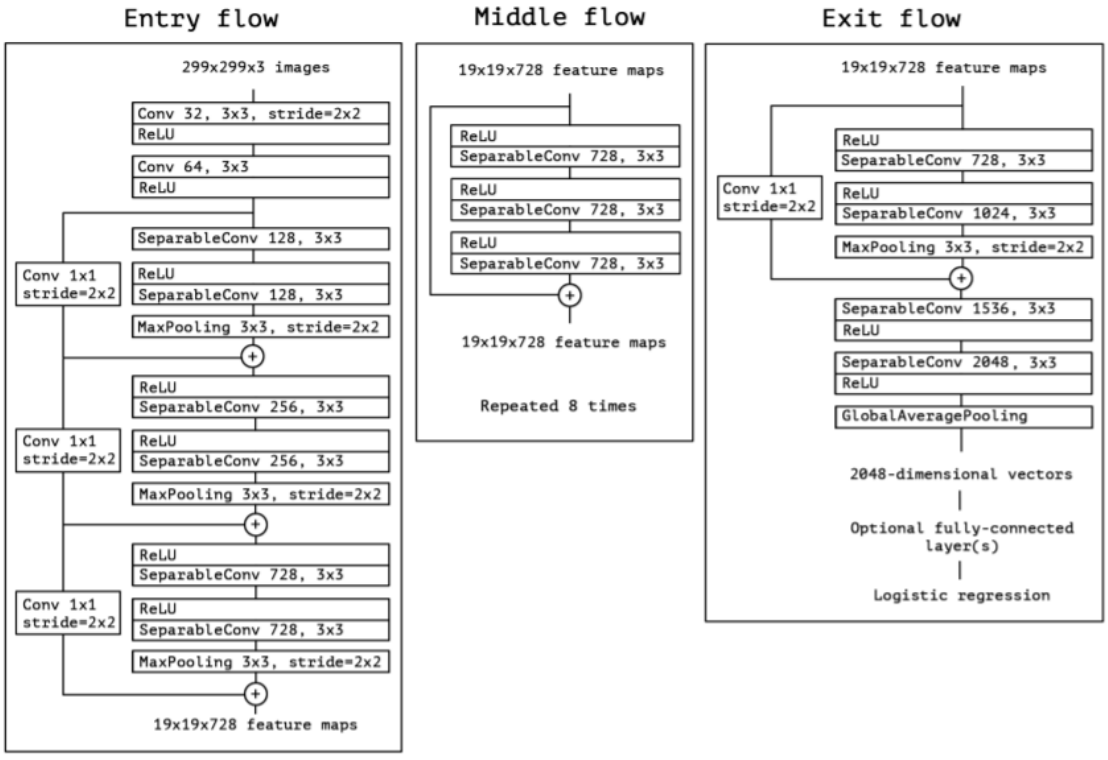

Figure 6. Xception architecture [11]

2.2.7. DenseNet201

The core idea of Densenet201 is to establish the connection relationship between different layers, make full use of features to further solve the dilemma of gradient disappearance when the network is deepened, and improve the efficiency of model training. In addition, using bottleneck layer, translation layer and smaller growth rate makes the network narrower and the parameters are reduced, which effectively suppresses over-fitting [12]. Densenet201 not only alleviates the vanishing gradient problem, but also strengthens feature propagation and greatly reduces the number of parameters; its architecture is shown in Figure 7.

Layers	Output Size	DenseNet-121($k = 32$)	DenseNet-169($k = 32$)	DenseNet-201($k = 32$)	DenseNet-161($k = 48$)
Convolution	112 × 112	7 × 7 conv, stride 2			
Pooling	56 × 56	3 × 3 max pool, stride 2			
Dense Block (1)	56 × 56	[1 × 1 conv; 3 × 3 conv] × 6	[1 × 1 conv; 3 × 3 conv] × 6	[1 × 1 conv; 3 × 3 conv] × 6	[1 × 1 conv; 3 × 3 conv] × 6
Transition Layer (1)	56 × 56	1 × 1 conv			
	28 × 28	2 × 2 average pool, stride 2			
Dense Block (2)	28 × 28	[1 × 1 conv; 3 × 3 conv] × 12	[1 × 1 conv; 3 × 3 conv] × 12	[1 × 1 conv; 3 × 3 conv] × 12	[1 × 1 conv; 3 × 3 conv] × 12
Transition Layer (2)	28 × 28	1 × 1 conv			
	14 × 14	2 × 2 average pool, stride 2			
Dense Block (3)	14 × 14	[1 × 1 conv; 3 × 3 conv] × 24	[1 × 1 conv; 3 × 3 conv] × 32	[1 × 1 conv; 3 × 3 conv] × 48	[1 × 1 conv; 3 × 3 conv] × 36
Transition Layer (3)	14 × 14	1 × 1 conv			
	7 × 7	2 × 2 average pool, stride 2			
Dense Block (4)	7 × 7	[1 × 1 conv; 3 × 3 conv] × 16	[1 × 1 conv; 3 × 3 conv] × 32	[1 × 1 conv; 3 × 3 conv] × 32	[1 × 1 conv; 3 × 3 conv] × 24
Classification Layer	1 × 1	7 × 7 global average pool			
		1000D fully-connected, softmax			

Figure 7. DenseNet201 architecture [12]

2.2.8. MobileNetV2

MobileNetV2 model was proposed by Mark Sandler et al [13]. The main branch of the original residual nets has three convolutions, the two point-by-point convolutions have more channels, so the residual nets are just the opposite. The number of convolution channels in the middle (still using the depth separation convolution structure) are more than the channels beside. In addition, removing nonlinear transformations in the main branch maintains the expressiveness of the model, its architecture is shown in Figure 8.

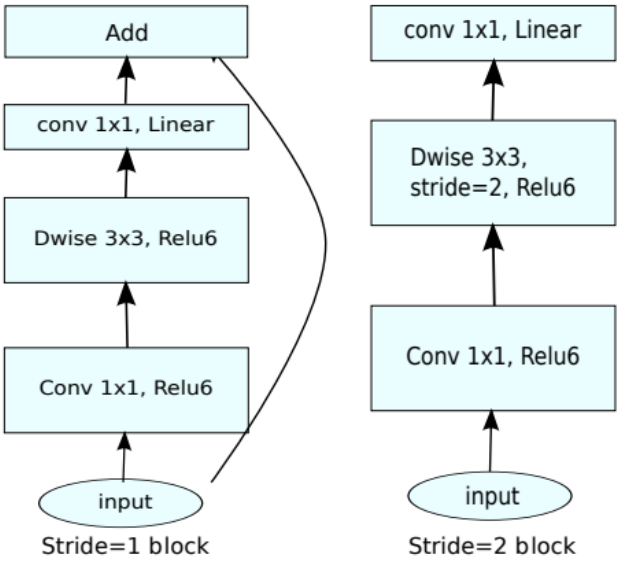

Figure 8. MobileNetV2 architecture [13]

2.2.9. ShuffleNet

ShuffleNet is a lightweight neural network convolution proposed by Megvii Technology Limited, the goal is to use limited computing resources to achieve the best accuracy, the design concept of ShuffleNet is to shuffle for different channels. The grouped convolution is to group different feature maps of the input layer, and then use different convolution kernels to convolve each group, thereby reducing the amount of convolution operations [14]. ShuffleNet uses two operations: pointwise group convolution and channle shuffle. Using channle shuffle can maximize the advantages of group convolution, and avoid its disadvantages, not only greatly reduces the amount of calculation but also ensures accuracy: its architecture is shown in Figure 9.

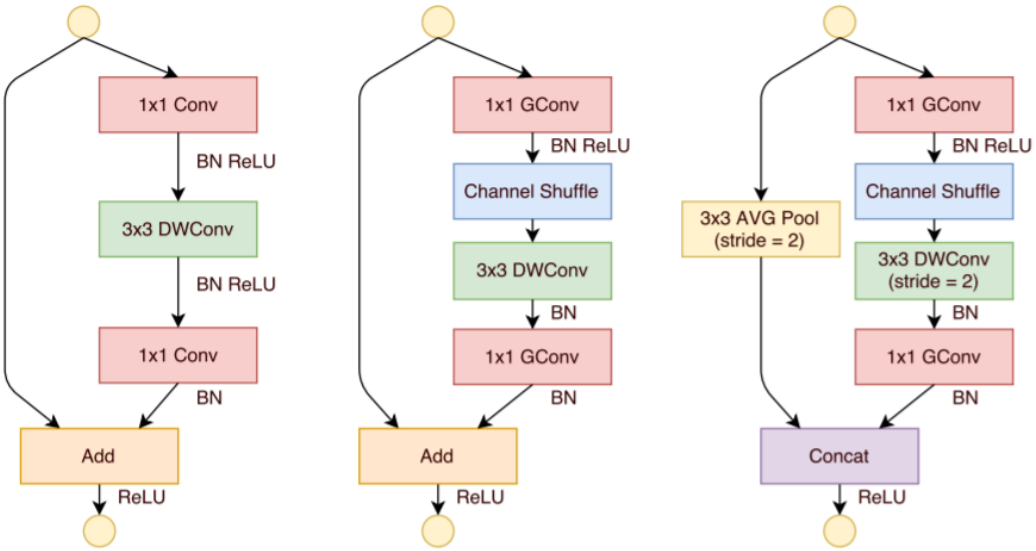

Figure 9. ShuffleNet architecture [14]

2.3. Activation function

The main function of the activation function is to convert the input signal of the node in the ANN model into an output signal. That is, the sum of the product of the input (X) and its corresponding weight (W) is converted into an output by the activation function f(X), and then sent to the next layer of neural network. This process repeats until the output layer makes a classification. This process repeats until the output layer makes a classification. In addition, we can use appropriate activation functions to improve the performance according to their application fields, including linear or nonlinear. The following are common activation functions [15].

- ReLU:

It is a function to obtain the maximum value. When the input value x is less than or equal to 0, the output value f(x) is all zero. When x is greater than 0, the output is equal to the input [16].

$$f(x) = \max(0, x) = \begin{cases} x_i, & if\ x_i \geq 0 \\ 0, & if\ x_i < 0 \end{cases}. \tag{1}$$

- Leaky ReLU:

ReLU sets all negative values to zero, while Leaky ReLU assigns all negative values a non-zero slope "a" (usually, a=0.01) [17].

$$f(x) = ax + x = \begin{cases} x, & if\ x > 0 \\ ax, & if\ x \leq 0 \end{cases}. \tag{2}$$

- ClippedReluLayer:

Input values less than zero are set to zero, and those above the clipping ceiling are set to the clipping ceiling. This clipping prevents the output from becoming too large [18].

$$f(x) = \begin{cases} 0, & if\ x < 0 \\ x, & 0 \leq x < ceiling \\ ceiling, & x \geq ceiling \end{cases}. \tag{3}$$

- ELU (Exponential linear unit):

When the input value x is greater than 0, the output is equal to the input. When the input value x is less than or equal to 0, the exponential nonlinear operation is performed. This function tends to converge the cost to zero faster to speed up learning, and avoid the vanishing gradient problem [19]. Also, the extra alpha constant term is usually set to 1.

$$f(x) = \begin{cases} x, & if\ x > 0 \\ a\ exp(x) - 1, & if\ x \leq 0 \end{cases}. \tag{4}$$

- TanhLayer:

Tanh keep the output between -1 and 1, value is 0.

$$f(x) = \frac{2}{1+e^{-2x}} - 1 \quad (5)$$

2.4. Loss function

Mean square error (MSE) is the most commonly used regression loss function, and the resulting value is the sum of the squares of the distance between the predicted value and the actual observed value. Since MSE only considers the average size of the error, not its direction. Therefore, predictions that deviate more from the true value will be penalized more than predictions that deviate less, so MSE is suitable for gradient calculation. When the value of MSE is smaller, it indicates that the prediction model has better accuracy in describing the experimental data.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \quad (6)$$

3. Proposed texture classification model

The task of this study is mainly to classify textured images, so we use various common pre-trainers to build a research model based on transfer learning techniques, as shown in Figure 10.

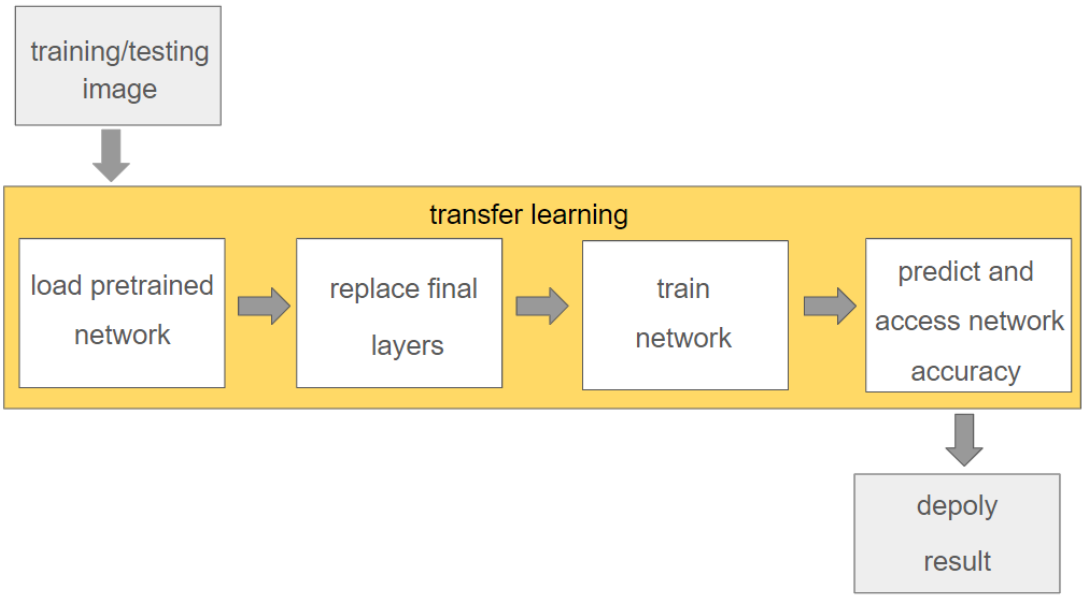

Figure 10. Texture classification model based on transfer learning and pre-trainer

3.1. Training and testing image

The texture image standard database (benchmark) is divided into two parts: training data and test data. First, input the training data to the texture classification model for training. After the training is completed, use the test data for validation. The entire validation above is described in the next section.

3.2. Transfer Learning

The ideal situation in machine learning is to have a large number of labeled training examples with the same distribution as the test data, however collecting enough training data is usually time-consuming, and semi-supervised learning can be used to solve this problem. Generally, semi-supervised methods only require a limited amount of labeled data and utilize a large amount of unlabeled data to improve learning accuracy. But in many cases, unlabeled instances are also difficult to collect, which usually makes the generated traditional models unsatisfactory. Transfer learning is a machine learning method that uses the original first task as the starting point for the second task model, allowing developers to directly transfer a neural network originally applied to another task to a new domain. [20] Transfer learning requires minimal retraining, and avoids the time-consuming and complex restarting of training models. It is suitable for situations where training data is insufficient or large-scale sample collection and sample labeling are required. The main purpose is to learn knowledge and experience from past problems and apply them to the next target area, which will achieve better results or solve problems faster.

3.3. Load pretrained network

The final layer of the pre-trained network is fine-tuned for the new classification problem, the original pre-trained network graph is retained from the pre-trained network, and the final layer is replaced with a fully connected layer, a softmax layer, and a classification output layer. Transfer these layers to a new classification task. Option to specify a new fully connected layer based on new data. Set the fully connected layer to be the same size as the number of classes in the new data. To learn faster than the transport layer in the new layer, increase the values of WeightLearnRateFactor and BiasLearnRateFactor of the fully connected layer.

3.4. Replace final layers

Use a pretrained image classification network that has been trained to extract informative features from images as a starting point for learning new tasks. Most pretrained networks are trained on a subset of the ImageNet database used for the ImageNet Large Scale Visual Recognition Challenge (ILSVRC). These networks have been trained on over a million images and can classify images into 1000 object categories such as animals, fruits, vehicles and many insects. Using a pretrained network with transfer learning is usually faster and easier than training the network from scratch.

3.5. Train network

The research model is retrained using the training data, which is a retraining of the existing neural network parameters, that is, freezing all the convolutional layers of the pretrained model and training only the fully connected layer of the last layer.

4. Experiment

4.1. Experimental environment

The development environment is Matlab R2020a [21] for Windows 10 (x64) operating system, and runs on a PC with Intel Core i5-11400 CPU, 3.2GHz and 16G RAM.

4.2. Texture image databases

In this study, two different texture image databases were tested, including the Describable Textures Dataset (DTD) database with complex textures and the University of Illinois at Urbana-Champaign (UIUC) database with regular textures.

4.2.1. Describable Textures Dataset (DTD) [22]

It is an image database with complex texture, divided into 47 different categories according to human perception, and each category has 120 images with a total of 5,640 images, and the image size ranges from 300x300 to 640x640. All images of DTD are obtained from Flickr and Google websites, as shown in Figure 11.

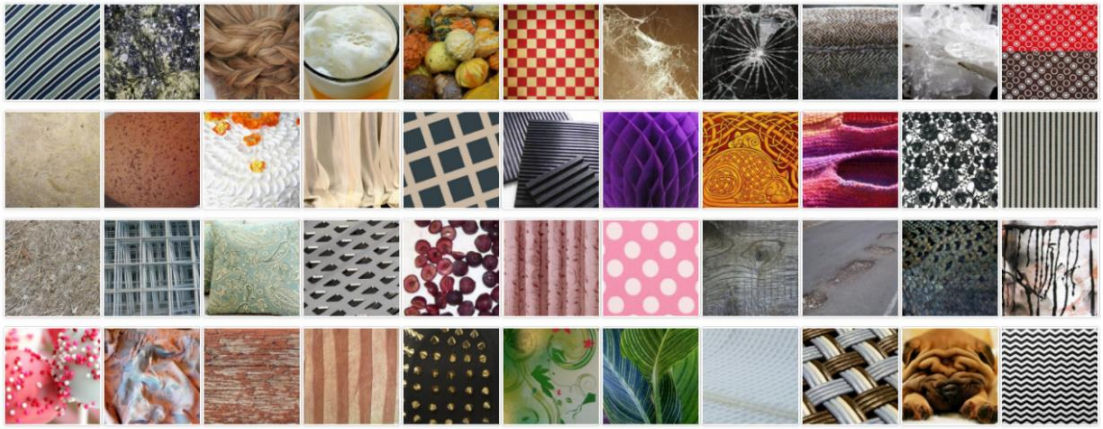

Figure 11. Describable Textures Dataset (DTD) [22]

4.2.2. University of Illinois at Urbana-Champaign(UIUC) Textures Dataset [23]

UIUC was proposed by the University of Illinois at Urbana-Champaign. There are 25 different categories of regular texture images, each category has 40 images with a total of 1000 images, and the image size is 640 × 480 grayscale images. The database contains artificial and natural textures, such as materials, fabrics, etc., as shown in Figure 12.

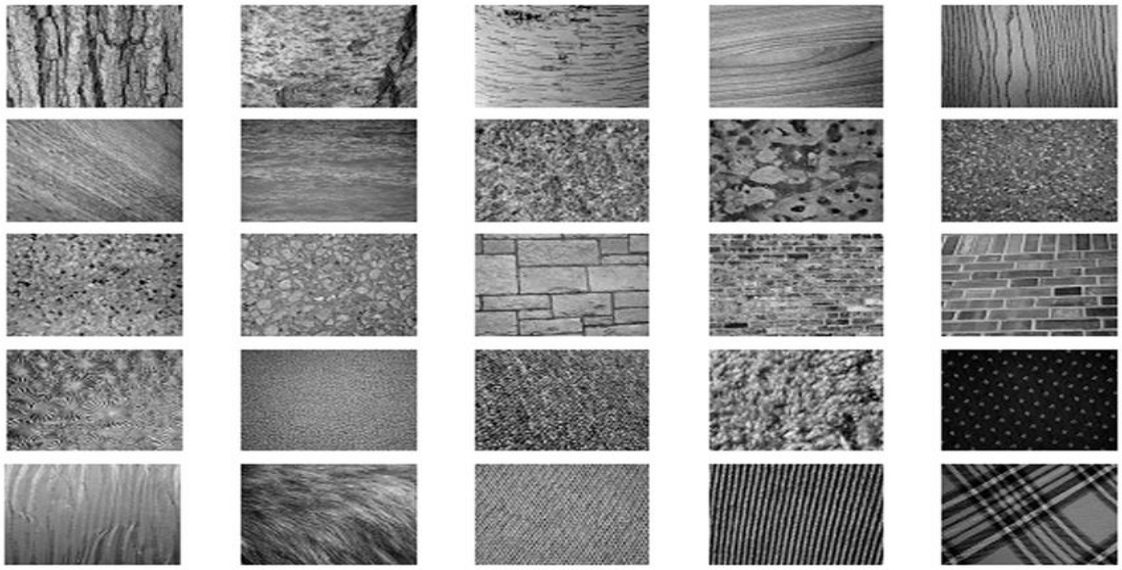

Figure 12. University of Illinois at Urbana-Champaign(UIUC) [23]

4.3. Evaluation Indicators

After classifying texture images, the model needs to be evaluated for performance. We use the confusion matrix of binary classification to record the prediction results of the classifier [24], as shown in Figure 13.

		Predicted class	
		Positive	Negative
True class	Positive	TP (True positive)	FN (False negative)
	Negative	FP (False positive)	TN (True negative)

Figure 13. Confusion matrix for binary classification

In the confusion matrix, True Positive (TP) indicates that the actual condition is "Positive", and the model predicts the number of "Positive". True Negative (TN) indicates that the actual condition is "Negative", and the model predicts the number of "Negative". False Positive (FP) indicates that the actual condition is "Negative", and the model predicts the number of "Positive". False Negative (FN) indicates that the actual condition is "Positive" and the model predicts the number of "Negative".

Then, we use 4 commonly used evaluation indicators, accuracy, precision, recall and F1-score to evaluate model performance. The accuracy is the proportion of correctly classified samples to the total number of samples, such as formula (8). Precision is the ratio of the number of correctly classified positive samples to the number of samples determined by the classifier as positive samples, as shown in formula (9). The recall is the ratio of the number of correctly classified positive samples to the number of true positive samples, such as formula (10). The F1-score is a comprehensive index that seeks a balance between precision and recall, such as formula (11).

$$\text{Accuracy} = \frac{TP+TN}{TP+FP+TN+FN} \quad (8)$$

$$\text{Precision} = \frac{TP}{TP+FP} \quad (9)$$

$$\text{Recall} = \frac{TP}{TP+FN} \quad (10)$$

$$\text{F1-score} = \frac{2\times Precision\times Recall}{Precision+Recall} \quad (11)$$

4.4. Validation

With sufficient validation data, in order to confirm the performance of various trainers, we use the holdout method for validation, that is, take most of the texture samples for training, and the rest for testing. We use 70% of the samples for training and the remaining 30% for testing, and finally employ the evaluation metrics accuracy, precision, recall and F1-score to compare the performance of various trainers.

4.5. Experimental results

4.5.1. Experimental results for DTD database

In this experiment, it can be observed that DenseNet201 has the best performance among all trainers, with an accuracy of 62.5%, a precision of 60.31%, a recall of 59.04% and an F1-Score of 59.67%, as shown in Table 1. In addition, it was also found that the performance of ResNet with the deepest depth did not increase accordingly. This means that the deep trainer is overfitting for complex textures.

Table 1. Experimental results of DTD

Model	Precision	Recall	F1-Score	Accuracy
AlexNet	47.65	44.14	45.83	47.32
VGGNet16	17.00	10.10	12.67	12.50
VGGNet19	6.06	6.20	6.13	8.03
ResNet18	57.00	53.00	55.00	56.00
ResNet50	58.00	53.00	56.00	57.00
ResNet101	57.19	55.67	56.42	58.63
GoogLeNet	56.00	54.00	55.00	55.00
Inception-v3	40.00	42.00	41.00	42.00
Xception	56.54	49.29	52.67	53.86
DenseNet201	60.31	59.04	59.67	62.50
MobileNetV2	46.00	44.00	45.00	44.00
ShuffleNet	51.89	49.29	50.56	49.29

4.5.2. Experimental results for UIUC database

In this experiment, it can be observed that ResNet18 and DenseNet201 have the best performance, the accuracy rate is 99.33%, the precision rate is 99.38%, the recall rate is 99.33% and the F1-Score is 99.35%, as shown in Table 2. And from Table 1 and Table 2, it can also be found that the performance of a trainer for different types of texture

databases is also very different. It is proved that the differences of texture datasets will affect the whole experimental results.

Table 2. Experimental results of UIUC

Model	Precision	Recall	F1-Score	Accuracy
AlexNet	94.74	93.66	94.20	93.66
VGGNet16	83.36	80.66	81.99	80.66
VGGNet19	75.65	70.00	72.71	70.00
ResNet18	99.38	99.33	99.35	99.33
ResNet50	98.74	98.66	98.70	98.66
ResNet101	98.51	98.00	98.25	98.00
GoogLeNet	97.03	96.66	96.85	96.66
Inception-v3	95.69	95.33	95.51	95.33
Xception	97.82	97.66	97.74	97.66
DenseNet201	99.38	99.33	99.35	99.33
MobileNetV2	98.47	98.33	98.40	98.33
ShuffleNet	96.35	95.66	96.00	95.66

4.5.3. Experimental results of activation function for DTD database

We experiment with different activation functions on the GoogleNet trainer. The results show that ReLU has the best performance, with 54.76% accuracy, 55.88% precision, 53.54% recall and 54.69% F1-Score, as shown in Table 3. This also shows that different activation functions can affect the overall trainer performance.

Table 3. Experimental results of activation functions for DTD database

Model	Precision	Recall	F1-Score	Accuracy
ReLU	55.88	53.54	54.69	54.76
Leaky ReLU	48.46	48.22	48.34	48.22
ClippedReluLayer	28.16	22.34	24.91	22.34
ELU	50.63	47.87	49.21	47.87
TanhLayer	N/A	N/A	N/A	N/A

4.5.4. Experimental results of activation function for UIUC database

We experiment with different activation functions on the GoogleNet trainer. The results show that ReLU has the best performance, with 96.13% accuracy, 96.71% precision, 96.13% recall and 95.89% F1-Score, as shown in Table 4. This also shows that different activation functions can affect the overall trainer performance.

Table 4. Experimental results of activation functions for UIUC database

Model	Precision	Recall	F1-Score	Accuracy
ReLU	96.71	96.13	95.89	96.13
Leaky ReLU	93.67	93.00	93.91	93.00

ClippedReluLayer	85.65	82.42	83.91	82.42
ELU	94.82	94.59	94.91	94.59
TanhLayer	N/A	N/A	N/A	N/A

4.5.5. Discussion of experimental results

We can observe from experiments 4.5.1 and 4.5.2 that whether using DTD or UIUC, two different databases, in terms of texture classification, the accuracy of using DenseNet201 network is better than other existing deep learning models. It can also be observed from experiments 4.5.3 and 4.5.4 that using the ReLU activation function is also better than other activation functions. From the above four experimental results, it can be known that the results of experiments 4.5.2 and 4.5.4 using the UIUC data set are obviously better than those using the DTD data set in 4.5.1 and 4.5.3. The reason is speculated that the texture data in the UIUC data set is relatively simple. In contrast, the patterns in the DTD data set are very different and complex in texture. In this experiment, 3948 texture images from the DTD dataset were used to train the DenseNet201 network. In this experiment, the learning rate was set to 0.0001, the epoch was 12, and the loss value converged below 0.01. The training time took a total of 311.10 minutes. The DenseNet201 network was trained using 840 texture images in the UIUC dataset. The learning rate was set to 0.0001, the epoch was 12, and the loss value converged below 0.01. The training time took a total of 17.49 minutes.

5. Conclusions

In order to understand the performance comparison of various well-known deep learning trainers against standard texture databases, to provide researchers with the ability to use appropriate trainers for texture classification of various subjects. Therefore, this study uses transfer learning technology to conduct experiments on the complex texture database DTD and the texture regular database UIUC. The experimental results on the DTD database show that DenseNet201 performs the best among all trainers, with an accuracy of 62.5%, precision of 60.31%, recall of 59.04%, and F1-Score of 59.67%. The experimental results of the UIUC database show that ResNet18 and DenseNet201 have the best performance, with an accuracy rate of 99.33%, a precision rate of 99.38%, a recall rate of 99.33%, and an F1-Score of 99.35%. Throughout the study we found that it is not the deeper the trainer, the higher the accuracy, because the complex texture makes the trainer overfitting. At the same time, the classification performance of a trainer for different types of texture databases is also very different, which proves that the difference of texture datasets will affect the whole experimental results. In addition, there are many studies proving that data augmentation can improve performance, and data augmentation will be carried out in the future to further demonstrate whether it can also increase performance in texture images.

References

[1] T. B. Chandra, K. Verma, B. K. Singh, D. Jain, S. S. Netam, "Automatic detection of tuberculosis related abnormalities in chest X-ray images using hierarchical feature extraction scheme," Expert Systems with Applications, Volume 158, 2020.

[2] X. Li, Y. Lu, J. Xiong, D. Wang, D. She, X. Kuai, D. Geng, B. Yin, "Presurgical differentiation between malignant haemangiopericytoma and angiomatous meningioma by a radiomics approach based on texture analysis," Journal of Neuroradiology, Volume 46, Issue 5, 2019, pp. 281-287.

[3] C. Wen, B.Y. Cao, Z.Q. Shi, Y. J. Ma, J. X. Wang, W. B. Yang, "Quantitative analysis on the oxygen diffusion in pyramidal textured surfaces of silicon and copper via transmission electron microscopy," Materials Science in Semiconductor Processing, Volume 121, Jan. 2021.

[4] M. Duan, X. Zhang, "Using remote sensing to identify soil types based on multiscale image texture features," Computers and Electronics in Agriculture, Volume 187, Aug. 2021.

[5] P. H. Koch, C. Lund, J. Rosenkranz, "Automated drill core mineralogical characterization method for texture classification and modal mineralogy estimation for geometallurgy," Minerals Engineering, Volume 136, 2019, pp. 99-109.

[6] M. Tuceryan, A. K. Jain, "Texture segmentation using voronoi polygons," IEEE Trans. Pattern Anal. Mach. Intell., Volume 12, Issue 2, Feb. 1990, pp. 211-216.

[7] J. Y. Tou, Y. H. Tay, P. Y. Lau, "One-dimensional greylevel co-occurrence matrices for texture classification," In International Symposium on Information Technology, Volume 3, Aug. 2008, pp. 1-6.

[8] U. Dixit, A. Mishra, A. Shukla, R. Tiwari, "Texture classification using convolutional neural network optimized with whale optimization algorithm," SN Applied Sciences, May 2019.

[9] R. Azadnia, A. Jahanbakhshi, S. Rashidi, M. Khajehzadeh, P. Bazyar, "Developing an automated monitoring system for fast and accurate prediction of soil texture using an image-based deep learning network and machine vision system," Measurement, Volume 190, Feb. 2022.

[10] F. H. C. Tivive, A. Bouzerdoum, "Texture classification using convolutional neural networks," TENCON 2006 - 2006 IEEE Region 10 Conference, Nov. 2006.

[11] E. Bębas, M. Borowska, M. Derlatka, E. Oczeretko, "Machine-learning-based classification of the histological subtype of non-small-cell lung cancer using MRI texture analysis," Biomedical Signal Processing and Control, Volume 66, April 2021.

[12] S. Kumar, L C. S. Redd, S. G. Joseph, V. K. Sharma, S. H, "Deep learning based model for classification of COVID-19 images for healthcare research progress," Materials Today: Proceedings, Volume 62, 2022, pp. 5008-5012.

[13] M. X. Bastidas-Rodrigueza, L. Polania, A. Gruson, F. Prieto-Ortiz, "Deep Learning for fractographic classification in metallic materials," Engineering Failure Analysis, Volume 113, July 2020.

[14] İ. Kırbaş, A. Çifci, "An effective and fast solution for classification of wood species: A deep transfer learning approach," Ecological Informatics, Volume 69, Jul. 2022.

[15] L. Wang, J. Wang, Z. Liu, J. Zhu, F. Qin, "Evaluation of a deep-learning model for multispectral remote sensing of land use and crop classification," The Crop Journal, Feb. 2022.

[16] A. Krizhevsky, I. Sutskever, G. E. Hinton, "Imagenet classification with deep convolutional neural networks," Advances in Neural Information Processing Systems, 2012.

[17] K. Simonyan, A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in CoRR, abs/1409.1556, 2014.

[18] K. He, X. Zhang, S. Ren, J. Sun, "Deep residual learning for image recognition," Proceedings of CVPR, Jun. 2016.

[19] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, "Going deeper with convolutions," Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Jun. 2015.

[20] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, Z. Wojna, "Rethinking the inception architecture for computer vision," Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, 2016.

[21] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," arXiv preprint arXiv:1610.02357, Apr. 2017.

[22] G. Huang, Z. Liu , L. Maaten, K. Q. Weinberger, "Densely connected convolutional networks," In CVPR, 2017.

[23] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 4510-4520.

[24] X. Zhang, X. Zhou, M. Lin, J. Sun, "ShuffleNet: An extremely efficient convolutional neural network for mobile devices," 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, Jun. 2017.

[25] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016.

[26] V. Nair, G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," In Proceedings of the 27th International Conference on Machine Learning, 2010, pp. 807-814.

[27] A. L. Maas, A. Y. Hannun, A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," Computer Science Department, Stanford University, CA 94305 USA, 2013.

[28] D.-A. Clevert, T. Unterthiner, S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (ELUs)," arXiv preprint arXiv:1511.07289, 2015.

[29] A. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates, A. Y. Ng, "Deep Speech: Scaling up end-to-end speech recognition," arXiv:1412.5567, 2014.

[30] S. J. Pan, Q. Yang, “A survey on transfer learning,” IEEE Transactions on Knowledge and Data Engineering, Volume 22, Issue 10, 2010, pp. 1345–1359.
[31] H. Demuth, M. Beale, “Neural network toolbox for use with MATLAB,” The Math Works, Inc., 1992.
[32] M. Cimpoi, S. Maji, I Kokkinos et al., “Describing textures in the wild,” In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Columbus, OH, USA, 2014, pp. 3606–3613.
[33] S. Lazebnik, C. Schmid, J. Ponce, “A sparse texture representation using local affine regions,” IEEE Transactions on Pattern Analysis and Machine Intelligence, Volume 27, Issue 8, 2005, pp.1265–1278.
[34] E. Alpaydin, Introduction to Machine Learning (4/e), Summit Valley Press, Mar. 2020.