Image Scene Classification

Project - Report:

Paul Bauriegel, Oliver Faber, Daniel Pavicic, Markus van Ballegooy

Contents

Project Challenge

One of the problems where deep learning excels is image classification. The goal in image classification is to classify a specific picture according to a set of possible categories. From a deep learning perspective, the image classification problem can be solved through transfer learning.

This project includes classifying images into one of six categories, using the pretrained Imagenet [1] model for transfer learning, as well as a comparison between the image classification results using this method and a traditional approach.

The final objective is to build a classifier that can distinguish images in different categories. The output of the system can be an input to the other modules in an image processing system.

To do that, some tasks needed to be accomplished:

1. Understand the content that is available in the dataset.

2. Pre-process the data to improve the final results.

3. Statistical image classifier

a. Build a statistical classifier to classify images into one of the categories.
b. Obtain statistics that can corroborate the results achieved and explain the reasons for these choices.

4. Deep neural classifier

a. Build a CNN (Convolutional Neural Network) based deep neural classifier to classify images into one of the categories.
b. Use the pretrained Imagenet model and fine-tune the weights in the CNN network based on the training images in the dataset.
c. Obtain some statistics that can corroborate the results obtained and explain the reasons for these choices

5. Use the different packages of visualization explained during the course to show sofindings from both approaches.

6. Compare both algorithms taking into account the outcomes and the statistics.

General Approach

In general, our approach to the challenge followed the steps given as tasks in the project briefing. We discussed our procedure in daily google meetings and chats. Decisions on the choice of a certain model, classifier or feature selection approach were taken by the group.

In particular, for the **statistical image classifier,** we chose a feature extraction along the vgg16 pretrained network and fed the 4096 extracted features into a support vector machine respectively a logistic regression. To use extracted features from a pretrained network seemed to be a reasonable approach as an alternative to a fully trained CNN.

Another, rather experimental version of a **statistical classifier** was provided with a “home made” RGB feature extractor which also included basic image features like edges and contrasts. These features were fed into three different classifiers. We were curious, if even basic image features can enable a classification into the given categories. If so, we might have found a rather “cheap” approach to image classification in terms of training effort and processing time.

Regarding the **neural networks,** we used a great variety of state of the art approaches starting from fully connected NNs and ending with CNNs combined in an ensemble setting. We tried to vary as many parameters (database for pretraining, fine tuning, augmentation) as possible to find the most powerful classification solution.

As an add-on, we developed an **image classification app**, which provides a “real time” - image classification based on the best classification model we found during our research.

Dataset Description & Preprocessing

Data Explorer

351.77 MB

- seg_pred
- seg_test
- seg_train

We got the Intel Image Classification Set[1].The dataset contains around 25 K Images of size 150 x 150) pixels. The Train, Test and Prediction data is separated in each zip file. There are around 14k images in Train, 3k in Test and 7k in Prediction.

The images data show Natural Scenes around the world.

There are 14034 images as train data and 3000 images as test data:

```
We have < 14034 > images in < TRAIN SET > (/content/seg_train/seg_train)
We have < 3000 > images in < TEST SET > (/content/seg_test/seg_test)
```

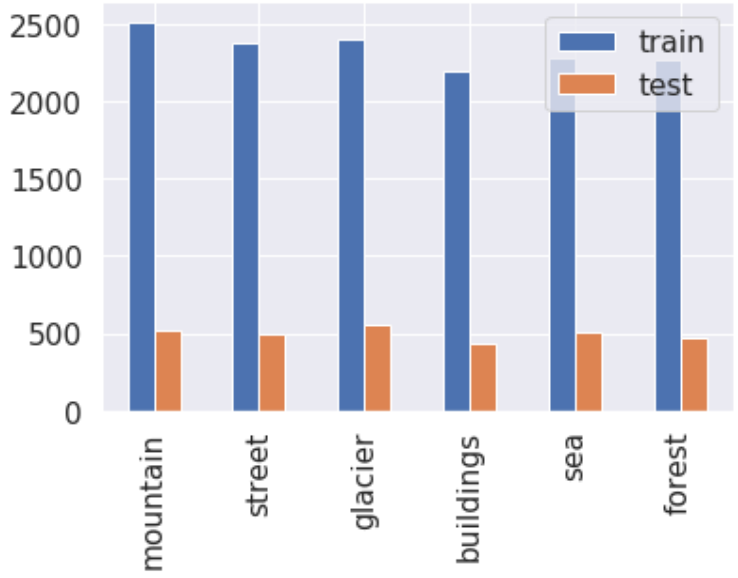
This data is distributed under 6 categories:

```
'buildings' -> 0
'forest' -> 1
'glacier' -> 2
'mountain' -> 3
'sea' -> 4
'street' -> 5
```

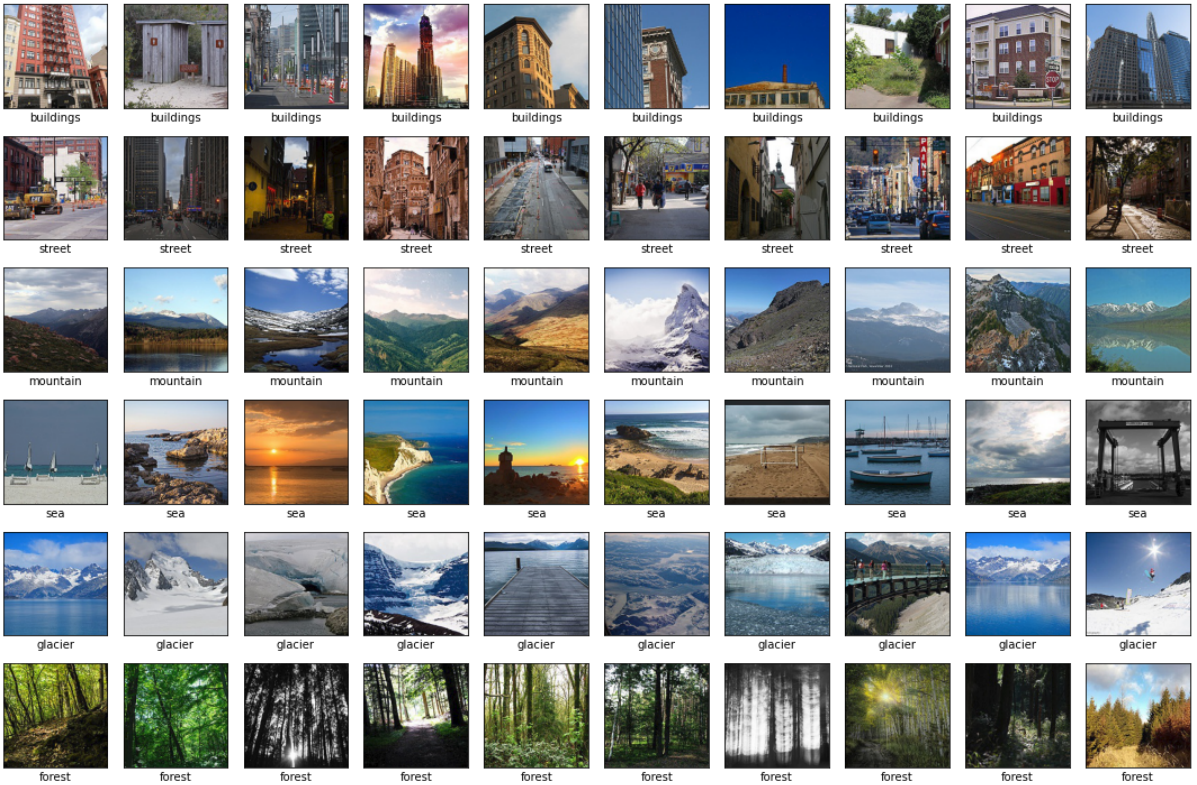
The data items have an implizit structure as follows:

Variable	Definition
image_name	Name of the image in the dataset (ID column)
label	Category of natural scene (target column)

Exploration / Distributions



[1] https://datahack.analyticsvidhya.com/contest/practice-problem-intel-scene-classification-challe



First examination of the datasets gave us the following insights.

- on first sight, there are no obviously mis-labeled data, with some rare exceptions, when a picture couldn't be assigned to any of the given classes)
- some of the pictures (e.g. mountains, glaciers , streets) show elements that can be placed into multiple categories (eg. street images show streets and often also buildings).

To estimate the impact of these *ambiguous labelings*, we tried to “re-cluster” the pictures based on 4096 vgg16 features with a kMeans clustering, forcing the algorithm to produce 6 clusters (possibly corresponding with the given 6 label classes.)

The resulting cluster solution shows that there is no clear correspondence between original labels and cluster membership at all. For a start, we didn’t further worry about this fact but rather trusted in the power of the stronger classification algorithms we were about to develop.

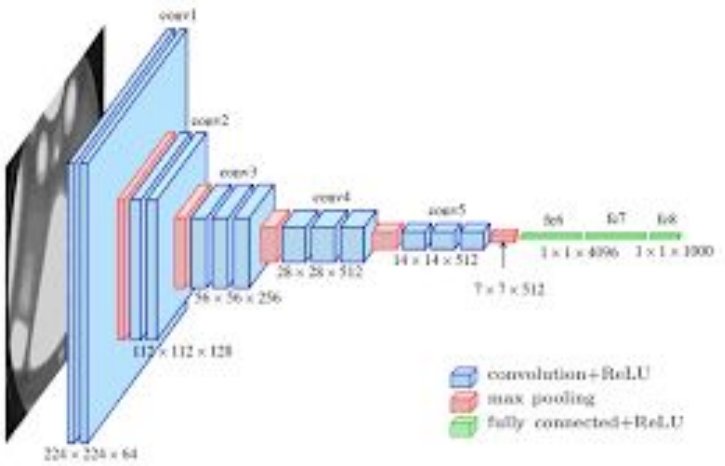
labls cluster	buildings	forest	glacier	mountain	sea	street	All
0	167	51	690	569	665	679	2821
1	602	717	6	602	636	0	2563
2	637	652	678	667	93	660	3387
3	0	803	0	0	0	0	803
4	651	280	675	301	464	1004	3375
5	341	0	646	645	645	304	2581
All	2398	2503	2695	2784	2503	2647	15530

Statistical Classifiers

The use of a “simple” statistical classifier for images can make sense, even if much more powerful methods (in terms of accuracy and versatility) are available. Using a reduced set of features and applying less training & processing time consuming models can produce - depending on the context requirements - satisfying results. For this reason, we ran two experiments with statistical classification using different kinds of image features.

Support Vector Machine Classifier with vgg16 net features

To set up a statistical classifier, we chose a support vector classifier (SVC) which we fitted on the 4096 features we obtained from a feature extraction using the feature vector predictions of the pretrained vgg16 network-




```
# load an image from file
image = load_img(k, target_size=(224, 224))
# convert the image pixels to a numpy array
image = img_to_array(image)
# reshape data for the model
image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
# prepare the image for the VGG model
image = preprocess_input(image)
# load model
model = VGG16()
# remove the output layer
model = Model(inputs=model.inputs, outputs=model.layers[-2].output)
# get extracted features
features = model.predict(image)
features = np.append(features, k)
bilderstapel_5000.append(features)
```

The results of that classification show a quite good test accuracy score of 91.08% for the SVC (training accuracy was 99.01%)
The confusion matrix also shows a quite efficient classification. Classification failures can be observed with images of mountains ⇔ glaciers and buildings ⇔ streets

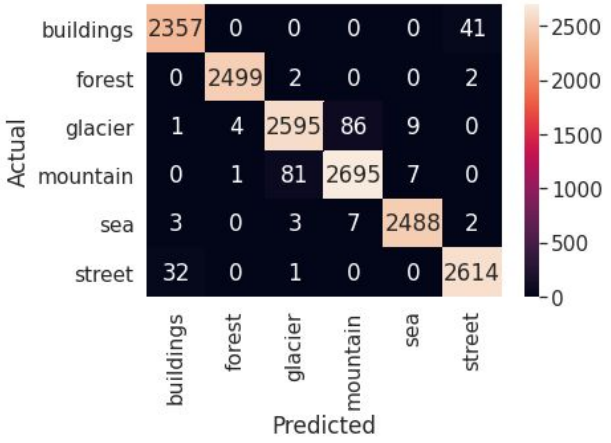


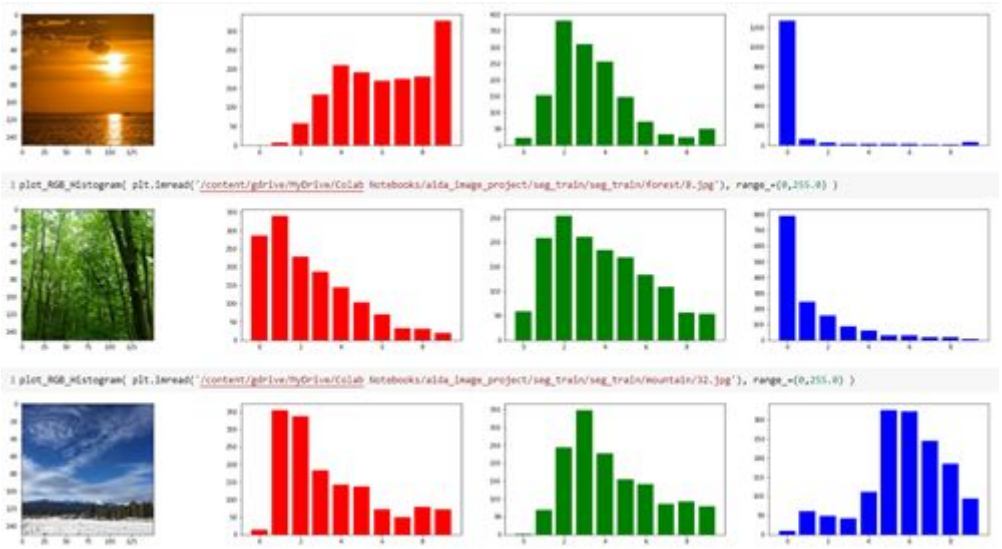
Fig: Confusion Matrix SVM+ VGG

Intuitive Feature Extractor

One approach was to investigate whether the classification was possible to improve with simple intuitive features such as the color or edge spectrum.

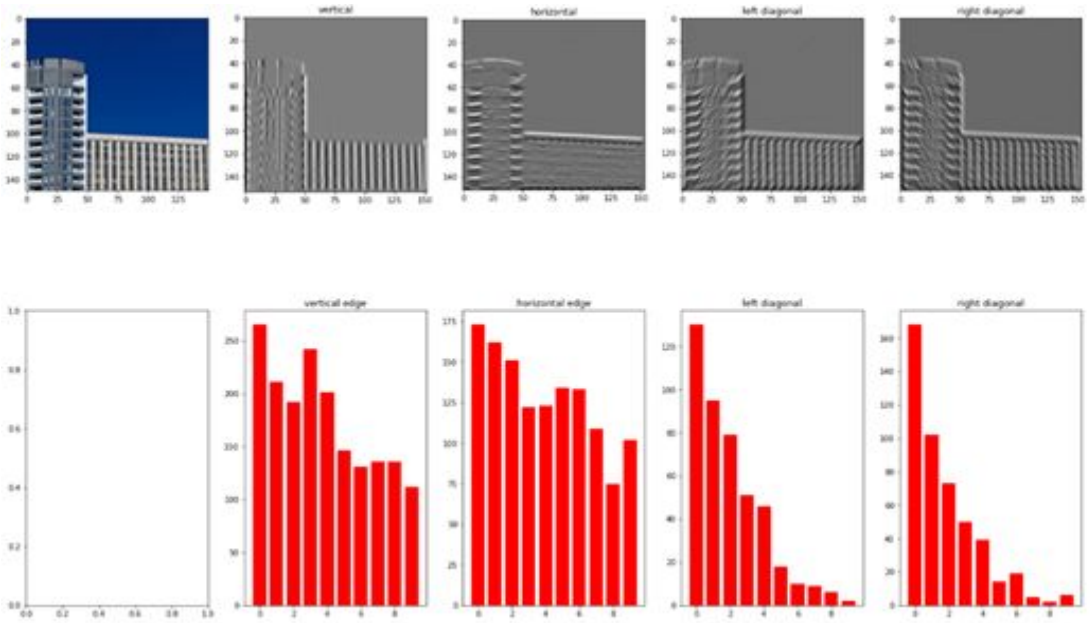
Color Spectrum

If you look at the pictures, you can see that the sundown is more red, the forest more green and the sky blue:



Edge Spectrum

Forests have mostly vertical edges because of the vertical trees. Buildings have many horizontal and vertical edges due to the windows.



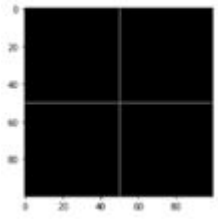
ORB, SURF, SWIFT Feature Descriptors

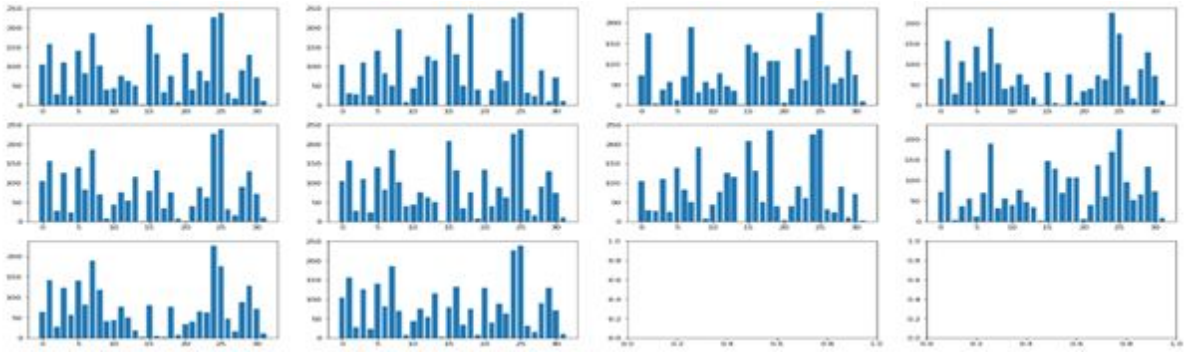
In a further approach, methods ORB, SURF, SWIFT for matching tasks were examined.

Result

These Feature-Descriptors generate correlation-patterns, which can be used by cross-correlation to find the same object/part with the same pattern in a given other picture. Count of features depending on object, edges, threshold, ... so it's not a generalizer for images like the color-spectrum.

Example: For the following test image arbitrarily count of redundant descriptors were generated, depending on thresholds and parameters.

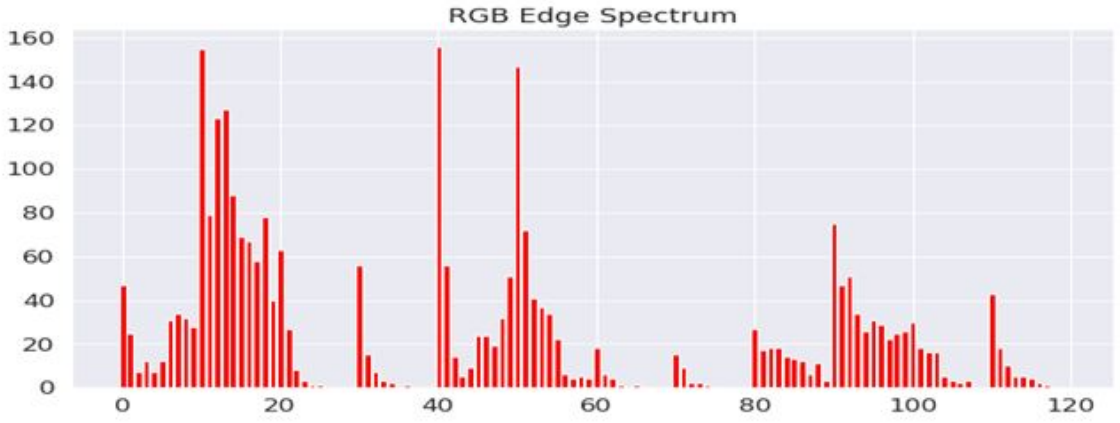




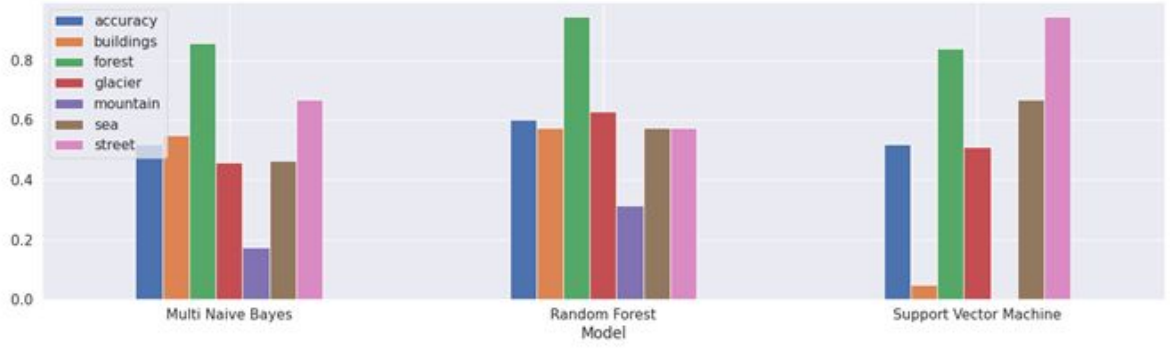
Feature Vector

The composed resulting Input-Feature-Vector consisted of 153 features, composed by the following parts (R=Red, G=Green, B=Blue):

[R_median[1], G_ median[1], B_median[1], Color_Spectrum[3,10], Edge_Spectrum[3,4,10]]



Resulting accuracies and per class accuracies with different classifiers



Conclusion

Best accuracy of 0.60 with Random Forest. This approach failed to achieve the desired result. These simple intuitive features are not complex enough to distinguish images.

Deep Neural Classifier I (CNN's)

Examine and evaluate selected Deep Learning architecture and methods for the given task

General approach:

Our main focus for solving the classification problem is building a Convolutional Neural Networks (CNN). CNNs have delivered the best results for this kind for several years now. If we have look at the top scoring results of the original competition[2,3,4,5] we will also notice that all top submissions use a different set-up of CNNs.

Goals for the model search

For this project we wanted to understand, on what the model performance depends. Specifically we had four main parameters and their effect on the results:
model architecture, image input size, tuning of learning rate and influence of transfer learning

To succeed with this goal we decided to implement the different approaches using Keras with a bit of Tensorflow when necessary. With Keras we could heavily leverage the already implemented architectures in Keras Applications.

The image input size of 150^2 is in below the common input of 224 of many networks.The input layer should be divisible by 2 many times[6]. Which gives us 3 options to try out: Upscaling to 224, downscaling to 112 (0.5*224) or no scaling at 150. If equal results can be archived with a smaller input size that would be preferable for computational reasons.

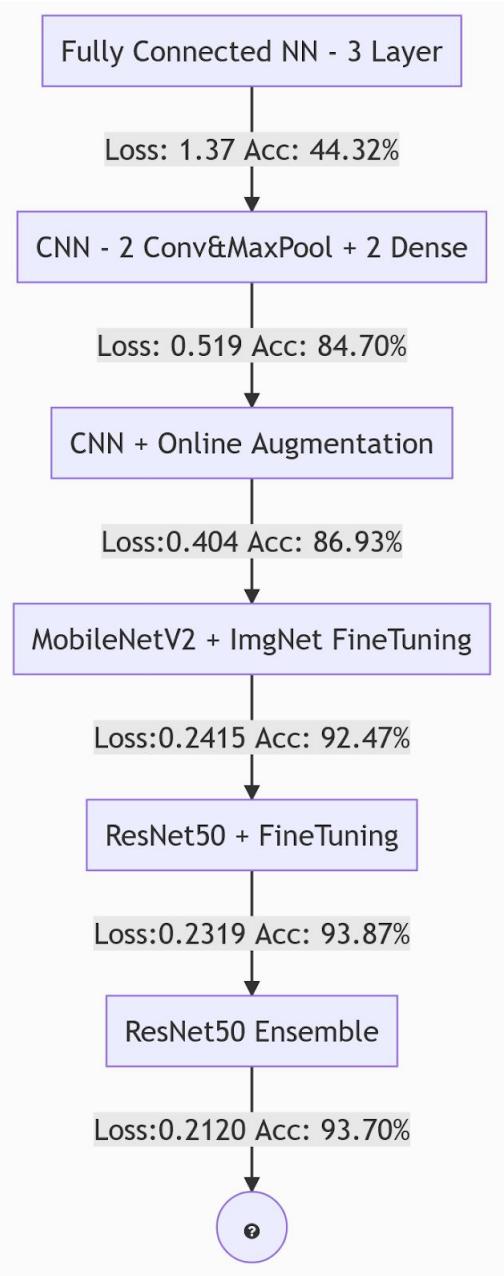
For the learning rate we want a fast model conversion, with good results. Therefore we will make use of the Keras Callback for early stopping and learning rate decay. For the initial learning rate we will refer to the recommendation from the papers

With about 15k images the dataset is reasonably big. Compared to ImageNet with 14mio[7], we are however still wondering, if fine-tuning imagenet weights might actually lead to better results than training from scratch. While researching we also found Places365[8]. Places365 contains 10mio images of 365 scenes and comes with pretrained models for some common architectures.

[2] https://datahack.analyticsvidhya.com/contest/practice-problem-intel-scene-classification-challe
[3] https://github.com/NishantBhavsar/intel-scene-classification
[4] https://github.com/afzalsayed96/intel-scene-classification
[5] https://towardsdatascience.com/1st-place-solution-for-intel-scene-classification-challenge-c95cf941f8ed
[6] https://cs231n.github.io/convolutional-networks/
[7] http://image-net.org/update-sep-17-2019
[8] http://places2.csail.mit.edu/index.html

Set-up and framework for examination

Step by step evaluations aligned to time and capacity restrictions



We trained the models on different machines with selected set-ups. We focussed on exemplary evaluating parameters but were limited to compare the training times of the different models.

Because training time was one of the main challenges for the project in the end, we decided to speed up the process of training we ported some of our models on Google Cloud TPU[9]'s (for details refer to the *Project_Image_Scene_Classification_TPU.ipynb* notebook).

Alligined to given time and capacity restriction, covering the complete search space to find the ideal model configuration, we decided to evaluate step by step: which method improved our result and further models we trained only with including those improvements.

Selected overview of examinations

The graphic to the left shows the successful modifications which had a positive impact on the model performance and therefore were included in the subsequent models.

Our first Baseline model[10] was a simple fully connected network that performed poorly at around 44% Accuracy.

It was followed by a very simple CNN using spatial convolution and max pool layers for the feature extraction and two fully connected layers for the classification, with which we could already double our accuracy. By Augmenting the training images, using Keras ImageDataGenerator we could minimize the loss by one fifth.

Afterwards we tried training bigger architectures from scratch such as ResNet50 (78%) or EfficientNet (83.5%), but couldn't really gain any improvement. By using the pretrained ImageNet weights we could further increase the accuracy on MobileNetV2 up to 92.5% and to 93.9% using the bigger ResNet50. Deeper or different architectures did really help to

[9] https://www.tensorflow.org/guide/tpu#train_a_model_using_keras_high_level_apis
[10] https://datauab.github.io/natural_scenes_classification/

improve the results any further. We also tried training on pretrained models of Places365, but we could really get over the 93.9%. We could improve the loss slightly by combining four via K-Fold trained ResNets into one Ensemble Network.

Evaluation of best performing Neural Network

As described in the previous chapter, we tried different neural networks architectures with various training parameters. We reached the best performance using the Ensemble model and we would like to report briefly on the strengths and weaknesses of this model. The model has an overall accuracy of 94%, predicting forests and seas almost perfectly. We have already seen in data exploration, that glacier-mountain and building-streets are the classes that tend to be hard to differentiate. The confusion matrix below confirms this problem also for our best NN.

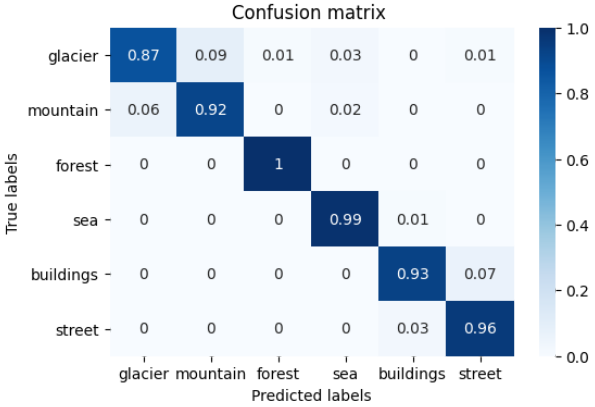
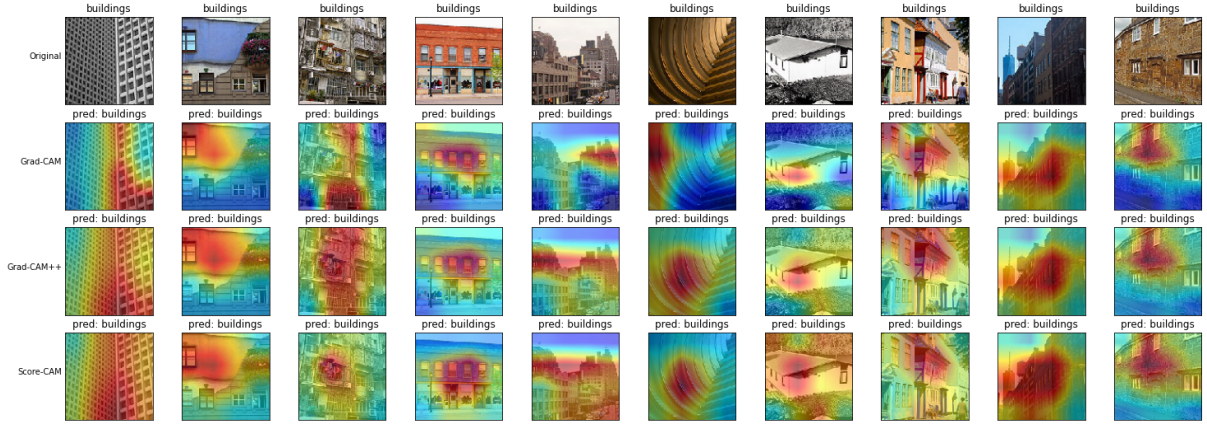


Fig: Confusion matrix- Ensemble NN

	precision	recall	f1-score	support
glacier	0.94	0.87	0.90	553.00
mountain	0.90	0.92	0.91	525.00
forest	0.98	1.00	0.99	474.00
sea	0.95	0.99	0.97	510.00
buildings	0.95	0.93	0.94	437.00
street	0.93	0.96	0.95	501.00
accuracy	0.94	0.94	0.94	0.94
macro avg	0.94	0.94	0.94	3000.00
weighted avg	0.94	0.94	0.94	3000.00

Fig: Classification-Report Ensemble NN

To get a better understanding of the weaknesses and strengths of the classifier we plotted the class activation maps for different pictures of different classes.



Further observations and further ideas for improvement

We also observed that adding Dense-Layers for Classification improved the results by up to 2%, we could however leverage that fact not to the full extent yet.

```
# Add layers at the end
X = base_model.output
X = layers.Flatten()(X)

X = layers.Dense(512, kernel_initializer='he_uniform')(X)
#X = layers.Dropout(0.5)(X)
X = layers.BatchNormalization()(X)
X = layers.Activation('relu')(X)

X = layers.Dense(16, kernel_initializer='he_uniform')(X)
#X = layers.Dropout(0.5)(X)
X = layers.BatchNormalization()(X)
X = layers.Activation('relu')(X)
```

```
# Rebuild top
x = layers.GlobalAveragePooling2D(name="avg_pool")(model.output)
x = layers.BatchNormalization()(x)

top_dropout_rate = 0.2z
x = layers.Dropout(top_dropout_rate, name="top_dropout")(x)
outputs = layers.Dense(NUM_CLASSES, activation="softmax",
                       name="pred")(x)
```

Additionally we couldn't test the effect on the Dropout layers sufficiently. In regards to the image input size we discovered that, we get better results the larger the pictures are.

To catch up with the top submissions from Intel competition, we would further need to implement additional techniques like Transfer Learning with Progressive Image Resizing or Test Time Augmentation. The authors of different submissions also stated that it helps to remove the existing misclassifications in the dataset.

Web App for presentation of Image Scene Classification results

To make our solution accessible for non-technical people, we deployed our model as a web application. The code for this web interface is taken from the Deploy Keras Model with Flask as Web App in 10 Minutes project developed by Xin Fu under GPL 3.0[11]. We changed the application code in a way that it can use our custom model instead of the ImageNet. The application uses a simple JavaScript frontend that sends the images encoded as Base64 to the REST API backend. The REST API backend is based on the sample code provided by Keras[12]. The average performance is about 210ms per prediction on the ResNet50 using a GTX 1650.

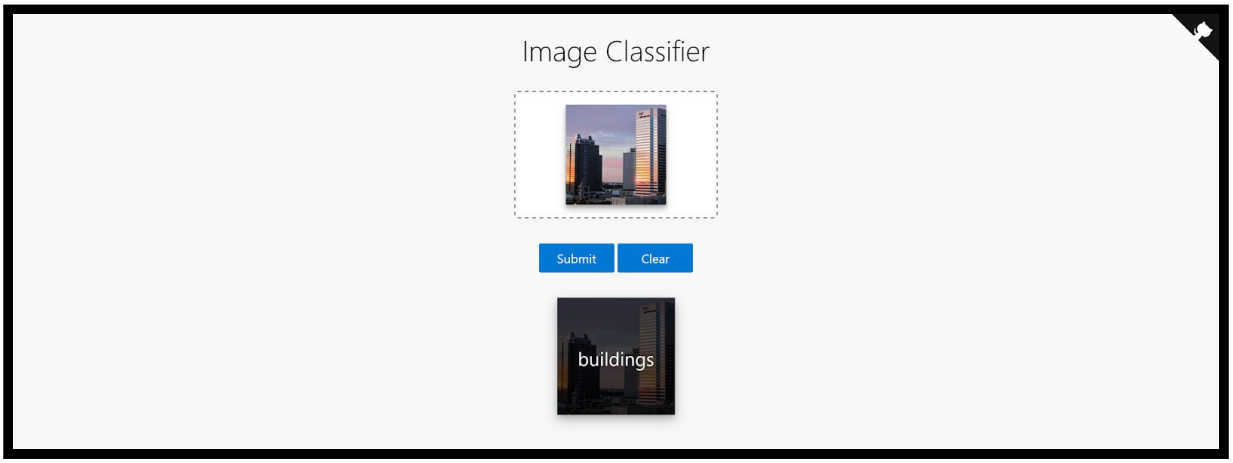


Fig: WebUI interface

To run the web application ensure that all required packages are installed or refer to the *requirements.txt* of this project. Then run the python app.py and you should be able to see the WebUI in your browser at http://localhost:5000
It's easy to install and run it on your computer.

```
$ git clone https://github.com/AIDA-DA/aida-project-image-scene.git
$ cd aida-project-image-scene

$ pip install -r requirements.txt

$ python app.py
```

There are multiple ways to go when deploying the model into production, depending on the infrastructure and hardware resources. Since we have no further specifications in the project requirements, we just deployed the model to a free IBM Cloud instance[13] for everyone to try out. See the Deploy_to_WML.ipynb notebook for details.

[11] https://github.com/mtobeiyf/keras-flask-deploy-webapp
[12] https://blog.keras.io/building-a-simple-keras-deep-learning-rest-api.html
[13] https://github.com/IBM/watson-machine-learning-samples/tree/master/cloud/notebooks/python_sdk/deployments

Conclusion

Model Selection

The final choice of a classification model for deployment in real-use scenarios should take into account several criteria which are relevant for a given context.

Besides the classification accuracy one should also take care of the models cost in terms of training and processing time. Also the amount of data needed to come to a converging model should be reflected for this kind of decision.

The following table tries to give a picture how the models we developed and tested meet the different criteria from our point of view.

criterion / model	**accuracy**	**processing time**	**amount of data needed**	**training effort**
statistical classifier	91%	+	0	++
fully connected NN	44%	0	0	-
CNN - MobileNetV2	92,47%	0	0	
CNN - resnet 50 finetuning	93,87%	-	0	+
ensemble (resnet50)	94 %	-	0	+

As we had no further requirements or restrictions regarding a specific context of usage, we voted for the model with the highest classification accuracy:

Ensemble (resnet50)

Open questions / further research

During our experiments with different models we consistently made the observation that we obtained more or less competitive overall classification results. However, for the classes mountains/glaciers and streets/buildings we observe a significant decline for the classification accuracy regardless which model is used. Taking into account that glaciers are often located in mountains and most buildings can be found next to roads, this finding is not

that astonishing: Some of the chosen images simply show both classes and therefore are being classified less confidently into one of them.

For the classification problem we have several options to solve this issue.

a. define “normative” decision rules that deal with lower confidences for these classes (e.g every time we find low confidences with mountain & glaciers, the image is classified as “glacier”, since glacier is regarded as the more “salient” category)
b. do a kind of hierarchical classification: After the 6-category model has done his work, we introduce another 2-category classification model that's especially trained to distinguish between glaciers and mountains/ streets and buildings. The model can decide, whether an image is “more” mountain or “more” glacier. This would require a review of labels within the image pool.
c. allow for multi-label-classification and apply the appropriate classification models, so that the classification can be done with multiple classes for each image.
d. define wider, but more selective classes that contain both subclasses (e.g “both” glacier & mountain) and try to find out how the classifier deals with these new classes. This also would lead to a new labeling of the image pool.

During this project, we did not find the time to address these questions but we strongly recommend to do so in following projects.

-

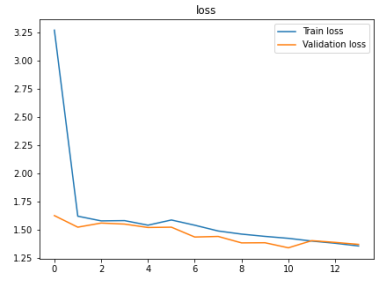
Appendix

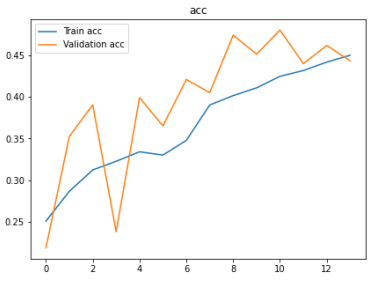
Results of single Neural Network evaluations

Algorithm	Image Size	Training-Method	Online-Augmentation	Validation	
				Acc-Score	Loss
ResNet-50	224	ImageNet FeatureExtraction - Only Dense Layer Training	yes	78,90%	0,5927
ResNet-50	224	ImageNet FineTuning - first layers	yes	93,50%	0,2465
ResNet-50	224	ImageNet - FineTuning - last layers	yes	93,87%	0,2319
ResNet-50	224	ImageNet FineTuning - all layers	yes	93,87%	0,2411
ResNet-50	150	Training from Scratch	yes	79,93%	0,5737
ResNet-50-Ensemble	150	Ensebling 4-KFold FineTuning Models	yes	93,70%	0,2120
MobileNetV2 Baseline	150	ImageNet FineTuning - first layers	yes	92,47%	0,2416
Simple Conv Baseline	150	Training from Scratch	yes	86,93%	0,4044
Simple Conv Baseline	150	Training from Scratch	no	84,70%	0,5192

Baseline CNN (simple, Augmentation, MobileNetV2) - notes

Simple Neural Network

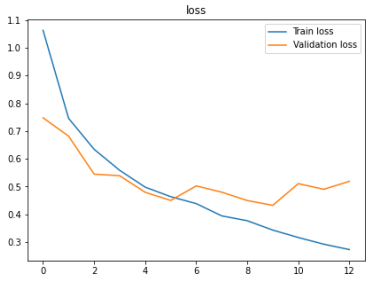


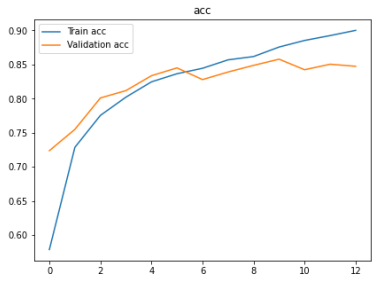



```
Parameter - first simple
steps_per_epoch  438
validation_steps  93
epochs  50
verbose  1
```

```
93/93 [==============================] - 3s 33ms/step - loss: 1.3715 - accuracy: 0.4432
Model MLP Test Loss: 1.371539831161499
Model MLP Test Accuracy: 0.4432123601436615
```

Simple CNN Model

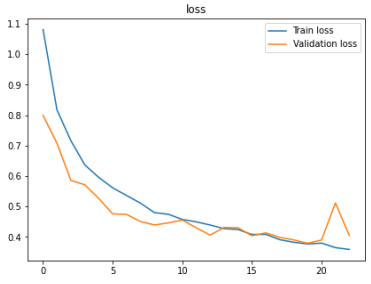


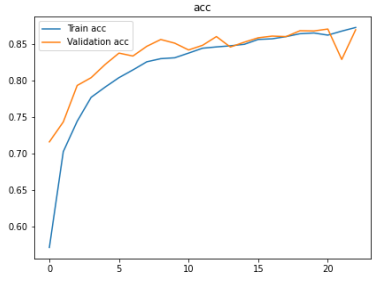



```
Parameter - CNN Model
steps_per_epoch 438
validation_steps 93
epochs 40
verbose  1
```

```
94/94 [==============================] - 4s 37ms/step - loss: 0.5192 - accuracy: 0.8470
Model CNN Test Loss: 0.5191602110862732
Model CNN Test Accuracy: 0.847000002861023
```

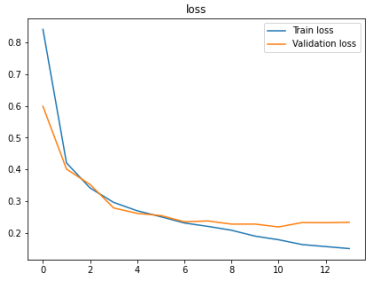
Data Augmentation

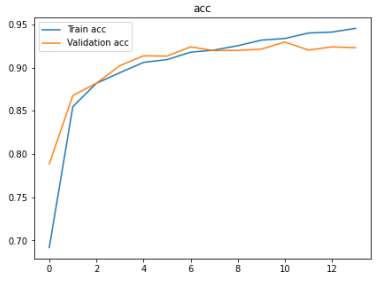





```
Parameter - Data Augmentation
steps_per_epoch 438
validation_steps 93
epochs 40
verbose  1
```

```
94/94 [==============================] - 15s 165ms/step - loss: 0.4044 - accuracy: 0.8693
Model with Data Augmentation Test Loss: 0.40435248613357544
Model with Data Augmentation Test Accuracy: 0.8693333268165588
```





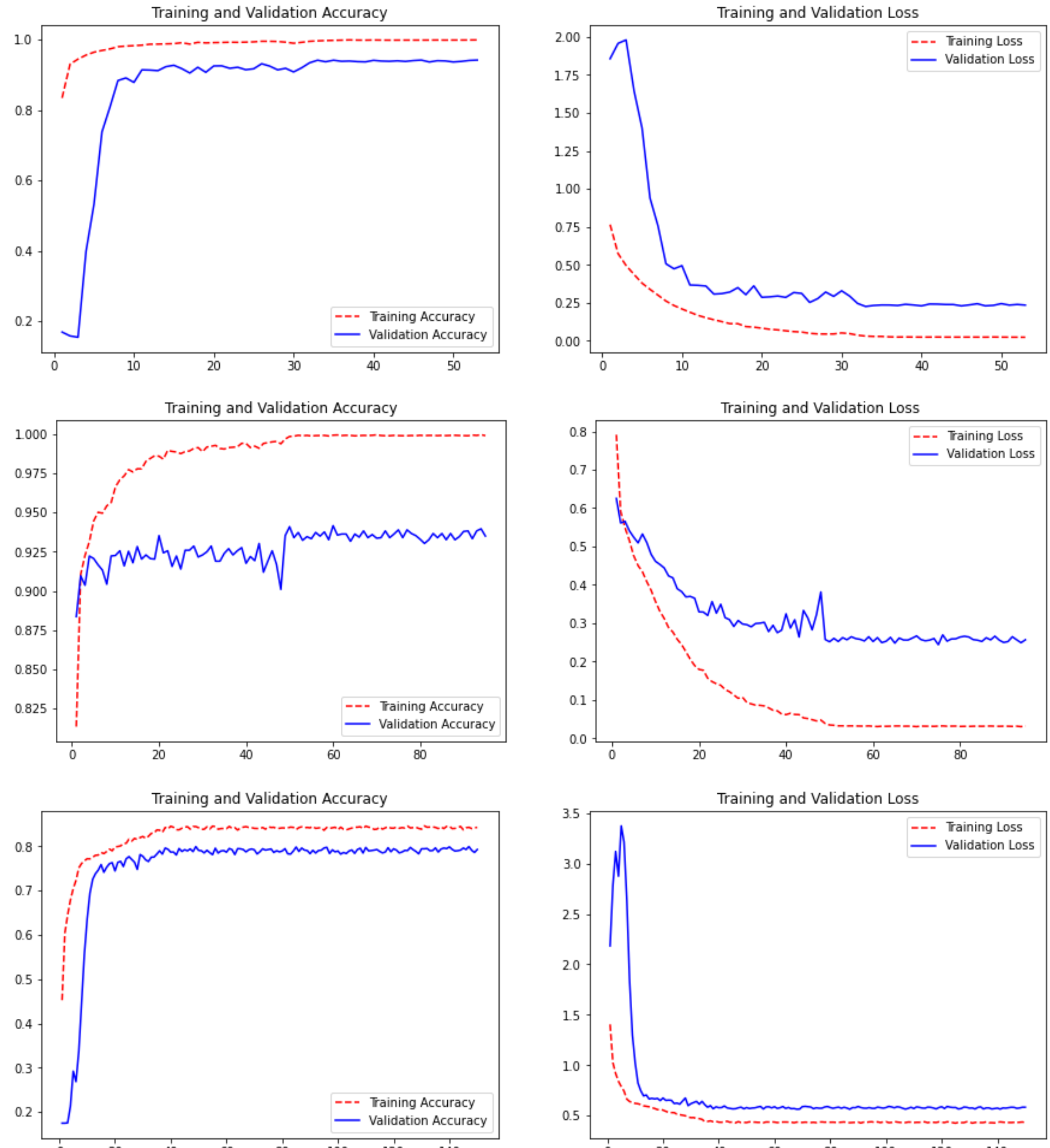

```
# Create model adding the pre-trained model mobileNetV2,
# adding GlobalAveragePooling2D layer
model = Sequential([mobile_model,
                    GlobalAveragePooling2D(),
                    Dropout(rate=0.5),
                    Dense(6, activation='softmax')])

model.compile(optimizer= RMSprop(lr=2e-5),
              loss='categorical_crossentropy',
              metrics=['accuracy'])

model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
mobilenetv2_1.00_224 (Functi (None, 5, 5, 1280)        2257984
_________________________________________________________________
global_average_pooling2d (Gl (None, 1280)              0
_________________________________________________________________
dropout_3 (Dropout)          (None, 1280)              0
_________________________________________________________________
dense_7 (Dense)              (None, 6)                 7686
=================================================================
Total params: 2,265,670
Trainable params: 1,870,278
Non-trainable params: 395,392
_________________________________________________________________
```

```
94/94 [==============================] - 16s 173ms/step - loss: 0.2416 - accuracy: 0.9247
Model Fine Tuning Test Loss: 0.24158638715744019
Model Fine Tuning Test Accuracy: 0.9246666431427002
```


Training and Validation Accuracy
Training Accuracy
Validation Accuracy
Training and Validation Loss
Training Loss
Validation Loss
Training and Validation Accuracy
Training Accuracy
Validation Accuracy
Training and Validation Loss
Training Loss
Validation Loss
Training and Validation Accuracy
Training Accuracy
Validation Accuracy
Training and Validation Loss
Training Loss
Validation Loss