remote sensing

Article

Simple Yet Effective Fine-Tuning of Deep CNNs Using an Auxiliary Classification Loss for Remote Sensing Scene Classification

Yakoub Bazi [1,*], Mohamad M. Al Rahhal [2], Haikel Alhichri [1] and Naif Alajlan [1]

1 Computer Engineering Department, College of Computer and Information Sciences, King Saud University, Riyadh 11543, Saudi Arabia; hhichri@ksu.edu.sa (H.A.); najlan@ksu.edu.sa (N.A.)

2 Information System Department, College of Applied Computer Science, King Saud University, Riyadh 11543, Saudi Arabia; mmalrahhal@ksu.edu.sa

* Correspondence: ybazi@ksu.edu.sa; Tel.: +96-610-146-962-97

Received: 19 October 2019; Accepted: 3 December 2019; Published: 5 December 2019

Abstract: The current literature of remote sensing (RS) scene classification shows that state-of-the-art results are achieved using feature extraction methods, where convolutional neural networks (CNNs) (mostly VGG16 with 138.36 M parameters) are used as feature extractors and then simple to complex handcrafted modules are added for additional feature learning and classification, thus coming back to feature engineering. In this paper, we revisit the fine-tuning approach for deeper networks (GoogLeNet and Beyond) and show that it has not been well exploited due to the negative effect of the vanishing gradient problem encountered when transferring knowledge to small datasets. The aim of this work is two-fold. Firstly, we provide best practices for fine-tuning pre-trained CNNs using the root-mean-square propagation (RMSprop) method. Secondly, we propose a simple yet effective solution for tackling the vanishing gradient problem by injecting gradients at an earlier layer of the network using an auxiliary classification loss function. Then, we fine-tune the resulting regularized network by optimizing both the primary and auxiliary losses. As for pre-trained CNNs, we consider in this work inception-based networks and EfficientNets with small weights: GoogLeNet (7 M) and EfficientNet-B0 (5.3 M) and their deeper versions Inception-v3 (23.83 M) and EfficientNet-B3 (12 M), respectively. The former networks have been used previously in the context of RS and yielded low accuracies compared to VGG16, while the latter are new state-of-the-art models. Extensive experimental results on several benchmark datasets reveal clearly that if fine-tuning is done in an appropriate way, it can settle new state-of-the-art results with low computational cost.

Keywords: scene classification; fine-tuning; vanishing gradient; auxiliary loss function; regularization layer

1. Introduction

In recent years, scene-level analysis has attracted much interest from the RS community thanks to the availability of RS images from a variety of earth observation platforms, including satellites, aerial systems, and unmanned aerial vehicles. The aim of scene classification methods is to classify the image based on a set of semantic categories in accordance with human interpretation. This task is challenging as it requires the definition of high-level features for representing the image content.

The latest developments based on deep learning methods have considerably boosted the classification accuracies compared to standard ones based on handcrafted features. The solutions based on CNNs are actually perceived as the most effective ones, in particular, those based on knowledge transfer from a pre-trained CNN. Indeed, using a pre-trained CNN has become a standard practice in many machine learning fields, including RS. One of the earliest ideas involves the removal of the top layer of the pre-trained network and its replacement with another fully connected layer that has a size equal to the number of classes (in the current problem) with a softmax activation layer. Then the network with its new top layer can be trained again on the RS dataset. This approach is typically known as fine-tuning. In general, the results reported up to now on several benchmark RS datasets show that the fine-tuning approach is less competing, in particular, for deeper networks due to the small size of the RS datasets (see literature review).

For this reason, other works opted instead to use the pre-trained network as a feature extractor by extracting features at different representations levels. This step may also include feature combinations. Then the resulting features are fed to an additional trainable module, which mainly acts as a classifier. Typical choices for the classification stage include fully connected layers and support vector machines (SVMs). On the other side, some works have proposed different methods and techniques to extract, combine, and fuse features from many pre-trained networks to enhance the classification accuracy. Thus, all of these methods in a sense are employing feature engineering one way or another, which defeats the main characteristic of deep neural networks, which is learning feature representations in an "end-to-end" manner from the dataset automatically. In fact, this characteristic has enabled them to significantly outperform handcrafted features in recent years. Hence, coming back to feature engineering practices while using deep neural networks seems counter-intuitive. So why fine-tuning of deeper networks is not competing with those based on feature extraction? The main reason reported so far in the literature is the deep nature of these networks, which causes them to suffer from the problem of "vanishing gradients" during training.

The question that we investigate in this work is: can we find a better way to combat the effect of vanishing gradients in deeper networks, such as GoogLeNet and Inception-v3 [1–4]? To this end, we will carry out an extensive analysis to show that these type of networks exhibit a phenomenon similar to the "Hughes effect" widely encountered in hyperspectral imagery. Then in a second step, we propose a simple solution to combat this effect by placing an auxiliary network on the top of an earlier layer to inject additional gradients. Then, we optimize both the primary and auxiliary loss functions using an opportune optimization method. It is worth recalling, that this idea was originally employed by Google during the training of GoogLeNet [1]. Yet this idea has been overlooked by researchers when transferring knowledge to new datasets during fine-tuning as they consider only the primary loss function. In the second set of experiments, we show this technique can boost the model performance significantly and permits to settle new state-of-the-art classification results on many benchmark datasets. The main contributions of this paper could be summarized as follows:

- Provide best practices for transferring knowledge from pre-trained CNNs with small weights;
- Show that deeper CNNs suffer from the vanishing gradient problem, and then propose a simple yet efficient solution to combat this effect using an auxiliary loss function;
- Confirm experimentally, that this simple trick allows settling new state-of-the-art results on several benchmark datasets with low computational cost (fine-tune for a maximum 40 iterations).

The remainder of the paper is organized in five sections. In Section 2 we review the main methods based on deep learning for scene classification in RS imagery. In Section 3, we introduce the different CNNs investigated in this work. Section 4, presents our fine-tuning method. Then, in Section 5, we present the experimental results on five well-known datasets followed by discussions in Section 6. Finally, we provide concluding remarks and future directions in Section 7.

2. Related Works

As mentioned in the introduction section, scene classification in RS imagery was approached using fine-tuning or feature extraction methods. The fine-tuning approach (Table 1) was investigated for AlexNet, GoogLeNet, and VGG16. As can be seen, the standard practice was to fine-tune the network up to thousands of iterations using gradient-based algorithms. One can also notice that, with the exception of Merced dataset, GoogLeNet clearly exhibits less performance compared to VGG16. In a very recent work [5], the authors reported better results for VGG16 using an Adam optimizer for only 50 iterations. Although many improved networks are available, it appears that VGG16 is still the primary choice for many RS researchers.

Table 1. Results obtained by the fine-tuning approach in previous studies.

Work	**Method**	**CNN**	**Acc [%]**	**Train [%]**
		Merced dataset		
Castelluccio et al. [6] 2015	SGD, with a learning rate of 0.001, and 20,000 iterations.	GoogLeNet	97.10%	80%
Cheng et al. [7] 2018	Adam, with a learning rate of 0.001 for the classification layer, and 0.001 for the other layers. The iteration number changes from 1000 to 15,000 with a stride of 1000.	VGG16+SVM GoogLeNet+SVM AlexNet+SVM	96.82 ± 0.20 97.14 ± 0.10 94.58 ± 0.11	80%
Nogueira et al. [8] 2017	SGD, with a learning rate of 0.001, and 20,000 iterations.	GoogLeNet	97.78 ± 0.97	80%
Sun et al. [5] 2019	SGD 0.0001 learning rate, 50 iterations	VGG16	97.14 ± 0.48 96.57 ± 0.48	80% 50%
		AID dataset		
Sun et al. [5] 2019	SGD 0.0001 learning rate, 50 iterations	VGG16 VGG16	93.60 ± 0.64 89.49 ± 0.34	50% 20%
		NWPU dataset		
Boualleg et al. [9] 2019	0.01 learning rate for the last layer, 0.001 for other layer, 15,000 iterations	VGGNet16: GoogLeNet: AlexNet:	87.15 ± 0.45 82.57 ± 0.12 81.22 ± 0.19	10%
Boualleg et al. [9] 2019	0.01 learning rate for the last layer, 0.001 for other layer, 15,000 iterations	VGGNet16: GoogLeNet: AlexNet:	90.36 ± 0.18 86.02 ± 0.18 85.16 ± 0.18	20%
Cheng et al. [10] 2017		VGG16	84.56	10%
		Optimal-31 dataset		
Wang et al. [11] 2018	No details were provided for the used parameters.	VGG16 GoogLeNet AlexNet	87.45 ± 0.45 82.57 ± 0.12 81.22 ± 0.19	80%
Sun et al. [5] 2019	SGD 0.0001 learning rate, 50 iterations	VGG16	89.52 ± 0.26	80%

Regarding feature extraction, the literature reports several methods with increased complexity. For example, Singh et al. [12] proposed a weakly supervised network that is able to classify the image and localize the most distinctive regions when trained on class labels only. They showed that training the model on the relevant regions instead of the entire scene can increase the classification accuracy. Liu et al. [13] proposed a scene classification triplet network trained entirely from scratch on weakly labeled data. The network is trained using image triplets. The cross-entropy loss is replaced with several loss functions based on the difference loss. In [14], the authors proposed an architecture that stacks multiple autoencoders to learn hierarchical feature representations. A Fisher vector pooling layer is used to build global feature representation of the image. Zhang et al. [15] proposed a gradient-boosting random framework that combines multiple CNNs to effectively classify remote sensing images. The authors in [16] proposed a feature representation method termed as a bag of convolutional features, that generates visual words from CNNs instead of handcrafted features. Wang at el. [11] used the recurrent neural networks (RNNs) to learn on the key regions of the scene. First, a pre-trained CNN is used as a feature extractor. Then, a mask matrix is applied to extract the key regions. The RNN is used to train the mask matrix and process the recurrent features sequentially. In [17], the authors used a pre-trained CNN to generate an initial feature representation of the images. These features are then coded using a sparse autoencoder for learning a new representation. The classification is performed by either adding a softmax layer on the top of the encoding layer and fine-tune the full network, or by training an autoencoder for each class. In [18], the authors used a pre-trained CNN for feature extraction. They removed the last fully connected layer and used extreme learning machine (ELM) for classification.

In other contributions, the authors tried to improve image representations by resolving the problem of object rotation and scale variations and training the model on scale-invariant features [19–21] or rotation-invariant features [22]. In [19], the authors proposed a two-branch network, one for fixed-size images and the other for varied-size images. The two branches are trained simultaneously via shared weights. Spatial pyramid pooling and similarity measure layers are added to force the network to learn the multiscale features. Liu et al. [23] proposed CNNs with varied-input sizes. Multiscale images are fed into their corresponding CNNs and spatial pyramid pooling is used to accelerate the training of these multiple networks. The best combination of the features is chosen by learning a multiple kernel method. In [22], the authors proposed a network that can combat variance in image orientation in large-scale remote sensing images by using a squeeze layer with average, active rotating filters.

Other group of works embedded a metric learning regularizer for better representation of remote sensing images [7,24]. Cheng at el. [6] proposed to embed a metric learning term along with the cross-entropy term to learn more discriminative features. This term maps image pairs that belong to the same class to be as close as possible, while images of different classes are mapped to be as farther as possible. In [24], the authors considered the contextual information between different pairs during training and proposed a diversity regularization method to reduce the redundancy of learned parameters.

One of the recent trends for image representation is using a combination of features where different features are extracted from multiple hidden layers of pre-trained CNNs and combined using a fusion strategy [25–30]. In [25], the authors trained three CNNs concurrently, each with distinct receptive field size. The image and two patches extracted from the image are fed into these networks. Then, a probability fusion model is used to combine the features. Liu at el. [26] integrated two pre-trained CNNs (i.e., CaffeNet and VGG16) and combined features from the lower layers of the network with the fully connected layer. The two resulting converted CNNs are adaptively integrated to further improve the classification accuracy. Marmanis at el. [27] proposed a two-stage framework. In the first stage, an initial set of features are extracted from a pre-trained CNN. Then, the obtained features are fed into a supervised CNN classifier to train the network.

Another work [28] used features extracted from pre-trained CNNs, namely VGG and ResNet. Two types of features are considered: the high-level features extracted from the last fully connected layer, and the low and mid-level features extracted from the intermediate convolutional layers. Extracted features are reduced by Principal Component Analysis (PCA) and then concatenated. Chaib et al. [30] used a pre-trained VGG network for feature extraction. The outputs of the first and second fully connected layers of the network are transformed using discriminant correlation analysis and then fused through concatenation and addition. In [29], the authors proposed a fusion strategy for features extracted from multiple layers of a pre-trained CNN. This strategy fused features encoded by a multiscale improved Fisher vector and the output of the last fully connected layer using PCA with a spectral regression kernel discriminant analysis.

3. Inception Networks and EfficientNets

3.1. Inception Networks

Inception networks are a family of CNNs that are developed by a group of researchers at Google [1,3,4]. They have a lot of heavily engineered tricks that set them apart from other conventional CNN architectures, such as AlexNet or VGG16. In general, inception networks have a lower number of weights than competitive networks, such as AlexNet and VGG16. Table 2 shows a comparative summary.

Table 2. Weights for different CNN architectures [31].

Network	#Parameters
AlexNet	60.97 M
VGG16	138.36 M
GoogLeNet [1]	7 M
Inception-v3 [3]	23.83 M
Inception-v4 [4]	42.71 M
Incep-Res-v2 [4]	55.97 M

The earliest version was introduced back in 2014 as GoogLeNet by Szegedy et al. [1]. Later on, this was also named as Inception-v1. In 2015, Szegedy et al. improved on their first version by introducing a new batch normalization layer in a second version called Inception-v2 [2] and the concept of factorizing convolutions in the third version Inception-v3 [3]. Finally, the team presented further improvements in the Inception-v4 and the Inception-ResNet versions in another paper [4]. Inception-v4 contained several simplifications which reduced the computational costs. While Inception-ResNet added the concept of residual blocks, which was inspired by the success of the ResNet architecture [32]. In the following subsection, we discuss in more detail the different inception versions.

3.1.1. GoogLeNet (Inception-v1)

The first version of inception networks, known as GoogLeNet, is composed of several inception modules. The inception modules deal better with variations in scale and location. They apply filters with multiple sizes on the same level. The network essentially gets a bit "wider" rather than "deeper". The initial design of the inception module is shown in Figure 1.

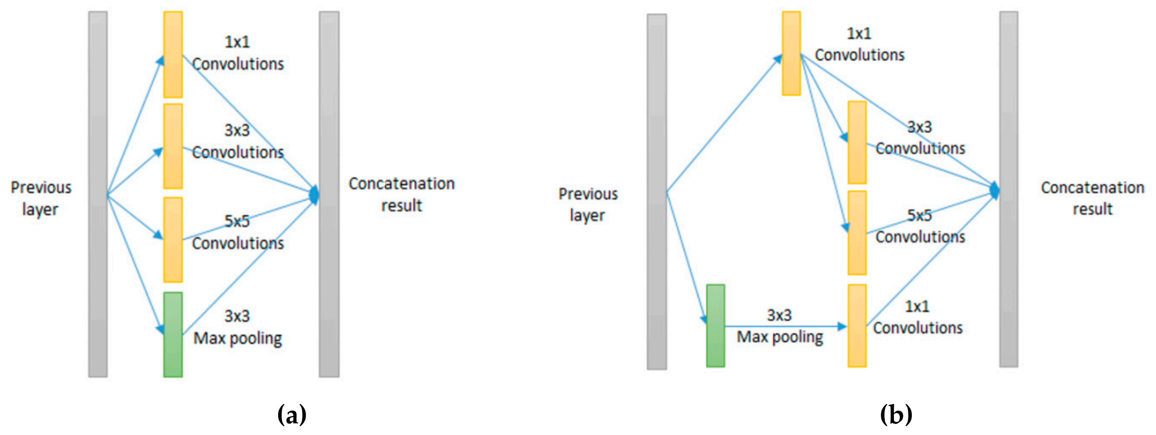


Figure 1. Inception modules where filters with different sizes are applied at the same level. (**a**) Naïve version of an inception module. (**b**) Inception module with dimension reduction.

The complete network is shown in Figure 2, where we see that it is composed of an initial stem followed by a sequence of inception modules, and finally global average pooling and a softmax layer.

As with any very deep network, Inception-v1 is subject to the vanishing gradient problem. As a solution, the authors in [1] introduced the two auxiliary classifiers as shown in Figure 2. In these two auxiliary classifiers, they applied softmax to the outputs of two inception modules and computed an auxiliary loss over the same labels. The network was trained using all three losses combined. The motivation behind this was to combat the effect of the "diminishing gradient" problem due to the depth of the network. According to the authors, only one auxiliary classifier near the end of the network was found to improve accuracy. This motivated our second set of experiments, where we use this technique while fine-tuning the inception networks.

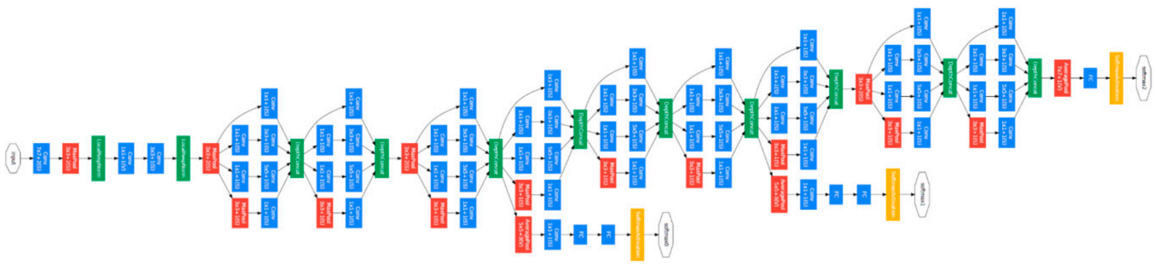


Figure 2. Inception-v1 (GoogLeNet): The orange box is the stem, which has some preliminary convolutions. The purple boxes are auxiliary classifiers. The wide parts are the inception modules [1].

3.1.2. Inception-v3

This network included more improvements in the architecture, including (1) the RMSprop optimizer, (2) factorized 7×7 convolutions, (3) batch normalization in the auxiliary classifiers, and (4) label smoothing, which is a type of a regularizing component added to the loss formula that prevents the network from becoming too confident about a class and prevents overfitting. The general architecture of the Inception-v3 network is shown in Figure 3. It was the first runner-up in the image classification competition in ImageNet Large Scale Visual Recognition Competition (ILSVRC) 2015 [3].

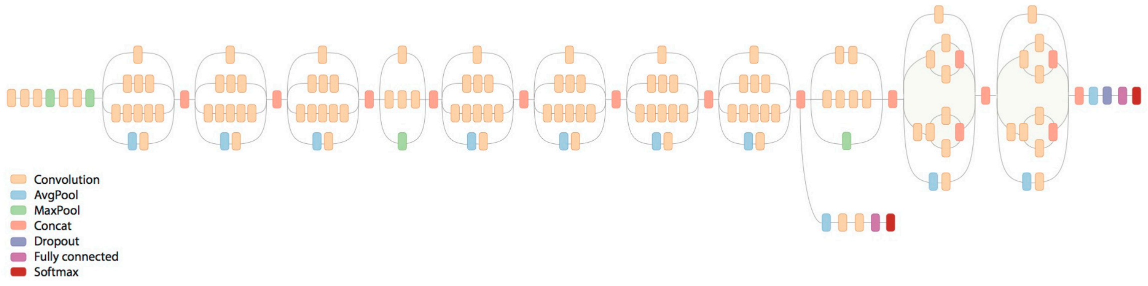
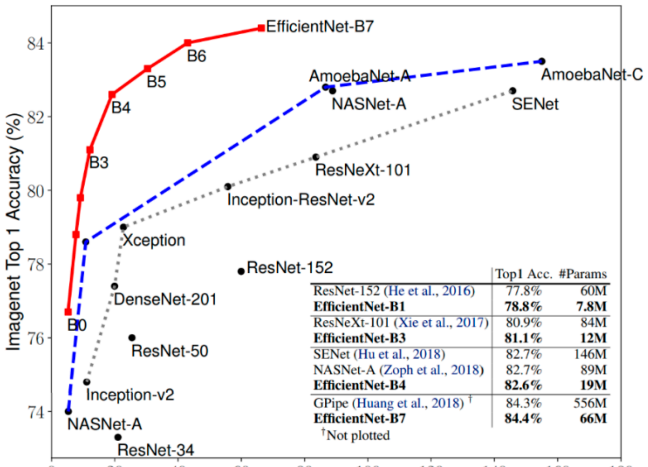


Figure 3. Inception-v3: The auxiliary loss function was used by the authors only in the training phase.

3.2. *EfficientNets*

EfficientNet is a new model scaling method, developed by Google recently [33], for scaling up CNNs. It uses a simple, greatly effective compound coefficient. EfficientNet works differently from traditional methods that scale dimensions of networks, such as width, depth, and resolution; and it scales each dimension with a fixed set of scaling coefficients uniformly. Practically, scaling individual dimensions improves model performance; however, balancing all dimensions of the network with respect to the available resources effectively improves the whole performance (Figure 4). Model scaling's efficacy depends strongly on the baseline network. To this end, a new baseline network is created by using the AutoML framework, which optimizes both precision and effectiveness (FLOPS), to perform a neural architecture search. Similar to MobileNetV2 and MnasNet, EfficientNet uses mobile inverted bottleneck convolution (MBConv) as the main building block. Additionally, this network uses a new activation function called swish instead of the Rectifier Linear Unit (ReLU) activation function. EfficientNet-B0 is shown in Figure 5. In this work, we propose to investigate the baseline architecture of EfficientNet-B0 and its deeper version, EfficientNet-B3.



	Top1 Acc.	#Params
ResNet-152 (He et al., 2016)	77.8%	60M
EfficientNet-B1	**78.8%**	**7.8M**
ResNeXt-101 (Xie et al., 2017)	80.9%	84M
EfficientNet-B3	**81.1%**	**12M**
SENet (Hu et al., 2018)	82.7%	146M
NASNet-A (Zoph et al., 2018)	82.7%	89M
EfficientNet-B4	**82.6%**	**19M**
GPipe (Huang et al., 2018) †	84.3%	556M
EfficientNet-B7	**84.4%**	**66M**

†Not plotted

Figure 4. Comparison between EfficientNets and other CNNs [33].

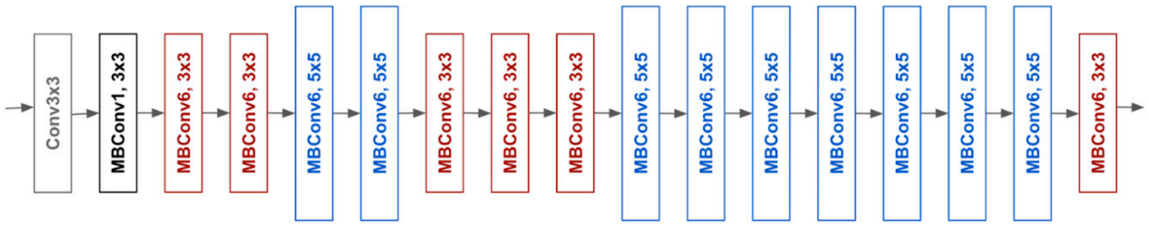
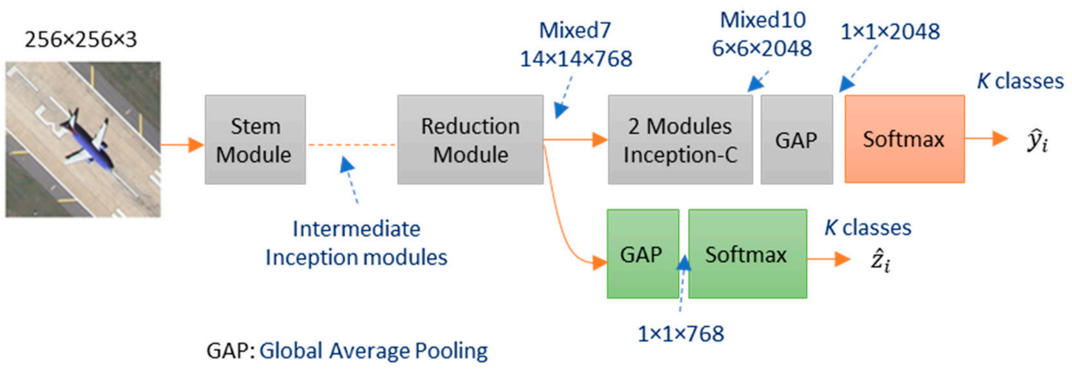


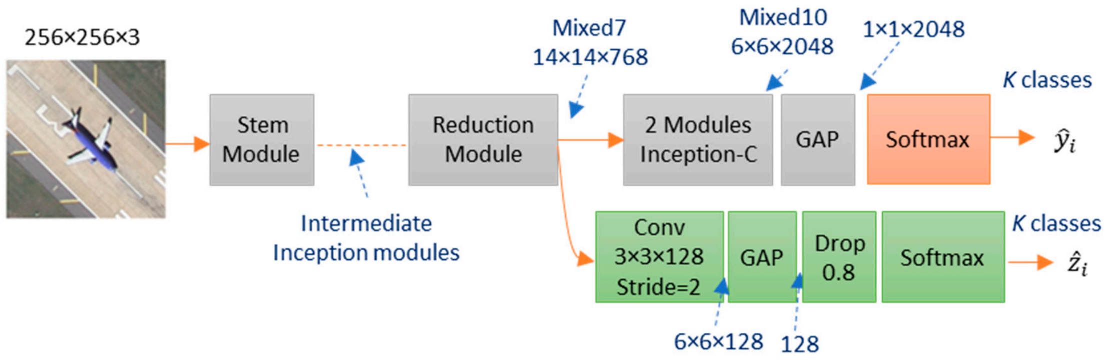
Figure 5. Architecture of the baseline EfficientNet-B0.

4. Proposed Fine-Tuning Method

Let us consider $D = \{X_i, y_i\}_{i=1}^{n}$ as a training set composed of n remote sensing images, each of dimension $256 \times 256 \times 3$ pixels. To each image X_i, we associate a corresponding binary label vector $y_i \in \mathcal{R}^K$, where K is the total number of classes. In a multiclass setting, the kth entry of this vector that corresponds to the kth class is set 1, while the other entries are set to zero (i.e., $\sum_{k=1}^{K} y_{ik} = 1$). Let us also define $\hat{y}_i$ as the output vector obtained from the main softmax function placed on the top of the network. In addition, we denote $\hat{z}_i$ as an auxiliary output vector obtained from another auxiliary softmax function placed on the top of an intermediate layer as shown in Figure 6 in the case of Inception-v3. The position of this auxiliary softmax function is important to boost the network performances. In the experiments, we will investigate this issue and show that placing this layer near to the top of the network allows combating the vanishing gradient problem in a better way and thus resulting in increased classification accuracies. To this end, we propose two different configurations for this auxiliary network using simple global average pooling (GAP) followed by a softmax or using additional convolution filters (Conv+BN+ReLU+GAP+Dropout(0.8)+Softmax). We recall that this auxiliary layer will be used for regularization purposes during the training phase only and will be removed in the test phase thus keeping the original network architecture. For EfficientNets and for conformity, we adopt the swish activation function instead of ReLU. Table 3 provides the suitable position of the auxiliary softmax function for each of the four networks considered in this work.



(a)



(b)

Figure 6. Example of adding an auxiliary softmax function to the Inception-v3 network. Two configurations are proposed: (**a**) a simple GAP+Softmax or (**b**) applying an additional convolution: Conv+BN+ReLU+GAP+Dropout(0.8)+Softmax.

Table 3. Position of the auxiliary softmax function for the four networks investigated in this work. A typical choice is near to the top of the network before the last spatial reduction step.

CNN	Placement of the Auxiliary Softmax	Layer Output
GoogLeNet	Mixed_4f_Concatenatated	$16 \times 16 \times 832$
Inception-v3	Mixed7	$14 \times 14 \times 768$
EfficientNet-B0	Swish34	$16 \times 16 \times 672$
EfficientNet-B3	Swish54	$16 \times 16 \times 816$

From a statistical point of view, the distribution of the network outputs can be regarded as a generalization of the Bernoulli distribution to more than two classes (i.e., $\Pi_{k=1}^{K}\hat{y}_{ik}^{y_{ik}}$ and $\Pi_{k=1}^{K}\hat{z}_{ik}^{y_k}$ for the main and auxiliary softmax layers, respectively). The weight W of the network can be learned by maximizing the following log-likelihood function:

$$L(D,W) = \lambda \sum_{i=1}^{N} \ln\left(\Pi_{k=1}^{K}\hat{y}_{ik}^{y_{ik}}\right) + (1-\lambda)\sum_{i=1}^{N} \ln\left(\Pi_{k=1}^{K}\hat{z}_{ik}^{y_k}\right), \tag{1}$$

where λ is a regularization parameter that controls the contributions of the two terms. Setting this parameter to $\lambda = 1$ will result in the standard fine-tuning approach.

The above problem is equivalent to minimizing the following loss function:

$$L(D,W) = -\lambda \sum_{i=1}^{N} \ln\left(\Pi_{k=1}^{K}\hat{y}_{ik}^{y_{ik}}\right) - (1-\lambda)\sum_{i=1}^{N} \ln\left(\Pi_{k=1}^{K}\hat{z}_{ik}^{y_{ik}}\right). \tag{2}$$

This expression could be further expressed as:

$$L(D,W) = -\lambda \sum_{i=1}^{N} \sum_{k=1}^{K} y_{ik} \ln \hat{y}_{ik} - (1-\lambda) \sum_{i=1}^{N} \sum_{k=1}^{K} y_{ik} \ln \hat{z}_{ik} \,. \tag{3}$$

By taking into consideration that the binary label vector y_i has only one entry set to 1, we obtain the so-called cross-entropy loss function:

$$L(D,W) = -\lambda \sum_{i=1}^{N} \sum_{k=1}^{K} 1\left(y_i = k\right) \ln \hat{y}_{ik} - (1-\lambda) \sum_{i=1}^{N} \sum_{k=1}^{K} 1\left(y_i = k\right) \ln \hat{z}_{ik} \,, \tag{4}$$

where $1(\cdot)$ is an indicator function that takes 1 if the statement is true, otherwise it takes 0. To optimize the cost function $L(D,W)$, we use the RMSprop optimization method, which is one of the most popular adaptive gradient algorithms introduced by Hinton to speed up the training deep neural networks. The RMSprop divides the gradient by a running average of its recent magnitude.

$$E\left[g^2\right]_t = \beta E\left[g^2\right]_{t-1} + (1-\beta)\left(\frac{\partial L}{\partial W}\right)^2, \tag{5}$$

$$W_t = W_{t-1} - \alpha\left(\frac{\partial L}{\partial W}\right)\frac{1}{\sqrt{E[g^2]_t}} \,, \tag{6}$$

where $E\left[g^2\right]_t$ is the moving average of squared gradients at iteration t, $\frac{\partial L}{\partial W}$ is the gradient of the loss function with respect to the weight W, while α is the learning rate and β is the moving average parameter. In the experiments, we set the parameter β to its default values ($\beta = 0.9$), while for the learning parameter α, we set it initially to 0.0001 and decrease by a factor of 1/10 every 20 epochs.

5. Experiments

In this section, we present an extensive experimental analysis to demonstrate the capabilities of the proposed solution. We test the proposed methods on five common RS scene datasets, namely, the University of California (UC) Merced dataset [34], the aerial image dataset (AID) [35], the Kingdom of Saudi Arabia (KSA) dataset [36], the NWPU-RESISC45 dataset [10], and the latest Optimal-31 dataset [11]. In the next section, we provide a more detailed description of each dataset. Then in the section that follows, we present the results of each experiment.

5.1. Dataset Description

UC Merced land-use dataset: The UC Merced dataset consists of 2100 RGB images measuring 256×256 pixels of 21 categorized land-use classes (100 images per class). The class labels are as follows: agricultural, airplane, baseballdiamond, beach, buildings, chaparral, denseresidential, forest, freeway, golfcourse, harbor, intersection, mediumresidential, mobilehomepark, overpass, parkinglot, river, runway, sparseresidential, storagetanks, and tenniscourt. The images were manually extracted by Yang and Newsam from the United States Geological Survey (USGS) [34]. These images have a pixel resolution of one foot.

Aerial image dataset: The aerial image dataset (AID) is a large-scale dataset consisting of more than 10,000 aerial sense images measuring 600×600 pixels, categorized within 30 classes [35]. AID is collected from Google Earth imagery which is carefully chosen from different countries and regions at different times and seasons around the world, mostly from China, the United States, England, France, Italy, Japan, and Germany. These images have a pixel resolution of about half a meter.

KSA dataset: This dataset is categorized into 13 classes acquired over different cities in the Kingdom of Saudi Arabia (KSA). It has 250 images per class of size 256×256 pixels [17,37]. The class labels are as follows: agriculture, beach, cargo, chaparral, dense residential, dense trees, desert, freeway, medium-density residential, parking lot, sparse residential, storage tanks, and water. The KSA dataset includes a total of 3250 RGB images with spatial high resolutions of 1 and 0.5 m.

NWPU-RESISC45: The NWPU-RESISC45 acquired from Google Earth imagery was created by the Northwestern Polytechnical University (NWPU) [4]. The dataset consists of 31,500 remote sensing images, divided into 45 classes. Each class contains 700 images with size cropped to 256 × 256 pixels. Most of the classes have spatial resolutions that vary from around 30 meters to 0.2 meters, except those with lower spatial resolutions: island, lake, mountain, and snowberg.

Optimal-31: This is a new dataset containing images from Google Earth imagery as shown in Figure 7. The images have a size of 256 × 256 pixels and their resolution is 0.5 meters. Optimal-31 categorizes 1860 images within 31 classes, and each class contains 60 images [11].

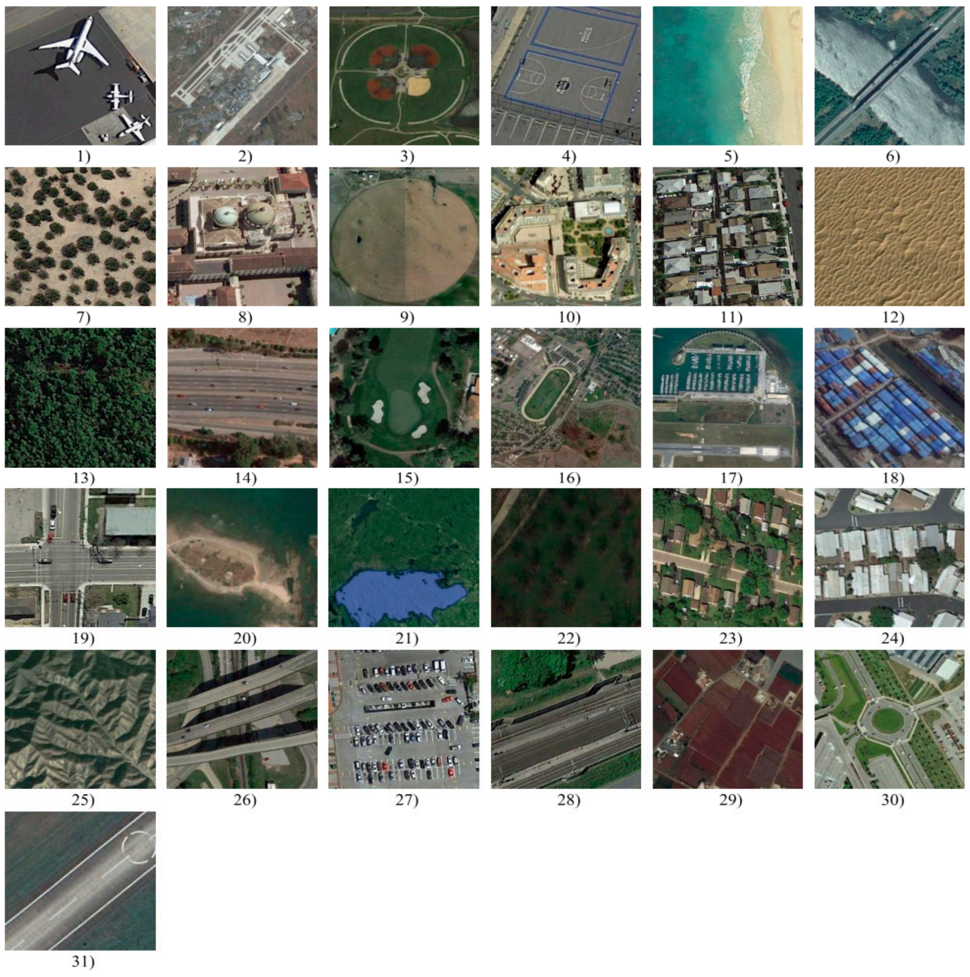


Figure 7. Optimal dataset.

5.2. Experimental Set-Up

We ran the experiments on an HP Omen Station with the following characteristics: central processing unit (CPU)-Intel core (TM) i9-7920x CPU @ 2.9GHz with a RAM of 32 GB and graphical processing unit (GPU) called NVIDIA GeForce GTX 1080 Ti (11 GB GDDR5X). All codes were implemented using Keras, which is an open-source deep neural network library written in Python. For each dataset, the results are presented in terms of overall accuracy (OA) and standard deviation (STD) over five trials. Table 4 summarizes the main parameters used in all the experiments.

Table 4. Parameters used in the experiments.

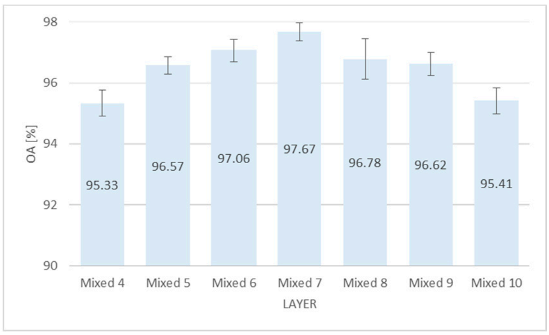
Parameter	Settings
Optimizer	RMSprop
Learning parameter α	Initial:0.0001, then decreased by a factor of 1/10 after each 20 iterations
Moving average β	0.9
Maximum number of epochs	40
Loss contribution λ	0.5
Images size	256×256 pixels
Mini-batch size	50
Number of trials for each experiment	5
Data augmentation	Not used

5.3. Experiments on Inception-v3

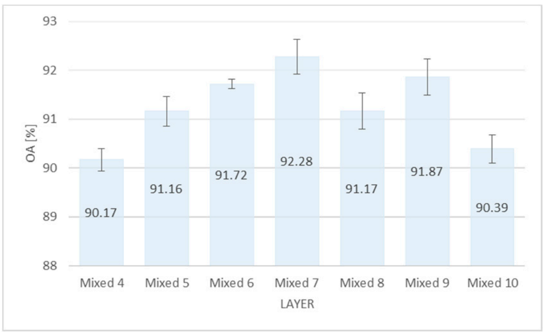
Inception-v3 is one of the standard pre-trained networks bundled with major deep learning frameworks. The inception-v3 network is a very deep network with 48 layers in total. In the first set of experiments, we analyzed its sensitivity with respect to the vanishing gradient problem. We built subnetworks by taking the output of an intermediate inception layer and discarding the remaining ones, then applying GAP followed by softmax and fine-tuning by using only the primary loss function. Figure 8 and Table 5 summarize the results obtained by stopping at different middle inception layers and for all the five datasets. The results clearly showed that the network suffers from the peaking paradox (widely known in the remote sensing community as the Hughes effect), which has been observed in many scientific problems in the past [38]. Basically, this paradox says that extracting more and more features from data may help at the beginning, but then at some point, the increased complexity of the feature extractor starts to hurt the performance of the model. In pattern recognition, many researchers have observed that in order to increase the recognition accuracy, we usually need to compute more detailed features. At some point, this becomes counterproductive and the recognition accuracy starts to decrease again. This is because computing more discriminative features requires the estimation of additional parameters in the model. If the training data is limited, then the estimation of the parameters will lead to overfitting problems. Table 5 reports the OA for all datasets and suggests that the inception layer termed as Mixed7 is the best for transferring knowledge from the network.

Table 5. Sensitivity analysis with respect to the network depth. The results are presented in terms of (OA ± STD) over five trials with different training and testing images.

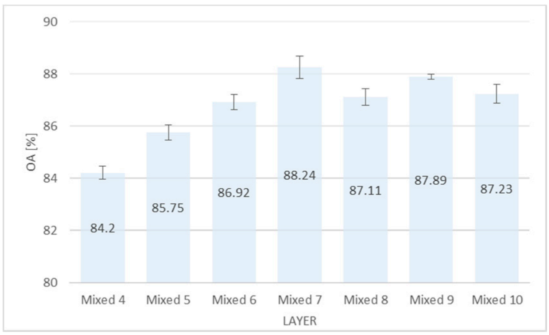
Inception Mixed Layer	Layer Size	Merced 50% Train	AID 20% Train	NWPU 10% Train	Optimal-31 80% Train	KSA 20% Train
Mixed4	(12,12,768)	95.33 ± 0.43	90.17 ± 0.23	84.20 ± 0.25	90.57 ± 0.30	94.11 ± 0.42
Mixed5	(12,12,768)	96.57 ± 0.28	91.16 ± 0.31	85.75 ± 0.30	91.66 ± 0.34	94.58 ± 0.38
Mixed6	(12,12,768)	97.06 ± 0.36	91.72 ± 0.10	86.92 ± 0.28	92.02 ± 0.87	94.80 ± 0.33
Mixed7	(12,12,768)	97.67 ± 0.29	92.28 ± 0.35	88.24 ± 0.43	93.11 ± 1.04	95.03 ± 0.50
Mixed8	(5,5,1280)	96.78 ± 0.67	91.17 ± 0.37	87.11 ± 0.31	91.88 ± 1.03	93.38 ± 0.66
Mixed9	(5,5,2048)	96.62 ± 0.38	91.87 ± 0.37	87.89 ± 0.10	92.79 ± 1.02	94.38 ± 0.55
Mixed10	(5,5,2048)	95.41 ± 0.42	90.39 ± 0.29	87.23 ± 0.36	91.45 ± 1.10	92.27 ± 1.65



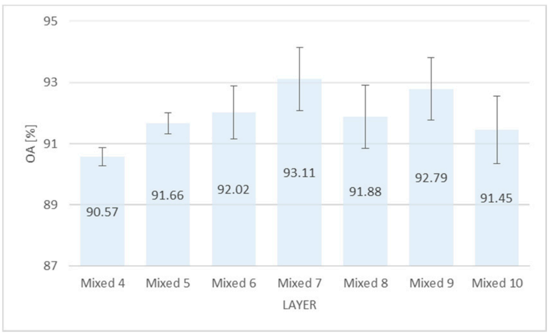
(a)



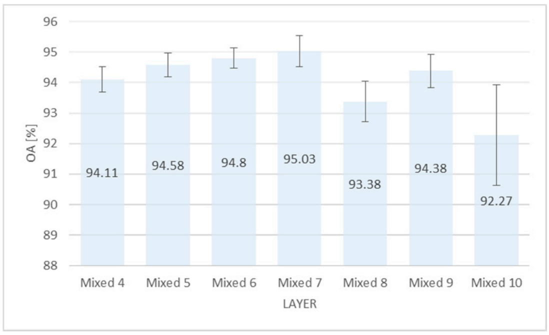
(b)



(c)



(d)



(e)

Figure 8. Classification accuracy represented in terms of Overall Accuracy±Stdandrd Deviation (OA ± STD) over five runs for (**a**) Merced, (**b**) AID, (**c**) NWPU, (**d**) Optimal-31, and (**e**) KSA datasets. Each time we fine-tuned the network up to a certain inception layer, while removing the upper layers.

In the second set of experiments, we adopted our proposed solution by adding an auxiliary loss function to the network on the top of Mixed7 (since we found in the previous experiment that it is the best choice for transferring knowledge) and then fine-tuned it by using both the primary and secondary loss functions. Table 6 shows the results obtained for the two different configurations proposed for the auxiliary loss function: Softmax and Conv+Softmax, as explained in the methodological section. Table 6 reports the overall accuracies obtained for the five datasets. The overall accuracy was calculated based on the classes predicted from the primary output only, while the auxiliary output was not considered. As can be seen from the results, the auxiliary classifier enabled a significant improvement (more than 2%) in the classification accuracy for all datasets, confirming that it plays an important role in combatting the vanishing gradient problem. By averaging the results over the five datasets, the inclusion of the auxiliary softmax layer allowed to boost the classification accuracy from 91.35% to 93.73%. On the other hand, the second version of the auxiliary classifier based on Conv+Softmax allowed the classification to reach an accuracy of 94.19%.

Table 6. Fine-tuning results obtained using an auxiliary loss function besides the primary one compared to the standard fine-tuning solution based only on the primary loss.

Dataset	Without Auxiliary	Auxiliary Softmax	Auxiliary Conv + Softmax
Merced	95.41 ± 0.42	97.35 ± 0.43	97.63 ± 0.20
AID	90.39 ± 0.29	92.69 ± 0.34	93.52 ± 0.21
NWPU	87.23 ± 0.36	89.28 ± 0.29	89.32 ± 0.33
Optimal31	91.45 ± 1.10	93.81 ± 0.51	94.13 ± 0.35
KSA	92.27 ± 1.65	95.55 ± 0.25	96.36 ± 0.24
Average	91.35 ± 0.76	93.73 ± 0.36	94.19 ± 0.27

5.4. Experiments on GoogLeNet

GoogLeNet (or Inception-v1) has been used in the RS literature by many researchers as indicated in the literature review. In general, the reported results were less competing compared to other models. In this section, we will show that this network has been penalized and demonstrate that it can provide better results compared to state-of-the-art. Table 7 shows the OA for all dataset averaged over five runs. As can be seen, this network seemed to be less sensitive to the depth effect compared to Inception-v3, as the results produced by the output layer Mixed_5c were better compared to the hidden inception layer Mixed_4f; however, adding the auxiliary classifier on the top of Mixed_4f produced improvement for all five datasets, albeit not as significant as for Inception-v3. By averaging the OA over the five datasets, the inclusion of the auxiliary softmax and auxiliary Conv+Softmax allowed to increase the accuracies from 92.95% to 93.25% and 93.92%, respectively.

Table 7. OA (%) Obtained by fine-tuning GoogLeNet without and with an auxiliary classifier.

	Mixed_4f	Without Auxiliary	With Auxiliary Softmax	With Auxiliary Conv + Softmax
Merced	97.12 ± 0.21	97.04 ± 0.26	97.52 ± 0.28	97.90 ± 0.34
AID	91.09 ± 0.20	92.09 ± 0.14	92.58 ± 0.22	93.25 ± 0.33
NWPU	85.52 ± 0.17	88.16 ± 0.13	88.06 ± 0.34	89.22 ± 0.25
Optimal-31	90.64 ± 0.90	92.36 ± 0.60	92.63 ± 0.36	93.11 ± 0.55
KSA	94.32 ± 0.68	95.10 ± 0.53	95.46 ± 0.50	96.14 ± 0.39
Average	91.74 ± 0.43	92.95 ± 0.33	93.25 ± 0.34	93.92 ± 0.37

Figure 9 shows the evolution of the loss function during training with and without the auxiliary classifier. Due to space limitation, we only show this for two datasets, namely Merced and AID datasets; however, we observed this behavior for all datasets. The use of the auxiliary classifier made the loss to converge at a slower rate, indicating that the gradient did not vanish in early iterations.

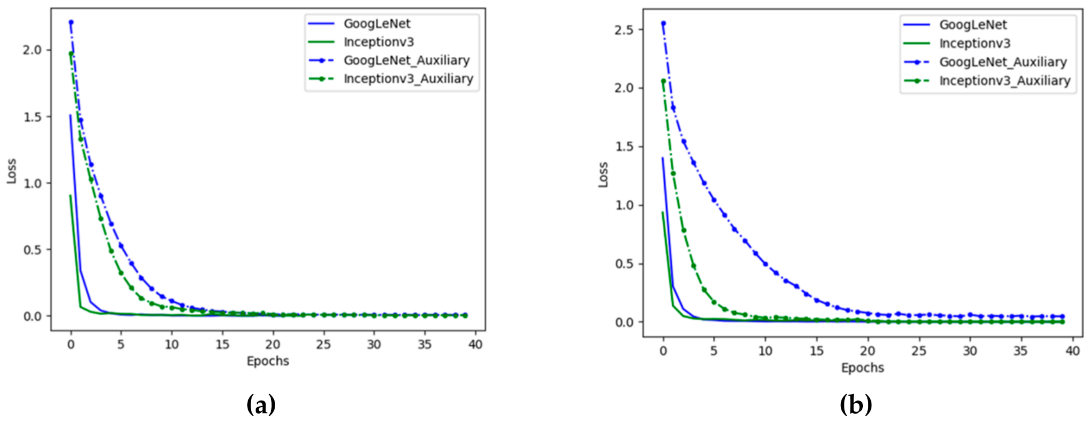
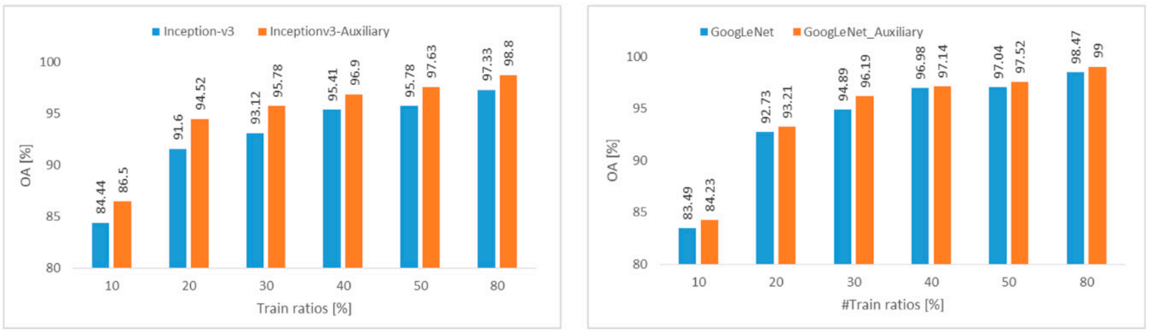


Figure 9. Loss versus the number of epochs obtained by Inception-v3 and GoogLeNet by training on (**a**) Merced and (**b**) AID datasets.

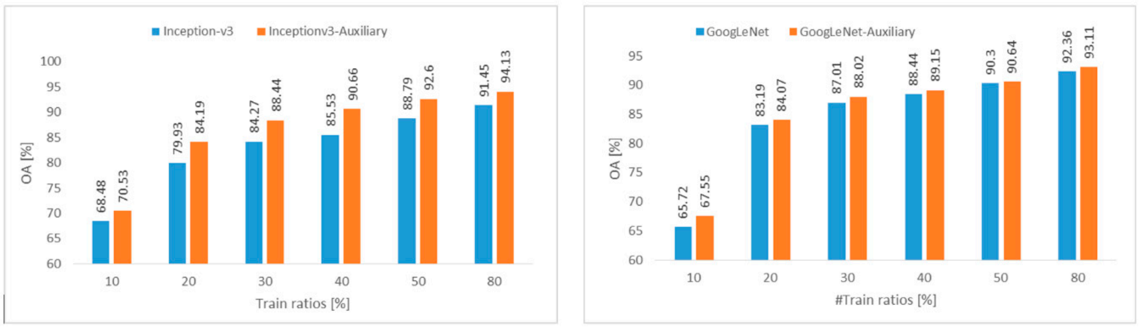
5.5. Sensitivity Analysis with Respect to the Training Size

In this experiment, we further analyzed the effect of the training set sizes on the performance of both networks with/without an auxiliary classifier. As for the auxiliary classifier, we considered only the second configuration based on (Conv+Softmax) as it yielded better improvements. We used the Merced and Optimal-31 datasets to perform this experiment since they are small datasets requiring less computational costs. Figures 10 and 11 show the OA for both GoogLeNet and Inception-v3 by varying the training set size from 10% to 80%. These figures clearly illustrated that both networks supplemented with an auxiliary classifier always provided improvements for all training set sizes. For Inception-v3, we obtained an average improvement of 2.08% and 3.7% in accuracies for the Merced and Optimal-31 datasets, respectively. On the other hand, GoogLeNet was found to be less sensitive to the depth effect, but we always observed an increase in the accuracy. The average improvement in accuracies for GoogLeNet was 0.8% and 0.92% for the Merced and Optimal-31 datasets, respectively.



(a) **(b)**

Figure 10. OA obtained on the Merced dataset for different training ratios by fine-tuning (**a**) Inception-v3 and (**b**) GoogLeNet without and with an auxiliary classifier (GAP+Conv+BN+ReLU+Dropout+Softmax).



(**a**) (**b**)

Figure 11. OA obtained on the Optimal-31 dataset for different training ratios by fine-tuning (**a**) Inception-v3 and (**b**) GoogLeNet without and with an auxiliary softmax layer (GAP+Conv+BN+ReLU+Dropout+Softmax).

5.6. Experiments using EfficientNets

In this experiment, we further analyzed the validity of the proposed solution using EfficientNets. We considered the baseline version EfficientNet-B0 and its deeper version, EfficientNet-B3. As can be seen from Table 8, EfficientNet-B0 was less sensitive compared to EfficientNet-B3; however, the inclusion of the auxiliary classification allowed to increase the accuracy for both in most cases. By averaging the results over the five datasets, we observed that the first network yielded 94.08% and 94.58% and the second network yielded 93.74% and 94.74% without and with an auxiliary classification loss, respectively. EfficientNets yielded slightly better accuracies compared to the inception-based ones. Yet our proposed solution for all four networks yielded better results compared to state-of-the-art methods with low computational costs, which will be discussed in the next section.

Table 8. OA (%) bbtained by fine-tuning EfficientNet without and with an auxiliary classifier.

	EfficeintNet-B0		**EfficeintNet-B3**	
Dataset	**Without Auxiliary**	**With Auxiliary**	**Without Auxiliary**	**With Auxiliary**
Merced (Train 50%)	97.69 ± 0.41	98.01 ± 0.45	97.33 ± 0.48	98.22 ± 0.49
AID (Train 20%)	93.61 ± 0.27	93.69 ± 0.11	92.64 ± 0.24	94.19 ± 0.15
NWPU (Train 10%)	89.83 ± 0.15	89.96 ± 0.27	89.46 ± 0.17	91.08 ± 0.14
Optimal-31 (Train 80%)	92.58 ± 0.92	93.97 ± 0.13	93.92 ± 0.73	94.51 ± 0.75
KSA (Train 20%)	95.71 ± 0.31	96.26 ± 0.35	95.35 ± 0.76	96.29 ± 0.49
Average	94.08 ± 0.26	94.58 ± 0.24	93.74 ± 0.47	94.85 ± 0.40

6. Discussions

In this section, we compare the results of our method with the state-of-the-art results reported so far in the literature of RS. Tables 9–12 show detailed comparisons for Merced, AID, NWPU, and Optimal-31 datasets, respectively. The training–testing set splits were different from one dataset to another depending on which splits were reported in the literature. In these comparisons, we reported the OA for the second configuration of the classification loss (Conv+Softmax). We termed our method in the different tables as GoogLeNet-aux, Inception-v3-aux, EfficientNet-B0-aux, and EfficientNet-B3-aux. As can be seen, our results obtained for GoogLeNet-aux were better (up to 10% difference for the AID and Optimal-31 datasets) than similar fine-tuning results reported in the literature using GoogLeNet, which confirmed clearly that this network has not been exploited in an appropriate way. It was also impressive to see that fine-tuning the different networks supplemented with an auxiliary classifier settled new state-of-the-art results compared to recent methods with complex feature engineering. For example, the recent state-of-the-art method in [5], based on VGG16 coupled with a complex module based on bidirectional LSTM, achieved 98.57% and 97.05% for Merced with 80% and 50% training ratios, respectively. Our proposed method yielded better results for the four networks, and the best result was achieved by EfficientNetB3-aux with 99.09% and 98.22% for 80% and 50% training ratios, respectively.

Table 9. Comparison with state-of-the-art methods for the Merced dataset.

Method	**80% Train**	**50% Train**
ARCNet-VGG16 [11]	99.12 ± 0.40	96.81 ± 0.14
VGG16+MSCP [39]	98.36 ± 0.58	—
Siamese ResNet50+RD [40]	94.50	91.71
OverfeatL+IFK [41]	98.91	—
Triplet networks [13]	97.99 ± 0.53	—
MCNN [23]	96.66 ± 0.90	
GoogLeNet+SVM [35]	94.31 ± 0.89	92.70 ± 0.60
AlexNet [42]	95.00 ± 1.74	-
VGG16+IFK [25]	98.57 ± 0.34	
D-DSML-CaffeNet [24]	95.76 ± 1.70	
ResNet [42]	97.19 ± 0.57	
Fusion by addition [30]	97.42 ± 1.79	
VGG16+EMR [28]	98.14	
Fine-tuning VGG16 [5]	97.14 ± 0.48	96.57 ± 0.38
GBNet [5]	96.90 ± 0.23	95.71 ± 0.19
GBNet+global feature [5]	98.57 ± 0.48	97.05 ± 0.19
Fine-tuning GoogLeNet [6]	97.10	—
Inception-v3-aux [ours]	98.80 ± 0.26	97.63 ± 0.20
GoogLeNet-aux [ours]	99.00 ± 0.46	97.90 ± 0.34
EfficientNet-B0-aux [ours]	99.04 ± 0.33	98.01 ± 0.45
EfficientNet-B3-aux [ours]	99.09 ± 0.17	98.22 ± 0.49

Table 10. Comparison with state-of-the-art methods for the AID dataset.

Method	50% Train	20% Train
ARCNet-VGG16 [11]	93.10 ± 0.55	88.75 ± 0.40
VGG16+MSCP [39]	94.42 ± 0.17	91.52 ± 0.21
MCNN [23]	91.80 ± 0.22	—
Fusion by addition [30]	91.87 ± 0.36	
Multilevel fusion [25]	95.36 ± 0.22	
VGG16 (fine-tuning) [5]	93.60 ± 0.64	89.49 ± 0.34
GBNet+global feature [5]	95.48 ± 0.12	92.20 ± 0.23
GoogLeNet+SVM [35]	86.39 ± 0.55	83.44 ± 0.40
CaffeNet [35]	89.53 ± 0.31	86.86 ± 0.47
VGG16 [35]	89.64 ± 0.36	86.59 ± 0.29
Inception-v3-aux [ours]	95.64 ± 0.20	93.52 ± 0.21
GoogLeNet-aux [ours]	95.54 ± 0.12	93.25 ± 0.33
EfficientNet-B0-aux [ours]	96.17 ± 0.16	93.69 ± 0.11
EfficientNet-B3-aux [ours]	96.56 ± 0.14	94.19 ± 0.15

Table 11. Comparison with state-of-the-art methods for the NWPU dataset.

Method	10% Train	20% Train
VGG16+MSCP [39]	85.33 ± 0.17	88.93 ± 0.14
Triplet networks [13]	—	92.33 ± 0.20
Fine-tuning VGG16 [10]	87.15 ± 0.45	90.36 ± 0.18
Fine-tuning GoogLeNet [10]	82.57 ± 0.12	86.02 ± 0.18
Inception-v3-aux [ours]	89.32 ± 0.33	92.18 ± 0.11
GoogLeNet-aux [ours]	89.22 ± 0.25	91.63 ± 0.11
EfficientNet-B0-aux [ours]	89.96 ± 0.27	92.89 ± 0.16
EfficientNet-B3-aux [ours]	91.08 ± 0.14	93.81 ± 0.07

Table 12. Comparison with state-of-the-art methods for the Optimal-31 dataset.

Method	80% Train
ARCNet-VGG16 [11]	92.70 ± 0.35
ARCNet-AlexNet [11]	85.75 + 0.35
ARCNet-ResNet [11]	91.28 + 0.45
Fine-tuning GoogLeNet [11]	82.57 ± 0.12
Fine-tuning VGG16 [11]	87.45 ± 0.45
Fine-tuning AlexNet [11]	81.22 ± 0.19
VGG16 [35]	89.12 ± 0.35
Fine-tuning VGG16 [5]	89.52 ± 0.26
GBNet [5]	91.40 ± 0.27
GBNet+global feature [5]	93.28 ± 0.27
Inception-v3-aux [ours]	94.13 ± 0.35
GoogLeNet-aux [ours]	93.11 ± 0.55
EfficientNet-B0-aux [ours]	93.97 ± 0.13
EfficientNet-B3-aux [ours]	94.51 ± 0.75

For further analysis, we carried out another experiment using data augmentation techniques. To this end, we augmented the datasets during the training phase by flipping the images vertically and horizontally. Table 13 reports the new accuracies obtained for EfficientNetB3-aux-aug with 100% augmentation. That is, we augmented each datasets by adding a set of images to the original training set by randomly flipping horizontally or vertically each image in the training set. This operation was done online for each mini-batch during the training phase. It is worth noting that more advanced augmentation methods could be adopted as well. As can be seen from Table 13, augmentation improved the accuracy and increased the computation cost.

Table 13. Classification results obtained without and with data augmentation (100%) using simple vertical and horizontal flips of the images.

	Merced	**Optimal-31**	**AID**
	50% Train	80% Train	20% Train
EfficientNet-B3-aux	98.22 ± 0.49	94.51 ± 0.75	94.19 ± 0.15
	14 minutes	20 minutes	27 minutes
EfficientNet-B3-aux-aug	98.38 ± 0.30	95.26 ± 0.46	95.56 ± 0.23
	42 minutes	46 minutes	1 hour

For additional comparison purposes, we carried out other experiments for EffientNet-B3 using feature extraction techniques. We froze the network weights and ran on the top three types of classifiers: a simple softmax, one fully-connected layer plus softmax, and two fully-connected layers plus softmax. The results, reported in Table 14, confirmed the superiority of the proposed fine-tuning method.

Table 14. Comparison with feature extraction methods using EfficientNet-B3: Freeze the weights of the pre-trained CNN and train an external classifier.

	Merced 50% Train	**Optimal-31 80% Train**
Softmax	93.90 ± 0.62	85.86 ± 1.67
Dense(128)+Softmax	94.34 ± 0.46	85.06 ± 1.70
Dense(128)+Dense(128)+Softmax	94.03 ± 0.71	86.02 ± 1.57
EfficientNet-B0-aux	98.01 ± 0.45	94.51 ± 0.75

Finally, to further assess the performance of the proposed fine-tuning method, we investigated an additional network called DenseNet [43]. This network (Figure 12) relies on the idea of connecting each layer to every other layer in a feed-forward fashion. That is for each layer, the feature maps of all preceding layers are used as inputs, while its own feature maps are used as inputs into all the subsequent layers. In this work, we placed the auxiliary classification loss on the top of the "pool4_conv" layer of DenseNet169 (with an output of 16 × 16 × 640) and the main softmax classification layer on the top of network output of dimension 8 × 8 × 1664. This network was fine-tuned by using the RMSprop optimizer with the parameters given in Table 4 without/with an auxiliary loss function and with data augmentation as for EfficientNet-B3. Table 15 shows the classification results for this network. These results again confirmed the promising capabilities of the fine-tuning approach when carried out in an appropriate way.

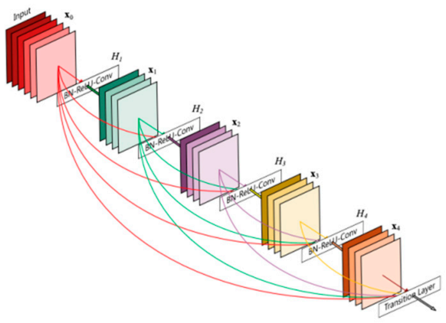


Figure 12. Example of a five-layer dense block, where each layer takes all preceding feature input maps as input [43].

Table 15. Classification reults obtained by fine-tuining the DenseNet169 network.

	Merced	**Optimal-31**	**AID**
	50% Train	80% Train	20% Train
DensNets169	98.15 ± 0.35 11 minutes	93.76 ± 1.06 15 minutes	94.38 ± 0.26 20 minutes
DensNets169-aux	98.22 ± 0.24 17 minutes	94.67 ± 0.31 19 minutes	94.88 ± 0.19 30 minutes
DensNets169-aux-aug	98.64 ± 0.33 48 minutes	95.37 ± 0.69 44 minutes	95.82 ± 0.12 1.20 hours

7. Conclusions

In this paper, we presented a simple yet effective method to combat the vanishing gradient problem for deeper CNNs with relatively small weights. We showed that supplementing the network with an additional auxiliary classification loss placed on the top of a hidden layer and the utilization of an appropriate optimization method can produce state-of-the-art results in terms of OA and convergence times compared to the complex methods based on complex feature extraction techniques. For future developments, we suggest to: (1) assess the method for other types of deeper networks, (2) investigate alternative solutions for combatting the vanishing gradient problem, and (3) improve the results by optimizing the CNN architecture by pruning redundant layers irrelevant to the classification task.

Author Contributions: Y.B. designed, implemented the method, and wrote the paper. M.M.A.R., H.A., and N.A. contributed to the analysis of the experimental results and paper writing.

Funding: This work was supported by the Deanship of Scientific Research at King Saudi University through the Local Research Group Program under Project RG-1435-055.

Acknowledgments: This work was supported by the Deanship of Scientific Research at King Saudi University.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.E.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; Rabinovich, A. Going deeper with convolutions. In Proceedings of the 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Boston, MA, USA, 7–12 June 2015; pp. 1–9.
2. Ioffe, S.; Szegedy, C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In Proceedings of the International Conference on Machine Learning, Lille, France, 6–11 July 2015; pp. 448–456.
3. Szegedy, C.; Vanhoucke, V.; Ioffe, S.; Shlens, J.; Wojna, Z. Rethinking the Inception Architecture for Computer Vision. In Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 27–30 June 2016; pp. 2818–2826.
4. Szegedy, C.; Ioffe, S.; Vanhoucke, V.; Alemi, A. Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. *arXiv* **2016**, arXiv:1602.07261.
5. Sun, H.; Li, S.; Zheng, X.; Lu, X. Remote Sensing Scene Classification by Gated Bidirectional Network. *IEEE Trans. Geosci. Remote Sens.* **2019**, 1–15. [CrossRef]
6. Castelluccio, M.; Poggi, G.; Sansone, C.; Verdoliva, L. Land Use Classification in Remote Sensing Images by Convolutional Neural Networks. *arXiv* **2015**, arXiv:1508.00092.
7. Cheng, G.; Yang, C.; Yao, X.; Guo, L.; Han, J. When Deep Learning Meets Metric Learning: Remote Sensing Image Scene Classification via Learning Discriminative CNNs. *IEEE Trans. Geosci. Remote Sens.* **2018**, *56*, 2811–2821. [CrossRef]
8. Nogueira, K.; Penatti, O.A.B.; Santos, J.A. Towards better exploiting convolutional neural networks for remote sensing scene classification. *Pattern Recognit.* **2017**, *61*, 539–556. [CrossRef]
9. Boualleg, Y.; Farah, M.; Farah, I.R. Remote Sensing Scene Classification Using Convolutional Features and Deep Forest Classifier. *IEEE Geosci. Remote Sens. Lett.* **2019**, *16*, 1944–1948. [CrossRef]

10. Cheng, G.; Han, J.; Lu, X. Remote Sensing Image Scene Classification: Benchmark and State of the Art. *Proc. IEEE* **2017**, *105*, 1865–1883. [CrossRef]
11. Wang, Q.; Liu, S.; Chanussot, J.; Li, X. Scene Classification With Recurrent Attention of VHR Remote Sensing Images. *IEEE Trans. Geosci. Remote Sens.* **2018**, *57*, 1155–1167. [CrossRef]
12. Singh, P.; Komodakis, N. Improving Recognition of Complex Aerial Scenes Using a Deep Weakly Supervised Learning Paradigm. *IEEE Geosci. Remote Sens. Lett.* **2018**, *15*, 1932–1936. [CrossRef]
13. Liu, Y.; Huang, C. Scene Classification via Triplet Networks. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2018**, *11*, 220–237. [CrossRef]
14. Wu, H.; Liu, B.; Su, W.; Zhang, W.; Sun, J. Deep Filter Banks for Land-Use Scene Classification. *IEEE Geosci. Remote Sens. Lett.* **2016**, *13*, 1895–1899. [CrossRef]
15. 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. [CrossRef]
16. Cheng, G.; Li, Z.; Yao, X.; Li, K.; Wei, Z. Remote Sensing Image Scene Classification Using Bag of Convolutional Features. *IEEE Geosci. Remote Sens. Lett.* **2017**, *14*, 1735–1739. [CrossRef]
17. Othman, E.; Bazi, Y.; Alajlan, N.; Alhichri, H.; Melgani, F. Using convolutional features and a sparse autoencoder for land-use scene classification. *Int. J. Remote Sens.* **2016**, *37*, 2149–2167. [CrossRef]
18. Weng, Q.; Mao, Z.; Lin, J.; Guo, W. Land-Use Classification via Extreme Learning Classifier Based on Deep Convolutional Features. *IEEE Geosci. Remote Sens. Lett.* **2017**, *14*, 704–708. [CrossRef]
19. Liu, Q.; Hang, R.; Song, H.; Li, Z. Learning Multiscale Deep Features for High-Resolution Satellite Image Scene Classification. *IEEE Trans. Geosci. Remote Sens.* **2018**, *56*, 117–126. [CrossRef]
20. 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. [CrossRef]
21. Alhichri, H.; Alajlan, N.; Bazi, Y.; Rabczuk, T. Multi-Scale Convolutional Neural Network for Remote Sensing Scene Classification. In Proceedings of the 2018 IEEE International Conference on Electro/Information Technology (EIT), Rochester, MI, USA, 3–5 May 2018; pp. 1–5.
22. Wang, J.; Liu, W.; Ma, L.; Chen, H.; Chen, L. IORN: An Effective Remote Sensing Image Scene Classification Framework. *IEEE Geosci. Remote Sens. Lett.* **2018**, *15*, 1695–1699. [CrossRef]
23. Liu, Y.; Zhong, Y.; Qin, Q. Scene Classification Based on Multiscale Convolutional Neural Network. *IEEE Trans. Geosci. Remote Sens* **2018**, *56*, 7109–7121. [CrossRef]
24. Gong, Z.; Zhong, P.; Yu, Y.; Hu, W. Diversity-Promoting Deep Structural Metric Learning for Remote Sensing Scene Classification. *IEEE Trans. Geosci. Remote Sens.* **2018**, *56*, 371–390. [CrossRef]
25. Yu, Y.; Liu, F. Aerial Scene Classification via Multilevel Fusion Based on Deep Convolutional Neural Networks. *IEEE Geosci. Remote Sens. Lett.* **2018**, *15*, 287–291. [CrossRef]
26. Liu, Y.; Liu, Y.; Ding, L. Scene Classification Based on Two-Stage Deep Feature Fusion. *IEEE Geosci. Remote Sens. Lett.* **2018**, *15*, 183–186. [CrossRef]
27. Marmanis, D.; Datcu, M.; Esch, T.; Stilla, U. Deep Learning Earth Observation Classification Using ImageNet Pretrained Networks. *IEEE Geosci. Remote Sens. Lett.* **2016**, *13*, 105–109. [CrossRef]
28. Wang, G.; Fan, B.; Xiang, S.; Pan, C. Aggregating Rich Hierarchical Features for Scene Classification in Remote Sensing Imagery. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2017**, *10*, 4104–4115. [CrossRef]
29. Li, E.; Xia, J.; Du, P.; Lin, C.; Samat, A. Integrating Multilayer Features of Convolutional Neural Networks for Remote Sensing Scene Classification. *IEEE Trans. Geosci. Remote Sens.* **2017**, *55*, 5653–5665. [CrossRef]
30. 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. [CrossRef]
31. Hasanpour, S.H.; Rouhani, M.; Fayyaz, M.; Sabokrou, M. Lets keep it simple, Using simple architectures to outperform deeper and more complex architectures. *arXiv* **2016**, arXiv:1608.06037.
32. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep Residual Learning for Image Recognition. In Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 27–30 June 2016; pp. 770–778.
33. Tan, M.; Le, Q.V. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. *arXiv* **2019**, arXiv:1905.11946.
34. Yang, Y.; Newsam, S. Bag-of-visual-words and Spatial Extensions for Land-use Classification. In Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems, San Jose, CA, USA, 2–5 November 2010; ACM: New York, NY, USA, 2010; pp. 270–279.

35. Xia, G.; 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. [CrossRef]
36. Othman, E.; Bazi, Y.; Melgani, F.; Alhichri, H.; Alajlan, N.; Zuair, M. Domain Adaptation Network for Cross-Scene Classification. *IEEE Trans. Geosci. Remote Sens.* **2017**, *55*, 4441–4456. [CrossRef]
37. Othman, E.; Bazi, Y.; Alhichri, H. Remote_Sensing_Dataset-Google Drive. Available online: http://bit.ly/ksa_dataset (accessed on 5 May 2019).
38. Ullmann, J.R. Experiments with the n-tuple Method of Pattern Recognition. *IEEE Trans. Comput.* **1969**, *100*, 1135–1137. [CrossRef]
39. He, N.; Fang, L.; Li, S.; Plaza, A.; Plaza, J. Remote Sensing Scene Classification Using Multilayer Stacked Covariance Pooling. *IEEE Trans. Geosci. Remote Sens.* **2018**, *56*, 6899–6910. [CrossRef]
40. Zhou, Y.; Liu, X.; Zhao, J.; Ma, D.; Yao, R.; Liu, B.; Zheng, Y. Remote sensing scene classification based on rotation-invariant feature learning and joint decision making. *J. Image Video Proc.* **2019**, *2019*, 3. [CrossRef]
41. Yang, Z.; Mu, X.; Zhao, F. Scene classification of remote sensing image based on deep network and multi-scale features fusion. *Optik* **2018**, *171*, 287–293. [CrossRef]
42. Liang, Y.; Monteiro, S.T.; Saber, E.S. Transfer learning for high resolution aerial image classification. In Proceedings of the 2016 IEEE Applied Imagery Pattern Recognition Workshop (AIPR), Washington, DC, USA, 18–20 October 2016; pp. 1–8.
43. Huang, G.; Liu, Z.; Van Der Maaten, L.; Weinberger, K.Q. Densely Connected Convolutional Networks. In Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 21–26 July 2017; pp. 2261–2269.