Ain Shams University
Faculty of Computer & Information Sciences
Bioinformatics Department

Neural Network Project Report

Project Idea:

“[NN'23] Sports Image Classification”

Team Members:

Member name	Student's ID	Department
1. Osama Mahmoud Ahmed Shaban	20191701022	Bioinformatics
2. Alaa Mohsen Mahmoud Ahmed	20191701030	Bioinformatics
3. Eman Elmoutaz-Bellah Mohamed	20191701044	Bioinformatics
4. Ahmed Ashraf Ahmed Morsy	20191701003	Bioinformatics
5. Shimaa Alaa Yousef	20191701113	Bioinformatics

➢ Project Idea Overview

In today's world of internet, a massive amount of data is getting generated every day and content-based classification of images is becoming an essential aspect for efficient retrieval of images and have attracted application in several fields and one of such field is sports. Building a model that can classify different sports activities into different categories could be useful for automated sports analysis tasks.

The dataset used in this project consists of 1,681 training images and 688 testing images. It has 6 classes representing different sports, these classes are: Basketball --> 0

Football-->1
Rowing-->2
Swimming-->3
Tennis-->4
Yoga-->5.

➢ 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 holdout 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

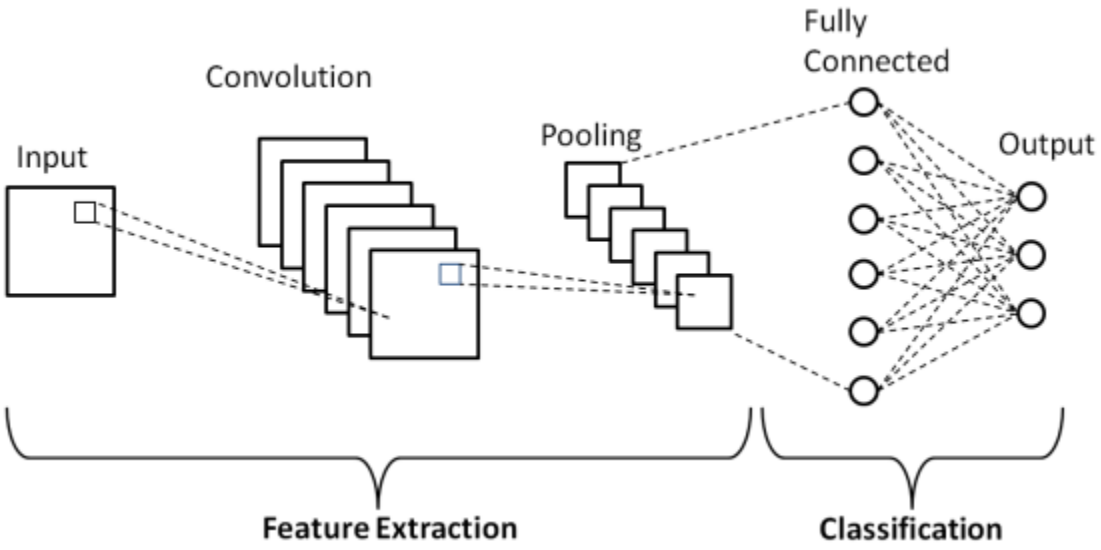
model observed during training as defined by a chosen performance measure on the validation dataset.

❖ Model 1: "Convolution Neural Network (CNN) Model"

- **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:



- **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 6 output classes**.

- **The used Techniques**
  - Train-Valid Split **(90% to 10%)**
  - Image Data Generator with **Data Augmentation**
  - Compile Function **(Adam Optimizer with eta=10^{-4})**

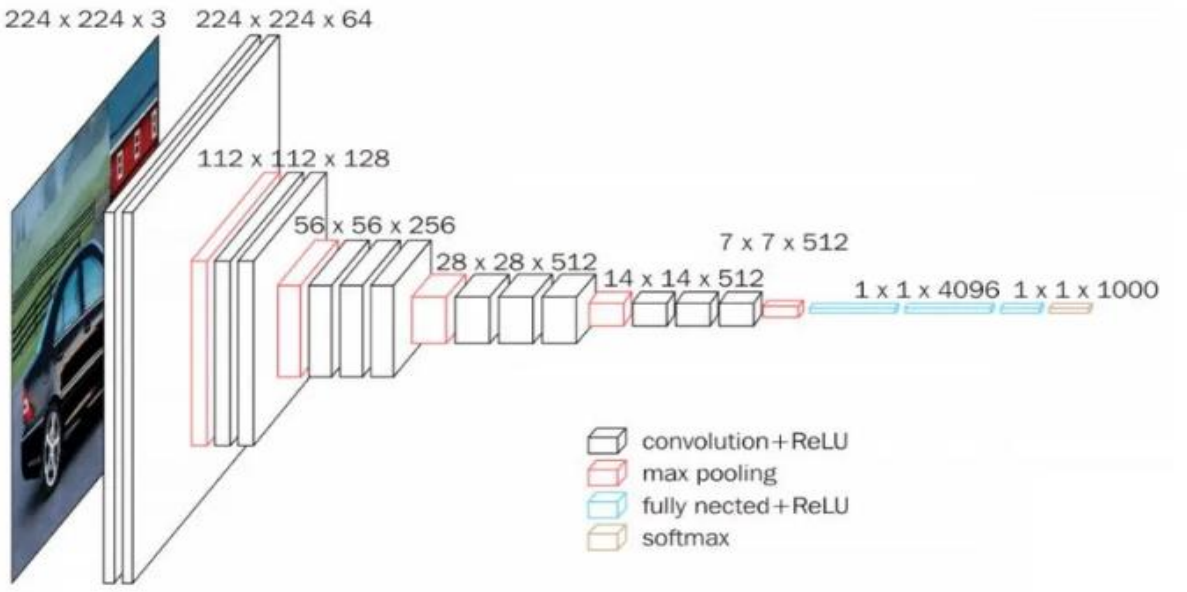
- **The Model Evaluation**
    - Validation Loss: **0.3472**
    - Validation Accuracy: **89.76%**
    - Test Score: **0.7753 (77.53%)**

❖ Model 2: "Vgg16 Model"

- **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 architectures.

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

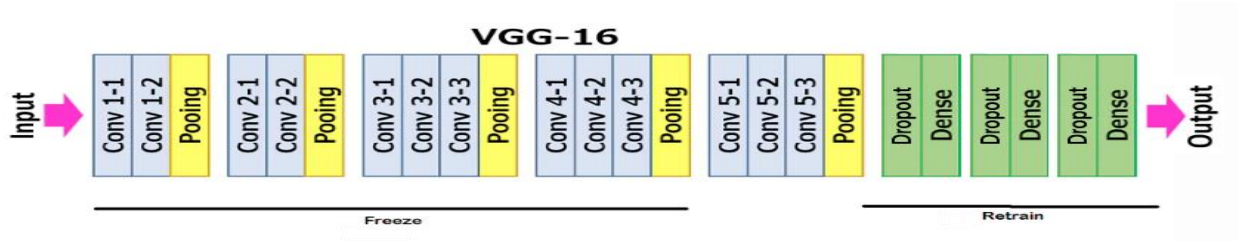


- **VGG16 Transfer Learning Approach**

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.

- **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.
  - Two Fully Connected (FC) layers follow a stack of convolutional layers: the first has 1024 channel and the second 256 channel, the final layer is the softmax 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 **6** with a softmax activation function.



- **The used Techniques and Hyper Parameters:**
  - Train-Valid Split (90% to 10%)
  - Image Data Generator with Data Augmentation
  - Early Stopping Callback **(monitor='val_loss', mode='min', patience=20)**
  - Compile Function **(Adam Optimizer with eta=10^{-4})**

- **Batch Size = 16**
- **Epochs = 60**

- **The Model Evaluation**
  - Validation Loss: **0.1830**
  - Validation Accuracy: **91.89%**
  - Test Score: **0.8659 (86.59%)**

Models 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 must 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.
- ✓ VGG16 Model with fine tuning gets better results than CNN Model.

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

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

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

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