Ain Shams University
Faculty of Computer & Information Sciences
Computer Science Department

Neural Network Project Report

Selected Project Idea:

"Weather Analysis Using Image Recognition"

Under Supervision of:

Dr. Ghada Hamed

Team Members:

1st Team Member Name:

Kareem Sherif Fathy

1st Team Member ID:

2018170283

2nd Team Member Name:

Kareem Saeed Ragab

2nd Team Member ID:

2018170282

3rd Team Member Name:

Abanoub Asaad Azab

3rd Team Member ID:

2018170001

4th Team Member Name:

Nada El Sayed Anies

4th Team Member ID:

2018170430

5th Team Member Name:

Nada Mohamed Abdelhamed

5th Team Member ID:

2018170434

Project Idea Overview

Many industries have the need to identify current and past weather conditions. The data helps them plan, organize, and/or optimize their operations. Using CNNs we are offering the potential to automate this by providing a digital eye. If an image recognition model could be built to identify conditions by looking at images of the weather, it could be deployed to automatically trigger smart devices.

The dataset used in this project consists of 5,551 training images and 1,300 testing images. It has 11 classes representing different weather conditions, these classes are: dew, fogsmog, frost, glaze, hail, lightning, rain, rainbow, rime, sandstorm, snow.

Approaches Used in all trials

a) Early Stopping

A basic problem that arises in training a neural network is to decide how many epochs a model should be trained. Too many epochs may lead to overfitting of the model and too few epochs may lead to underfitting of the model.

In this technique, we can specify an arbitrarily large number of training epochs and stop training once the model performance stops improving on a hold out validation dataset.

b) Model Check Point

The **EarlyStopping** callback will stop training once triggered, but the model at the end of training may not be the model with the best performance on the validation dataset.

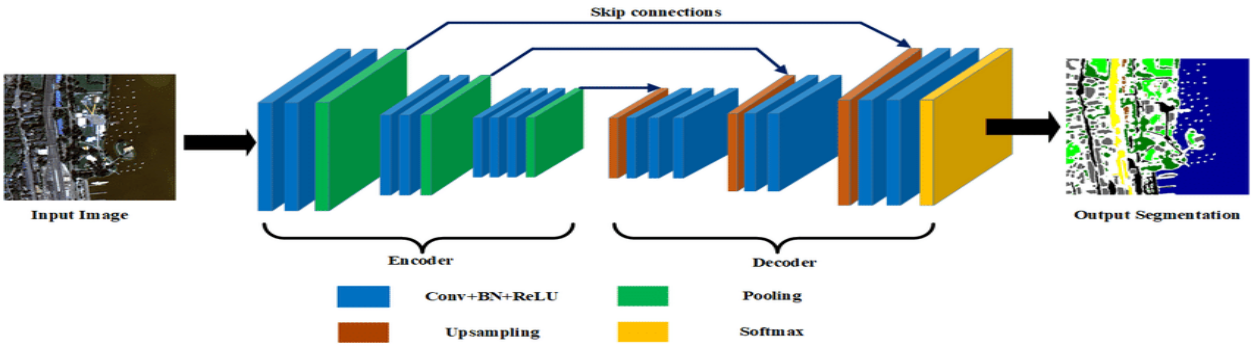
An additional callback is required that will save the best model observed during training for later use. This is known as the **ModelCheckpoint** callback. The ModelCheckpoint callback is flexible in the way it can be used, but in our case, we will use it only to save the best model observed during training as defined by a chosen performance measure on the validation dataset.

Trial #1: "Encoder & Decoder Model"

1.1 Base Model Architecture

The First Step of the Model Architecture named as the Encoder (Down Sampling) holds a Convolution Layers followed by Batch-Normalization and Max-Pooling Layers. Afterwards, The Architecture reverses the layers' flow (any convolution layer is replaced by a Un-Convolution Layer, etc...) the step named as the Decoder (Up Sampling) Step. It keeps reversing until the image size reaches the original size.

The below architecture clarifies that:



1.2 The Tried Model Architecture (in details)

The architecture consists of 7 Layers of convolution, Batch-Normalization and Max-Pooling layers with **kernel size (3, 3)**, **Same Padding** and **relu** activation function then Up-Sampling Layer. Afterwards the Previous Layers are reversed with the same kernel size and activation function. Ending up with a Flatten Layer followed by two Fully-Connected Layers, The Last Layer has a **softmax** as the activation function **with 11 output classes**.

1.3 The used Techniques

- Label Encoder
- One Hot Encoder

1.4 The Model Evaluation

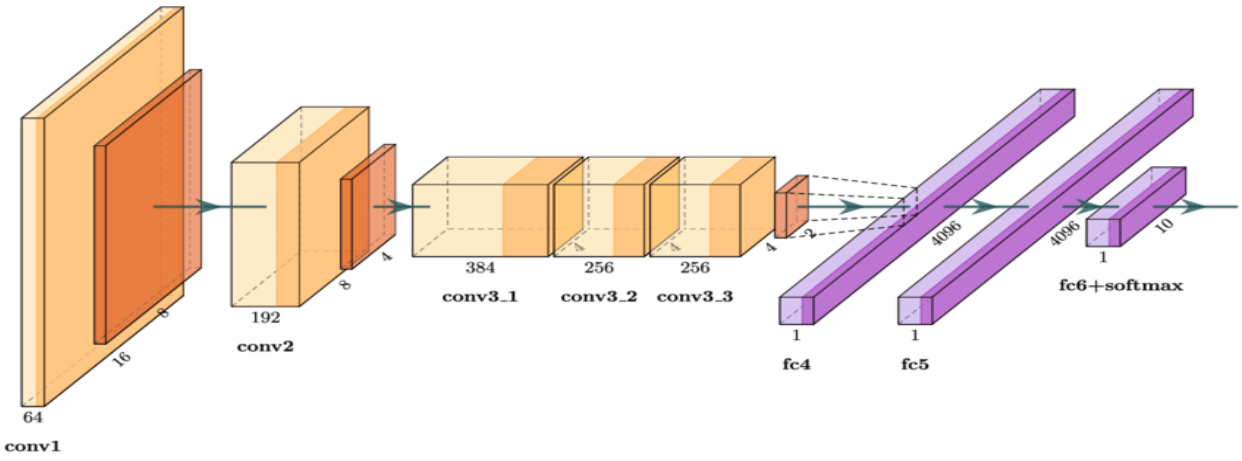
- Validation Loss: **0.606**
- Validation Accuracy: **54.2%**
- Test Score: **0.519 (51.9%)**

Trial #2: "Alex Net Model"

2.1. Base Model Architecture

The Model Architecture is a **Sequential Model** consists of Convolution Layers followed by Batch-Normalization and Max-Pooling Layers. Ending up with two Dense (Fully Connected) Layers.

The below architecture clarifies that:



2.2. The Tried Model Architecture (in details)

The architecture consists of 5 Layers of convolution, Batch-Normalization and Max pooling Layers with **kernel size (3, 3)**, **Same Padding** and **relu** activation function. Ending up with a Flatten Layer, two Fully Connected, The Last Layer has a **softmax** as the activation function **with 11 output classes**.

2.3. The used Techniques

- Label Encoder
- One Hot Encoder

2.4. The Model Evaluation

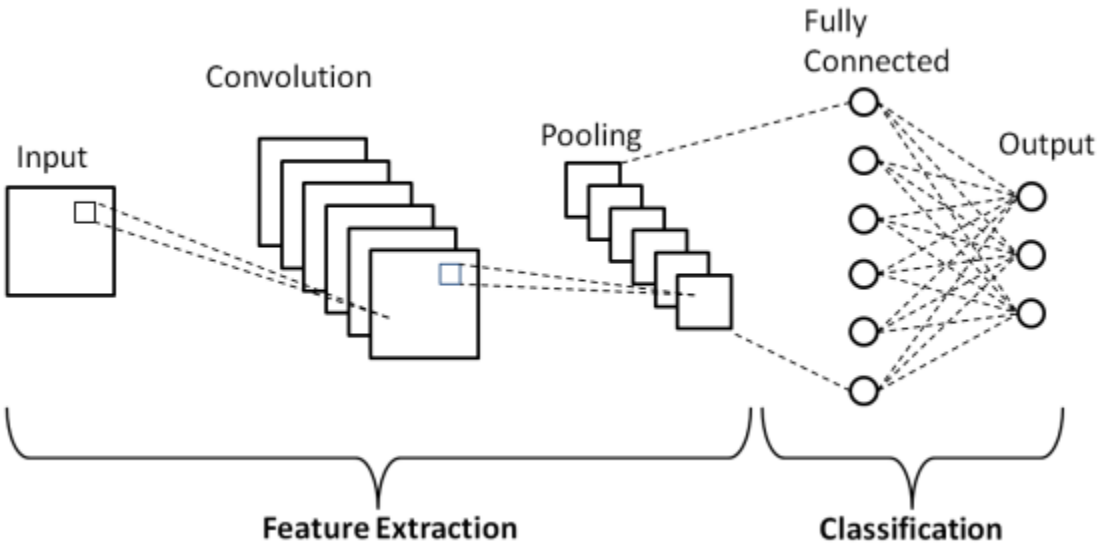
- Validation Loss: **0.156**
- Validation Accuracy: **74.3%**
- Test Score: **0.746 (74.6%)**

Trial #3: "Convolution Neural Network (CNN) Model"

3.1. Base Model Architecture

CNN uses a multilayer system consists of the input layer, output layer, and a hidden layer that comprises multiple convolutional layers, pooling layers, fully connected layers.

The below architecture clarifies that:



3.2. The Tried Model Architecture (in details)

The architecture consists of 17 Layers of convolution, Max-Pooling and Dense Layers with **kernel size (3, 3)** for every convolution layer, **kernel size (2, 2)** for every pooling layer and **relu** activation function. Ending up with a 3 Fully Connected Layers, The Last Layer has a **softmax** as the activation function **with 11 output classes**.

3.3. The used Techniques

- Train-Valid Split **(85% to 15%)**
- Image Data Generator with **Data Augmentation**
- Compile Function **(SGD Optimizer with eta=10^{-4} and momentum=0.9)**

3.4. The Model Evaluation

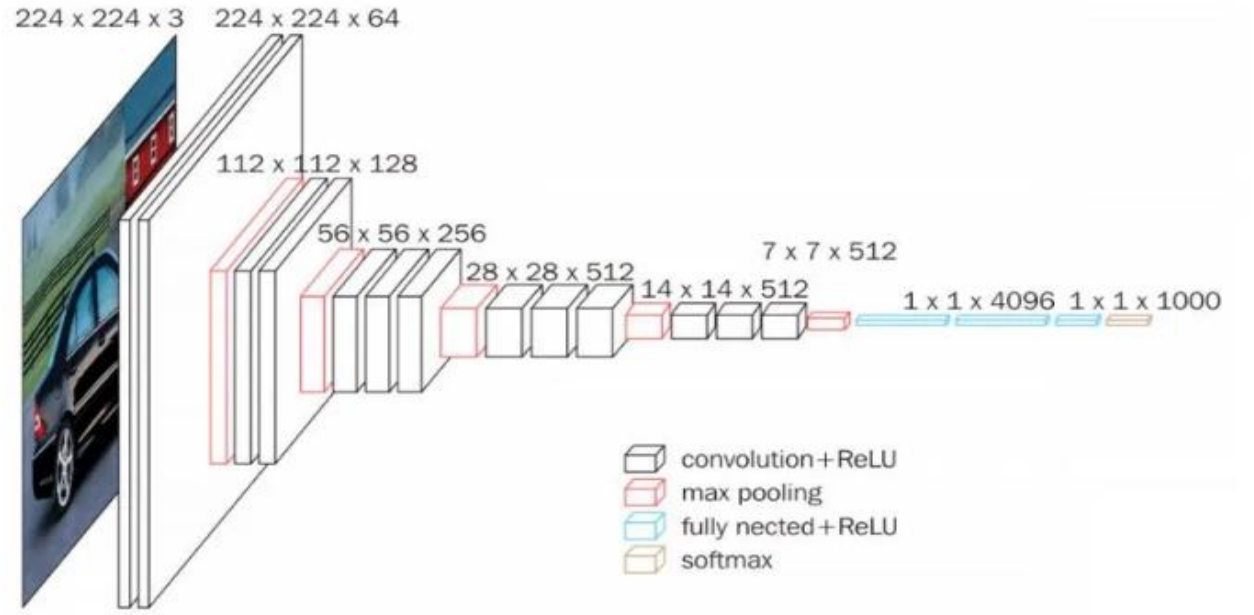
- Validation Loss: **0.148**
- Validation Accuracy: **63.2%**
- Test Score: **0.605 (60.5%)**

Trial #4: "Vgg16 Model"

4.1. Model Overview

VGG16 is a convolutional neural network architecture that was the runners up in the 2014 ImageNet challenge (ILSVR) with **92.7% top-5 test accuracy** over a dataset of 14 million images belonging to 1000 classes. Although it finished runners up it went on to become quite a popular mainstream image classification model and is considered as one of the best image classification architecture.

The below architecture clarifies that:



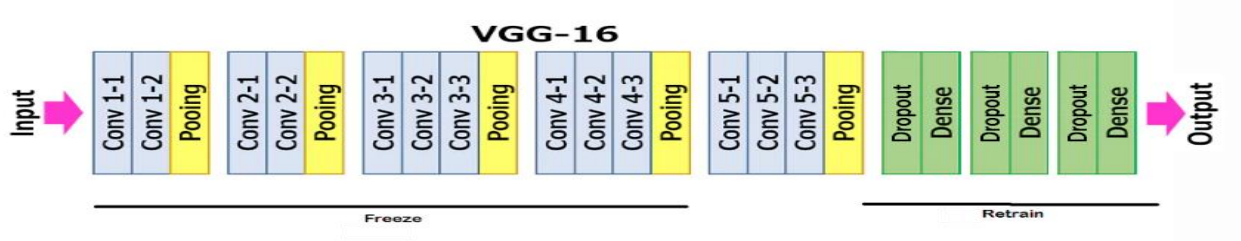
4.2. VGG16 Transfer Learning Approach

Deep Convolutional Neural networks may take days to train and require lots of computational resources. So to overcome this and a way to improve our test results we will use Transfer Learning for implementing VGG16 with Keras.

Transfer learning is a technique whereby a deep neural network model that was trained earlier on a similar problem is leveraged to create a new model at hand. One or more layers from the already trained model are used in the new model.

4.3. The Tried Model Architecture (in details)

- In VGG16 there are **13** convolutional, **5** Max-Pooling and **3** FC Layers which sum up to **21** layers but it has only **16** weight layers.
- It has convolution layers of **kernel 3×3** with a **stride 1** and always use the **same padding** and pooling layers of **kernel 2×2** with **stride 2**.
- Conv-1 Layer has 64 number of filters, Conv-2 has 128 filters, Conv-3 has 256 filters, Conv-4 and Conv-5 has 512 filters.
- Three Fully-Connected (FC) layers follow a stack of convolutional layers: the first two have 4096 channels each, the third performs 1000-way ILSVRC classification and thus contains 1000 channels (one for each class). The final layer is the soft-max layer.
- The input to Conv-1 layer is of fixed size **224x224** RGB image.
- We reused the model weights from **pre-trained model** that was developed for standard computer vision benchmark datasets like **ImageNet**. We have downloaded the pre-trained weights that do not have top layers weights.
- We have performed some changes in the dense layer. In our model, we have replaced it with our own three group of dropout layers followed by a dense layer for each. Finally the last dense layer is of dimension **11** with a softmax activation.



4.4. The used Techniques

- Train-Valid Split (90% to 10%)
- Image Data Generator with Data Augmentation
- Compile Function **(SGD Optimizer with eta=10^{-4} and momentum=0.9)**

4.5. The Model Evaluation

- Validation Loss: **0.077**
- Validation Accuracy: **83.1%**
- Test Score: **0.829 (82.9%)**

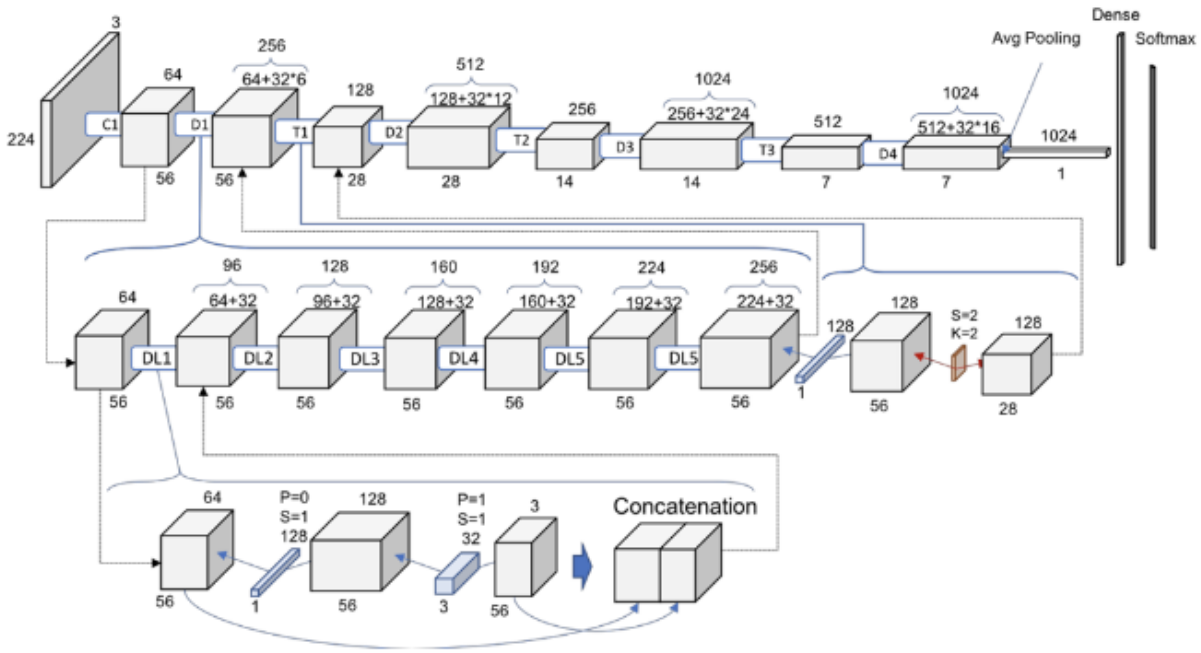
Trial #5: "DenseNet Model"

5.1. Model Overview

DenseNet is quite similar to ResNet with some fundamental differences. ResNet uses an additive method (+) that merges the previous layer (identity) with the future layer, whereas DenseNet concatenates (.) the output of the previous layer with the future layer.

DenseNet was developed specifically to improve the declined accuracy caused by the vanishing gradient in high-level neural networks. In simpler terms, due to the longer path between the input layer and the output layer, the information vanishes before reaching its destination.

<u>The below architecture clarifies that:</u>



An output of the previous layer acts as an input of the second layer by using *composite function operation.* This composite operation consists of the convolution layer, pooling layer, batch normalization, and non-linear activation layer.

These connections mean that the network has **L(L+1)/2** direct connections. L is the number of layers in the architecture.

The DenseNet has different versions, like DenseNet-121, DenseNet-160, DenseNet-201, etc. The numbers denote the number of layers in the neural network. The number 121 is computed as follows:

5+(6+12+24+16)*2 = 121

5 – Convolution and Pooling Layer

3 – Transition Layers(6, 12, 24)

1 – Classification Layer (16)

2 – DenseBlock (1x1 and 3x3 conv)

5.2. The used Techniques and Hyper Parameters

- Train-Valid Split **(90% to 10%)**
- Image Data Generator with **Data Augmentation**
- Compile Function **(Adam Optimizer with eta=10^{-4})**
- Early Stopping Callback **(monitor='val_acc', mode='max', patience=10)**
- **Batch Size = 16**
- **Epochs = 40**

5.3. The Model Evaluation

- Validation Loss: **0.072**
- Validation Accuracy: **86.2%**
- Test Score: **0.851 (85.1%)**

Trials Conclusion

Data Normalization affects the model performance hugely as well as The **Data Augmentation**. The architecture with a few layers does not occasionally offer satisfactory performance.

Class Imbalance is a serious issue in Deep Neural Network Data, such an issue should be resolved before passing the data to the model which will take sides and surely we don't want that to happen.

The Initial Layers learn very general features and as we go higher up the network, The Layers tend to learn patterns more specific to the task it is being trained on. So In **Transfer Learning** if we want to use these properties of the layer then we have to keep the initial layers intact (freeze that layer) and retrain the later layers for our task.

The advantage of **fine-tuning** is that we do not have to train the entire layer from scratch and hence the amount of data required for training is not much either. Also, parameters that need to be updated are less and hence the amount of time required for training will also be less.

References

[1] TensorFlow: https://www.tensorflow.org/guide/low_level_intro

[2] OpenCV: https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_tutorials.html

[3] CNN: https://medium.com/technologymadeeasy/the-best-explanation-of-convolutional-neural-networks-on-the-internet-fbb8b1ad5df8

[4] Vgg16: https://neurohive.io/en/popular-networks/vgg16/

[5] DenseNet: https://www.pluralsight.com/guides/introduction-to-densenet-with-tensorflow

[6] Image Data Generator: https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator

[7] Early Stopping: https://keras.io/api/callbacks/early_stopping/