Image Classification using Keras

[1]Argha Chatterjee, [1]Bidesh Roy, [1]Koushik Majumder, [1]Roni Mondal, [1]Shamim Ahmed
[1]Data Scientist, [1]Software Engineer, [1]Project Manager, [1]Electronics Engineer, [1]Data Analyst

Abstract— **We now live in an era of Artificial Intelligence and the Internet of Things, and our devices generate a huge amount of data. Most of this data is in the form of photos, videos, and images. Most of these image data obtained from cameras and censors are unstructured. Hence, we must depend on lots of modern techniques of machine learning and artificial intelligence to analyze these data and make efficient use of them. Image classification comes as an effective method to analyze and interpret these unstructured image data. Based on some specific rules image classification tries to categorize by assigning labels to groups or pixels or vectors present in an image. Its application can be in various fields such as medical imaging, satellite imagery, traffic control systems, computer vision, and many more.**

Recently it has also become a very useful technology for remote sensing. Earlier the resolution of the images obtained from satellites would be very low, and the pixels would not be clear hence it would be difficult to analyze them. Hence researchers also thought of Object-based analysis which broke down the image into meaningful components from the scene that distinguishes the image from others. The scientists discovered that object-based analysis is better compared to the previous method of using pixels. With the advent of machine learning even semantic methods where semantic level classes were defined for the images obtained from remote send=sing and segregated into airplanes, forests, grassland, water bodies, etc.

Deep learning methods made the process faster and simpler as it helps in analyzing and interpreting large amounts of data easily. Its multiple processing layers help to understand more features from the data along with maintaining high-level abstraction. A convoluted Neural Network is a type of deep learning method that can assign various importance to aspects of the image which are also objects and differentiate one from the other. CNN requires lesser preprocessing compared to all other classification models. It is analogous to the neural network of the human brain. In this research, we have trained the CNN model using the train and validation dataset and once it is trained, we use it to predict the images under the prediction folder having images of classes buildings, forests, glaciers, mountains, sea, and street.

Index Terms—**image processing, keras, deep learning, CNN.** *(key words)*

I. INTRODUCTION (HEADING 1)

Image classification is one of the significant parts of digital image analysis. It can find applications in healthcare, marketing, transportation, e-commerce, and many other industries. We can do image classification using both supervised and unsupervised learning techniques. In this project, we make use of Keras which is an open-source Python library that can be used to develop and evaluate deep learning models. Keras was developed by Google for implementing neural networks and is written in Python. It makes multiple backend neural network computations. Keras has TensorFlow running in the backend and can be used for deep learning as it has predefined modules for all types of Neural network computations. Using Keras we initialize the model, add convolutional layers, and max-pooling for feature extraction, flatten the convolutional layers and add a deep neural network to the flattened dataset The output layer has multiple classes. 'Categorical_crossentropy' is being used to calculate the loss while compiling the model for some reason (multiple classes).

For building the model in Keras we applied the following steps:

Defining the network: The different layers and the connections of the model are defined. The two main models are Sequential and Functional models. One can select the type needed and then define the data flow.Network compilation: In the Keras model.compile() method compiles the code for the machine to understand. To compile the code we need to define three functions, one is the loss to calculate the losses in our model, the second is an optimizer that reduces the loss in the model, and a metrics to identify the accuracy. Fitting the Network: This is used to fit the model and the data to train the model on the data.Evaluating the Network: The error in the model is evaluated in thisMaking Predictions: model. predict() function is used to predict new data using the same model.

Literature review

Neha Jain, Vibhor Jain and Anju Mishra did a similar project to analyze the performance of popular convolutional neural networks in detecting objects in real time video feeds. It was focused on analyzing the performance of three networks Alex Net, Google Net, ResNet50. The results obtained proved that Google Net, ResNet50 were able to detect objects better as compared to Alex Net.

- Mingyuan Xin and Yong Wang used error back propagation algorithm to propose innovative training criterion of DNN for maximum interval and minimum classification error. The obtained results showed that M3 CE can enhance cross entropy and M3 CE-CEc obtained better results in both MNIST and CIFAR-10 databases.
- Deepika Jaiswal, Sowmya Vishvanathan and Soman Kp used CNN in image classification in remote sensing data of aerial images and scene images form SUN database. The algorithm is evaluated on quality metric Mean Square Error and classification accuracy.
- P.Deepan and LR Sudha in their work titled Object classification of remote sensing image using Deep CNN discusses the visual geometry group deep CNN for scene classification, provide detailed description about benchmark data and describe the experimental results and analysis.

II. UNDERSTANDING THE DATA

We have secured a dataset of ~25k images from a wide range of natural scenes from all around the world. Our task is to identify which kind of scene can the image be categorized into. We are going to classify six different categories of Images:

-0: buildings
- 1: forests
- 2: glaciers
- 3. Mountains
- 4. Sea
- 5. Street

Knowing the image dimensions for :
BUILDINGS image-set

In [8]:

```
def buildings_statistics():
    heights = []
    widths = []
    img_count = 0
    for img in os.listdir(buildings_path):
        path = os.path.join(buildings_path, img)
        if "DS_Store" not in path:
            data = np.array(Image.open(path))
            heights.append(data.shape[0])
            widths.append(data.shape[1])
            img_count += 1
    avg_height = sum(heights) / len(heights)
    avg_width = sum(widths) / len(widths)
    print("Average Height: " + str(avg_height))
    print("Max Height: " + str(max(heights)))
    print("Min Height: " + str(min(heights)))
    print('\n')
    print("Average Width: " + str(avg_width))
    print("Max Width: " + str(max(widths)))
    print("Min Width: " + str(min(widths)))
buildings_statistics()
```

```
Average Height: 149.9881332724783
Max Height: 150
Min Height: 124


Average Width: 150.0
Max Width: 150
Min Width: 150
```

Forest Image Set:

In [9]:

```
def forest_statistics():
    heights = []
    widths = []
    img_count = 0
    for img in os.listdir(forest_path):
        path = os.path.join(forest_path, img)
        if "DS_Store" not in path:
            data = np.array(Image.open(path))
            heights.append(data.shape[0])
            widths.append(data.shape[1])
            img_count += 1
    avg_height = sum(heights) / len(heights)
    avg_width = sum(widths) / len(widths)
    print("Average Height: " + str(avg_height))
    print("Max Height: " + str(max(heights)))
    print("Min Height: " + str(min(heights)))
    print('\n')
    print("Average Width: " + str(avg_width))
    print("Max Width: " + str(max(widths)))
    print("Min Width: " + str(min(widths)))
forest_statistics()
```

```
Average Height: 149.89476001761338
Max Height: 150
Min Height: 108


Average Width: 150.0
Max Width: 150
Min Width: 150
```

Glacier Image Set

In [10]:

```
def glacier_statistics():
    heights = []
    widths = []
    img_count = 0
    for img in os.listdir(glacier_path):
        path = os.path.join(glacier_path, img)
        if "DS_Store" not in path:
            data = np.array(Image.open(path))
            heights.append(data.shape[0])
            widths.append(data.shape[1])
            img_count += 1
    avg_height = sum(heights) / len(heights)
    avg_width = sum(widths) / len(widths)
    print("Average Height: " + str(avg_height))
    print("Max Height: " + str(max(heights)))
    print("Min Height: " + str(min(heights)))
    print('\n')
    print("Average Width: " + str(avg_width))
    print("Max Width: " + str(max(widths)))
    print("Min Width: " + str(min(widths)))
glacier_statistics()
```

```
Average Height: 149.78535773710482
Max Height: 150
Min Height: 76


Average Width: 150.0
Max Width: 150
Min Width: 150
```

III. METHODOLOGY

Building CNN Model

Keras is being used to create the model. Model building comprises of:

initializing the model

- adding convolutional layers and maxpooling for feature extraction. 4 convolution layers are generated with different no. of filters under each convolution layer
- flattening the convolutional layers
- adding deep neural network to the flattened dataset; 4 hidden layers and 1 output layer is generated
- The output layer has multiple classes, thus 'softmax' function is being used. Also 'categorical_crossentropy' is being used to calculate the loss while compiling the model for same reason (multiple classes)

In [12]:

```
model = Sequential()

model.add(Conv2D(32, kernel_size = (3, 3), activation='relu', input_shape=(img_height,
img_width, 3)))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(BatchNormalization())
model.add(Dropout(0.2))

model.add(Conv2D(64, kernel_size=(3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(BatchNormalization())
model.add(Dropout(0.2))

model.add(Conv2D(128, kernel_size=(3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(BatchNormalization())
model.add(Dropout(0.2))

model.add(Conv2D(256, kernel_size=(3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(BatchNormalization())
model.add(Dropout(0.2))

model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.2))

model.add(Dense(6, activation = 'softmax'))

model.summary()

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics = ['accuracy'
])
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 148, 148, 32)      896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 74, 74, 32)        0
_________________________________________________________________
batch_normalization (BatchNo (None, 74, 74, 32)        128
_________________________________________________________________
dropout (Dropout)            (None, 74, 74, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 72, 72, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 36, 36, 64)        0
_________________________________________________________________
batch_normalization_1 (Batch (None, 36, 36, 64)        256
_________________________________________________________________
dropout_1 (Dropout)          (None, 36, 36, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 34, 34, 128)       73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 17, 17, 128)       0
_________________________________________________________________
batch_normalization_2 (Batch (None, 17, 17, 128)       512
_________________________________________________________________
dropout_2 (Dropout)          (None, 17, 17, 128)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 15, 15, 256)       295168
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 7, 7, 256)         0
_________________________________________________________________
batch_normalization_3 (Batch (None, 7, 7, 256)         1024
_________________________________________________________________
dropout_3 (Dropout)          (None, 7, 7, 256)         0
_________________________________________________________________
flatten (Flatten)            (None, 12544)             0
_________________________________________________________________
dense (Dense)                (None, 128)               1605760
_________________________________________________________________
dropout_4 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               16512
_________________________________________________________________
dropout_5 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               16512
_________________________________________________________________
dropout_6 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_3 (Dense)              (None, 128)               16512
_________________________________________________________________
dropout_7 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_4 (Dense)              (None, 6)                 774
=================================================================
Total params: 2,046,406
Trainable params: 2,045,446
Non-trainable params: 960
_________________________________________________________________
```

Image Pre-processing and Augmentation

Train dataset is being augmented using flip and zoom techniques. This is being done to avoid overfitting of model. Train and Validation generators are created using dataset from directories (folders).

```
In [13]:
DIR = 'F:/Data Science/Python/Python Projects/intel-image-classification'

train_datagen = ImageDataGenerator(
        rescale=1./255,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True)

test_datagen = ImageDataGenerator(rescale=1./255)


train_generator = train_datagen.flow_from_directory(
        DIR + '/seg_train',
        target_size=(img_height, img_width),
        batch_size=train_batch_size,
        class_mode='categorical',
        seed=777,
        shuffle=True)

validation_generator = test_datagen.flow_from_directory(
        DIR + '/seg_test',
        target_size=(img_height, img_width),
        batch_size=test_batch_size,
        class_mode='categorical',
        seed=777,
        shuffle=True)
```

```
Found 14034 images belonging to 6 classes.
Found 3000 images belonging to 6 classes.
```

Checking local device used by system, GPU is preferred for image processing

The no. of cores in GPU is high vis-a-vis CPU and thus GPU is preferred for CNN processing CUDA package has been installed to ensure that keras uses GPU by default, if available

In [15]:

```
from tensorflow.python.client import device_lib
print(device_lib.list_local_devices())
```

```
[name: "/device:CPU:0"
device_type: "CPU"
memory_limit: 268435456
locality {
}
incarnation: 13872645089973790120
, name: "/device:GPU:0"
device_type: "GPU"
memory_limit: 3141979340
locality {
  bus_id: 1
  links {
  }
}
incarnation: 16332711329353266311
physical_device_desc: "device: 0, name: GeForce GTX 1050 Ti, pci bus id: 0
000:01:00.0, compute capability: 6.1"
]
```

Fitting CNN model to image datasets taken from training and validation directories

In [16]:

```
step_train = train_generator.n//train_generator.batch_size
step_valid = validation_generator.n//validation_generator.batch_size

history = model.fit_generator(generator = train_generator,
                    steps_per_epoch = step_train,
                    validation_data = validation_generator,
                    validation_steps = step_valid,
                    epochs=epochs,
                    verbose=1)
```

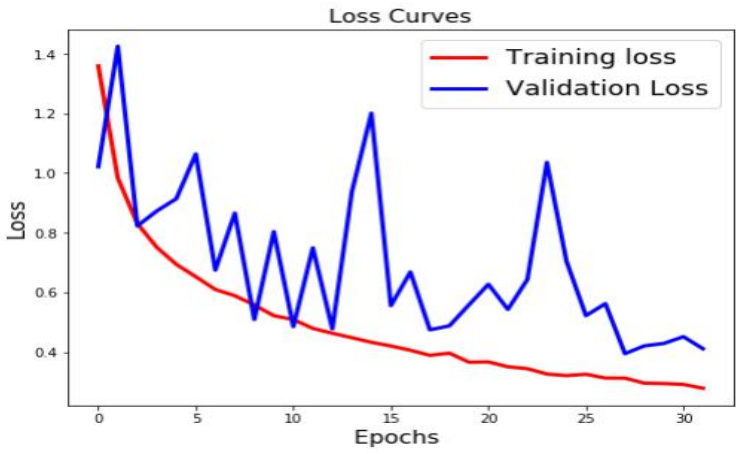
Visualising Model Results:
Plotting the Loss Curves

In [17]:

```
plt.figure(figsize=[8,6])
plt.plot(history.history['loss'],'r',linewidth=3.0)
plt.plot(history.history['val_loss'],'b',linewidth=3.0)
plt.legend(['Training loss', 'Validation Loss'],fontsize=18)
plt.xlabel('Epochs ',fontsize=16)
plt.ylabel('Loss',fontsize=16)
plt.title('Loss Curves',fontsize=16)
```

Out[17]:

```
Text(0.5, 1.0, 'Loss Curves')
```



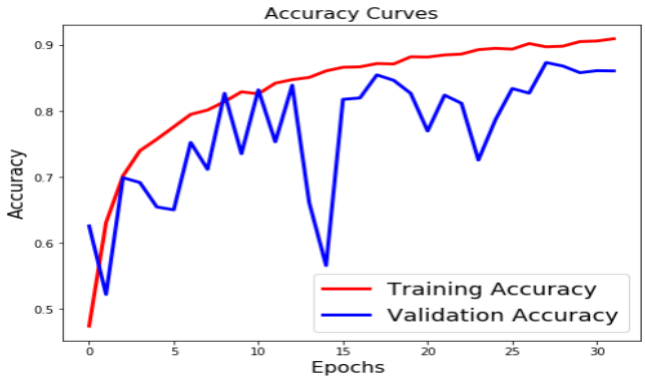
Plotting the Accuracy Curves

In [18]:

```
plt.figure(figsize=[8,6])
plt.plot(history.history['accuracy'],'r',linewidth=3.0)
plt.plot(history.history['val_accuracy'],'b',linewidth=3.0)
plt.legend(['Training Accuracy', 'Validation Accuracy'],fontsize=18)
plt.xlabel('Epochs ',fontsize=16)
plt.ylabel('Accuracy',fontsize=16)
plt.title('Accuracy Curves',fontsize=16)
```

Out[18]:

```
Text(0.5, 1.0, 'Accuracy Curves')
```



Saving the model for future use

In [21]:

```
model.save('project_model.h7')
```

```
WARNING:tensorflow:From C:\Users\USER\Anaconda3\lib\site-packages\tensorfl
ow_core\python\ops\resource_variable_ops.py:1781: calling BaseResourceVari
able.__init__ (from tensorflow.python.ops.resource_variable_ops) with cons
traint is deprecated and will be removed in a future version.
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
INFO:tensorflow:Assets written to: project_model.h7\assets
```

Retrieving the saved model for predictions

In [22]:

```
from tensorflow.keras.models import load_model

model = load_model('project_model.h7')
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics = ['accuracy'
])
```

Predicting against the prediction image-set

In [23]:

```
images = []

for img in os.listdir(DIR + '/seg_pred'):
    img = os.path.join(DIR + '/seg_pred', img)
    img = image.load_img(img, target_size=(img_width, img_height))
    img = image.img_to_array(img)
    img = np.expand_dims(img, axis=0)
    img = img/255
    images.append(img)
```

In [24]:

```
# stack up images list to pass for prediction
images = np.vstack(images)
images.shape
```

Out[24]:

```
(7301, 150, 150, 3)
```

Compute predictions

In [25]:

```
predictions = model.predict(images)
y_pred = [np.argmax(probas) for probas in predictions]
```

Prediction results using random images

In [26]:

```
image_list=[]
n=10

for i in range(n):
    image_list.append(random.randint(0, len(y_pred)))

print("The Randomly selected Images are:", image_list)
```

```
The Randomly selected Images are: [2765, 277, 4176, 2200, 1073, 1597, 425
6, 1165, 2619, 1793]
```

IV. RESULTS

V. CONCLUSION

The model predictions are fairly accurate. However, there is always a possibility for further improvements by:

- Optimizing the hyper parameters, like batch size, epochs, etc.
- Tuning the no. of convolution and deep neural networks
- Fine tuning the dropouts to avoid overfitting of model.

In this project, we have created a model for scenic image classification. However, CNN can be using in many industrial and Research and Development, namely:

- Steel surface defect detection
- Concrete crack detection for buildings and structures
- Health research for disease detection, etc.

REFERENCES

[1] Deepika Jaswal, Sowmya.V, K.P.Soman, "Image Classification Using Convolutional Neural Networks" International Journal of Scientific & Engineering Research, Volume 5, Issue 6, June-2014ISSN 2229-5518J.
[2] 6. Kang, G.; Liu, H. Surface defects inspection of cold rolled strips based on neural network. In Proceedings of the 2005 International
[3] Conference on Machine Learning and Cybernetics(ICMLC 2005), Guangzhou, China, 18–21 August 2005; pp. 5034–5037.
[4] 7. Di, H.; Ke, X.; Peng, Z.; Zhou, D. Surface defect classification of steels with a new semi-supervised learning method. Opt. Laser.
[5] Eng. 2019, 117, 40–48. [CrossRef]
[6] 8. Schlegl, T.; Seeböck, P.; Waldstein, S.M.; Schmidt-Erfurth, U.; Langs, G. Unsupervised Anomaly Detection with Generative
[7] Adversarial Networks to Guide Marker Discovery. In Proceedings of the 25th International Conference on Information Processing
[8] in Medical Imaging (IPMI 2017), Boone, NC, USA, 25–30 June 2017; pp. 146–157.
[9] 9. Lee, S.Y.; Tama, B.A.; Moon, S.J.; Lee, S. Steel Surface Defect Diagnostics Using Deep Convolutional Neural Network and Class