Remote Sensing Image Scene Classification by three State-of-the-Art Convolutional Neural Networks using Transfer Learning

ENGG*6100 Machine Vision

Spencer Walls, 0734584

April 29th, 2019

Contents

1. Introduction

The rise of Earth observation technology has produced an abundance of high-resolution imagery of the surface of the Earth, generated via technologies such as synthetic aperture radar (SAR) [1] and multi/hyperspectral imaging [2]. This abundance of satellite imagery, existing at many different spatial, spectral, and temporal resolutions [3], provides opportunities for the extraction of high-level semantic information regarding the many diverse features of the Earth's surface. Sophisticated software tools and algorithms that can accurately identify and classify such features from images taken by these space- and air-borne platforms are therefore in high demand. For example, such techniques are of utmost importance to the classification of land-use and land cover (LULC) imagery [4]. This has given birth to the research area of *Remote sensing image scene classification* [3, 4], a component of the broader field of *Aerial and satellite image analysis* [3] (also referred to as *Remote sensing image interpretation* [4]). Remote sensing image scene classification has been given significant attention in recent years as an active research area of computer vision [3, 4, 8, 10, 11, 12 – 21, 25, 32 – 34].

Remote sensing image scene classification aims to automatically assign explicit semantic labels to a wide range of remote sensing image scene patches with respect to their contents. The specific goal is to categorize all relevant scene images into a discrete set of LULC classes as accurately as possible [3]. Remarkable advances have been made in this field in recent years, namely due to the numerous practical applications for which this problem is fundamental, such as land resource management, traffic control, urban planning, and disaster monitoring [5 – 8]. Such progress may also be attributed to the growing prominence of deep learning based approaches to computer vision, which have achieved state-of-the-art performance on classification tasks [3, 4, 8, 25, 32 – 34]. Nevertheless, despite the drastic improvement to classification accuracy in recent years, certain scene classes are still frequently mis-classified by deep learning models, particularly when multiple classes with many visual similarities are present [4]. Thus, far from a solved problem, remote sensing image scene classification remains an active research area in computer vision.

The present study applies three state-of-the-art Convolutional Neural Network (CNN) models to the land-use classification problem defined by the *UC Merced Land-Use* dataset [40], a comprehensive benchmark dataset for remote sensing image scene classification that is comprised of 21 different land-use classes. The architectures consist of *Xception* [58], *VGG-19* [59], and *MobileNet* [60], none of which has yet been applied to this specific dataset. These CNN architectures have all been pre-trained on the landmark *ImageNet* [56] dataset, for which they produce state-of-the-art results. The present study thus employs transfer learning, whereby the pre-trained CNNs extract feature maps from the images, which are then passed to a fully-connected Multilayer Perceptron (MLP) that makes the final land-use classification. Results are compared to other state-of-the-art methods that have been applied to the UC Merced Land-Use dataset.

2. Literature Review

The field of aerial and satellite image analysis commenced in the early 1970s [3]. At this time, the spatial resolution of satellite imagery was quite low, and the size of pixels was therefore usually coarser than the actual objects of interest [9]. As a result, most of the techniques for aerial and satellite image analysis that emerged during this time were geared towards per-pixel or even sub-pixel analysis [9]. With the drastic improvement of remote sensing technology over the last couple decades, today the spatial resolution of the objects of interest is typically comprised of many different pixels, which has essentially eliminated the need to meticulously categorize scene images at the level of individual pixels.

Rather than being concerned with the statistical analysis of individual pixels, researchers began to focus on the spatial patterns generated by multitudes of adjacent pixels. This idea was proposed by Blaschke and Strobl in 2001 [10], which gave birth to a paradigm-shift from pixel-level interpretation to what is called object based image analysis (OBIA). As the name suggests, this involves delineation at the object level, whereby “object” denotes a semantic entity or scene component that is discernable in a digital image. The central goal of OBIA is to accurately partition the semantic objects contained in an image into categories or classes which are defined in terms of specific color, spectral, and texture information [3].

Far superior to earlier methods that were concerned with single pixel analysis, OBIA continued to dominate the field of aerial and satellite image analysis for decades [3, 10, 11]. Nevertheless, despite exceptional performance on some standard land-use identification tasks, the approach of OBIA did not allow for true semantic-level interpretation and understanding of

LULC classes from digital imagery. Not until certain theoretical advancements took place did true semantic-level remote sensing image scene classification become possible. Individual scene images, which are typically local image patches that have been extracted from a larger remotely sensed image, must be successfully labelled with a precise and explicit semantic class.

Today, the different approaches taken towards remote sensing image scene classification may be broadly grouped into three categories: handcrafted feature based methods, unsupervised feature learning based methods, and deep feature learning based methods [3]. Important research within these areas is briefly summarized below. It is evident that these three areas are not entirely independent of one another as various methodologies can overlap. The vast majority of state-of-the-art methods, in addition to the present study, employ some sort of deep feature learning. Thus, considerable weight is given to this third category in the following discussion.

2.1 Handcrafted Feature Based Methods

The earliest research in remote sensing image scene classification depended on handcrafted features [12 – 15]. Such methods typically demand significant feature-engineering skills as well as domain-specific expertise, in effort to design numerous handcrafted features such as shape, texture, colour, spatial, and spectral information, and combinations therein [3]. Some noteworthy methods include *scale invariant feature transform* (SIFT) [16], *texture descriptors* [17], and *color histograms* [18].

The SIFT method describes subregions via gradient information that encompasses specific identified keypoints. The ‘standard’ or ‘sparse’ SIFT approach integrates histogram based gradient representation and keypoint detection. SIFT is very distinctive and invariant to fluctuations in rotation, scale, and lighting [16]. In regard to the texture descriptors approach, examples that have been applied to remote sensing image scene classification include *Gabor filters* [19] and *co-occurrence matrix* [20]. These approaches are appropriate for textural scene image identification, since the features are typically calculated by putting primitives in local image subregions and observing the relative differences [17].

Lastly, the global color histogram feature is arguably the simplest when considering all handcrafted features; however, it is a powerful visual feature for scene classification. Besides mathematical simplicity, a defining characteristic of color histograms is that “they are invariant to rotation and translation about the viewing axis" [3]. Nevertheless, this approach is still limited as it fails to convey spatial information and is sensitive to quantization errors [3].

2.2 Unsupervised Feature Learning Based Methods

Unsupervised feature learning from unlabelled images has emerged as a viable approach to remote sensing image scene classification in recent years, instead of the limited method of handcrafted feature selection. This approach, which involves the automation of feature learning by specific machine learning algorithms, has made substantial progress in recent years. Unsupervised feature learning strives to learn a range of filters or basis functions that are used for feature encoding, whereby the function's input is an array of handcrafted features and the output is an array of features that have been learned. This process of feature learning allows for more discriminative features to be obtained than that which can be from using strictly handcrafted features [3]. Some noteworthy unsupervised feature learning methods are *k-means clustering* and *autoencoder* [3, 21].

The *k*-means clustering approach involves applying a vector quantization technique in effort to partition datapoints into *k* different clusters, whereby datapoints in the same cluster are more similar to each other than datapoints in other clusters. This approach is considered to be a standard unsupervised learning algorithm which is typically employed on datasets that are unlabelled [3]. The autoencoder approach, which has been successfully applied to remote sensing image scene classification by several authors [3, 21, 22], is a symmetrical neural network that is employed to learn a compressed feature representation from a high dimensional feature space [3]. This approach achieves this by "minimizing the reconstruction error between the input data at the encoding layer and its reconstruction at the decoding layer" [3] by means of a back-propagation based learning algorithm.

Overall, unsupervised feature learning based methods have performed quite well for remote sensing image scene classification problems in comparison to the handcrafted feature based approaches. Nevertheless, the category labels typically provide insufficient semantic information, which as a result does not ensure that the best possible discernment between classes is being learned. The unsupervised methods are indeed limited because they do not exploit class information. In order to optimize classification performance, labelled data must be employed. This allows for powerful supervised feature learning methods to extract rich and meaningful features [3]. This current state-of-the-art approach to remote sensing image scene classification is discussed in the following subsection.

2.3 Deep Feature Learning Based Methods

In 2006, a profound breakthrough was made by Hinton and Salakhutdinov in the area of deep feature learning [21]. Since then, the vast majority of the state-of-the-art approaches for remote sensing image scene classification have relied on deep supervised learning to acquire adequate representations of features. The primary objective of researchers today is thus to replace handcrafted features with flexible and trainable deep neural networks. In addition to the successful application to many other computer vision problems, these models have demonstrated very impressive feature representation capability in the area of remote sensing image scene classification [3, 4, 23 – 25].

Deep neural networks are incredibly powerful, namely due to their ability to automate the feature learning process, whereby they learn from data via a back-propagation based learning algorithm such as stochastic gradient descent. This technology is a big step from relying on handcrafted features and the corresponding domain-specific expertise required. Furthermore, in contrast to the unsupervised learning approaches which typically consist of quite shallow architectures, deep learning models are comprised of numerous processing layers that allow for more adequate and sophisticated feature representation. For example, each subsequent layer in a given network builds on top of the feature representations of the previous layer, resulting in increasingly abstract feature representations being built from layer to layer. As the input data makes its way through the network it is therefore represented at multiple levels of abstraction. This allows the final layers of the network to represent images in terms of abstract and semantic properties [3].

Deep learning applications in computer vision are currently dominated by a class of deep neural networks called Convolutional Neural Networks (CNNs), which are employed in the present study. CNNs are able to process information contained within multi-dimensional arrays such as multi-spectral images, which are comprised of several two-dimensional arrays pertaining to pixel intensities in the red green blue (RGB) color space [3]. Beginning with the profound success of *AlexNet* [26], many successful CNN model architectures have been proposed for image classification, such as *VGGNet* [27], *GoogLeNet* [28], *ResNet* [29], and *NASNet* [30]. Four fundamental ideas that exploit inherent properties of natural signals drive the theoretical underpinnings of CNNs: "local connections, shared weights, pooling, and the use of many layers" [3, 31]. The standard architecture of a CNN is defined by a series of interconnected layers, namely *convolutional layers*, *pooling layers*, *normalization layers*, and *fully-connected layers*.

The convolutional layers are primarily responsible for feature extraction, whereby the first few layers learn low-level features, such as lines, edges, and corners, and the layers existing deeper within the network subsequently extract higher-level semantic features, such as physical objects and shapes. Each convolutional layer is followed by a pooling layer, whereby a type of local non-linear mathematical operation is executed over a specific region of an image that contains a certain feature. Pooling layers improve the computational flexibility of CNNs by ensuring adequate feature representation even in the presence of small translations and rotations within images [3, 31]. This is of particular importance when dealing with remote sensing image scene classification problems, whereby input images have typically been cropped from substantially larger images. Multiple methods of pooling exist; two frequently employed approaches include max-pooling and global-average-pooling [31, 58 – 60].

The normalization layers, which are inspired by the inhibition schemes used by biological neurons, aim to augment the generalization capacity of the CNN [3]. However, normalization layers have been implemented much less in recent years, due to the conclusion that these layers have minimal impact on the accuracy of a CNN [4, 31]. The fully-connected layers typically consist of a few layers that exist at the very end of a CNN. This section of the model represents a classic, feed-forward MLP, which takes the output of the features extracted from the previous layers as input. The fully-connected section typically consists of an input layer, one or two hidden layers, and an output layer, which corresponds to probability-based predictions with respect to the specific classes the model aims to classify. Overall, the fully-connected section of a CNN removes constraints and is thus better able to summarize the features extracted by the previous lower-level layers [3], and finally make a reasonable prediction with respect to the features learned from the input data.

Indeed, CNNs have become tremendously popular for remote sensing image scene classification in recent years [3, 4, 8, 25, 32 – 34]. Today, numerous publicly available, high-resolution remote sensing image datasets exist which are used to evaluate different CNN models in this specific area. Noteworthy datasets include the *WHU-RS19* [35], *SIRI-WHU* [36], *RSSCN7* [37], *RSC11* [38], and *Brazilian Coffee Scene* [39] datasets, in addition to the *UC Merced Land-Use* [40] dataset which is employed in the present study.

3. Materials

3.1 UC Merced Land-Use Dataset

The UC Merced Land-Use dataset is comprised of 2100 aerial scene images which are partitioned into 21 land-use classes, whereby precisely 100 images are present for each individual class. Sample images from each of the 21 classes are provided in Figure 1, which was extracted from Zheng et al. [4]. The pixel size of the images is 256 x 256, and the spatial resolution is 0.3m in the RGB color space. All images of the dataset were selected from aerial orthoimagery that was downloaded from the United States Geological Survey (USGS) National Map. The images were taken in the following regions of the United States: Birmingham, Boston, Buffalo, Columbus, Dallas, Harrisburg, Houston, Jacksonville, Las Vegas, Los Angeles, Miami, Napa, New York, Reno, San Diego, Santa Barbara, Seattle, Tampa, Tucson, and Ventura.

The problem presented by the UC Merced Land-Use dataset is challenging namely due to the wide diversity of land-use categories that are present. For example, the dataset contains some significantly overlapped classes, such as sparse residential, medium residential, and dense residential, which really only differ in terms of the precise density of physical structures that are present. This makes differentiation and thus classification between these specific classes quite difficult [4]. The UC Merced Land-Use dataset is the most widely employed dataset for the task of remote sensing image scene classification [4, 41 – 51].

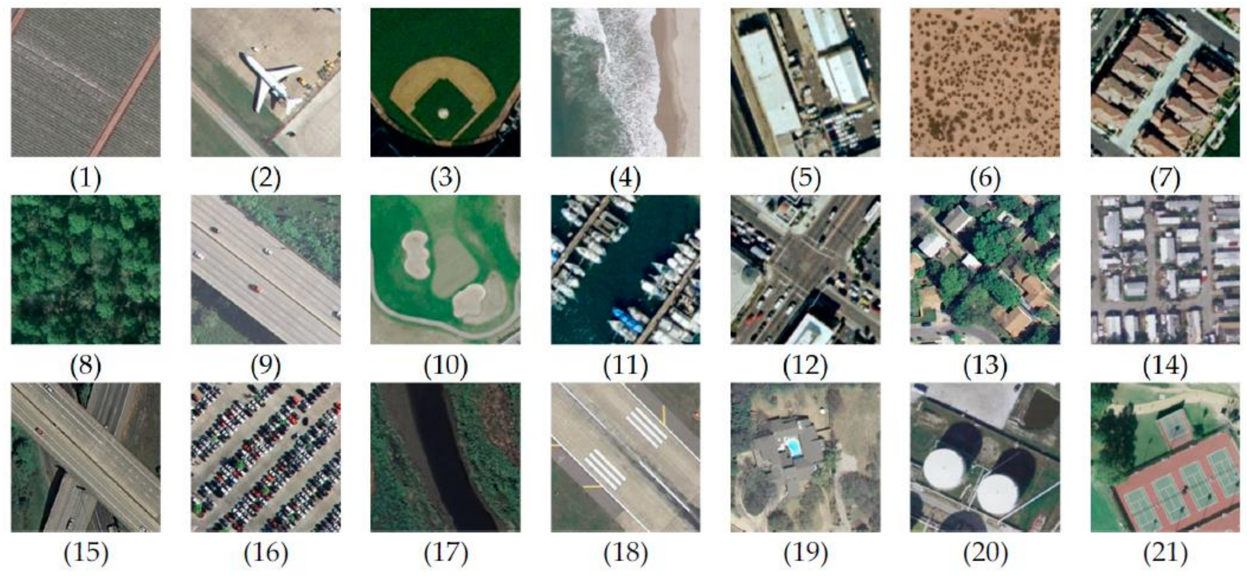


Figure 1. Sample images from the 21 classes of the UC Merced Land-Use dataset: (1) agricultural; (2) airplane; (3) baseballdiamond; (4) beach; (5) buildings; (6) chaparral; (7) dense residential; (8) forest; (9) freeway; (10) golf course; (11) harbor; (12) intersection; (13) medium residential; (14) mobile home park; (15) overpass; (16) parking lot; (17) river; (18) runway; (19) sparse residential; (20) storage tanks; and (21) tennis court [4].

3.2 Training Infrastructure

All models were implemented using the *Keras* framework that is present in the Python programming language [61]. Furthermore, all experimentation was conducted on a Windows 10 operating system with one 2.50 GHz 2-core i5-7200CPU processor and 8GB of memory. Limited computational resources indeed restricted the present study, as no graphics processing unit (GPU) was available to accelerate computing.

4. Methodology

This section provides the following: a brief overview of the theory behind CNNs and transfer learning; in-depth descriptions of the three CNN architectures employed; the specifications of the fully-connected section that makes the final land-use classifications; and the evaluation protocol.

4.1 Overview of CNN theory

A CNN is a certain type of Artificial Neural Network (ANN). An ANN model is essentially a nonlinear statistical technique that is inspired by and loosely modelled after the learning capabilities of a biological brain. ANNs are simplified mathematical models of biological neural networks which consist of thoroughly interconnected processing nodes that are arranged in layers and connected by connection weights [52]. The weights are iteratively tuned by stochastic-gradient-based heuristic processes over a loss function, which is a computationally demanding task. Nevertheless, provided that sufficient data are available, these models of computation have a profound ability to learn and generalize by processing training examples and recognizing patterns in the behaviour of a system [53]. Through their profound generalization capacity, the models learn from examples that contain consistent outputs relative to different combinations of inputs [54]. These sophisticated computational models not only can reproduce the results of training examples but more importantly can predict the results of new and unseen examples with exceptional accuracy.

The CNN is thus a type of ANN that is widely used in computer vision tasks such as image classification, object detection, and recommender systems [4]. The theory behind this model is loosely based off of the functionality of the visual cortex in a biological brain [55]. The basic architecture of a CNN is given in Figure 2, which was extracted from Zheng et al. [4].

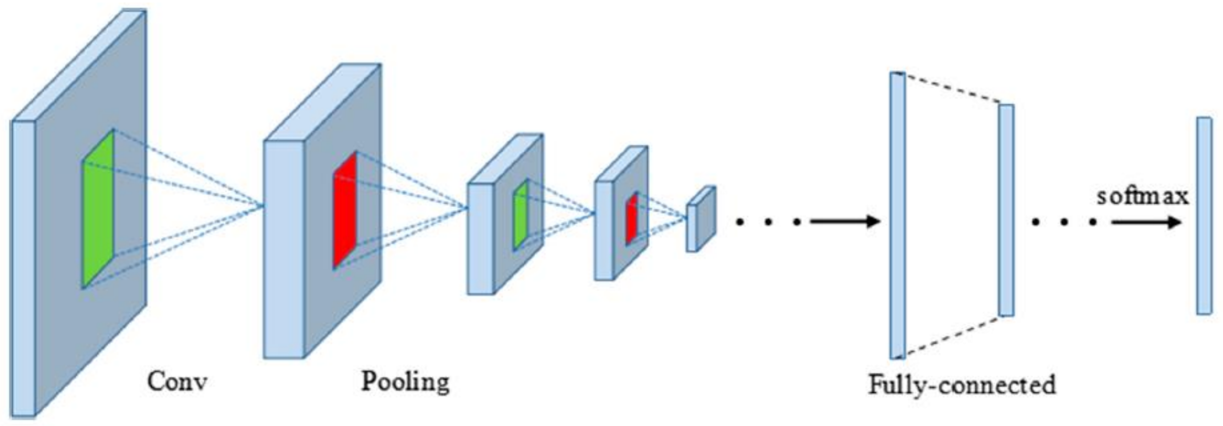


Figure 2. Standard CNN architecture [4].

The CNN algorithm is essentially comprised of two fundamental objects: convolutional layers, which are the primary building-blocks of the CNN, and pooling layers [4]. The algorithm commences by taking an image as input which is inputted to the first convolutional layer. In a convolutional layer, the individual image is scanned by many different feature detectors or filters, each of which is designed to detect a certain feature that might be present in the image.

The feature detectors are iteratively tuned by a stochastic-gradient-based learning algorithm that works to optimize a loss function, typically *cross-entropy*. As each feature detector separately scans the input image, dot products between separate local regions in the input image and the feature detectors are computed [4]. The output of this operation is numerous feature maps, each of which is the result of the convolutions between the local regions of the input image and a specific feature detector. The output of the convolutional layer is thus comprised of many different feature maps. This output is then inputted to a pooling layer, after being processed through the *rectified linear unit* (ReLU) activation function to eliminate any possible linearity in the feature maps, which squashes the output of the CNN to be within a certain numerical range (see sec 4.4.2 for a description of this function) [4, 55].

Pooling essentially applies a downsampling operation to each sub-region of the feature maps [4]. Multiple types of pooling exist. Max pooling, for example, involves scanning each feature map that was produced in the previous layer by a small grid, for example a 3 x 3 pixel window. The maximum value of these nine cells would be extracted and placed in a new smaller feature map as the representative of this 3 x 3 grid. The result is thus a compressed version of the original feature map. The pooling layer simply consists of the feature maps from the convolutional layer after pooling has been applied to them. This helps reduce the dimensionality of the image which ultimately leads to more efficient image processing by the model. A CNN typically contains numerous stacks of convolutional and pooling layers that a given image is passed through prior to entering the *fully-connected* section of the model. After the final pooling layer, the pooled images are flattened (transformed into a vector) and inputted to a fully-connected, feed-forward MLP, whereby each individual pixel of every feature map is represented by an input neuron in the network [55].

After the feature maps of the image are inputted pixel by pixel into the fully connected neural network, they are then processed through one or more hidden layers. The neurons of the hidden layer contain the aforementioned ReLU activation function. After the data has been processed through the hidden layers, the output is finally inputted to the output layer, which contains a number of neurons equal to the number of classes that are being considered. Each output neuron contains the *softmax activation function* (see sec 4.4.2 for a description of this function). This activation function is very important, as it allows for the output of each of the output neurons to be a value between 0 and 1, which corresponds to the probability that the input image belongs to the class that the neuron represents. Thus, the input to a CNN is an image, and the output of a CNN is a collection of probabilities pertaining to each class. The class that is assigned the greatest probability is interpreted as a prediction that the image belongs to this class [55]. In order for the CNN to function optimally, i.e. return acceptable output in a reasonable amount of time, and also to avoid overfitting, it is necessary to normalize the input images. This is done simply by dividing each pixel intensity value by 255, which normalizes each pixel to a value between 0 and 1 prior to being passed to the network.

4.2 Overview of Transfer Learning

The present study revolves around the concept of *transfer learning*. Transfer learning involves using a CNN architecture that has already been trained on a very large dataset with many different classes as a *base model*. This base model is essentially employed as a feature extractor for the target classification problem. Specifically, the base model extracts feature maps from the input images, which are then used as inputs to the fully-connected section. Thus, transfer learning involves employing a base model with pretrained weights as a feature extractor, with the aim of *transferring* important general-purpose features learned previously by the base model to the problem at hand.

In recent years, this approach has become increasingly common in computer vision research. For a wide range of computer vision problems, it has become standard practice to pre-train a CNN on the famous ImageNet dataset, a massive dataset comprised of 1.2 million images that belong to a total of 1000 classes. After being trained on ImageNet, these models are able to adapt the features learned from this dataset to a specific target classification problem [57]. The problem of remote sensing image scene classification is no exception, as the most optimal results to date have been yielded using CNN architectures that are first pre-trained on ImageNet [4, 8 , 32, 33 45 – 47, 50]. Indeed, transfer learning using the ImageNet dataset allows for CNNs to learn excellent general-purpose features prior to being exposed to a specific remote sensing image scene classification problem, which is especially warranted when data is limited.

The present study thus employs the Xception, VGG-19, and MobileNet architectures, all of which have been pre-trained on the ImageNet dataset. These three architectures are used to extract fundamental, high-level features from the images of the UC Merced Land-Use dataset. Thus, a transfer learning approach is employed, whereby the general-purpose features previously learned by the CNNs from the ImageNet dataset are augmented and adapted to the dataset of the present study. This is the first time that these specific architectures have been applied to the land-use classification problem defined by the UC Merced Land-Use dataset.

4.3 State-of-the-Art CNN Architectures

4.3.1 Xception

The Xception CNN architecture was proposed by Francois Chollet, the creator of the famous *Keras* deep learning library for Python [61], in 2017 [58]. This architecture contains ~ 22.9 million trainable parameters [61], and is inspired by a previously developed model, *Inception*, that was proposed by Szegedy et al. in 2014 [62]. The *Keras* implementation of Xception is approximately 88 megabytes (MB) [61].

A traditional convolutional layer, with the aim of learning feature extractors in a 3D space (two spatial dimensions and one channel dimension), employs individual feature extractors to concurrently map both cross-channel and spatial correlations. The central idea behind the Inception approach is that this convolution process can be made much more efficient if it is separated into two independent processes involving cross-channel correlations and spatial correlations [58, 62].

A standard *Inception module* begins by first observing cross-channel correlations by means of a set of three or four different 1 x 1 filters that map the input data into three or four respective channel-spaces which are smaller than the initial input space. The correlations within these smaller individual 3D spaces are subsequently mapped by means of normal 3 x 3 or 5 x 5 filters. Thus, the cross-channel correlations and spatial correlations are effectively decoupled in the convolution process [58, 62]. A simplified example of this is illustrated in Figure 3 (extracted from Chollet [58]) which displays an Inception module that uses a 1 x 1 filter for cross-channel correlations and a 3 x 3 filter for spatial correlations. The Xception approach takes this Inception module one step further, and assumes that the mapping of cross-channel correlations and spatial correlations can be entirely independent. This results in an “extreme version of an Inception module” [58], also given in Figure 3 (extracted from Chollet [58]), which initially uses a 1 x 1 filter to map the cross-channel correlations, and then independently maps the spatial correlations of every individual output channel [58].

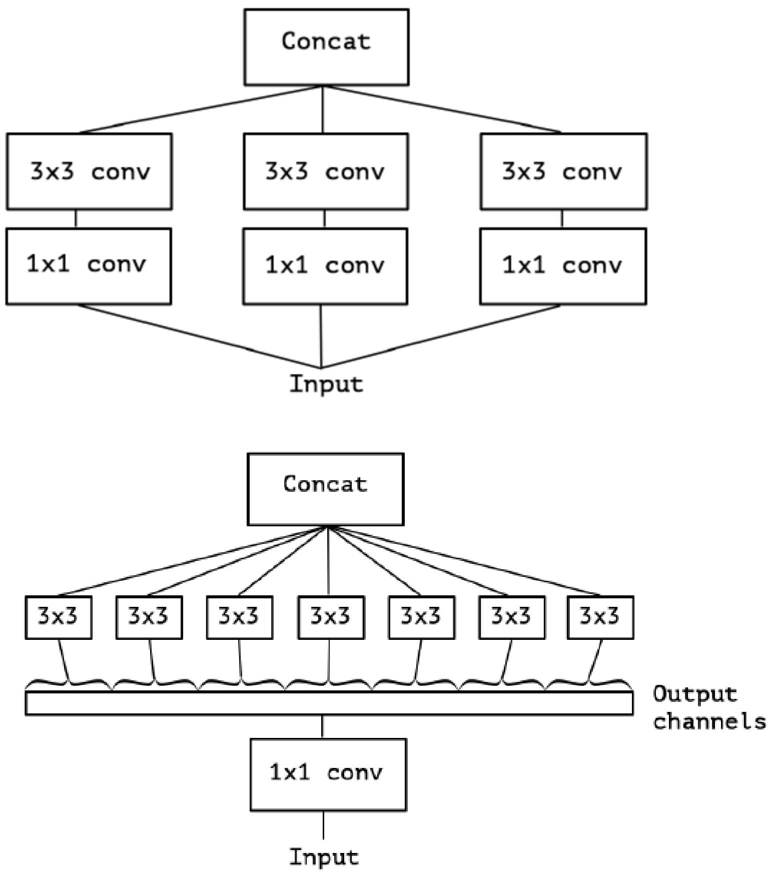


Figure 3. Simplified Inception module (top); Simplified "extreme Inception" module (bottom).

Chollet [58] maintains that this "extreme" version of an Inception module given in Figure 3 is nearly identical to the *depthwise separable convolution* operation, first proposed for use in neural networks in 2014 [63]. This operation consists of a *depthwise convolution*, which is a spatial convolution that is executed separately over every channel of an input, and a pointwise convolution, which is a 1 x 1 convolution that projects the channels output via the depthwise convolution onto a completely new channel space. Chollet [58] notes that two minor differences exist between the "extreme" Inception module denoted in Figure 3 and a true depthwise separable convolution: 1) Inception commences with the pointwise convolution which is then followed by the depthwise convolution; however, the depthwise separable convolution operation reverses this order of operations, whereby the depthwise convolution is executed first and then followed by the pointwise convolution; and 2) Inception incorporates the ReLU activation function after both depthwise and pointwise convolutions to eliminate linearity, whereas depthwise separable convolutions sometimes omit this. Chollet [58] suggests that reversing the order of operations is overall insignificant, but that the second difference may warrant investigation [58].

Chollet [58] ultimately contends that the Inception architecture can be improved by replacing all Inception modules with depthwise separable convolutions. The Xception architecture, derived from the term “extreme Inception”, is thus exclusively comprised of many stacks of depthwise separable convolutions. The fundamental idea behind this approach is that “the mapping of cross-channel correlations and spatial correlations in the feature maps of convolutional neural networks can be *entirely* decoupled” [58]. The comprehensive description of the Xception architecture is given in Figure 4, which was extracted from Chollet [58]. The feature extraction base of the network is comprised of 36 convolutional layers which are further partitioned into 14 modules. With the exception of the first and last module, every module is encompassed by linear residual connections. Furthermore, every convolutional layer and depthwise separable convolutional layer, denoted in Figure 4 by *Conv* and *SeparableConv*, respectively, is followed by batch normalization [58]. It should be noted that since the state-of-the-art architectures were only employed for feature extraction, the implementation of the present study only includes up to the “Global Average Pooling” section of the “Exit flow” [58].

The Xception architecture has boasted state-of-the-art performance on the ImageNet dataset, outperforming the highest-performing model of the Inception family, i.e. *Inception V3* [64]. Chollet [58] contends that this difference in performance cannot be attributed to increased model capacity, since Xception and Inception V3 have roughly the same number of parameters (~ 23 million) [61]. Therefore, it may be concluded that improvements to performance are ultimately the result of more efficient utilization of model parameters by Xception [58].

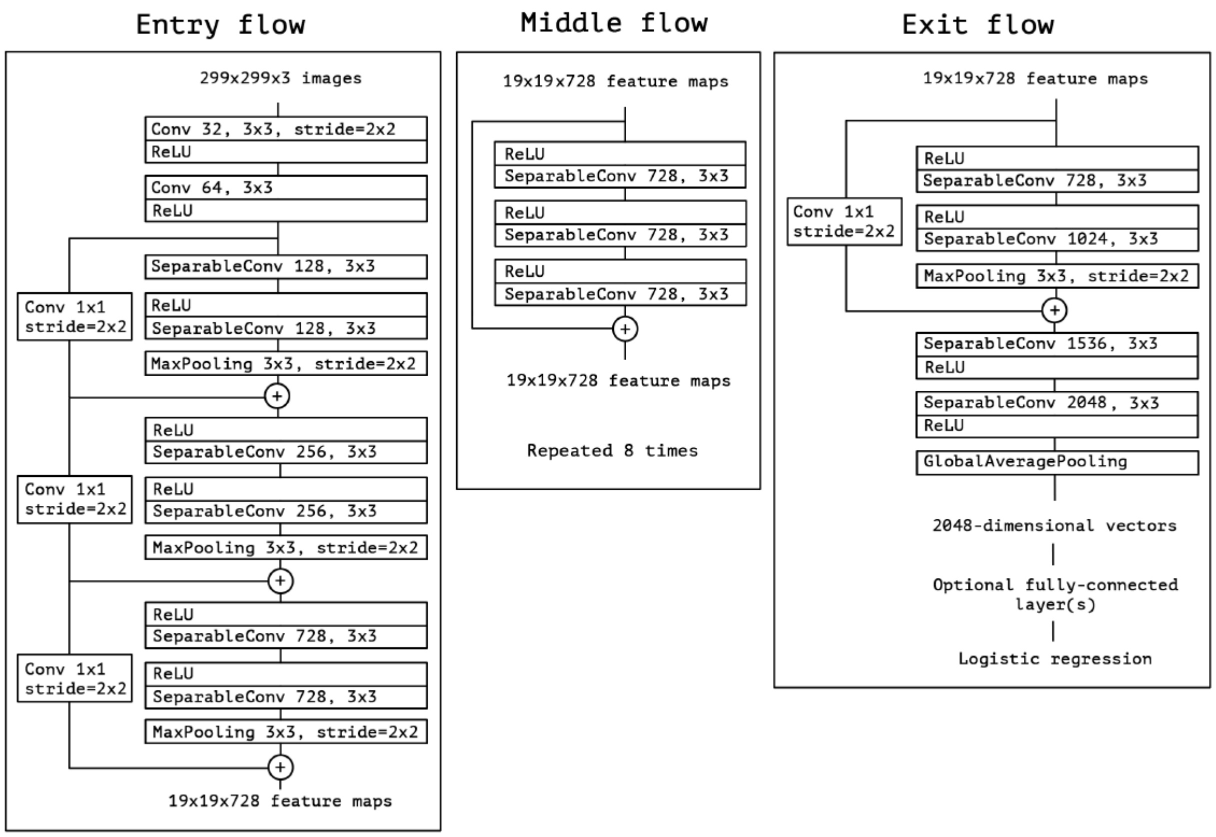


Figure 4. Xception architecture: first, the input data is passed through the entry flow; then, it is passed through the middle flow (the middle flow is repeated eight times); lastly, the data is passed through the exit flow. Note that the batch normalization which occurs after certain layers is not present in the diagram [58].

When trained on the ImageNet dataset, Xception employed the stochastic gradient descent (SGD) learning algorithm with an initial learning rate of 0.045. Furthermore, a learning decay rate of 0.94 every two epochs, and a weight decay of $1e-5$, were implemented when the model was trained on ImageNet [58].

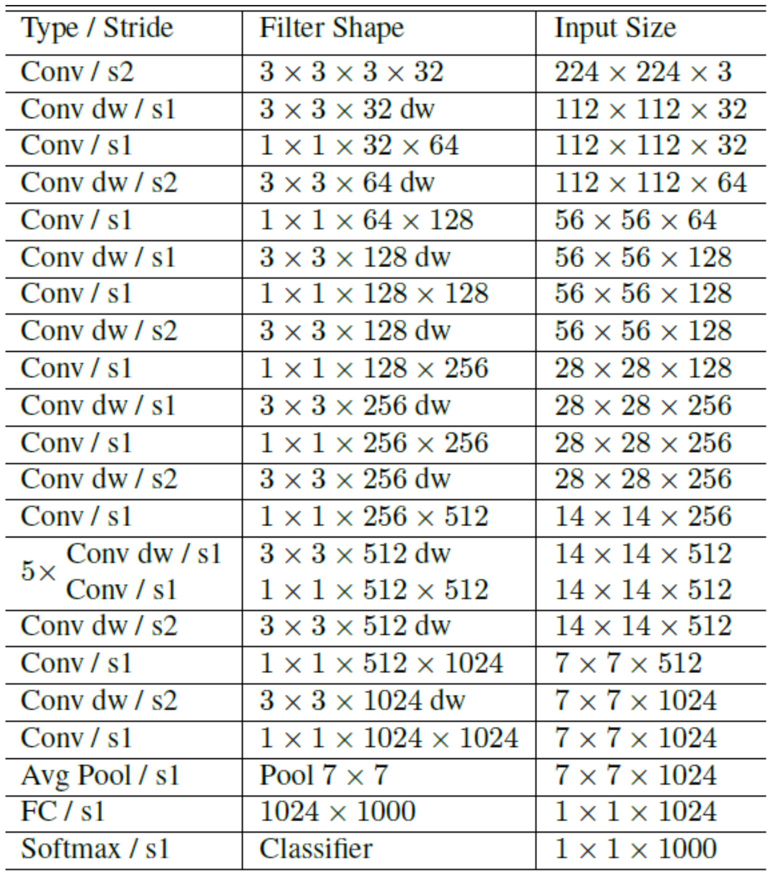
4.3.2 MobileNet

MobileNet was proposed by Howard et al. [60], a research group at Google Inc, in 2017. This architecture was developed primarily for mobile and embedded computer vision applications [60]. Much like the Xception architecture, MobileNet mainly employs the depthwise separable convolution operation, the theory behind which is outlined in Section 4.3.1. However, MobileNet has only ~ 4.2 million trainable parameters [61], significantly less than that of Xception. This architecture is streamlined and accompanied by two fundamental hyperparameters that provide a trade-off between accuracy and latency. These hyperparameters include a *width multiplier,* which essentially thins a model uniformly at each layer by reducing its number of input and output channels, and a *resolution multiplier*, which reduces the resolution of an input image [60]. However, the present study employs the baseline MobileNet architecture which does not make use of these hyperparameters. Howard et al. [60] contend that networks like MobileNet are in high demand in application areas such as autonomous vehicles, robotics, and augmented reality, whereby recognition tasks must be executed in real-time on a platform with limited computational resources [60]. The *Keras* implementation of MobileNet is only 16 MB [61], making it the most efficient model employed in terms of both time and space.

A complete description of the MobileNet architecture is provided in Table 1, extracted from Howard et al. [60], whereby *Conv* stands for "pointwise convolution" and *Conv dw* stands for "depthwise convolution". With the exception of the first layer, which is a full convolution layer, the MobileNet architecture consists of depthwise separable convolutions in every layer. All layers are followed by both batch normalization and the ReLU activation function to eliminate linearity; Table 1 does not display these operations for brevity. The down sampling operation is implemented via strided convolution in both the first layer and the depthwise convolutions that follow. Furthermore, final average pooling is incorporated prior to the fully connected layer, which reduces the spatial resolution to one. It should be noted that for the implementation of the present study, only up to the "Avg Pool / s1" [60] section is relevant. MobileNet is comprised of 28 layers overall, if pointwise and depthwise layers are considered separately [60].

When trained on the ImageNet dataset, MobileNet employed the root mean square propagation (RMSprop) learning algorithm. A detailed description of this algorithm is given in Section 4.4.3 (as this algorithm is used to train the fully-connected MLP). A different learning algorithm was therefore employed for this architecture than that which was used for Xception when training on ImageNet. In contrast to the training of large models, MobileNet also employed less data augmentation and regularization methods, since overfitting is less of a problem with smaller models. Lastly, since the depthwise filters contain very few parameters, either no or very little weight decay was implemented within these filters when this architecture was trained on ImageNet [60].

Table 1. Network Architecture of MobileNet [60]. Note: "dw" here stands for "depthwise"; "s" for "stride"; "Conv" for "point-wise convolution"; and "Conv dw" for "depthwise convolution".



Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

4.3.3 VGG-19

The VGG-19 CNN architecture was proposed by Simonyan and Zisserman in 2016 [59]. This model was therefore conceived of prior to the Xception and MobileNet architectures, outlined in Sections 4.3.1 and 4.3.2, respectively. VGG-19 is different from the aforementioned architectures namely in that it does not employ the depthwise separable convolution operation. Contrarily, Simonyan and Zisserman [59] build off of the previous works of Ciresan et al. [65] and Krizhevsky et al. [66], simply increasing the depth of the network to include a total of 19 weight layers. Simonyan and Zisserman [59] demonstrate that this simple model augmentation can translate into significant improvements in performance. The VGG-19 architecture contains substantially more trainable parameters than Xception and MobileNet, with a total of nearly 143.7 million [59]. The *Keras* implementation of VGG-19 is 549 MB [61], making it the least efficient model in the present study.

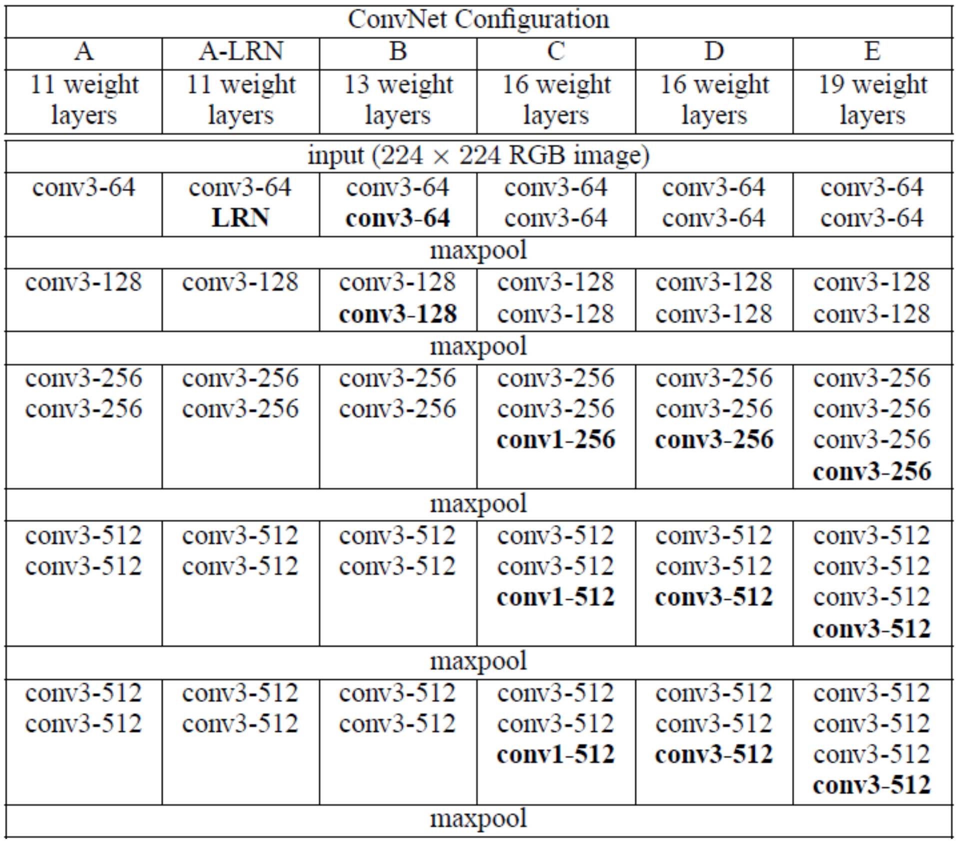
In the VGG-19 model, images are passed through a stack of convolutional layers whereby filters of the smallest possible configuration are present, i.e. 3 x 3. The convolution stride of one pixel is maintained, and the spatial pooling is executed via five separate max-pooling layers. These spatial pooling layers, which employ a stride of two and use a 2 x 2 pixel window, follow some but not all of the convolutional layers [59].

The VGG-19 architecture is presented in detail in Table 2, which was extracted from Simonyan and Zisserman [59]. It is important to note that this table in fact presents the configurations of six different architectures, because Simonyan and Zisserman [59] developed numerous *VGGNet* models which only vary in terms of their depth, i.e. the number of weight layers. Thus, only the rightmost column, *E*, is of importance to the present study, as this contains the configuration of VGG-19.

The width of the convolutional layers is quite small, beginning with 64 in the first layer which then increases by a factor of two following every max-pooling layer, until finally reaching 512 in the last convolutional layer. Simonyan and Zisserman [59] contend that their architecture is somewhat different from that which dominated the state-of-the-art prior to their implementation. For example, *VGG-19* uses very small filters (3 x 3 pixels), whereas previous authors had employed 7 x 7 or even 11 x 11 filters, except for Ciresan et al. [65] and Krizhevsky et al. [66].

When being trained on ImageNet, the VGG-19 architecture followed the implementation of Krizshevsky et al. [66]. The mini-batch gradient descent learning algorithm was employed, and the momentum parameter was set to 0.9. Furthermore, a weight decay of $5e-4$ was used, and the learning rate was initialized as 0.01. Overall, the VGG-19 architecture was very successful when tested on ImageNet, as it outperformed all other state-of-the-art models to be found in the literature at its time of publication, such as AlexNet [36] and GoogLeNet [28].

Table 2. Configuration of six different VGGNet architectures developed by Simonyan and Zisserman [59]. VGG-19 is the only of these employed in the present study, and can be found in the rightmost column, E. The parameters of the convolutional layers are signified as "conv(filter size)-(number of channels)" [59]. The fully-connected section that the authors employed has been omitted from the table, since only up to the "maxpool" section was utilized in the present study. Note that ReLU, which is present after the convolutional layers, has also been omitted for brevity.



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					

4.4 Full-Connection Model Parameters

Equipped with high-level features obtained from the state-of-the-art CNN models to use as inputs, the fully-connected section is able to make accurate classifications with respect to the 21 land-use classes. The fully-connected section is essentially a classic feed-forward MLP that consists of an input layer, a hidden layer, and an output layer. The present study thus employs the state-of-the-art CNNs to extract feature maps from the UC Merced Land-Use dataset, which are then used as inputs to the MLP. The outputs of the MLP are probability-based class predictions. Therefore, no training of CNNs was implemented in the present study; network training is entirely left to the fully-connected MLP. The specifics of the MLP, in addition to the hyperparameters that control its operation, are discussed in detail below. All hyperparameters were kept constant across the three implementations of the present study.

4.4.1 Network Architecture

The network architecture of an MLP pertains to the number of hidden layers it includes as well as the number of neurons that are present within each layer. In the present study, myriad input neurons exist; specifically, one input neuron for every pixel in every flattened feature map outputted from the last layer of the CNN is present. Thus, the number of input neurons varies depending on the CNN architecture employed for feature extraction, may it be Xception, MobileNet, or VGG-19. One sole hidden layer is employed, and the number of neurons within it is kept constant at 256 across the three models. Furthermore, the number of output neurons is naturally kept constant at 21, whereby one neuron is present for each land-use class.

4.4.2 Activation Functions

The MLP employs two different activation functions; one in the hidden layer and one in the output layer. The hidden layer employs the aforementioned ReLU activation function [31]:

$$ReLU\colon\ f(x) = \max(0, x)$$

This activation function is almost exclusively employed in the hidden layer of the fully-connected section of CNNs [28, 36, 58, 59, 60, 65, 66]. Extensive use of this function is also made in the CNN architectures employed for feature extraction in this study. In regard to the output layer, the softmax activation function was employed [55]:

$$softmax\colon\ f_j(z) = \frac{e^{z_j}}{\sum_k e^{z_k}}$$

The softmax function is fundamental as it results in the output of each of the 21 neurons being a value between 0 and 1. For each neuron, its output corresponds to the probability that the input image belongs to the class that it represents. This activation function thus allows the output of the MLP to be a collection of probabilities pertaining to each land-use class. The class that is assigned the greatest probability is interpreted as a prediction that the image belongs to that class.

4.4.3 Learning Algorithm

The learning algorithm is what minimizes the objective function and is thus the parameter that is primarily responsible for the learning capacity of the model. The present study employs the aforementioned RMSprop learning algorithm [67]:

$$RMSprop\colon\ w_{new} = w_{old} - \frac{\alpha}{\sqrt{MeanSquare(w,t)}}\nabla E(w_{old})$$

where w_{new} = the updated weight; w_{old} = the previous value of the weight; α = the learning rate = 0.001; E = output error computed by the objective function; γ = the forgetting factor = 0.9; and t = the target output. The philosophy behind the RMSprop approach is thus to keep a moving average of the magnitudes of a given weight's recent gradients in terms of mean square:

$$MeanSquare(w,t) = \gamma\ MeanSquare(w,t-1) + (1-\gamma)(\nabla E(w))^2$$

and subsequently divide the learning rate by this quantity. Thus, the learning rate is initialized at 0.001, however is quickly adapted by this moving average. This provides the MLP with a powerful and dynamic learning capacity.

4.4.4 Objective Function

The objective function (or loss function) measures the performance of the classification model. The present study employed the cross-entropy objective function, which is commonly used in CNNs [31]:

$$cross - entropy\colon\ -\sum_{c=1}^{M} y_{o,c}\log(p_{o,c})$$

where *M* = the number of classes; *y* = a binary indicator if the class label (*c*) has been assigned correctly for the given observation (*o*); and *p* = probability prediction that observation *o* is a member of class *c*.

4.4.5 Dropout

It is integral that the fully-connected section of any CNN used for transfer learning implements the *dropout* parameter. This involves randomly setting a fraction of input units to zero at each update when the model is being trained [61], which has been found to be a very simple solution to the overfitting problem of neural networks [69]. The present study employed the frequently used dropout rate of 0.5, i.e. 50% of input units were randomly set to zero at each update during the training phase.

4.4.6 Train/test split

MLPs are trained by running a large number of observations through the network which frequently comprises the majority of the total dataset, selected at random. This is referred to as the training phase whereby the model learns from the data and iteratively tunes its weights so that it can perform adequately in the testing phase, where there is no target output and thus no error to learn from. Data therefore must be randomly split into two datasets, one for training and one for testing. In the present study, a train/test split of 80/20 was employed, thus 80% of the input images were used for training and the remaining 20% were used for testing.

4.4.7 Training Parameters

When training the MLPs, the maximum number of epochs was set to 100. The number of epochs equates to the number of times the total training dataset passes forward and backward through the model. Thus, one full pass forward and backward = one epoch. Too few epochs results in an MLP that is insufficiently trained to model a given system; however, too many epochs can result in overfitting. During training, the "EarlyStopping" function of *Keras* [61] was utilized, which monitored the model's performance; if the accuracy of the MLP did not improve after six consecutive epochs, the model was considered to have converged, and training thus terminated. The batch size, which is the number of observations that pass through the model before the weights are updated during each epoch, was kept constant at 16.

4.5 Evaluation Protocol

Three MLPs were evaluated, i.e. one pertaining to each of the state-of-the-art CNN architectures employed for feature extraction. The overall accuracy and confusion matrix were employed as performance criteria to evaluate the performance of each of the models. The overall accuracy is defined as the number of correctly classified test images divided by the total number of test images. This metric ranges between 0 and 1, and may also be computed by taking the trace of a confusion matrix and dividing this by the total number of classes. The confusion matrix provides a concise visual representation of a model's performance with respect to specific classes, and may be employed to analyse confusion between different classes and overall model performance. This matrix contains rows which represent instances of the actual classes and columns which represent instances of the predicted classes. Each element x_{ij} in a confusion matrix thus pertains to the percentage of images that in fact belong to class *i* however are predicted by a model to belong to class *j*. For example, the diagonal elements (x_{ii}) of a model that boasts perfect accuracy would all be 100%, and all non-diagonal elements would resultantly be 0% [4].

Each of the three models were independently trained and tested a total of five times. Model performance varied only slightly (+/– 2%) across the five iterations executed for each model. For brevity, only the most optimal overall accuracy (and corresponding confusion matrix) produced by each model is provided in the section that follows, whereby the overall accuracy of each model is compared to that of other state-of-the-art approaches.

5. Results and Discussion

The overall accuracy of each model is given in Table 3, along with that of other state-of-the-art approaches that have been applied to the UC Merced Land-Use dataset. In addition, the number of trainable parameters for each model is provided, as well as a yes/no indicator denoting if data augmentation was used, and the total size of the dataset as a result of this augmentation. The CNN architectures employed in the present study are provided together in the last three rows of Table 3 for clarity. It should be noted that Table 3 does not provide an exhaustive list of all architectures that have been applied to this problem to date, but just some approaches that have been taken (including the most optimal model yet employed). Szegedy et al. [28] employed the GoogLeNet architecture which achieved an overall accuracy of 94.31%; Xia et al. [68] employed the VGG-16 architecture which achieved an overall accuracy of 95.21%; Castelluccio et al. [44] employed the GoogLeNet architecture and fine-tuned it to achieve an overall accuracy of 97.10%; and lastly, Zhang et al. [4] employed a hybrid architecture coined the *Inception-v3-CapsNet*, which boasts the best overall accuracy that has been achieved on this dataset to date, i.e. 99.05%. The three models employed in present study, i.e. Xception, VGG-19, and MobileNet, achieved overall accuracies of 72.38%, 83.57%, and 92.62%, respectively.

Table 3. Overall accuracy of the proposed methods compared to that of state-of-the-art methods.

State-of-the-art method	**Overall accuracy (%)**	**# Trainable parameters (million)**	**Data augmentation / total # images**
GoogLeNet [68]	94.31	~ 11.2	Yes / not specified
VGG-16 [68]	95.21	~ 138.4	Yes / not specified
Fine-tuned GoogLeNet [44]	97.10	~ 11.2	Yes / not specified
Inception-v3-CapsNet [4]	99.05	~ 25.0	No / 2100
Xception (present study)	72.38	~ 22.9	No / 2100
VGG-19 (present study)	83.57	~ 143.7	No / 2100
MobileNet (present study)	92.62	~ 4.2	No / 2100

Illustrated in Table 3, the architectures of the present study did not outperform other state-of-the-art approaches that have been applied to this problem. Xception and VGG-19, in particular, boasted significantly worse performance. Nevertheless, the performance of the MobileNet architecture is close to that of other methods. Furthermore, MobileNet's flexible and efficient architecture may deem it highly applicable to a wide range of practical remote sensing image scene classification problems, particularly involving embedded or mobile vision systems [60]. A table summarizing the *precision* and *recall* metrics pertaining to each class for the three models is provided in Appendix A. A discussion of the performance of each architecture is provided in the subsections that follow, including confusion matrices for each approach.

5.1 Xception Results

The Xception architecture performed least optimally among the approaches employed, with an overall accuracy of 72.38%. This is unexpected, considering that an architecture quite similar to this, i.e. Inception-v3-CapsNet, was reported by Zhang et al. [4] to perform very well (99.05%). Nevertheless, instead of using a traditional fully-connected MLP to follow the CNN architecture, Zhang et al. [4] use a *capsule network*, which employs a collection of neurons as a capsule that has the power to encode feature characteristics including spatial information [4]. This may partially explain why the Xception approach, which is very similar to the original Inception [62] model, produced substantially less impressive results. Table 4 conveys a confusion matrix summarizing the performance of Xception with respect to the specific land-use classes of the dataset.

Table 4. Confusion matrix for the Xception architecture on the UC Merced Land-Use dataset. Each element x_{ij} pertains to the percentage of images that are of class i but are predicted to be of class j. Note: the different land-use classes are fully listed and numbered 1 – 21 for the actual classes in the rows; for brevity, just the class numbers are listed in the columns which represent the class predictions. "0%" elements have been omitted.

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**
1. agricultural	**100**																				
2. airplane		**95**																			5
3. baseballdiamond		5	**0**							30									35	25	5
4. beach				**100**																	
5. buildings		10			**70**		10													10	
6. chaparral						**100**															
7. denseresidential					10		**70**					5					5		5		5
8. forest	5					20		**0**									75				
9. freeway									**100**												
10. golfcourse										**85**							10		5		
11. harbor											**95**									5	
12. intersection					5							**80**			5				5		5
13. mediumresidential					10		35					10	**0**						20		25
14. mobilehomepark		5			5		80					5		**0**			5				
15. overpass									15			5			**80**						
16. parkinglot																**100**					
17. river	10	5															**85**				
18. runway																	5	**95**			
19. sparesresidential					5														**95**		
20. storagetanks					5							5							5	**85**	
21. tenniscourt							5					5							5		**85**

Xception produces reasonably accurate results for most classes. The model boasts perfect accuracy (100%) for the *agricultural*, *beach*, *chaparral*, *freeway*, and *parkinglot* classes. Furthermore, the *airplane*, *harbor*, *runway*, and *sparseresidential* class predictions boast near-optimal accuracy (95%). However, for the *baseballdiamond*, *forest*, *mediumresidential*, and *mobilehomepark* classes, Xception yields entirely unacceptable results, as it makes no correct classifications (0% accuracy) for these classes. These four anomalous classes indeed explain the mediocre performance of Xception overall, as they significantly lower its overall accuracy to 72.38%. For the remaining classes, the model accuracy is reasonable (70% – 85%).

Overall, for the classes for which Xception cannot yield any correct classifications, the confusion tends to be spread across numerous classes, i.e. the confusion does not specifically pertain to just two or three individual classes. However, the model frequently confused the *forest* and *river* classes, i.e. 75% of images labelled as *forest* were falsely classified as *river*. In addition, *mobilehomepark* and *denseresidential* were frequently confused, i.e. 80% of images labelled as *mobilehomepark* were falsely classified as *denseresidential*.

5.2 VGG-19 Results

The VGG-19 architecture produces decent results with an overall accuracy of 83.57%. However, like Xception, VGG-19 was expected to perform better overall than it did. This is namely because previous authors, specifically Xia et al. [68], implemented an almost identical architecture, *VGG-16*, reporting an overall accuracy of 95.21%. The only difference between VGG-19 and VGG-16 is that the former contains three more layers than the latter, hence the naming convention. Indeed, it is surprising that the performance of VGG-19 is significantly less optimal than VGG-19, as a difference in overall accuracy of over 11% can be observed. Table 5 conveys a confusion matrix that summarizes the performance of VGG-19 on the land-use classes comprising the dataset.

Table 5 - Confusion matrix for the VGG-19 architecture on the UC Merced Land-Use dataset. Each element x_{ij} pertains to the percentage of images that are of class i but are predicted to be of class j. Note: the different land-use classes are fully listed and numbered 1 – 21 for the actual classes in the rows; for brevity, just the class numbers are listed in the columns which represent the class predictions. "0%" elements have been omitted.

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**
1. agricultural	**100**																				
2. airplane		**95**																		5	
3. baseballdiamond			**75**							25											
4. beach				**90**													10				
5. buildings			5		**80**										10						5
6. chaparral						**90**		10													
7. denseresidential					5		**50**						15	20					5		5
8. forest								**95**									5				
9. freeway									**95**	5											
10. golfcourse										**70**							30				
11. harbor											**100**										
12. intersection					5							**85**			5		5				
13. mediumresidential					5		30						**45**	5					15		
14. mobilehomepark					5		25							**70**							
15. overpass															**100**						
16. parkinglot								5								**95**					
17. river										10							**85**		5		
18. runway	10																	**90**			
19. sparesresidential																			**100**		
20. storagetanks			5		5														5	**85**	
21. tenniscourt			5		5		5			5									20		**60**

Overall, the VGG-19 architecture produces satisfactory results for the majority of classes. The model boasts perfect accuracy (100%) for the *agricultural*, *harbor*, *overpass*, and *sparseresidential* classes (interestingly, none of these are classes for which Xception produced perfect accuracy, save for *agricultural*). Furthermore, the *airplane*, *forest*, *freeway*, *runway*, and *parkinglot* class predictions boast near-optimal accuracy (95%), along with *beach*, *chaparral*, and *runway* (90%). Unlike Xception, there are no classes for which VGG-19 made zero correct class predictions. The classes with the three lowest accuracies are *mediumresidential* (45%), *denseresidential* (50%), and *tenniscourt* (60%). For these three classes, no explicit class-specific confusion can be observed, i.e. the confusion is spread across numerous classes. These anomalies indeed bring down the overall accuracy of VGG-19. Classes other than the aforementioned classes were predicted with reasonable accuracy.

5.3 MobileNet Results

The MobileNet architecture performed the most optimal overall, boasting an overall accuracy of 92.62%. Among the investigated methods, only the performance of this architecture is close to that of the state-of-the-art approaches given in Table 3. These results are very impressive given the size of MobileNet, as the model contains a mere ~ 4.2 million trainable parameters and takes up only 16 MB of memory [61]. Xception has over 18 million more parameters than MobileNet, and VGG-19 has over 139 million more parameters than MobileNet [61]. This underscores the true power of this architecture, as it produces substantially more optimal results than its counterparts, albeit with less computational units, space, and overall processing power. Indeed, MobileNet is more efficient than both Xception and VGG-19, as well as the other state-of-the-art methods given in Table 3. The architecture performing the best on this dataset to date, i.e. the Inception-v3-CapsNet, boasts an overall accuracy that is ~ 6% greater than that of MobileNet; however, this model also contains nearly 20 million more trainable parameters than MobileNet [4, 61], making it remarkably less efficient. For applications involving mobile and embedded vision, Inception-v3-CapsNet may be too inefficient and computationally expensive for practical implementation. For such applications, MobileNet is indeed the recommended method to employ.

The high performance of MobileNet is not entirely unprecedented, given that a variation of this architecture has achieved exceptional results on the somewhat related task of *Large Scale*

Geolocalizaton [60], whereby a model must classify certain geographic locations. Nevertheless, this implementation was on a significantly larger scale, whereby the model contained millions of more parameters than the present implementation, and was trained on millions of images. Table 6 displays a confusion matrix summarizing the impressive performance of MobileNet with respect to the specific classes of the dataset.

Table 6. Confusion matrix for the MobileNet architecture on the UC Merced Land-Use dataset. Each element x_{ij} pertains to the percentage of images that are of class i but are predicted to be of class j. Note: the different land-use classes are fully listed and numbered 1 – 21 for the actual classes in the rows; for brevity, just the class numbers are listed in the columns which represent the class predictions. "0%" elements have been omitted.

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**
1. agricultural	**100**																				
2. airplane		**100**																			
3. baseballdiamond			**100**																		
4. beach				**100**																	
5. buildings					**90**																
6. chaparral						**95**		5													
7. denseresidential		5			5		**50**						20	10	5				5		
8. forest								**100**													
9. freeway									**80**						5			15			
10. golfcourse			5							**95**											
11. harbor											**100**										
12. intersection												**95**			5						
13. mediumresidential							10						**90**								
14. mobilehomepark													5	**95**							
15. overpass															**100**						
16. parkinglot																**100**					
17. river				15						10	5						**65**	5			
18. runway																		**100**			
19. sparesresidential										5			5						**90**		
20. storagetanks																				**100**	
21. tenniscourt																					**100**

Overall, the MobileNet architecture produces very impressive results for the vast majority of classes. The model boasts perfect accuracy (100%) for over half of the classes in the dataset, including the *agricultural*, *airplane*, *baseballdiamond*, *beach*, *forest*, *harbor*, *overpass*, *parkinglot*, *runway*, *storagetanks* and *tenniscourt* classes. Furthermore, the *chaparral*, *golfcourse*, *intersection*, and *mobilehomepark* class predictions boast near-optimal accuracy (95%), along with *buildings*, *mediumresidential*, and *sparseresidential* (90%). The model performed least optimally on the *denseresidential* (50%), *river* (65%), and *freeway* (80%)

classes. Indeed, MobileNet boasts accuracy that is ≥ 90% for all classes other than these three classes, which are responsible for reducing the overall accuracy of this model to slightly below that of other famed state-of-the-art approaches.

5.4 Further Analysis

In Section 5.3, it was determined that MobileNet far outperforms the other two approaches employed, i.e. Xception and VGG-19. Furthermore, Sections 5.1 and 5.2 demonstrated that the performance of the Xception and VGG-19 implementations were significantly less optimal than anticipated. Indeed, previous implementations of architectures that are quite similar to Xception and VGG-19, such as Inception-v3-CapsNet and VGG-16, have been reported to boast significantly better performance than that of the present implementations. This may be attributed to a number of factors. Firstly, the present study did not employ *data augmentation*, a technique that is frequently used for deep learning approaches to remote sensing image scene classification, especially when the relevant dataset is small. Data augmentation provides plentiful training and testing data by executing certain transformations on the raw images randomly, such as shearing, shifting, zooming, and flipping. This results in a more abundant and dynamic dataset overall, which prevents the model from overfitting and improves its ability to generalize [26, 27]. The present study did not employ such an approach, which provides an explanation as to why MobileNet, a drastically smaller architecture, vastly outperformed both Xception and VGG-19.

Smaller architectures, such as MobileNet, are much less susceptible to overfitting than larger architectures such as Xception and VGG-19 [61]. Due to having less trainable parameters, MobileNet is less likely to simply memorize, rather than actually learn, from the input data. Larger architectures generally require more input data than smaller architectures, and the UC Merced Land-Use dataset is considerably small, containing only 100 images per class. Therefore, MobileNet outperformed Xception and VGG-19 likely because of its smaller and more efficient architecture which is much more suitable to the small dataset employed. Moreover, if data augmentation were to be used, the performance of the larger Xception and VGG-19 architectures would likely improve significantly. Nevertheless, data augmentation is a computationally expensive operation, and in the present study it was purposely omitted. Working with smaller datasets is an import research area of computer vision, especially when considering embedded and mobile vision applications whereby computational resources are limited [61].

Secondly, a popular approach to transfer learning, i.e. *fine-tuning*, was also omitted from the present study. Opposed to just employing the CNN architectures to extract features to use as inputs to an MLP, as was done in the present study, fine-tuning involves actually using the CNN itself during the training phase. Specifically, this approach involves “freezing” all layers of a CNN architecture, except for the final “block” or last few convolutional layers, which are trained concurrently with the fully-connected section during the training phase. Thus, this denotes a holistic approach, whereby a small part of the CNN is trained alongside the fully-connected MLP. This approach to transfer learning has yielded state-of-the-art results in remote sensing image scene classification [3, 4, 8, 25, 32 – 34]. Nevertheless, fine-tuning is typically dependent on data augmentation, which was of course omitted. Overall, the present study indeed traded off some optimality (in terms of accuracy) for more efficient modelling. This resulted in the two larger architectures of Xception and VGG-19 performing suboptimally; however, this trade-off proved to be worthwhile for the MobileNet architecture overall.

Across the three architectures investigated, certain trends can be observed in terms of class-specific accuracy. For example, all three models scored perfect accuracy (100%) on the *agricultural* class. It may therefore be concluded that this is the easiest class for a given model to predict. Furthermore, it is evident that models tend to struggle at differentiating between the three residential area classes, i.e. *mediumresidential*, *sparseresidential*, and *denseresidential*. These three classes are very similar in that they only really differ in terms of the concentration of housing present in the images. Thus, these classes are considered to be “overlapped”; previous researchers have also observed this challenging aspect of the present dataset [3, 4, 8, 25]. Zhang et al. [4] developed an elegant solution to this problem by means of the aforementioned Inception-v3-CapsNet architecture, which employs a *CapsNet* that encodes the spatial relationships of features more effectively than a traditional fully-connected section. This approach is currently the top state-of-the-art architecture for solving the land-use classification problem defined by the UC Merced Land-Use dataset.

6. Conclusions

The advent of deep learning and CNNs has dramatically improved the state-of-the-art for remote sensing image scene classification in recent years. Using transfer learning, the present study applied three state-of-the-art CNN architectures to the land-use classification problem defined by the UC Merced Land-Use dataset, which contains 21 land-use classes. The overall accuracy was employed as the primary performance metric, provided alongside confusion matrices for each approach; the precision and recall metrics were appendicized. A brief literature review, pertaining to the history of remote sensing image scene classification up to the present state-of-the-art methods, was also presented. The present study used state-of-the-art models to extract feature maps from the dataset, which were then used as inputs to a fully-connected model, i.e. an MLP algorithm. The architectures include Xception, VGG-19, and MobileNet, which boast overall accuracies of 72.38%, 83.57%, and 92.62%, respectively.

The MobileNet architecture significantly outperformed the other two methods, which is very impressive due to its comparatively small size (only 16 MB and includes ~ 4.2 million trainable parameters), as this model is intended for mobile and embedded vision applications. The Xception and VGG-19 models did not perform as optimally as anticipated, since similar architectures implemented by previous authors boast impressive results. This can namely be attributed to the omission of data augmentation and fine-tuning, techniques which frequently accompany models of such great magnitude. The present study took a more efficiency-oriented approach, observing the power of the MobileNet architecture when data is limited.

The MobileNet architecture predicted over half of the classes (11 classes) in the dataset with perfect accuracy (100%), whereas VGG-19 and Xception predicted four and five of the classes with perfect accuracy, respectively. Furthermore, all three architectures predicted the *agricultural* class with perfect accuracy. The models struggled most with differentiating between the three residential land-use classes, i.e. *mediumresidential*, *sparseresidential*, and *denseresidential*, which typically cause confusion due to containing the same features with different levels of concentration. This issue has also been reported by previous authors, and pertains to the limited capacity of CNNs to encode spatial information. For other classes, class-specific confusion is scarcely observed, i.e. the confusion is typically spread across numerous classes, opposed to two or three classes being explicitly mistaken for each other.

Considering the results of the present study, MobileNet is naturally the suggested architecture to employ when solving this specific remote sensing image scene classification problem. Among the investigated methods, only MobileNet boasts an overall accuracy that is near that of other state-of-the-art methods. In comparison to these methods, MobileNet is however substantially more efficient in terms of both time and space, making it ideal for applications in which computational resources are limited. Future suggested work is to further investigate the MobileNet architecture, as well as a second version of this model called *MobileNetV2* [61], which is even smaller than the original MobileNet model. Furthermore, data augmentation and fine-tuning may further improve the already impressive performance of MobileNet, and dramatically improve that of Xception and VGG-19. These techniques should also be investigated in the future for the present architectures.

References

1. Hubert, M.J.; Carole, E. Airborne SAR-efficient signal processing for very high resolution. Proc. IEEE 2013, 101, 784–797.
2. Plaza, A.; Plaza, J.; Paz, A.; Sanchez, S. Parallel hyperspectral image and signal processing. IEEE Signal Process. Mag. 2011, 28, 119–126.
3. Cheng, G.; Han, J.; Lu, X. Remote sensing image scene classification: Benchmark and state of the art. Proc. IEEE 2017, 105, 1865–1883.
4. Zhang, W.; Tang, P.; Zhao, L. Remote Sensing Image Scene Classification Using CNN-CapsNet. Remote Sens. 2019, *11*, 494.
5. Cheriyadat, A.M. Unsupervised feature learning for aerial scene classification. IEEE Trans. Geosci. Remote Sens. 2014, 52, 439–451.
6. Shao,W.; Yang,W.; Xia, G.S. Extreme value theory-based calibration for the fusion of multiple features in high-resolution satellite scene classification. Int. J. Remote Sens. 2013, 34, 8588–8602.
7. Estoque, R.C.; Murayama, Y.; Akiyama, C.M. Pixel-based and object-based classifications using high- and medium-spatial-resolution imageries in the urban and suburban landscapes. Geocarto Int. 2015, 30, 1113–1129.
8. Zhang, X.;Wang, Q.; Chen, G.; Dai, F.; Zhu, K.; Gong, Y.; Xie, Y. An object-based supervised classification framework for very-high-resolution remote sensing images using convolutional neural networks. Remote Sens. Lett. 2018, 9, 373–382.
9. L. Janssen and H. Middelkoop. Knowledge-based crop classification of a Landsat Thematic Mapper image. Int. J. Remote Sens. 1992, 13, 2827–2837.
10. T. Blaschke and J. Strobl. What's wrong with pixels? Some recent developments interfacing remote sensing and GIS. GeoBIT/GIS. 2001, 6, 12–17.
11. H. Li; H. Gu; Y. Han; and J. Yang. Object-oriented classification of high-resolution remote sensing imagery based on an improved colour structure code and a support vector machine. Int. J. Remote Sens. 2010, 31, 1453–1470.
12. G. Cheng; P. Zhou; J. Han; L. Guo; and J. Han. Auto-encoder-based shared mid-level visual dictionary learning for scene classification using very high resolution remote sensing images. IET Computer Vision. 2015, 9, 639–647.
13. Yang, Y.; Newsam, S. Comparing SIFT descriptors and Gabor texture features for classification of remote sensed imagery. In Proceedings of the 15th IEEE International Conference on Image Processing (ICIP), San Diego, CA, USA, 12–15 October 2008; 1852–1855.
14. G. Cheng, J. Han, P. Zhou, and L. Guo. Multi-class geospatial object tdetection and geographic image classification based on collection of part detectors. ISPRS J. Photogramm. Remote Sens. vol. 98, 119-132, 2014.
15. H. Sridharan and A. Cheriyadat. Bag of lines (bol) for improved aerial scene representation. IEEE Geosci. Remote Sens. Lett., vol. 12, no. 3, 676-680, 2015.
16. D. G. Lowe. Distinctive image features from scale-invariant keypoints. Int. J. Comput. Vis. vol. 60, no. 2, 91-110, 2004.

17. R. M. Haralick, K. Shanmugam, and I. h. Dinstein. Textural features for image classification. IEEE Trans. Syst. Man Cybern., vol. 3, 610-621, 1973.
18. M. J. Swain and D. H. Ballard. Color indexing. Int. J. Comput. Vis., vol. 7, no. 1, pp. 11-32, 1991.
19. A. K. Jain, N. K. Ratha, and S. Lakshmanan, “Object detection using Gabor filters,” *Pattern Recog.,* vol. 30, no. 2, pp. 295-309, 1997.
20. R. M. Haralick, K. Shanmugam, and I. h. Dinstein, “Textural features for image classification,” *IEEE Trans. Syst. Man Cybern.*, vol. 3, pp. 610-621, 1973.
21. G. E. Hinton and R. R. Salakhutdinov, Reducing the dimensionality of data with neural networks. *Science*, vol. 313, no. 5786, pp. 504-507, 2006.
22. F. Zhang, B. Du, and L. Zhang, “Saliency-guided unsupervised feature learning for scene classification,” *IEEE Trans. Geosci. Remote Sens.*, vol. 53, no. 4, pp. 2175-2184, 2015.
23. X. Yao, J. Han, G. Cheng, X. Qian, and L. Guo, “Semantic Annotation of High-Resolution Satellite Images via Weakly Supervised Learning,” *IEEE Trans. Geosci. Remote Sens.*, vol. 54, no. 6, pp. 3660-3671, 2016.
24. Q. Zou, L. Ni, T. Zhang, and Q. Wang, “Deep learning based feature selection for remote sensing scene classification,” *IEEE Geosci. Remote Sens. Lett.,* vol. 12, no. 11, pp. 2321-2325, 2015.
25. G. Cheng, P. Zhou, and J. Han, “Learning Rotation-Invariant Convolutional Neural Networks for Object Detection in VHR Optical Remote Sensing Images,” IEEE Trans. Geosci. Remote Sens., vol. 54, no. 12, pp. 7405-7415, 2016.
26. A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. Conf. Adv. Neural Inform. Process. Syst.*, 2012, pp. 1097-1105.
27. K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. Int. Conf. Learn. Represent.*, 2015, pp. 1-13.
28. C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proc. IEEE Int. Conf. Comput. Vision Pattern Recognit.*, 2015, pp. 1-9.
29. K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Int. Conf. Comput. Vision Pattern Recognit.*, 2016, pp. 770-778.
30. Zoph, B., Vasudevan, V., Shlens, J., Le, Q.V.: Learning transferable architectures for scalable image recognition. In: CVPR. (2018).
31. Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature,* vol. 521, no. 7553, pp. 436-444, 2015.
32. Y. Zhong, F. Fei, and L. Zhang, “Large patch convolutional neural networks for the scene classification of high spatial resolution imagery,” J. Appl. Remote Sens., vol. 10, no. 2, pp. 025006-025006, 2016.
33. D. Marmanis, M. Datcu, T. Esch, and U. Stilla, “Deep Learning Earth Observation Classification Using ImageNet Pretrained Networks,” IEEE Geosci. Remote Sens. Lett., vol. 13, no. 1, pp. 105-109, 2016.

34. M. Längkvist, A. Kiselev, M. Alirezaie, and A. Loutfi, “Classification and Segmentation of Satellite Orthoimagery Using Convolutional Neural Networks,” Remote Sensing, vol. 8, no. 4, pp. 329, 2016.
35. G. Sheng, W. Yang, T. Xu, and H. Sun, “High-resolution satellite scene classification using a sparse coding based multiple feature combination,” *Int. J. Remote Sens.*, vol. 33, no. 8, pp. 2395-2412, 2012.
36. B. Zhao, Y. Zhong, G.-S. Xia, and L. Zhang, “Dirichlet-derived multiple topic scene classification model for high spatial resolution remote sensing imagery,” IEEE Trans. Geosci. Remote Sens., vol. 54, no. 4, pp. 2108-2123, 2016.
37. Q. Zou, L. Ni, T. Zhang, and Q. Wang, “Deep learning based feature selection for remote sensing scene classification,” IEEE Geosci. Remote Sens. Lett., vol. 12, no. 11, pp. 2321-2325, 2015.
38. L. Zhao, P. Tang, and L. Huo, “Feature significance-based multibag-of-visual-words model for remote sensing image scene classification,” *J. Appl. Remote Sens.*, vol. 10, no. 3, pp. 035004-035004, 2016.
39. O. A. Penatti, K. Nogueira, and J. A. dos Santos, "Do deep features generalize from everyday objects to remote sensing and aerial scenes domains?," in Proc. IEEE Int. Conf. Comput. Vision Pattern Recognit. Workshops, 2015, pp. 44-51.
40. Y. Yang and S. Newsam, "Bag-of-visual-words and spatial extensions for land-use classification," in *Proc. ACM SIGSPATIAL Int. Conf. Adv. Geogr. Inform. Syst.*, 2010, pp. 270-279.
41. Zhu, Q.; Zhong, Y.; Zhao, B.; Xia, G.; Zhang, L. Bag-of-visual-words scene classifierwith local and global features for high spatial resolution remote sensing imagery. IEEE Geosci. Remote Sens. Lett. 2016, 13, 747–751.
42. Zhang, F.; Du, B.; Zhang, L. Scene classification via a gradient boosting random convolutional network framework. IEEE Trans. Geosci. Remote Sens. 2016, 54, 1793–1802.
43. Liu, Y.; Zhong, Y.; Fei, F.; Zhu, Q.; Qin, Q. Scene Classification Based on a Deep Random-Scale Stretched Convolutional Neural Network. Remote Sens. 2018, 10, 444.
44. Castelluccio, M.; Poggi, G.; Sansone, C.; Verdoliva, L. Land use classification in remote sensing images by convolutional neural networks. arXiv 2015, arXiv:1508.00092.
45. Hu, F.; Xia, G.S.; Hu, J.; Zhang, L. Transferring deep convolutional neural networks for the scene classification of high-resolution remote sensing imagery. Remote Sens. 2015, 7, 14680–14707.
46. Chaib, S.; Liu, H.; Gu, Y.; Yao, H. Deep feature fusion for VHR remote sensing scene classification. IEEE Trans. Geosci. Remote Sens. 2017, 55, 4775–4784.
47. Gong, X.; Xie, Z.; Liu, Y.; Shi, X.; Zheng, Z. Deep salient feature based anti-noise transfer network for scene classification of remote sensing imagery. Remote Sens. 2018, 10, 410.
48. Chen, G.; Zhang, X.; Tan, X.; Cheng, Y.F.; Dai, F.; Zhu, K.; Gong, Y.;Wang, Q. Training small networks for scene classification of remote sensing images via knowledge distillation. Remote Sens. 2018, 10, 719.

49. Zeng, D.; Chen, S.; Chen, B.; Li, S. Improving remote sensing scene classification by integrating global-context and local-object features. Remote Sens. 2018, 10, 734.
50. Chen, J.;Wang, C.;Ma, Z.; Chen, J.;He,D.; Ackland, S. Remote sensing scene classification based on convolutional neural networks pre-trained using attention-guided sparse filters. Remote Sens. 2018, 10, 290.
51. Zou, J.; Li, W.; Chen, C.; Du, Q. Scene classification using local and global features with collaborative representation fusion. Inf. Sci. 2018, 348, 209–226.
52. Kumar, M., Raghuwanshi, N. S., Singh, R. (2011). Artificial neural networks approach in evapotranspiration modeling: A review. Irrigation Science, 29(1), 11–25.
53. Baldassi, C., Borgs, C., Chayes, J., Ingrosso, A., Lucibello, C., Saglietti, L., & Zecchina, R. (2016). Unreasonable Effectiveness of Learning Neural Networks: From Accessible States and Robust Ensembles to Basic Algorithmic Schemes. PNAS.
54. Tufféry, S. (2007) Data mining et statistique décisionnelle. L'intelligence de données, Modèles linéaire, Régression logistique, Réseaux de neurones, Scoring et Text mining. Edition TECHNIP. Paris.
55. Wu, Jianxin (2017). Introduction to Convolutional Neural Networks. National Key Lab for Novel Software Technology, Nanjing University, China.
56. O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. International Journal of Computer Vision (IJCV), 2015.
57. Minyoung Huh, Pulkit Agrawal, and Alexei A Efros. 2016. What makes ImageNet good for transfer learning? arXiv preprint arXiv:1608.08614.
58. F. Chollet. Xception: Deep learning with depthwise separable convolutions. In CVPR, 2017.
59. K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In ICLR, 2015.
60. A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv:1704.04861, 2017.
61. Francois Chollet et al. (2015). Keras documentation. Retrieved from: https://keras.io/
62. C. Szegedy,W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.
63. L. Sifre. Rigid-motion scattering for image classification, 2014. Ph.D. thesis.
64. C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. *arXiv preprint arXiv:1512.00567*, 2015.
65. Ciresan, D. C., Meier, U., Masci, J., Gambardella, L. M., and Schmidhuber, J. Flexible, high performance convolutional neural networks for image classification. In *IJCAI*, pp. 1237–1242, 2011.
66. Krizhevsky, A., Sutskever, I., and Hinton, G. E. ImageNet classification with deep convolutional neural networks. In *NIPS*, pp. 1106–1114, 2012.
67. Hinton G, Srivastava N, Swersky K (2012) Neural Networks for Machine Learning – Lecture 6e – rmsprop: Divide the gradient by a running average of its recent magnitude.

68. Xia, G.S.; Hu, J.; Hu, F.; Shi, B.; Bai, X.; Zhong, Y.; Zhang, L.; Lu, X. AID: A benchmark data set for performance evaluation of aerial scene classification. IEEE Trans. Geosci. Remote Sens. 2017, 55, 3965–3981.
69. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. & Salakhutdinov, R. Dropout: a simple way to prevent neural networks from overfitting. J. Machine Learning Res. 15, 1929–1958 (2014).

Appendix A – Precision and Recall

The *precision* and *recall* metrics for each model are provided in the table below. Precision is defined as (tp / (tp + fp)) * 100 and recall is defined as (tp / (tp + fn)) * 100, whereby tp = the number of true positives, fp = the number of false positives, and fn = the number of false negatives.

	Xception		**VGG-19**		**MobileNet**	
Class	**Precision (%)**	**Recall (%)**	**Precision**	**Recall (%)**	**Precision**	**Recall (%)**
1. agricultural	87	100	91	100	100	100
2. airplane	79	95	100	95	95	100
3. baseballdiamond	0	0	83	75	95	100
4. beach	100	100	100	90	87	100
5. buildings	64	70	73	80	95	90
6. chaparral	83	100	100	90	100	95
7. denseresidential	35	70	45	50	83	50
8. forest	0	0	86	95	95	100
9. freeway	87	100	100	95	100	80
10. golfcourse	74	85	61	70	86	95
11. harbor	100	95	100	100	95	100
12. intersection	70	80	100	85	100	95
13. mediumresidential	0	0	75	45	75	90
14. mobilehomepark	0	0	74	70	90	95
15. overpass	94	80	87	100	87	100
16. parkinglot	100	100	100	95	100	100
17. river	46	85	63	85	100	65
18. runway	100	95	100	90	83	100
19. sparesresidential	54	95	67	100	95	90
20. storagetanks	68	85	94	85	91	100
21. tenniscourt	65	85	86	60	100	100