Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels

UNIVERSITAT POLITÈCNICA DE CATALUNYA

MASTER THESIS

TITLE: Convolutional Neural Networks for classifying studio/non-studio frames in TV news programs

MASTER DEGREE: Master's degree in Applied Telecommunications and Engineering Management (MASTEAM)

AUTHOR: Alexis Nevot Ezpeleta

ADVISOR: Francesc Tarrés Ruiz

DATE: July, 6th 2020

Title: Convolutional Neural Networks for classifying studio/non-studio frames in TV news programs

Author: Alexis Nevot Ezpeleta

Advisor: Francesc Tarrés Ruiz

Date: July 06, 2020

Abstract

This project has studied an unusual topic on image recognition and classification that can have numerous utilities on multimedia content. The approach studied on this project has been scenes recognition on TV news programs, which is a good starting point to distinguish scenes on visual content displayed on TV. The work in this project can be the first steps to segment scenes on movies, series, documentaries, etc…

The methodology used to build a program capable to recognise and classify images is Deep Learning, a subset of Machine Learning based on artificial neural networks that learn from representations of data obtained after performing some operations on the input images. As in Machine Learning, a training process with multiple example outputs is necessary to make the system capable to figure out the rules to obtain those outputs (categories) from the given inputs (frames). To build an image recognition system, the most suitable framework (previous comparison has been made between some of them) has been installed to code the operations that lead to the image classification and the results to evaluate its performance.

Another important part to complete this project is the data collection and management, the data volume has to be large and diverse in order to increase the number and type of patterns or features learnt from the representations, and therefore, to increase the system's capability to generalize. It also has to be managed properly, in order to not difficult the learning process with additional, human errors.

The last part of this project consists on training the classifier in the different architectures, check which input parameters lead to the best performance and compare the results to decide which of them is the most appropriate to classify TV news program images. After the training is finished, the system must be tested with never-seen data in order to check its real performance and obtain a proper evaluation.

CONTENTS

INTRODUCTION

This project’s objective is to classify images, in particular, frames belonging to TV news programs. The approach performed in this project could be the first step in multimedia applications, to provide the capability to search specific scenes in multimedia content.

The methodology used for that purpose is Deep Learning, which is a subset of Machine Learning and AI. DL has similarities with ML in terms of learning rules, a new programming paradigm that suits perfect to automate known tasks, but the difference is that DL is a methodology capable to learn the features coming from the representations obtained in the Convolutional Neural Networks, a very effective tool to recognise and classify images. Its current hype is increasing its popularity, attracting the interest of more people that introduce new topics susceptible to use image classification thanks to the possibility of transferring the learning obtained with high-performance equipment capable to compete on image recognition challenges.

The first step is to obtain the necessary number of frames to train (learning process) the system that performs the classifications. With the collection of a data set, the next step is to build a system that, after the training process, is capable to classify the frames corresponding to the provided categories. To use the functionalities of the Convolutional Neural Networks, a DL framework must be installed in order to load all the images, do the training process and obtain the classification results. The choice of the framework is subjective and normally it depends on which is the most suitable for the user, and for that reason a comparison of different frameworks has been made on the second chapter of this project. Apart from the frameworks, a common element in frameworks has been compared: the optimizer, which is important to minimize errors during the classification task. The architectures used to do the classifications are also explained in Chapter 2.

After collecting enough videos of the newscasts with the methods explained on chapter 3, these videos have been sliced into frames to build the dataset. Once the dataset is built, and, in order to maximize the system’s accuracy, the frames have been properly tagged with their corresponding categories, ensuring not additional, human errors worsen the classification results. Usually, not all frames will be correctly classified, reason for which is necessary to prepare properly the data and add some functions to maximize the effectiveness of the training process. To improve the system’s performance, some techniques in Convolutional Neural Networks that have proved to be effective will be used to overcome the problem of using a small dataset, which is the case of this project.

With the data collected, the system can begin the training and display the results: In the first place, the validation results with known data will give an idea of how accurate is the system and the overall error during the classification, and the last evaluation, which consists on classification of never-seen data will reveal how efficient is the classifier. The validation results allow to tune some input parameters with the objective of improving the results in classifications (validation and test). The results obtained in this work are the best that the dataset and the training parameters allow.

The results obtained with the tested architectures show that the results, and therefore, the generalization ability is better with shallow architectures, as stated by the Occam's razor principle[1].

The structure of this work is described here:

- Chapter 1 explains the corresponding field that contains the methods to analyse the images and get the representations from which it can perform the classifications.
- Chapter 2 is focused on the environments from which image recognition and classification can be performed, explains which functions have been used, some of the available optimizers and the description of the architectures used to do the classifications.
- Chapter 3 explains how the videos of the TV news programs are obtained, and from these videos, the frames to build the dataset to train the system, along with the proper management that sorts the data to ensure we obtain the expected results from the classification task.
- Chapter 4 shows and explains the results obtained after using the different architectures coming from the ILSVRC contest.

CHAPTER 1. CONVOLUTIONAL NEURAL NETWORKS

This chapter explains the general architecture of a Convolutional Neural Network (CNN), its functions and the background that compose the full system that takes as input the images of the dataset and performs the classification. First of all, we explain the methodology used to build the system: Deep Learning (DL) and how this methodology has been used in this project to reach its main goal.
The next section explains the set of operations that compose the core of DL because this part comprises some concepts of Artificial Intelligence (AI) and the way the system learns.

The last section explains the DL background in terms of hardware and programming, the evolution of programming paradigm and some recent history of DL, topic used to introduce the architectures used in this project that will be explained extensively in Chapter 2.

1.1. Deep Learning and architecture overview

The goal of this project is to build a system capable to classify different types of scenes in TV News Programs, those taken at the TV studio and those taken from video reports. We will call these frames simply as *studio* and *non-studio* frames.

One of the most popular methodologies to classify and recognise images is Deep Learning (DL). Today, most Deep Learning architectures for image recognition are based on Convolutional Neural Networks (CNN) to learn representations of data at different levels of abstraction. Actually, the depth in a neural network is related to the number of layers used to represent the information of the image at different levels of detail. In general, first layers represent a low-level abstraction of the original image such as lines, corners, edges, etc.; higher layers usually represent higher-level interactions between images' objects. NN name comes as reference to neurobiology despite there is no evidence that the human brain has a similar learning method [2].

Despite DL has achieved more breakthroughs apart from image recognition and classification, these will not be commented here because they are not in focus within this project. Before going into details about CNNs, an architecture overview is shown to provide a bird's eye-view [3]. In general, CNNs stack these layers to conform the next structure:

- Input: The incoming information that handles this layer are pixel values of the image with the three colour channels RGB.
- Convolution: These layers compute the output of neurons connected to local regions in the input or other layers.
- MaxPooling: Necessary to downsample extracted images after convolutions for computational reasons.
- Fully Connected (FC): Required layer to perform the classification, with a neuron per class.

Another layer is necessary first to convert the convolution format to the FC in order to get the classifications, the flatten layer, explained in section 1.1.4.

The elements to describe in neural networks are listed here:

- The layers and the combination of these to create the network.
- The input data: Original input images to classify.
- The Loss function: Feedback about predicted and real categories.
- Optimizer: Determines the algorithm for adapting the parameters in the learning procedure.

1.1.1. Convolution operation layer

In order to recognise features in images, each layer extracts many representations (also called *feature maps,* from now on, the word *feature map* can mean either the original input image or a feature map) from a single input image by sliding *convolutional filters* (also called *kernels*) across its width and height.

From a single input image, a feature map per filter is obtained, and to bring some clarity, a main parameter in convolution layers is the number of filters to apply, in the case, using 64 filters will output 64 feature maps, different with each other, from a single convolution layer. Filters are small matrixes with values *(weights)* designed to detect objects, features, shapes[4]… that are used in Image processing to sharpen, detect edges, blur images, estimate vertical contours…

The size of these filters is usually 3x3, 5x5 pixels, this feature allows learning patterns in any position of the image, thanks to the values of the filters' weights, and depending on the image dimension, the filter size may cause a dimension reduction on feature maps during successive convolutions due to border effects [2]. This way of learning reduces the number of total weights, which means as well a reduction of the number of parameters to process, and therefore, the number of necessary images to learn.

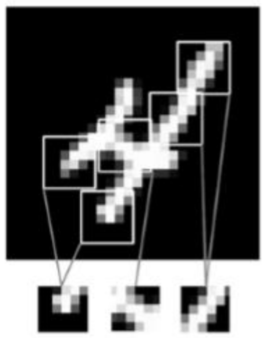


Fig 1.1 Example of convolution operation on number 4 (MNIST database)

In the previous example about convolution, there are filters sliding throughout the image with their pre-set values and indications about which features of the image are being learnt. To provide information about filters and their use, an example is shown:

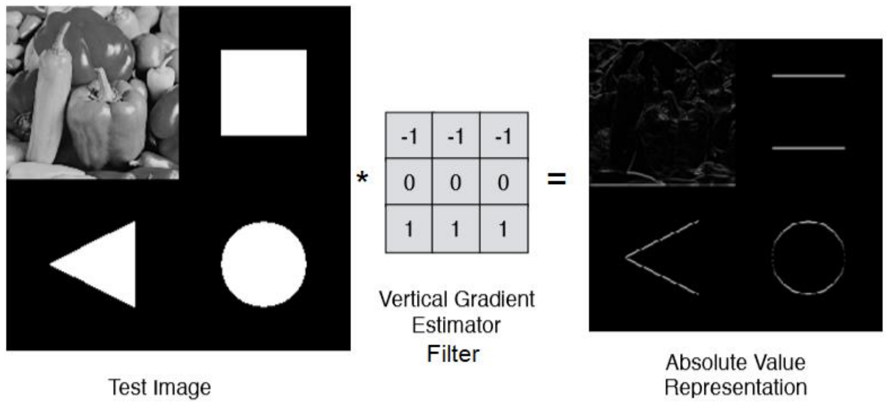

Test Image
*
-1 -1 -1
0 0 0
1 1 1
Vertical Gradient Estimator Filter
=
Absolute Value Representation

Fig 1.2 Example of filtered grey image

Computationally speaking, an image is a grid of pixels, which values ranges from 0 to 255 in three channels corresponding to the RGB model, but before managing values on that range, these will be first normalized between 0 and 1 for convenience during the processes yet to come. The filters slide over the image carrying out an element-wise product and sum of the filter matrix with the part of the input it is currently on (see [4]). Each layer applies multiple filters with initial small random values of their weights [5], so the output will be far from expected, but with every example processed the weights are adjusted a little in the right direction, reducing loss score, a term that will be explained in section 1.1.3. These weights are adjusted gradually depending on a feedback signal in order to minimize the Loss function, which is the main objective of the learning process, also called *training*, a mechanism that makes Machine Learning and Deep Learning systems capable to distinguish and classify information.

1.1.2. Max Pooling operation layer

This operation has the main objective of downsampling the feature maps. To avoid that downsampling may introduce a loss in significant information, it is convenient to slide windows over the images and output the maximum value of each channel. This concept is similar to convolution because a window slides throughout all the images to perform the Max Pooling operation (see [6]), but instead of transforming patches via element-wise product, the transformation is carried out by extracting the maximum value of the pixels contained in the Max Pooling window with size 2x2 pixels. The reasons to downsample the images are simple:

- To reduce the number of coefficients per sample and so, the computational load of the network. Too many coefficients suppose *overfitting*[7], no ability to generalize when recognising learnt features if these appear with slight differences.
- To eliminate spatial hierarchy when learning features, this means recognising patterns or features regardless their position and orientation.

Another difference with the convolution operation is the number of *strides*: in the convolution operation, once the transformation is completed, the next step is to repeat the same operation in the next column of the grid of pixels, that is stride 1. After the Max Pooling operation, the same operation is repeated skipping one column, being the stride in this case 2, this operation supposes an advantage respect to FC layers, in which all neurons are connected and therefore, more parameters have to be processed.

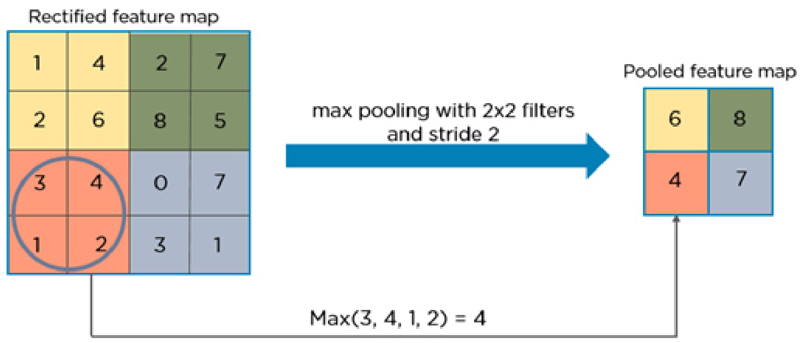


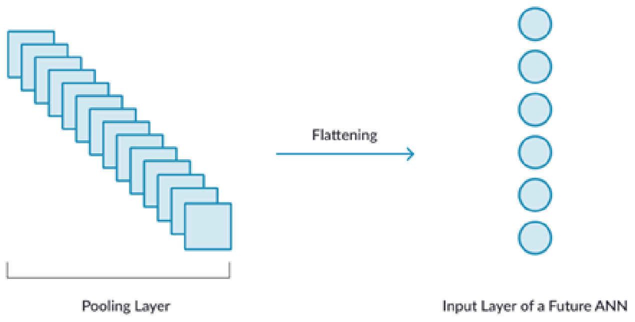
Fig 1.3 Max Pooling scheme

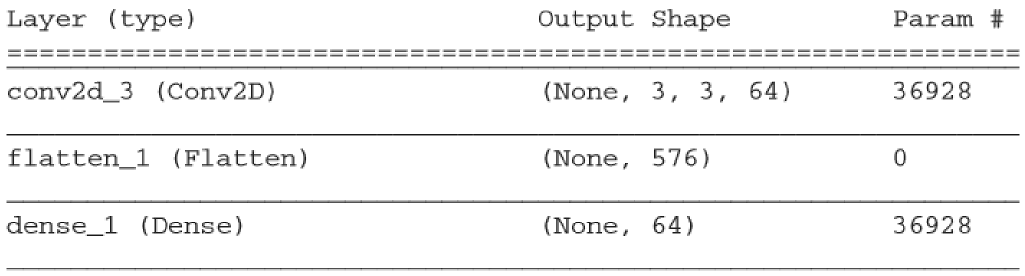
The target of MaxPooling is to extract the pixel with maximum value because the features recognised during convolution tend to be encoded over these pixels with maximum value, which recalls a concept called *maximal presence* [8]. Not all type of pool operations use this mechanism to downsample images. There are variations such as *average pooling* that also downsample the feature maps but instead of extracting the pixel of maximum value of its sliding window, it calculates the average value of all the pixels in the window and outputs that value, but the truth is that there are no alternatives to max pool that work better. In comparison, the information obtained by MaxPooling becomes diluted if the downsample method is average pooling.

1.1.4. Flatten layer

Since the layer in charge of classification is FC type, it won't admit as input the multiple feature maps delivered by the MaxPooling layer, which has a similar format to convolution layer, but with height and width reduced. Actually their output format is: *(height, width, number of feature maps)*. This format defines the feature map dimensions and the number of feature maps obtained after the convolution operation in which multiple filters have been applied.

But this kind of format cannot be an input to a FC layer, the only format FC layers admit is 1-D type, the number of elements to process. It is necessary then to convert this format *(height, width, number of feature maps)* to this *(height x width x number of feature maps)*, but before feeding the classification layer, all the delivered data coming from previous layers must be converted to densely connected format [2], where the numbers of parameters to process is the same. The appropriate layer to perform this task is the flatten layer. (Fig 1.4) shows the different processes of formats (shapes) and number of parameters when going from convolution layers to FC layers.





```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 3, 3, 64)          36928
_________________________________________________________________
flatten_1 (Flatten)          (None, 576)               0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                36928
_________________________________________________________________
```

Fig 1.4 Flattening scheme

1.1.4. Loss function and optimizer

As mentioned in previous sections, filters' initial weights are random, so probably, the first classifications will be far from correct. The mechanism in charge to compare the networks' predictions obtained at its output with the real category is the Loss function [5]. Of course, we are the ones who decide to which category belong the input images, which means we are responsible of the images matching their category, in this case, as example, we have to check carefully in the dataset that a picture of just a landscape ('Video_New' category) is not tagged as 'Reporter' for example.

Going one step beyond, the mechanism used by CNN to learn which representations are mapped to the right category and which are mapped to a wrong category underlies on the filters' weights: with their current value, the network performs Convolution and Max Pooling operations enough times to map learnt patterns and features to categories to generate a prediction. This prediction and the true target (real category) are introduced as input to the Loss function (also called *Objective function*), which returns a Loss score, a value that indicates the network how far is the prediction from the true target.

This score is used as feedback by the *Optimizer* to update the filters' weights every iteration (closer to the true target) until they achieve the *correct* value, those that make the Loss function output the lowest Loss score and therefore, maximizing classification's accuracy [2]. This is the central algorithm in Deep Learning, which is called *Backpropagation* and the continuous updates of the filters' weights is the part in which the network is trained.

Basically backpropagation comprises multiple variables (weights) and one target: To find the values of the weights that minimize the Loss function. These elements call Gradient operation, in this case an algorithm related to gradient operation: (Gradient Descent).

As there is more than one Optimizer in CNN, different types, will be compared in terms of convergence time and accuracy along this document. A scheme about Loss function and Optimizer interaction is shown below in figure (Fig. 1.5).

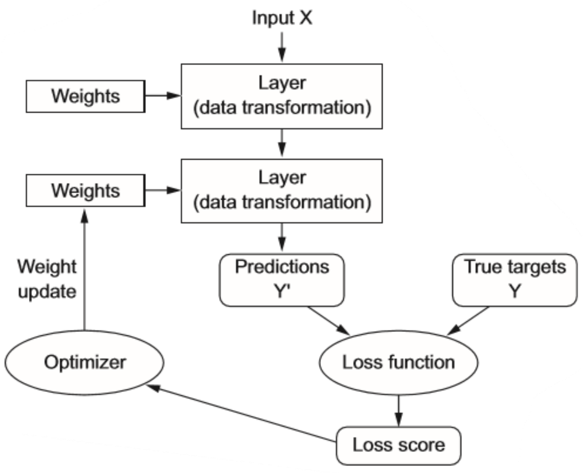


Fig 1.5 Interaction between Layers, Loss function and Optimizer

1.1.5. Activation functions in neurons

An activation function defines an output from a neuron for a given set of inputs, the term activation is inspired by activity in human brains, where different neurons fire, or are activated by different stimuli, but in the case of artificial neurons, the model of a single neuron (McCulloch & Pitts – 1943) contemplates the existence of these elements [5]:

- Inputs (X_1,…, X_N)
- Weights (W_1,…, W_N)
- Bias (b)
- Output (Y)

A neuron can have multiple inputs and multiple outputs, and the relationship between the mentioned elements is the formula:

$$Y = S[\sum_{i=1}^{N} W_i X_i + b] \tag{1,1}$$

For this model, S is the activation function that in order to avoid linearity has this mathematical definition:

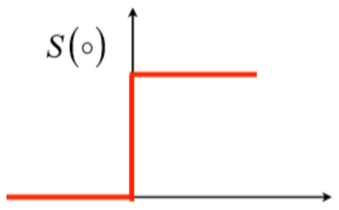


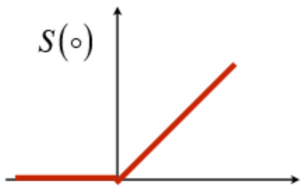
Fig 1.6 Non-linear activation function

However, this model has some drawbacks to carry out images' recognition and classification as intended in this project:

- Binary output
- Weights and activation threshold are already predefined.
- Lacks flexibility.

The neurons in the next layers cannot be fed with the just the output delivered by the neurons from the previous layer (see [2]), if so, we just would get a linear combination and multilayer would be equivalent as a single layer. This output must be defined in a way that depending on its value and the established threshold, a neuron is activated in the next layer or not, an alternative is mapping input signals into output signals needed for the neural network to function. The most common activation functions [9] for image recognition are:

- ReLU (Rectified Linear Unit): mathematically is defined as Y= max (0,X), it is used by convolution layers because it provides the network a fast convergence and its non-linearity allows backpropagation (despite looking similar to a linear function, it is not and its derivative is a step function).



$$S(x) = \begin{cases} x & if\ x > 0 \\ 0 & otherwise \end{cases}$$

Fig 1.7 ReLU activation function

- Sigmoid: Despite being computationally expensive, this function performs better for binary classifications thanks to its soft gradient and the clarity of its predictions for values x > 2 and x < -2, Y values tend to be to the edge of the curve, very close to 0 or 1. The output of this function is the probability to belong to one of the two classes.
- Softmax: All classification problems require a final dense/fully connected layer with one neuron per class to deliver as output the category to which belongs the input frame in multiple categories. This output is the probability distribution to belong to each class, normalized between 0 and 1.

1.2. Convolutional Neural Networks background

Until now, we have explained the operations performed in CNN to obtain the ability of recognising the images and how to classify them correctly, but those operations are not at hand for any commodity equipment, this way of programming is a new paradigm (ML/DL paradigm) compared to classical programming, in which it is possible to run any program in almost any computer. Hereunder, some requirements to run DL programs are explained.

1.2.1. GPU (Graphical Processing Unit)

Unlike classical programming paradigm, in which data is processed according to programmed rules to obtain an output, the ML paradigm [5] (the same as DL) seems slightly different: supplying data and answers expected from that data, the output should be the rules that produced the original answers from the input data, as it is indicated in (Fig. 1.8).

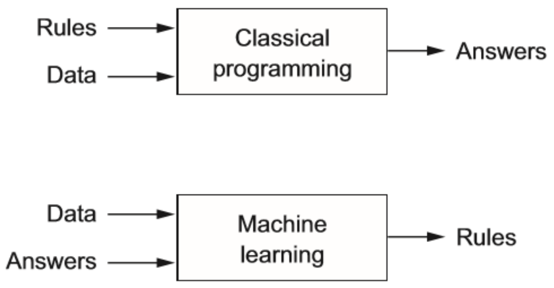


Fig 1.8 Different programming paradigms

Despite it seems a small change, the reality is that obtaining the rules from data and answers has required several years and technological outbreaks. The main difference about these two paradigms is the intelligence: in classical programming it is absolutely necessary to provide specific data with specific steps about what to do according to the programmed rules, and what to do if an exception comes to all expected results. Differently from the ML paradigm, a program coming from this paradigm lacks intelligence. An example could be identifying a reporter: It would be necessary to indicate the presence of elements such a person appearing in the centre of the image, holding a microphone on the hand, or in the suit… Endless indications that make unfeasible to contemplate all possible cases.

In the Machine Learning paradigm (valid for DL as well), the system eventually *learns* to figure out how to make the right decision to obtain a known output for a given input by performing the CNN operations. An important advantage of this, is translation invariance, back to the previous example, identifying a reporter, the system would learn which representations in the different feature maps match the category 'Reporter'. A possible pattern for this category could be the microphone, a not too distant person from the camera… This paradigm conforms the basis of AI, which consists in automating intellectual tasks performed by humans.

Changing from the classical programming paradigm to the machine learning paradigm does not consist only on changing procedures or ways to program, it is necessary to change, at least, the equipment. The classical programming equipment, however, probably is not capable by itself to perform convolutions in which patterns and features are identified and used to classify images. The essential equipment used in DL paradigm to make possible to do all the necessary convolutions over thousands of images and classify them is the GPU [10] (Graphical Processing Unit), which is a popular element in gamers' computers: The graphic card.

Another element necessary in the ML paradigm is storage, not only the capability to perform all the operations in the CNNs. The number of images to build the data set can be about tens of thousands and with a proper resolution, only the dataset used in this project reaches perfectly 20 GB, which makes a huge difference respect the classical programming paradigm.

Apart from a fast CPU to process data and to achieve the computational power required to run DL models, companies like NVIDIA or AMD have focused their efforts on developing fast, massively parallel chips to render complex 3D scenes on commodity equipment in real time. In 2007 NVIDIA created a computing platform: CUDA (Compute Unified Device Architecture) with the idea of using the GPU for parallel programing to solve complex problems[].

The last significant difference is the time to see the results. The training process can go from hours to days, which supposes the biggest difference between the two paradigms. The paradigm shift, however, has fostered the acquisition of fully equipped computers with multiple GPUs to keep training during days to move forward in DL, as described below.

1.2.2. ImageNet database

Some years after the creation of the CUDA platform, an image recognition challenge arose in March 2010: ILSVRC (ImageNet Large Scale Virtual Recognition Challenge). ImageNet is a visual database designed to implement visual object recognition software with currently more than 14 million of images and more than 20.000 categories. It runs an image recognition contest every year in which the winner is the one that detects, classifies and locates correctly the highest number of objects and scenes over a dataset provided by ImageNet.

A very important milestone in the DL history is the victory of the CNN AlexNet (created by Alex Krizhevsky), in the ILSVRC in the year 2012, it is considered a milestone because the classification error rate decreased a 61% respect to the previous error rate in 2011 (from 0.26% in 2011 to the 0.16%) [12], an important reduction that has been considered a very important breakthrough in DL. Checking the top 5 ranking from that year, the second best error rate was a 10.8% higher.

The importance of ImageNet in this project is not about DL or CNNs history, the architectures of CNNs used along history in all the contests, contains several convolution, MaxPooling, fully connected layers that are necessary to manage the quantity of parameters that are necessary to recognise scenes and objects of more than 20.000 categories over more than 14 million of images nowadays. As can be deducted, commodity equipment is not prepared for such daunting task; the computation necessary to compete in contests such as ILSVRC requires dedicated computers with multiple GPUs training for a long time, a requirement that is not feasible to complete this project.

Notwithstanding, the progress of those trainings can be saved and transferred to other networks, advantage that has been taken from that feature, to load those parameters in the different architectures used in this project.

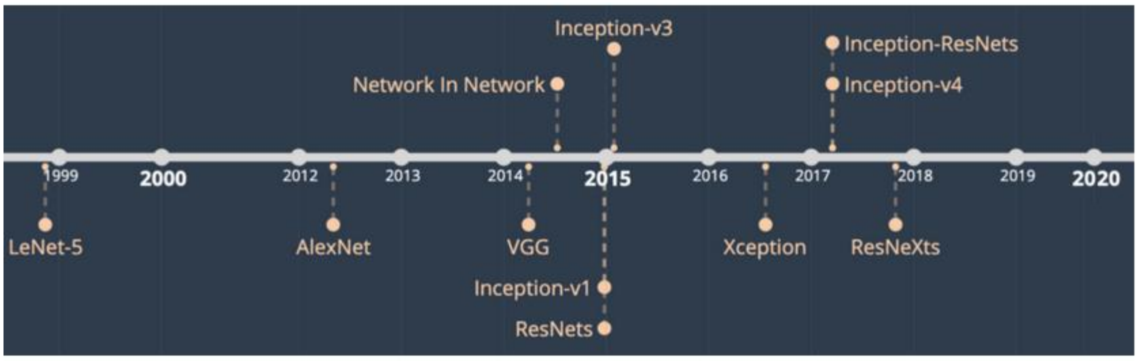


Fig 1.9 Small overview of ImageNet architectures

(Fig. 1.9) shows the winning architectures of all ImageNet challenges over the years as introduction, because most of them will appear in this document as tested architectures to perform the classification task.

The equipment used for this project is described in Annex A1 (Hardware specifications).

CHAPTER 2. KERAS – DEVELOPMENT TOOLS

This chapter describes the environment used to build the different CNNs in this project, the description explains the code that has made possible obtaining the results of the image classification, but before explaining the environment, the first section here under compares different DL frameworks available to create neural networks.
The code to build the network in Python will be detailed, beginning with the load of architectures and ending with the classifications [2]. This comprises creating the models with layers, transforming the original input images into data that can be fed to the CNN, compiling the model, training and finally displaying the results. Despite being in a new programming paradigm, problems that move us further from our objective may arise, but some strategies to cope with them will be explained as well.
Different loss functions and optimizers will be explained in detail to analyse which is the most suitable to perform the image classification of this project.
Also the different network architectures will be explained and detailed in this chapter, the comparison about the results will be shown and explained in Chapter 4.

2.1 Keras, TensorFlow, PyTorch and Caffe comparison

More businesses and companies are moving to the use of AI in order to automate tasks and improve their performance, bringing intelligence to machines thanks to the use of ML and/or DL systems. Depending on the technology they work with and the desired purpose, the choice of the framework to develop the system is crucial. Normally the perfect framework for each case is chosen in function of results, fast business and ease to deploy. In the case of image recognition and classification, the necessary technology has been explained, remaining only the environment in which to build the program that loads images and returns their classification.

Four DL frameworks are compared in a table to see in a clear, quick way which are the important features to take into account before beginning to program a full CNN such as API level, ease to code, architecture, debugging, support[13]…

Since the four frameworks can perfectly lead to the same objective, it is normal that some functionalities are shared amongst them, as the case of Keras, which has an user friendly API, but doesn’t handle low-level functions, therefore, frameworks with low level API (specifically TensorFlow) have to be installed to provide Keras with these functionalities (the name of this concept is *backend)*. With low-level functions we mean mathematical operations such as generalized Matrix-Matrix multiplication and NN basics such as the element-wise operation in convolutions.

Table 2.1. Framework comparison

	Keras	*TensorFlow*	*PyTorch*	*Caffee*
API level	High level	High & low level	Low level	Low level
Speed	Slow	For high performance	For high performance	Fast (1ms / image)
Architecture	Simple	Complex	Complex	Complex (big networks)
Coding	Single line code	Reduced size of model with high acc.	Complex	Complicated but extensible
Debugging	Not frequently needed	Difficult	Better debugging capabilities	Not complicated
Support	Small community support	Backed by community tech companies	Stronger community support	No commercial support
Datasets	Small	High performance models	Large datasets	Small and large
Popularity	High due to its simplicity	Highest popularity	High	Low

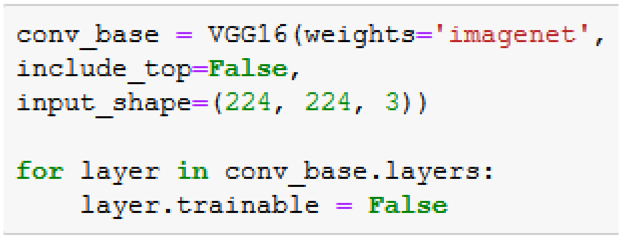
The main features of these frameworks are exposed in the table and after considering them all, here is the explanation of why the choice for this project has been **Keras**. The main reason is that it is the perfect framework to begin working with CNNs due to its simple architecture and high level API. This way, pre-trained architectures can be easily loaded to obtain results in short time to check them. Another important feature about Keras is the dataset size: Since it uses TensorFlow as backend, it results to be slow for big sizes, and the dataset of this project has been built from scratch, reason for which is quite difficult to reach sizes of tens of thousands (see ImageNet or OpenImage datasets). Each category trains, validates and tests with 1000 frames per category, but small datasets can be augmented with a technique called Data Augmentation to achieve similar results to big sizes, explained in section 2.2.3.

2.2 Building the CNN

This section shows and explains the code used to build the CNNs. The operations explained in Chapter 1 will appear here with the proper parameters to reach the objective, the part of code explained here is the basic code that allows to perform the necessary operations to perform the classification:

- Load the different architectures
- Add our own classifier with our categories
- Load and convert the frames to the proper format
- Train and validate the model
- Test the model with predictions of never-seen data

2.1.2. Loading pre-trained architectures



```
conv_base = VGG16(weights='imagenet',
include_top=False,
input_shape=(224, 224, 3))

for layer in conv_base.layers:
    layer.trainable = False
```

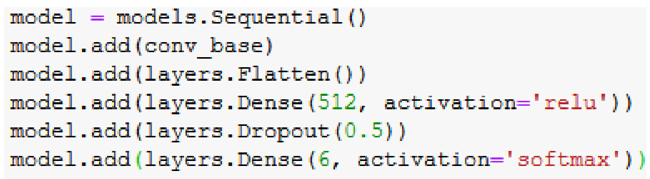
Fig 2.1 Convolutional base load

This is the part in which the different architectures are loaded in Keras, *conv_base* will play the role of all architectures tested in this project: VGG16/19, Inception V3 and Xception, it is a common approach for computer vision problems in which datasets are small. By setting *weights* to ‘imageNet’, we indicate we want to use the networks’ values obtained from the training with multiple GPUs for a long time, using ImageNet dataset.

It is also important not to train the whole network, for that, the classifier is removed (include_top=False) and it is necessary to *freeze* all layers from the pre-trained CNN (layer.trainable = False) as shown on (Fig 2.1). The reason to not use the original classifier from the architecture is to make sure that the representations learnt are specific to our classes and data, which are not as specific as in the pre-trained classifier [2].

All the input images are resized to square shape with height and width values = (224, 224) for all architectures.

2.2.2 Adding classifier on top of architectures



```
model = models.Sequential()
model.add(conv_base)
model.add(layers.Flatten())
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(6, activation='softmax'))
```

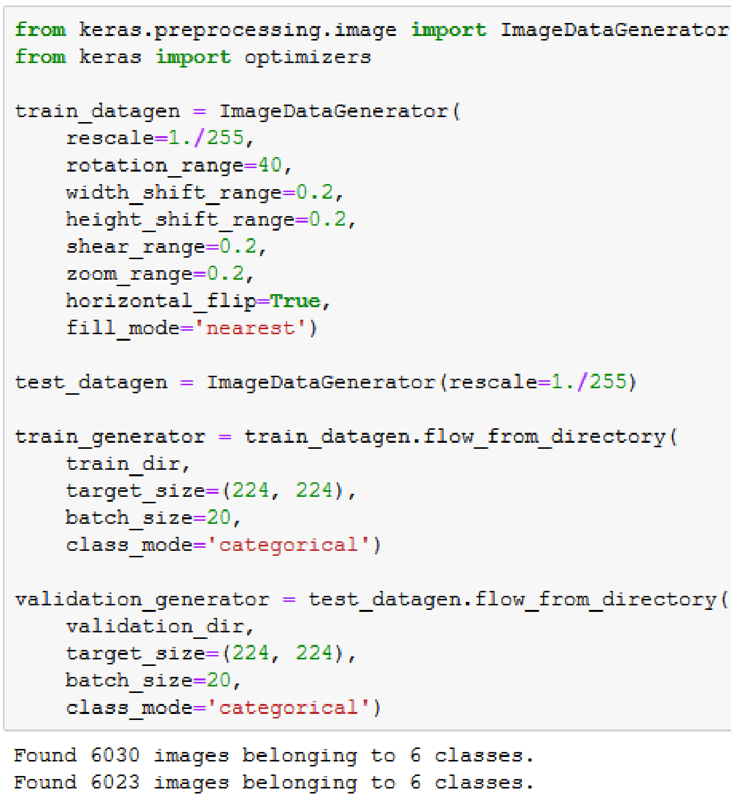
Fig 2.2 Classifier added to convolutional base

We add this classifier on top of the architectures, which converts the format coming from convolution layers into the proper format to perform the classification. The Dropout layer is to randomly select and deactivate a number of neurons (0.5 factor is to deactivate half of the neurons), leaving the rest of neurons solving the problem. This is one the measures to cope with *overfitting*, which means a network lacks capability to generalize. [5]. Note that the last layer has one neuron per class, with *softmax* activation for multiple classes, which will output the probability distribution, selecting the class with the highest probability.

2.2.3. Generating data to CNN from original input images

From this point, it is necessary to convert the dataset images, stored in their corresponding folders, into a format that can be fed as input to CNNs. The functionality in Keras able to do that is the ImageDataGenerator class, which requires some arguments depending on the use of the data. Note that the validation and test data will only be converted to the proper format and their pixels value will be rescaled from the interval [0 - 255] to [0-1]. The training data on the other hand, will be *augmented* with extra arguments on the ImageDataGenerator class to generate additional training data to increase the model's accuracy with small quantity of data. Data is augmented according to the parameters provided to the generator with ranges of zoom, rotations, flips, displacement (see [14])…

This concept is called *Data augmentation*, which is the second measure to cope with overfitting, by introducing modified frames that seem different from the original frames.



```
from keras.preprocessing.image import ImageDataGenerator
from keras import optimizers

train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest')

test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(224, 224),
    batch_size=20,
    class_mode='categorical')

validation_generator = test_datagen.flow_from_directory(
    validation_dir,
    target_size=(224, 224),
    batch_size=20,
    class_mode='categorical')
```

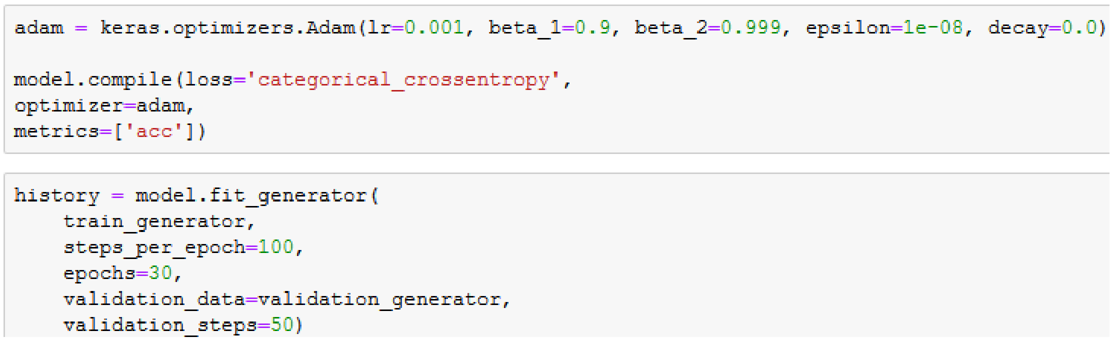
```
Found 6030 images belonging to 6 classes.
Found 6023 images belonging to 6 classes.
```

Fig 2.3 Converting frames to CNN format

2.2.4. Compiling and training the model

With all this preparation, it is the moment to configure an optimizer, a Loss function to minimize and the metrics to measure. Different optimizers have been used in this project are explained in section 2.3. Once the optimizer and the loss function have been compiled, the training and validation process (model.fit) can begin. The duration of this process depends on the number of epochs, it must be high enough to monitor accuracy after convergence.

All models have been trained for a long enough period where the models converge and, thus, a bird view of the global behaviour can be taken.



```
adam = keras.optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-08, decay=0.0)

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

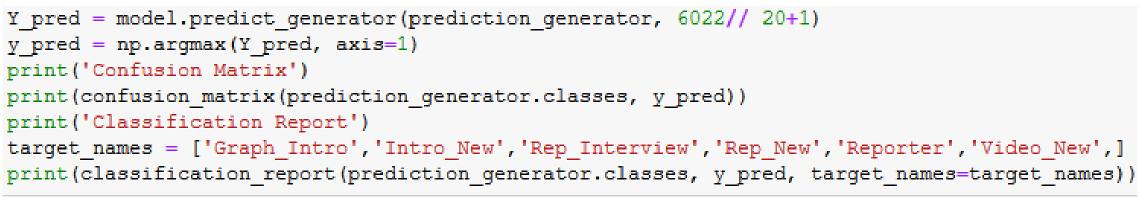
```
history = model.fit_generator(
    train_generator,
    steps_per_epoch=100,
    epochs=30,
    validation_data=validation_generator,
    validation_steps=50)
```

Fig 2.4 Configure optimizer and train the model

2.2.5. Generating predictions

After obtaining the results from the training and validation process, the next step is to evaluate the model with never-seen data. Using a validation set allows to tune the model by changing input arguments, number of layers, number of epochs, etc… for the cases in which the results are not good enough, but to have a complete evaluation of the model it is important to distinguish if it performs well both in seen as in never-seen data.

The best way to show the evaluation of the system is placing each prediction in a *confusion matrix* [15]. These matrixes will be shown in Chapter 4, but basically, each column represents a category, and the rows represent the predicted categories. This tool allows knowing which categories confuse the system because it shows the number of mismatches and in which category is classified. The ideal confusion matrix is diagonal with the number of elements of each category, because it means that the predicted categories coincide with true categories. (In this case, a diagonal matrix with the value 1000).



```
Y_pred = model.predict_generator(prediction_generator, 6022// 20+1)
y_pred = np.argmax(Y_pred, axis=1)
print('Confusion Matrix')
print(confusion_matrix(prediction_generator.classes, y_pred))
print('Classification Report')
target_names = ['Graph_Intro','Intro_New','Rep_Interview','Rep_New','Reporter','Video_New',]
print(classification_report(prediction_generator.classes, y_pred, target_names=target_names))
```

Fig 2.5 Evaluating the model with never-seen data

2.1. Loss function and learning algorithms

As mentioned in Chapter 1, to perform automatic classification the system must learn to classify correctly through the training process, reducing the Loss score provided by the Loss function, which basically compares the network's prediction with the true value and outputs a Loss score. This function has to be minimized in order to achieve maximum accuracy, which is the Optimizer's task.

To accomplish this task, the Optimizer manages the Loss score and the filters' weights, which are updated depending on that score, the lower the score, the softer the updates in the weights. The Optimizer looks for the values for which the gradient of the Loss function is zero (as close as possible), but on the basis that all Optimizers have the same goal, there are different types of Optimizers with different ways to find the values that minimize the Loss Function. An important parameter for the Optimizers is the Learning Rate (LR), which is decisive to reach convergence: If LR is too low, reaching convergence can take forever, if it is too high, the algorithm will never converge, for that reason it is important to adapt the LR while searching the minima. The most basic optimizer is Gradient Descent (GD), but achieving the proper values is not feasible when handling millions of parameters, so a faster version, Stochastic Gradient Descent (SGD) is introduced as basis of the Optimizers used in this project:

2.3.1. Loss Function

First of all, we need to check the problem to face, in this case it is a multiclass classification problem, so the corresponding Loss Function is [5]:

$J(y) = \sum_{i=1}^{M} y_d \cdot ln y' + (1 - y_d) \cdot ln(1 - y')$, (y_d is the true target) **(2,1)**

$y' = S(W^T \cdot X)$ (y' Is the predicted value) **(2,2)**

$\mathbf{W}_{k+1} = \mathbf{W}_k - \mu \nabla \mathbf{w} J(y)$ Is the expression for weight learning **(2,3)**

As can be observed on these equations, the changes that can lead the Loss Function $J(y)$ to its minimum value, will depend on how good or bad are the predictions obtained (y'), and the predictions' quality depend at the same time on the weights learnt through the Backpropagation mechanism, that will be constantly updated during the training process. To sum up, the Loss Function values depend exclusively on the weights.

2.3.2. Learning algorithm (SGD)

Similar to GD, the only input value for this optimizer is the LR, taking big steps when the predicted value is far from the true target (high LR) and small steps when the predictions are close to the true target (low LR), the difference is that redundant samples form clusters (minibatches) to implement the mathematical calculations with the gradients (see [16]). This feature allows the Optimizer to work in just that subset of data, instead of all the information, which results faster because the number of steps are reduced in a factor corresponding to the number of minibatches or clusters (faster than GD). SGD does not compute the exact derivative of the Loss Function, it is an estimation for each batch, therefore, the gradients 'oscillate' instead of moving uniformly, causing that not all steps to the minima are in the right direction. Moreover, working on minibatches provide more stable estimates of the parameters in fewer steps.

2.4. Optimizers

The optimizers perform mathematical operations to reach the minima as fast as possible, despite they use different ways to find the minima, the objective is the same. The different optimizers [17] used in this project are described below:

2.4.1. SGD with Momentum

Instead of taking the direction on the maximum gradient, the strategy of this method, with momentum, is moving average of the gradients and the value from previous gradients. With this technique, the optimizer tends to oscillate, and the way it deals with these oscillations is averaging the estimated gradients (weighed averages), that would be closer to the original function, accelerating them in the right direction. The way to the minima is described by the formula:

$$m_t = \beta m_{t-1} + (1-\beta)\, g_t \qquad \textbf{(2,4)}$$

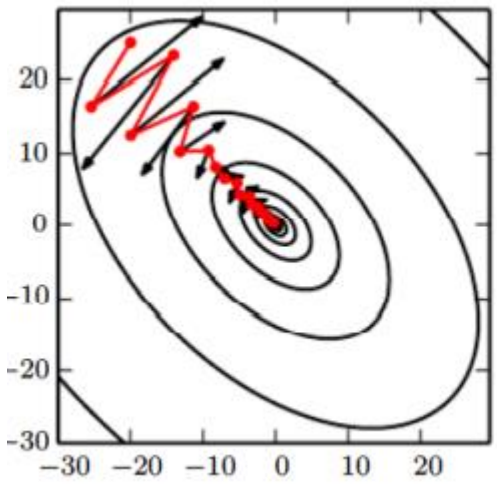


Fig 2.6 SGD with Momentum

2.4.2. RMSprop

Similar to SGD, the main parameter is the LR, but RMSprop looks for a fast way to the minima, dumping oscillations not routed in the right direction. The method is to penalize the updates of those parameters that cause the Loss function to suffer wide oscillations, with the objective not to adapt quickly to such changes for the weights' values. The advantage of the updates that are correctly routed to the minima is that they do not suffer any penalty and therefore, there will not be big oscillations in the way to the local minima.

$$v_t = \rho v_{t-1} + (1-\rho)\, {g_t}^2 \qquad \textbf{(2,4)}$$

This formula indicates how the gradient are treated: The square of the average gradient is computed, multiplied by $(1-\rho)$ and added the previous exponential average (v_{t-1}) of the gradients to obtain the result. The reason to do it in this way, is to weigh the more recent gradients respect the previous.

RMSprop can also deal with large datasets thanks to the use of minibatches, averaging the gradients for each weight, and taking all the steps with the same magnitude. The magnitude of the steps can be regularized dividing the gradient by the square root of the mean gradient, reason for which the Optimizer's name is RMSprop, the LR is also divided for a weight that is the average of the magnitudes of the recent gradients that had that weight.

2.4.3. ADAM (Adaptive Moment estimation)

ADAM is an adaptive LR optimization algorithm, computing each LR for different parameters, like RMSprop, it uses squared gradients to scale the LR and computes them individually for each parameter and similar to SGD with momentum, it moves average over the gradients.

ADAM uses estimations of the first and second moment (expectations of the gradients' values, g_t):

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t \qquad \textbf{(2,5)}$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \qquad \textbf{(2,6)}$$

However, developing some iterations in these formulas and assuming that the expectations of the gradients come from the same distribution, it can be checked easily that there is an inherent bias that must be corrected according to the expression below:

$$\widehat{m}_t = \frac{m_t}{1-\beta_1} \qquad \textbf{(2,7)}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2} \qquad \textbf{(2,8)}$$

As these equations show, this optimizer has more hyperparameters than the previous ones, the recommended values are:

- β_1= 0.9
- β_2 = 0.999

β_1 and β_2 are used to average the first and the second moment to scale the LR for each parameter, being η the initial LR. There is also an additional parameter (ϵ) to ensure no division by zero in the weight update:

$$w_t = w_{t-1} - \eta \frac{\widehat{m}_t}{\sqrt{\hat{v}_t}+\epsilon} \qquad \textbf{(2,9)}$$

With recommended value $\epsilon = 10^{-8}$.

2.5. Used CNN architectures

Different CNN architectures have been tested to check which of them provides the maximum accuracy to perform the classification of the newscast frames. These architectures belong to networks that have competed in ILSVRC, obtaining very good results in terms of classification. At the beginning, their first approach was achieving maximum depth in these architectures, but experience revealed that insistence on deep networks didn't always lead to better results. For this reason, approaches with changes on convolution operation and feed-forwarding outputs were successful as explained below.

2.5.1. VGG Net

Presented in ILSVRC 2014, this architecture can optionally use 16 or 19 convolutional layers with 3x3 filters. It becomes progressively simplified in terms of width, using 2x2 MaxPooling filters with stride 2, increasing the total number of filters in the upper layers. Concatenating 3x3 convolution layers, the reception field is increased, and the 3x3 size reduces the total number of parameters. Another important feature of this architecture is the use of ReLU activation on each convolution layer, it makes the decision function more discriminatory, achieving a more accurate classification[5].

VGG Net has become a reference architecture and has contributed to a better understanding on the NNs. As 3x3 convolutional layers are added, each feature depends on a 3x3 region from the original image, in the second 3x3 layer, each output element will depend on a 5x5 region from the original image, on a 7x7 region on the third layer and so on… This way, after each MaxPooling operation, each output will depend on a 14x14 region.

VGG16 number of parameters is 138 million, a large number for a configuration of 16 layers, however, its conceptual simplicity makes this network widely used on multiple applications since it is implemented in the majority of frameworks as pre-trained network on ImageNet. The architecture is shown in (Fig 2.7).

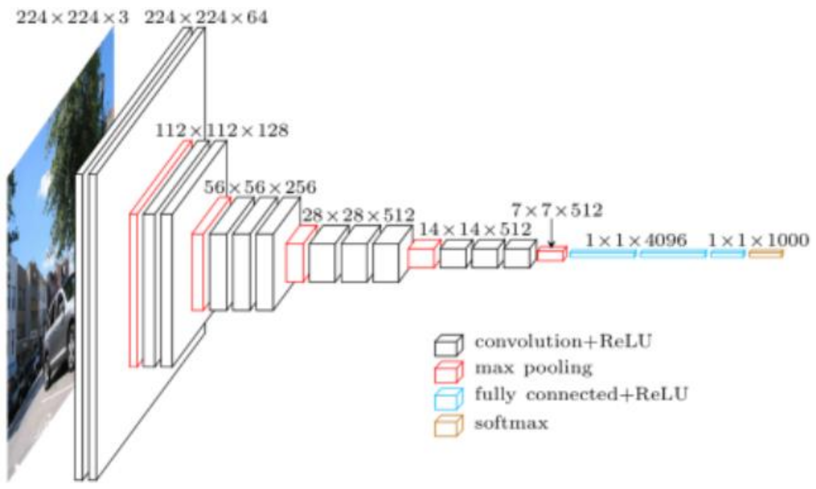


Fig 2.7 VGG Net architecture

2.5.2. GoogLeNet Inception module

GoogLeNet's architecture brought a significant contribution to image classification, thanks to the use of the *inception modules,* an element that simplified the layers design of the neural networks [5]. The inception modules allow the use of simple structures, concatenated in series or parallel, combining different sizes of convolution filters, considering even 1x1 size to not overgrow the number of parameters and difficult the network training.

These blocks combine all the features of the immediately preceding layer, reducing the number of elements and simplifying the number of parameters of the successive convolutional blocks, as shown in (Fig 2.8).

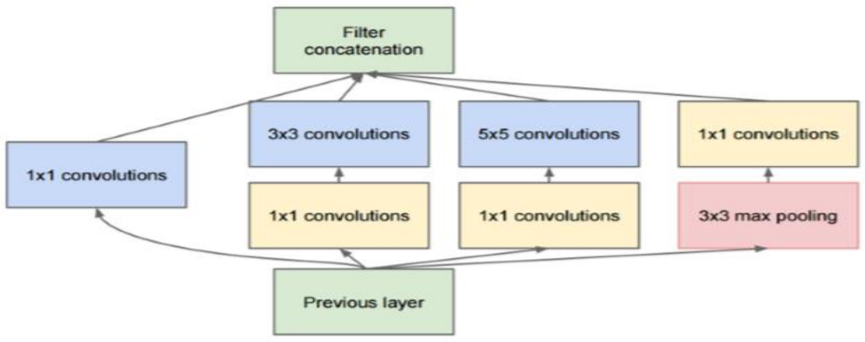


Fig 2.8 Inception module

The 3x3 and 5x5 convolution operations on (Fig 2.8) are only permitted if previous simplifications have been done with the 1x1 filter. The reason of this constrain is to simplify the learning process, increasing the architecture's depth and maintaining its complexity in acceptable levels despite the depth increase, which leads to better results. However, the learning process in this architecture is complex because the errors are backpropagated through numerous layers, which supposes a slowdown. For this reason, GoogLeNet included a couple of parallel replicated outputs in the central part of the architecture, with the idea of moving errors to the first layers and therefore, enhance the weight adaptation to ease the global learning of the network. A figure of the basic architecture is shown in (Fig 2.9)

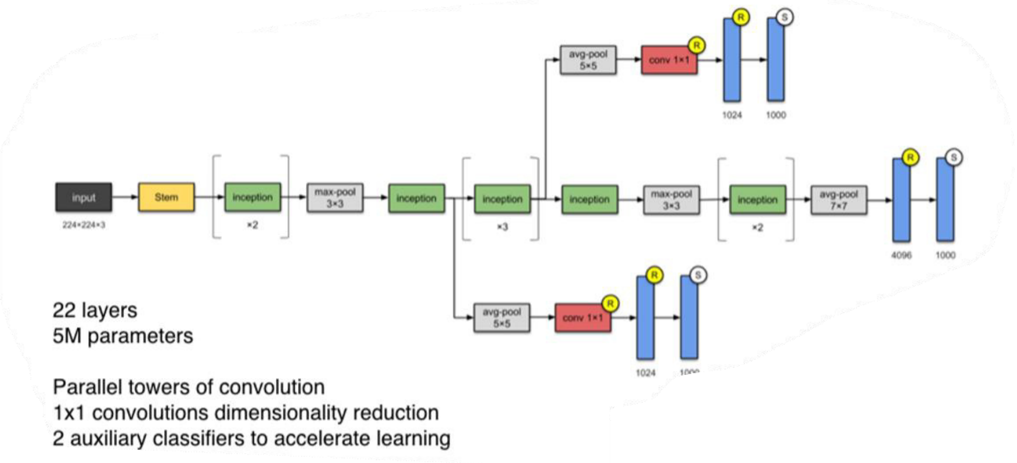


Fig 2.9 Inception architecture

This concept evolved subsequently in more sophisticated models with higher flexibility on its basic inception module, which consists on increasing the number of convolution filters and simultaneously reducing the number of parameters. With a NxN filter is decomposed into a Nx1 filter and subsequently a 1xN filter, with the same number of elements but with a minor number of parameters. The depth of the network can be increased maintaining its complexity in this way.

2.5.3. XCEPTION

This model is similar to Inception but with some computational improvements: the inception modules (eXtreme inception) perform depthwise convolutions, in which a 3x3 filter is decomposed into two filters: a 3x3 spatial filter that is applied to all channels equally and a 1x1 filter, in depth, that is applied to the channels dimension in each layer. The depthwise convolution reduces considerably the number of parameters respect a real 3D convolution, with a different order, performing first the 1x1 convolution and next the 2D convolutions, and between these convolutions, a non-linear element (ReLU) is added. This strategy is shown in (Fig 2.10), being the architecture similar to Inception. [18]

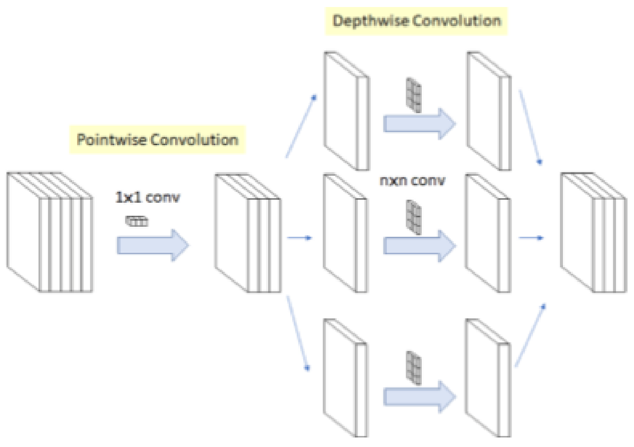


Fig 2.10 Xception depthwise convolution strategy

CHAPTER 3. BUILDING THE DATASET

This chapter explains how the data set has been built, from videos' acquisition to tagged frames with their corresponding category, ready as network's input. In the middle of the building process there has been a proper management and manual classification to prepare the network to reach the ability to classify the frames with maximum accuracy.
This data is the fuel that keeps the network training and it has to be properly analysed to ensure that the data we supply to the network is the right to learn what we want the network to learn, otherwise, errors in classification would suppose sharing common features between categories, complicating the learning process. Another mistake to avoid is sharing identical frames between training and validation: The training results would be great, but the test results would be worse.
The network has to classify correctly never seen data, or in other words, it has to be able to generalize correctly. Specific criteria has been applied during the manual classification to contribute improving its generalization ability.

3.1. Obtaining videos

To build the whole data set, it has been necessary to obtain as many videos as possible of TV news programs. This multimedia content is not at hand and easy to obtain despite videos are broadcasted all over the world on TV and uploaded on the program's website.
The main methods to obtain the videos for this project are:

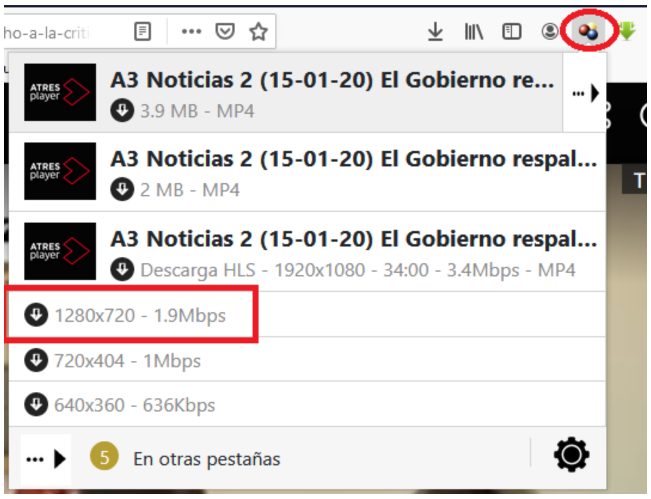
3.1.1. From deferred programs: Video on Demand

In the case of national TV channels, the videos are available on the channels' website, but the download option is not always available, so the solution to obtain these videos has been to install an add-on for browsers that allows downloading videos and different elements in a website: DownloadHelper.

When DownloadHelper detects a streaming source in a web, it displays a menu with multiple options (Resolution – Bitrate):

- Video to download, if a web hosts more than one video, it will be detected as multi streaming source and video names will appear on menu.
- Video Resolution – Video Bitrate.
- Video format: In some cases, there is more than one format available to download.

As can be seen marked in red in figure below, the convention to download the videos for this project have been selected with (resolution – bitrate) = (1280x720 – 1.9 Mbps).


ho-a-la-crit

A3 Noticias 2 (15-01-20) El Gobierno re...
3.9 MB - MP4

A3 Noticias 2 (15-01-20) El Gobierno respal...
2 MB - MP4

A3 Noticias 2 (15-01-20) El Gobierno respal...
Descarga HLS - 1920x1080 - 34:00 - 3.4Mbps - MP4

1280x720 - 1.9Mbps

720x404 - 1Mbps

640x360 - 636Kbps

5 En otras pestañas

Fig 3.1 DownloadHelper menu

3.1.2. Live streaming: Personal Video Recorder (PVR)

To obtain videos from Live streaming programs, a satellite decoder is a perfect tool to record news via USB port: Freesat V8 Media Super (GTMedia).

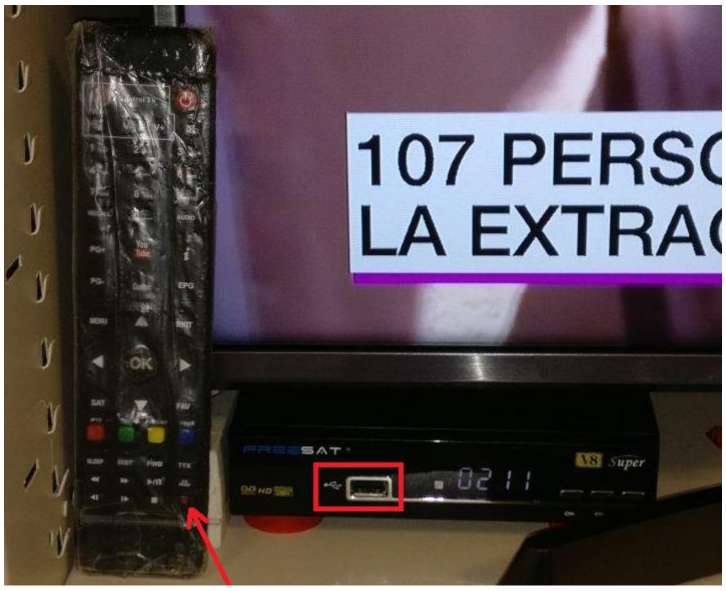


Fig 3.2 Decoder used to record TV news

The way to obtain videos with this device is much simpler than using web browser add-ons: Just inserting an USB dongle in the decoder’s USB port and press ‘record’ button on the remote control. Default record time is two hours, long enough to get frames from all categories. At the end of the record, a transport stream file (.ts) of size (1920x1080) is stored on a folder created for that purpose: PVR (Personal Video Recorder).

3.1.3. Videos’ properties

Since the obtained videos are different depending on the acquisition method, the differences between them are explained below, but it can be mentioned in advance that it will not be a problem because the network’s output will be the same regardless the method used to obtain the video. This small table contains properties of all obtained video data in this project:

Table 3.1. Video properties

TOOL	SIZE	CODEC	CONTAINER
DownloadHelper	1280x720	H.264	MPEG-4 (part 10)
Personal Video Recorder (PVR)	1920x1080	H.262	MPEG-2 (part 2)

3.1.3.1. *MPEG Transport Stream (TS)*

It is also a digital multimedia container format to store programs of coded data in a set of sub-streams (video, audio, data) according to ITU-T Rec. H.262 | ISO/IEC 13818-2 and ISO/IEC 13818-3.

Transport stream encapsulates Packetized Elementary Streams (PES), providing error correction and synchronization pattern to ensure data integrity in case of errors in the channel. This coding method is ideal to use on lossy environments and its main use is for broadcasting systems such as DVB, ATSC and IPTV and as mentioned before, satellite TV.

The codec format used for MPEG-2 part 2 is H.262, which has some mechanisms to compress stream in order to fit in the bandwidth of available TV channels and to reduce overload. There are three ways to code frames:

- I-frames (intra-coded frames): Skips spatial redundancy and takes advantage of the inability of the human eye to detect changes.
- P-frames (predictive-coded frames): This type of frames base compression on previous frames using them as reference.
- B-frames (bidirectional predictive-coded frames): Similar to P-frames, this type uses both previous and subsequent reference frames, achieving a higher compression than P-frames.

Not all the frames are the same type, an example of a group (also called Group of Pictures – GOP) can be this[20]:

…IBBPBBPBBPBBPBBI…

Note that around 15th frame (the standard is flexible) there has to be an I-frame. Not identically but in general, both CODECs use this frame classification to achieve high compression ratio. The TS packets have a constant length of 188 bytes, 4 of them used as header in order to control synchronism, cypher, and errors.

3.1.3.2. *MPEG-4*

A standard method that defines compression of audio and visual digital data. The extension .mp4 file corresponds to the MPEG-4 part 14 (also called MP4), which consists on a digital multimedia container format used commonly to store video, audio and other data such as subtitles. It also codes the frames the same way as in MPEG-Transport Stream.

The data stream containing video (MP4/H.264) offers:

- High compatibility on all operative systems and browsers.
- Higher efficiency when coding than its original version (MPEG-4 part 2)
- Good image quality even if bit rate used to code is low.
- High robustness versus errors in transmission, which makes it a very good option for streaming.
- More suitability when broadcasting.
- Possibility to interact with scene generated at the receiver.

Its only drawback is the high complexity to code and decode, which requires a fast CPU/GPU.

3.1.3.3. *MPEG-2 and MPEG-4 comparison*

Differences between MPEG-4 (part 10) and MPEG-2 (part 2):

- MPEG-4 has a higher image quality with the same bit rate.
- MPEG-4 compression rate is 30-50% more effective.
- MPEG-4 requires less bandwidth than MPEG-2.
- MPEG is more oriented to broadcast multimedia content.

3.2. From videos to frames – FFMPEG

To extract the frames from the videos, a very useful tool is at hand: FFMPEG (Fast Forward Moving Pictures Experts Group). FFMPEG is an open-source software for the multimedia handling that has a wide range of functions on video treatment, actually it is a very popular application used by a vast majority of video treatment applications, it is capable to convert to different video formats, handle audio, merge frames to create a new video…

In this project, the use of FFMPEG has been useful to convert the downloaded videos to frames, but FFMPEG can use many types of element as input, from regular files to website streams, it is capable to convert arbitrary simple rates while resizing streaming videos[21]. The main advantage of using FFMPEG is it has a broad support:

- Image formats
- Video and Audio CODECs
- Video and Audio containers

This support is possible thanks to its libavformat and the libavcodec libraries for media formats that contain muxers and demuxers for audio/video containers. The libavfilter library is also useful for management matters.

The elements compatible with FFMPEG needed to build the data set with frames extracted from videos are:

- CODEC H.264/AVC/MPEG-4 AVC/MPEG-4 part 10 (encoders: libx264, libx264rgb, compatible for de/coding and pixel format).
- CODEC MPEG-2 Video (compatible for de/coding and pixel format).
- CODEC PNG format support to save frames.

The most important feature of FFMPEG for this project is its capability to recognise a vast number of CODECs and formats to recognise the input (MPEG-2/4 videos) and to get the desired output (PNG frames).

3.2.1 FFMPEG syntax

The FFMPEG program reads the content of the specified input with **–i** option, the options used in FFMPEG for this project have been these:

The resulting command that converts a video into frames has this structure:

> ffmpeg -i [video_name.extension] [frame_name.extension]

As example:

> ffmpeg -i RTVE-i1.mp4 RTVE-i1-%d.png (%d is to have numbered frames)

This command slices a video into frames one by one, and according to the standard frame rate in European countries, each second contains 25 frames, but it results in many redundant frames, so the solution proposed is obtaining 1 frame per second instead of 25 per second:

> ffmpeg -i [video_name.extension] –vf fps=1 [frame_name.extension]

-vf is an option to indicate the use of a video filter, the option of the video filter selected for this project is to set frames per second from 25 (default frame rate on MPEG-2/4) to 1.

As example:

> ffmpeg -i RTVE-i1.mp4 –vf fps=1 RTVE-i1-%d.png

This way, FFMPEG will extract less frames from videos, no extra-storage will be required to keep redundant frames and the manual classification will be easier.

3.2.2. Naming frames

Naming downloaded videos is necessary to identify channel and program, when a video is downloaded, the convention to identify channel and program is:

Channel_name-i[number of video].extension

An example to see it clearly can be:

TV3-i6.avi

The number next to *i* keeps no relation with the days of the month, it is just an index to enumerate the videos. In the same day, two or three TV news program can be emitted in the same channel (at morning, evening, night).

To have more variety in frames, for a given channel, a good option is to download and separate morning, evening, night and weekend newscast from different days. The choice in this project has been this:

- Download all morning newscasts from a fixed month.
- Download all evening newscasts from the next month.
- Download all the night newscasts from the next month after evening newscast.
- Download the equivalent number of newscasts of the weekend edition.

A way to separate them in this project has been to distribute the newscasts between training folder, and the validation folder as indicated further in this document. This criterion is to ensure that Intro_New category does not contain almost identical frames: the presenters will be different this way.

The convention to name the frames is:

Channel_name-i[number of video]-[frame_index].extension

Again, an example is provided: *A3-i4-148.png*

3.3. Frames' classification

After obtaining all the frames from a video, the next step is to classify them; the method to classify the frames has been this:

- Check all the frames
- In case a frame belongs to one of the categories, move it to the folder corresponding to that category.
- Continue checking frames until the last one.

The main goal of this project is to distinguish studio frames from non-studio frames, and to achieve that, the proposed categories to classify are these:

- Graph_Intro: Corresponds to graphic frames, digitally created by computer, not from any camera. This category belongs to studio because the displayed info comes from data obtained by the TV program. For this reason, it is a studio category.

- Intro_New: This category refers to presenters' explanation before a reporter explains the new; it can introduce as well a set of graphics. This is clearly a studio category.

- Reporter: Frame where a reporter gathers all possible info about the event and provides a status-report about the place. It can be the beginning of the non-studio category, or can, as well can indicate the end of non-studio.

- Rep_Interview: This kind of frame shows a reporter interviewing a person about the event that made the journalists come to that place. This category cannot be studio.

- Rep_New: Reporter explaining new with camera support. This category normally comes next after Reporter category to provide additional info with cameras to television viewers. This is a non-studio category.

- Video_New: New development, the core of the non-studio category, the raw image of the camera showing the mentioned event happening. It normally contains vehicles, buildings, monuments, distant crowd, landscapes…

The reason to choose these categories is to make the network identify which features belong to studio and which features do not. Examples of these frames can be found on Annex A3.

3.3.1. Data management

To make our network capable to identify each frame it is necessary first to collect enough samples of each category. The network must do the training part first; it means to make the model classify the *training* samples with no prior knowledge. At the beginning, the model will classify frames incorrectly, but it will keep learning though errors, until it learns patterns and detect elements that guide it during the convolution process.

Once the training has been completed, it is necessary then to measure the model's accuracy and loss (this stage is called *validation,* a way to see the training's results). The data from validation is a fragment of the training data, which means that the data is similar but not identical to the model and it is not data from which it has learnt. Validation data is important in order to tune our network's configuration (selecting the number of layers, the number of epochs, the batch size, and it is a way to see at first sight if the network suffers underfitting or overfitting. Another interesting validation technique is the *K-fold cross validation*, which consists on splitting the training set into *k* groups use one of them as a validation set. Do the training process for the rest of the groups and validate with the extracted group, then repeat this operation for all groups and the resulting loss accuracy is the average of the resulting *k* trainings.

After having obtained system's accuracy and loss, the next step is to make the model classify never-seen data to have an objective result, this is the Test part.

3.3.1.1. Data proportions

All models in ML/DL must satisfy a proportion [5] of data in order to obtain the desired results (accuracy as close as possible to 100% and almost null loss). The proportion of data that has provided the best results until now has been this:

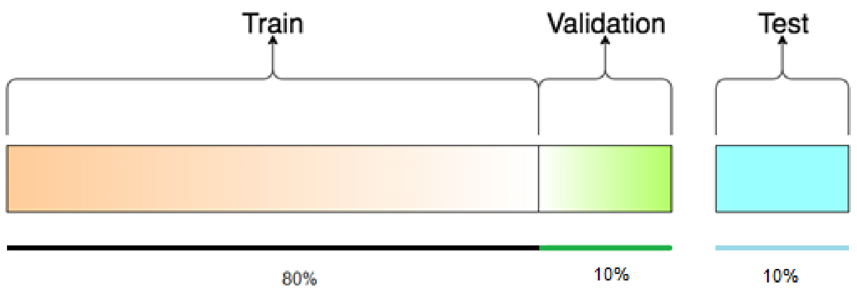


Fig 3.3 Data proportions for Training, Validation and Test

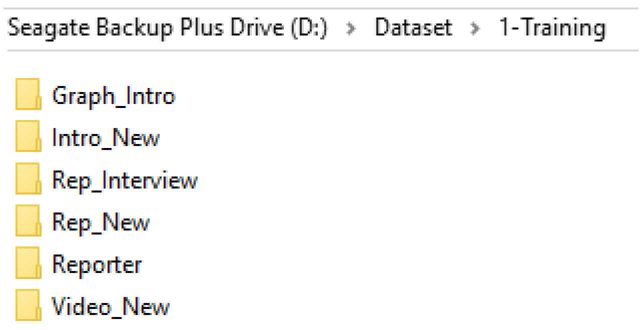
In previous Deep Learning problems, the order of magnitude of the training samples is tens of thousands (40.000-60.000 frames) having validation and test the proportional part as shown on the figure (Fig 3.3).

In the case of this project, the number of frames for each category has been 1000 for training, 1000 for validation and 1000 for test. Previously the proportion has not been accomplished, but Keras has functionalities that can boost accuracy with small data sets, the functionality applied in this project is *Data Augmentation*, achieving accuracy around [85-90]% depending on the network architecture used. This functionality is an approach that creates more training data from existing samples by generating random transformations such as rotation, zoom, horizontal flip…

With these transformations, the trick is that the frame is similar but not identical. Therefore, the network will not consider it as a redundant element during the training part, feature that allows the model to cope with overfitting and helps increasing its generalization ability.

3.3.1.2. Dataset and channels' structure

The way to load data in Keras is to place the frames in their corresponding category, the data set for this project consists in a folder named 'Data set' which contains three folders, '1-Training', '2-Validation', '3-Test'. Each of these folders contains one folder per category, which in this case, if we take as example the folder '1-Training', it contains 6 folders with the names of the categories of this project, as shown in (Fig 3.4)


Seagate Backup Plus Drive (D:) › Dataset › 1-Training

- Graph_Intro
- Intro_New
- Rep_Interview
- Rep_New
- Reporter
- Video_New

Fig 3.4 Folders' structure in Dataset

A folder with the channel's name that contains as many folders as downloaded videos, following the previous convention commented in FFMPEG naming, the folder with the name of the channel contains various folders: 'i1', 'i2'…'iN'. Not all the channels have the same number of TV news programs because there are channels that provide more useful frames than others do with the same amount of frames.

After obtaining some folders with frames from FFMPEG, the next recommended task is to analyse these folders immediately and not generate new folders until the appropriate frames are on their corresponding folders.

3.3.1.3. Distribution of TV channels in Training, Validation and Test

The distribution of the frames in the Traning, Validation and Test folders has been done according to this convention: To train the network with national newscasts and check its performance by testing frames from international TV channels like BBC, CNN, CNBC…

Table 3.2. National newscast edition for training and validation

TRAINING	VALIDATION
Morning, evening, night	Weekend

Table 3.3. International newscast edition for training and validation

TRAINING	VALIDATION
Morning	Evening

The reason for this distribution is to provide as maximum variety as possible in national channels, with a limited number of presenters, studios and reporters, and for the case of international TV channels, which are endless, the diversity will reach its maximum with just a sample of one studio, one presenter, one reporter… per channel. Since the international channels are outnumbered, the list of all channels is on the Annex A2.

CHAPTER 4. TRAINING AND RESULTS

This chapter shows the results after having fed the dataset to each architecture, the results shown are the training loss and accuracy along with validation loss and accuracy of all the trainings, and finally, the prediction accuracy.

- Training loss is the error on the training set of data.
- Validation loss refers to errors in the validation data classification.
- Training accuracy is the ratio between successful classifications and total number of frames during the training process.
- Validation accuracy is the ratio between successful classifications and total number of frames during the validation process.
- Prediction accuracy is the ratio between successful classifications and total number of frames during the prediction process.

The measure that determines if a result is good or bad after the classifications is the prediction accuracy, which is considered an objective and unbiased measure, the validation accuracy, in the other hand, is a small part that comes from the training set, reason for which that measure cannot be absolutely considered as never-seen data. The numerical results are displayed in these tables (30th epoch), obtained after finishing the training for 30 epochs, saving the model and begin training again with a lower LR to obtain better, but stationary results. The accuracy graphs and the confusion matrixes of all architectures can be found on Annex A4.

4.1. Results for architecture VGG16

Table 4.1. Results for VGG16

	Train Loss	Train Acc	Val. Loss	Val. Acc	Pred. Acc
SGD with M	0.1795	0.9400	0.2703	0.9070	0.86
RMSprop	0.0264	0.9920	0.3624	0.9190	0.87
ADAM	0.0085	0.9995	0.5621	0.9210	0.87

VGG16 is the architecture that leads to the best results obtained in this project, reaching the maximum prediction accuracy. Despite SGD prediction accuracy is slightly lower, the validation graph shows a small overfitting affection (validation curve is not distant from training curve, which is not the case of RMSprop and ADAM, where overfitting in both cases begins in epoch 6), and thus, has a better generalization capability. This result is accord with the Occam's razor principle.

4.2. Results for architecture VGG19

Table 4.2. Results for VGG19

	Train Loss	Train Acc	Val. Loss	Val. Acc	Pred. Acc
SGD with M	0.1504	0.9605	0.2735	0.9010	0.86
RMSprop	0.0232	0.9925	0.5029	0.9110	0.86
ADAM	0.0045	0.9980	0.4003	0.9150	0.86

The VGG19 results do not differ much from VGG16 architecture. Again, SGD optimizer is slower reaching overfitting, and in the case of in RMSprop, the training and validation curve don’t split until epoch 10 and with ADAM, the validation accuracy curve (stationary in 0.99) is not far off the training accuracy curve (stationary in 0.91). RMS prop and ADAM converge fast, but not to the best solution.

4.3. Results for Inception V3

Table 4.3. Results for Inception V3

	Train Loss	Train Acc	Val. Loss	Val. Acc	Pred. Acc
SGD with M	**0.4001**	**0.8585**	**1.6072**	**0.6210**	**0.56**
RMSprop	**0.5319**	**0.8150**	**1.7329**	**0.5940**	**0.53**
ADAM	**0.1697**	**0.9400**	**2.0595**	**0.6600**	**0.59**

The results with Inception architecture begin to worsen, prediction accuracy decreases from 0.90 to values lower than 0.60 and the validation accuracy suffers a similar decrease. Despite many LRs have been tested to improve the results of the table, the validation accuracy remains stationary regardless the number of epochs in all architectures. A reason for this behaviour is the convergence on solutions that are good for training, but for this architecture, the validation results are poor and far from solution if compared to VGGs.

4.4. Results for Xception

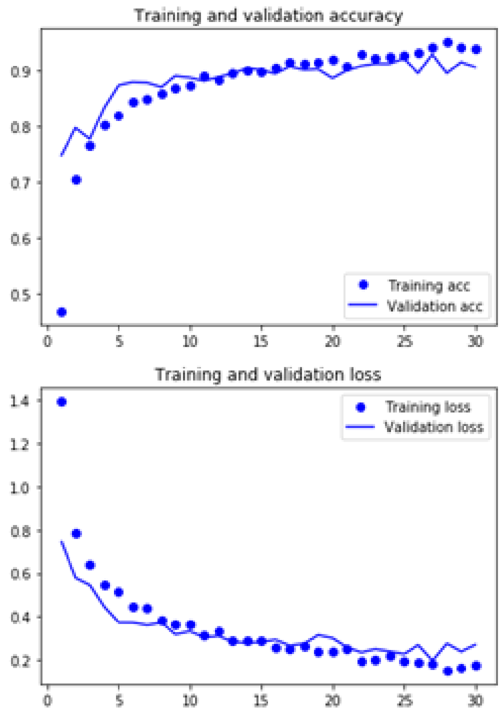
Table 4.4. Results for Xception

	Train Loss	Train Acc	Val. Loss	Val. Acc	Pred. Acc
SGD with M	**0.1904**	**0.9460**	**0.9841**	**0.7060**	**0.62**
RMSprop	**0.2743**	**0.9045**	**1.1564**	**0.7070**	**0.61**
ADAM	**0.2260**	**0.9260**	**1.3683**	**0.7500**	**0.65**

The results obtained in Xception, however, are numerically a bit better than the Inception results, but the graphs show a clear overfitting in which is impossible to obtain better validation results with the small quantity of frames obtained. For this architecture, validation curve never reaches the training curve, providing poor results for the predictions.

The reason that concludes this behaviour in all the architectures is the small quantity of frames in the dataset. Due to this small quantity, all the architectures suffer overfitting, in the case of RMSprop and ADAM convergence is reached fast, but the solutions obtained are very good for training and bad for validation, keeping this trend during all epochs. SGD, in the other hand, reaches convergence slower than RMSprop and ADAM, reason for which obtains better results: overfitting arrives later for this optimizer.

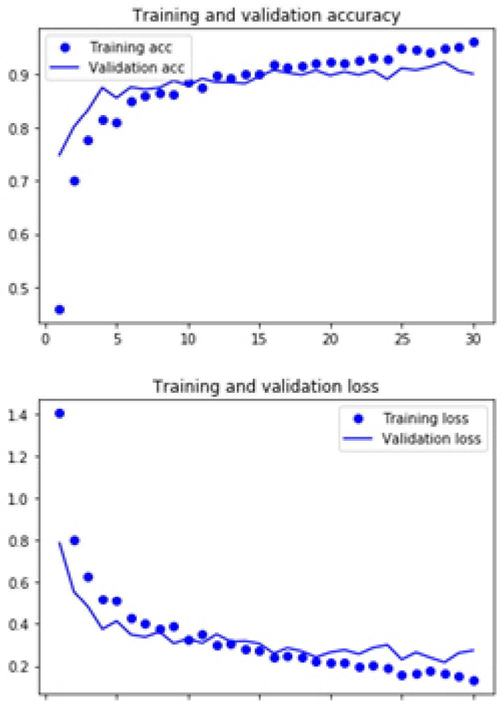
Some of the best results are shown here as example:



```
                Confusion Matrix
 Graph_Intro [[890  11   1   1   8  96]
   Intro_New  [ 14 790  23  30 130  19]
Rep_Interview [  5  89 779   6 122   4]
     Rep_New  [  1   7  10 936  34  13]
    Reporter  [  0  60  92   4 835  13]
   Video_New  [ 42  13   3   6   7 934]]
```

```
Classification Report
              precision    recall  f1-score   support

  Graph_Intro      0.93      0.88      0.91      1007
    Intro_New      0.81      0.79      0.80      1006
Rep_Interview      0.86      0.78      0.81      1005
      Rep_New      0.95      0.94      0.94      1001
     Reporter      0.74      0.83      0.78      1004
    Video_New      0.87      0.93      0.90      1005

     accuracy                          0.86      6028
    macro avg      0.86      0.86      0.86      6028
 weighted avg      0.86      0.86      0.86      6028
```

Fig 4.1 VGG16 SGD

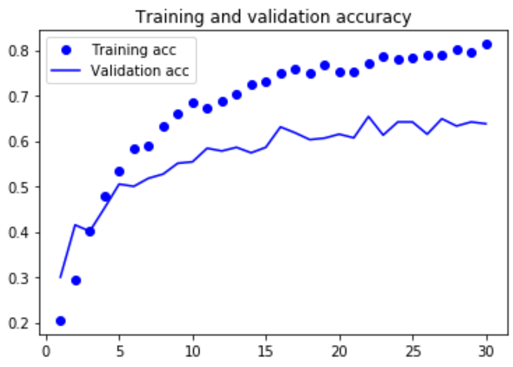


```
                 Confusion Matrix
  Graph_Intro [[953  12   1   0   5  36]
    Intro_New  [ 34 830   9  12 111  10]
Rep_Interview  [  8 128 705   4 157   3]
      Rep_New  [  3   9   6 952  26   5]
     Reporter  [  5  98  60   5 828   8]
    Video_New  [122  15   1   4  11 852]]
```

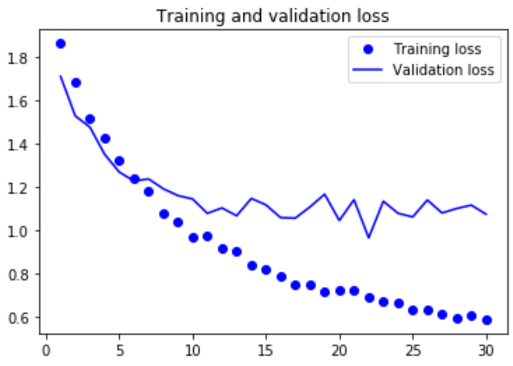
```
Classification Report
              precision    recall  f1-score   support

                   0.85      0.95      0.89      1007
                   0.76      0.83      0.79      1006
                   0.90      0.70      0.79      1005
                   0.97      0.95      0.96      1001
                   0.73      0.82      0.77      1004
                   0.93      0.85      0.89      1005

     accuracy                          0.85      6028
    macro avg      0.86      0.85      0.85      6028
 weighted avg      0.86      0.85      0.85      6028
```

Fig 4.2 VGG19 SGD

Some bad results examples:

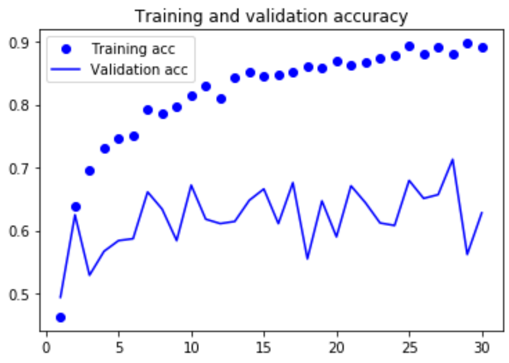


```
                    Confusion Matrix
Graph_Intro [[478  68   3 444  11   3]
  Intro_New  [  3 402   6 512  83   0]
Rep_Interview [  0  79 377 411 138   0]
    Rep_New  [  0  10   6 968  17   0]
   Reporter  [  0  47  85 286 586   0]
  Video_New  [ 15  27  22 499  83 359]]
```

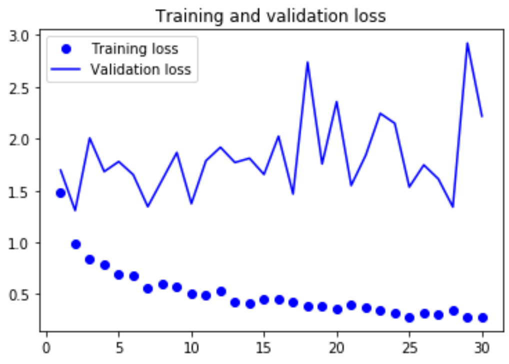


```
Classification Report
              precision    recall  f1-score   support

  Graph_Intro       0.96      0.47      0.64      1007
    Intro_New       0.64      0.40      0.49      1006
Rep_Interview       0.76      0.38      0.50      1005
      Rep_New       0.31      0.97      0.47      1001
     Reporter       0.64      0.58      0.61      1004
    Video_New       0.99      0.36      0.53      1005

     accuracy                           0.53      6028
    macro avg       0.72      0.53      0.54      6028
 weighted avg       0.72      0.53      0.54      6028
```

Fig 4.3 Xception RMSprop



```
                    Confusion Matrix
Graph_Intro [[419 164  14 390  10  10]
  Intro_New  [  6 562  11 333  92   2]
Rep_Interview [  0 119 516 187 183   0]
    Rep_New  [  0   2   0 997   2   0]
   Reporter  [  0  98  47 152 707   0]
  Video_New  [ 17  59  12 487  95 335]]
```



```
Classification Report
              precision    recall  f1-score   support

  Graph_Intro       0.95      0.42      0.58      1007
    Intro_New       0.56      0.56      0.56      1006
Rep_Interview       0.86      0.51      0.64      1005
      Rep_New       0.39      1.00      0.56      1001
     Reporter       0.65      0.70      0.68      1004
    Video_New       0.97      0.33      0.50      1005

     accuracy                           0.59      6028
    macro avg       0.73      0.59      0.59      6028
 weighted avg       0.73      0.59      0.59      6028
```

Fig 4.4 Inception V3 ADAM

CONCLUSIONS

The work in this project has studied and documented the basics in Deep Learning, a subset of Machine Learning popular in the field of image classification. The steps followed to obtain the classifications that distinguish the frames between the studio and non-studio categories have been these:

- Document the basic operations in CNNs.
- Compare the different DL frameworks and install the most suitable.
- Implement the CNNs with the necessary operations shown in Chapter 1.
- Build the dataset, by obtaining videos and extracting the frames.
- Load the architectures, train the networks with different optimizers and techniques to obtain the results and compare them.

With the obtained results and after all the research to do this project, the reached conclusions are these:

- The interest and research in DL is increasing, with the appearance of new classification datasets, to solve different problems such as clothes, sports, supermarket product classification…
- Thanks to the loaded architectures it is possible achieve very good results in shallow architectures.
- The deepest networks result non-effective in small datasets.
- The generalization capability is higher with shallow architectures, according to Occam’s razor principle, which states that the simplest explanation is normally the right one.
- Data augmentation parameters in the training frames must generate frames that have no additional features respect the validation set and the test, for example, the validation set will rarely contain a rotated frame in a newscast.
- Depending on the problem, Data Augmentation can boost accuracy to 95% with small datasets, not here due to small quantity and restricted augmentation.

The Future lines of development are these:

- The first improvement has an easy but long implementation, consists on installing as many DL frameworks as possible and check the performance with image classification, taking into account DL is not exclusively associated to computer vision. Along with this implementation, building a bigger dataset with more frames would be interesting to compare performances obtained in this project for further research. The results would be better in the deepest architectures with a bigger dataset.

- The work in this project classifies the different type of scenes that appear on the newscasts. In subsequent versions of the created system in this project, it would be interesting to identify the presence of the sign language that appears on some newscasts. A first approach can be done by creating additional categories, which would consist on the previous categories with the presence of language sign, this means that all categories would go from *Graph_Intro* and *Graph_Intro (language sign)* to *Video_New* and *Video_New (Language sign)*. The reason to use this structure is to not only detect the language sign, but to detect the category in which appears the language sign. It wouldn't suppose much additional difficulty comparing to the creation of the dataset explained in Chapter 3 because most of the channels dispose explicitly of videos with language sign.

- The last approach that could be useful from this system can be finding scenes in movies instead of rewinding backwards or forward in multimedia content. The way to achieve this functionality is repeating the work done in this project and creating the necessary categories and therefore, building the required dataset. This concept by itself is extremely wide, for this reason, the suggested approach is to do this work in different films genre, being this way apparently more feasible to reach results.

Two aspects to consider as part of the economic impacts in this thesis, first of all the equipment used to train the models require remarkable GPUs to obtain results in a reasonable time, which means that the progress on image classification programs is based on the use of high performance equipment, normally quite expensive. It is also important to consider that the results in this project have been obtained thanks to the possibility of transferring the training results to the frameworks.

On second place, a feature that has been mentioned along this thesis: the model training. It takes quite long, and it is highly recommended not to do the training part with energy optimization option enabled because entering sleep mode will most likely end in repeating model training with energy optimization option disabled, which takes us to the second economic consideration: the price for the energy consumption for long periods of time. This consideration is referred mostly to the equipment used in the ILSVRC, but despite not being negligible, the environmental impact of this project is not comparable to high-performance computers that are training models for long periods to improve their accuracy.

The last consideration has not only economic considerations but environmental: the electric energy consumption for long periods contribute to CO_2 emissions due to necessary training of dedicated equipment with no energy saving feature.

The technology used in this project is able to perform facial recognition, but it has been sought not to do so, in order to avoid manipulation from third parties. As shown in the code on Chapter 2, the main objective has been only to recognise scenes to distinguish those frames that belong to the categories explained in this project.

Unless consent is given, not respecting the right to privacy is an offence that may entail legal measures by those affected, as exposed on article 197.7 of the Spanish criminal code and therefore, not resulting an interest point for this project.

ACRONYMS

ADAM	Adaptive Moment Estimation
AI	Artificial Intelligence
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
CNN	Convolutional Neural Network
DL	Deep Learning
FC	Fully Connected
FFMPEG	Fast Forward Motion Picture Experts Group
GD	Gradient Descent
GPU	Graphics Processing Unit
ILSVRC	ImageNet Large Scale Visual Recognition Challenge
LR	Learning Rate
ML	Machine Learning
MNIST	Modified National Institute of Standards and Technology
NN	Neural Network
ReLU	Rectified Linear Unit
RMSprop	Root Mean Square propagation
SGD	Stochastic Gradient Descent
VGG	Visual Geometry Group

REFERENCES

[1] How Occam's Razor works, date accessed: 06/03/2020, from https://science.howstuffworks.com/innovation/scientific-experiments/occams-razor.htm

[2] Chollet, F. Deep Learning With Python, New York, NY, USA: Manning Publications Co, 2017

[3] Albawi, S; Mohammed. T; Al-Zawi. S" Proceedings of 2017 International Conference on Engineering and Technology", *ICET*, Pp. 1-6. (2018)

[4] Patel, Krut, Convolutional Neural Networks — A Beginner's Guide – Towards Data Science, date accessed: 10/01/2020, https://towardsdatascience.com/understanding-rmsprop-faster-neural-network-learning-62e116fcf29a?gi=c1cf86979d5

[5] Tarrés, F. Applied Image Processing, Neural Networks and Deep Learning [PDF], UPC Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels. Barcelona, obtained from https://atenea.upc.edu

[6] J. Ricco, What is max pooling in convolutional neural networks? https://www.quora.com, date accessed: 15/12/2019, from https://www.quora.com/What-is-max-pooling-in-convolutional-neural-networks

[7] Qué es overfitting y underfitting y cómo solucionarlo | Aprende Machine Learning, date accessed: 04/12/2019, from https://www.aprendemachinelearning.com/que-es-overfitting-y-underfitting-y-como-solucionarlo/

[8] What are Max Pooling, Average Pooling, Global Max Pooling and Global Average Pooling? date accessed: 18/03/2020, from https://www.machinecurve.com/index.php/2020/01/30/what-are-max-pooling-average-pooling-global-max-pooling-and-global-average-pooling/

[9] 7 Types of Neural Network Activation Functions: How to Choose?, date accessed: 11/03/2020, from https://missinglink.ai/guides/neural-network-concepts/7-types-neural-network-activation-functions-right/

[10] GPU (Graphics Processing Unit) Definition, date accessed: 20/03/2020, https://techterms.com/definition/gpu

[11] What is CUDA? How is it linked to NVIDIA? What does CUDA have to do with GPU? – Quora, date accessed: 23/02/2020, from: www.quora.com/What-is-CUDA-How-is-it-linked-to-NVIDIA-What-does-CUDA-have-to-do-with-GPU

[12] ImageNet Large Scale Visual Recognition Challenge (ILSVRC), date accessed: 04/02/2020 http://image-net.org/challenges/LSVRC/2012/index

[13] Keras vs TensorFlow vs PyTorch | Deep Learning Frameworks | Edureka, date accessed: 04/04/2020, from www.edureka.co/blog/keras-vs-tensorflow-vs-pytorch/

[14] How to Configure Image Data Augmentation in Keras, date accessed: 18/12/2019, from machinelearningmastery.com/how-to-configure-image-data-augmentation-when-training-deep-learning-neural-networks/

[15] A simple example: Confusion Matrix with Keras flow_from_directory.py · GitHub, date accessed: 22/12/2019, from gist.github.com/RyanAkilos/3808c17f79e77c4117de35aa68447045

[16] Ruder S, "An overview of gradient descent optimization algorithms", *ArXiv* id: 1609.04747, (2016)

[17] Optimizers Explained - Adam, Momentum and Stochastic Gradient Descent, date accessed: 22/12/2019, from https://mlfromscratch.com/optimizers-explained/#/

[18] Chollet, F, "Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition", CVPR 2017, *ArXiV* id: 1610.02357, Pp. 1800-1807 (2017)

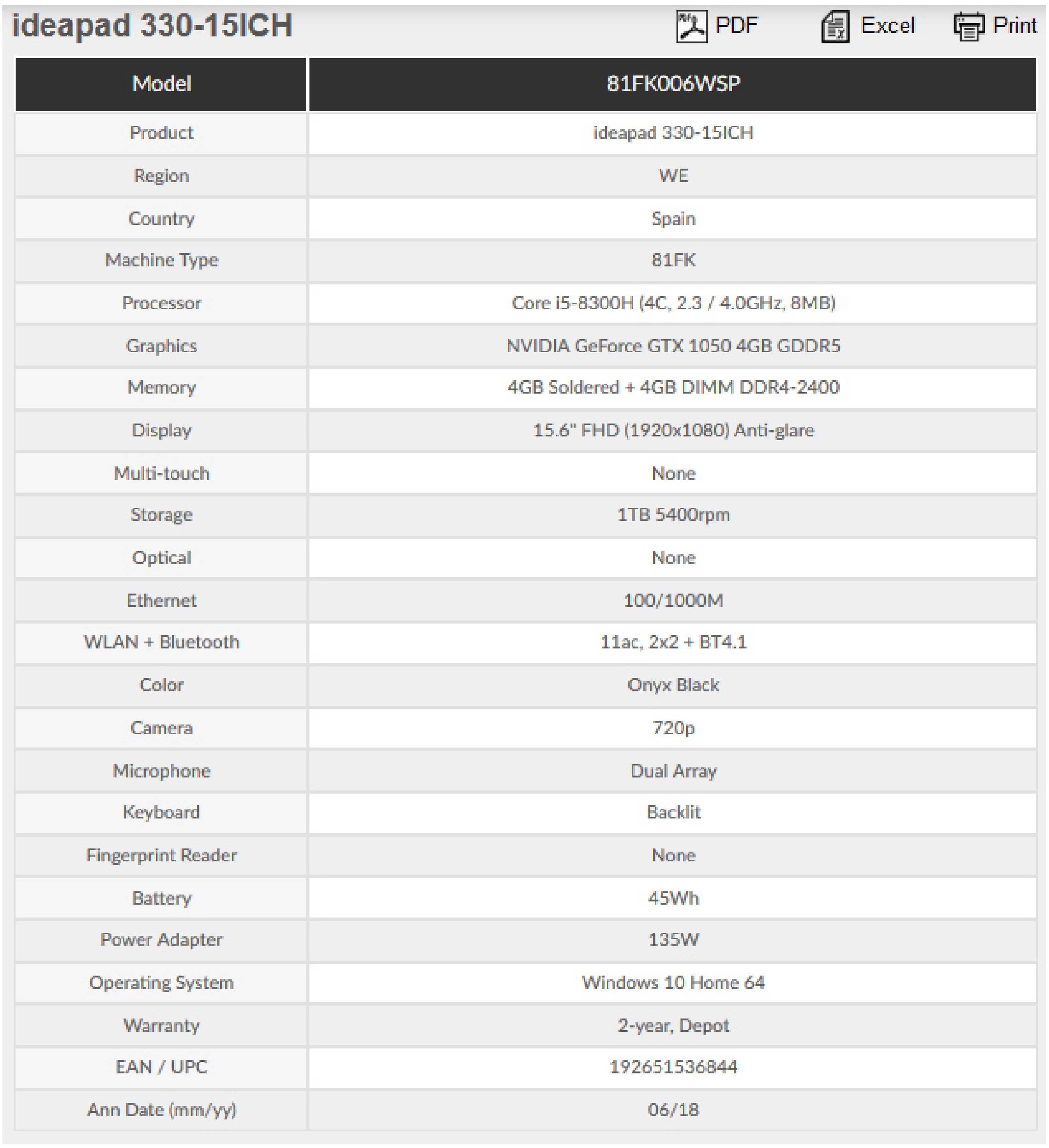
[19] He, K; Zhang, X; Ren, S; "Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition", *ArXiV* id: 1512.03385, Pp. 770-778, (2016).

[20] Tarrés, F."Televisión digital" Sistemas audiovisuales I. Televisión analógica y digital, Pp. 273-380, (2004)

[21] Korbel, F. "FFMPEG fundamentals" FFmpeg Basics, Pp. 15-29, (2012)

Annexes

A.1 Hardware specs used in this project



ideapad 330-15ICH

PDF Excel Print

Model	81FK006WSP
Product	ideapad 330-15ICH
Region	WE
Country	Spain
Machine Type	81FK
Processor	Core i5-8300H (4C, 2.3 / 4.0GHz, 8MB)
Graphics	NVIDIA GeForce GTX 1050 4GB GDDR5
Memory	4GB Soldered + 4GB DIMM DDR4-2400
Display	15.6" FHD (1920x1080) Anti-glare
Multi-touch	None
Storage	1TB 5400rpm
Optical	None
Ethernet	100/1000M
WLAN + Bluetooth	11ac, 2x2 + BT4.1
Color	Onyx Black
Camera	720p
Microphone	Dual Array
Keyboard	Backlit
Fingerprint Reader	None
Battery	45Wh
Power Adapter	135W
Operating System	Windows 10 Home 64
Warranty	2-year, Depot
EAN / UPC	192651536844
Ann Date (mm/yy)	06/18

Fig. A1.1 Used equipment datasheet

This is the computer used to train the networks, the most important features for this project are these:

- CPU Core i5, 4C (2.30 – 4.00) GHz
- GPU NVIDIA GeForce GTX 1050 – 4GB
- RAM memory: 8GB

With this equipment, the average times to get the results are:

- Training/validation: 1 hour 2 minutes and 47 seconds
- Predictions: 1 minute and 58 seconds

A2.1 Training and validation Channels

Table A2.1 Channels to build the training and validation set

RTVE	C|NET	TLC
A3	GESSELSCHAFT	KGET 17
LA6	GBC	CGTN
TV3	TVGI	EC
CUATRO	GOL TV	TGR
T5	BEIN SPORT	CORSE
8TV	ERT	COURT TV
BETEVE	EFEKTO TV	ED
BDN	TEG ESAN	EKF
TELEB	INTERECONOMIA	ITV
L'H	TELEWEBION	NEWS NOW
CyL	()24	RUV
CMM	MGC	ANN
ESASTURIASTV	LA2	JNE
NAVARRA TV	LVZ+	KAFTAN TV
RTVC	dD	ONE KC
ANDORRA TV	TV MELILLA	ABC NEWS
TAGESSCHAU	EL TRECE	LALIGA
I1 (ITALY)	TG 4	LCI
I2(ITALY)	NWZ NEWS	TF1
I7(ITALY)	OMNI NEWS	TV HEMM
FRANCE24	PORTO CANAL	MEXICO TV
TVGA	ORBE TV	CANAL 9
ABC 7	OTP	NOVA TN
KGET 7	NV	TV PERU
AFRICANEWS	POLCU TV	RPP
POWERLUNCH	POLAND TV	STV
TELETICA	ECHO 24	9 NEWS
TN7	NTC	TELEDOCE
SUNRISE	RAINEWS 24	TMC
SUN 7	RMADRID TV	TRT
RTP	RSSING	TVC
TVI24	1TV (RUSSIA)	TV URUGUAY
BADIA	STONIA TV	VOA
CAPITAL	SVT1 (SWEDEN)	ETV
UNO	TELESUR	EXTREMADURA TV
7 (MIAMI)	TERAZ TV	YLE
EXTRATV	TG 2000	BFM
9 (AUSTRALIA)	TN 21	TG1
SGTN	1 (TUNEZ)	YLE
SOGOU	TVP	BFM
C1 (PRAHA)	TV UG	EXTREMADURA TV
DMC	PRESS TV	VITEC
NILE TV	I24	TV CEUTA
ETV	A TV	

A2.2 Test Channels

Table A2.2 Channels to build the test set

TeleMadrid
IB3
CNN
BBC
RT
CBS
2DF
ARAGON TV
ARIRANG
BDMADRID
BLOOMBERG
CANAL SUR
CNBC
CUBAVISION
FOX NEWS
NEWSROOM
TVN
NHK
TV10
SKYNEWS
TV5 MONDE
TV RECORD SD

A3. Example frames for each category

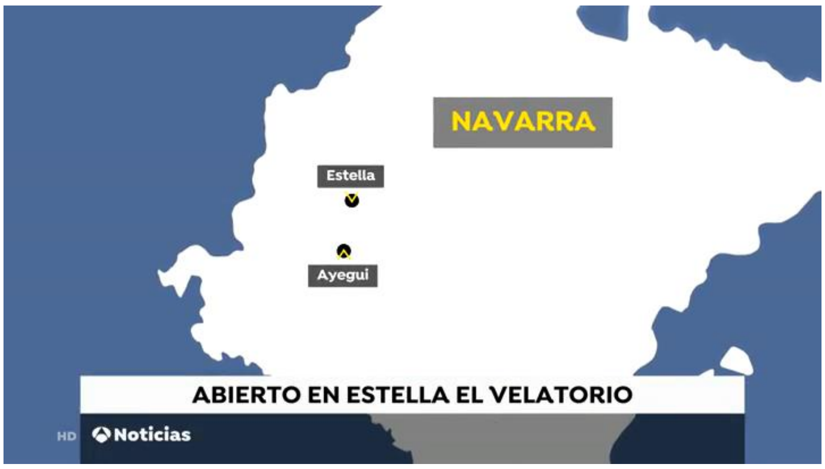


Fig A3.1 Graph_Intro frame

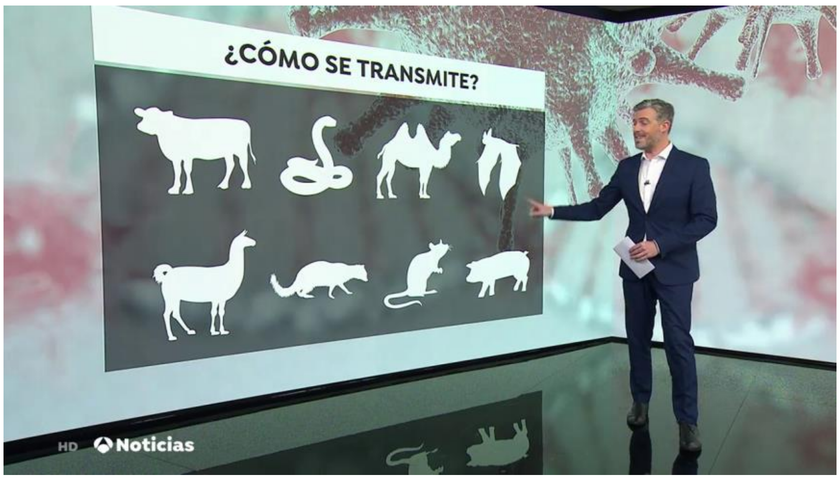


Fig A3.2 Intro_New frame

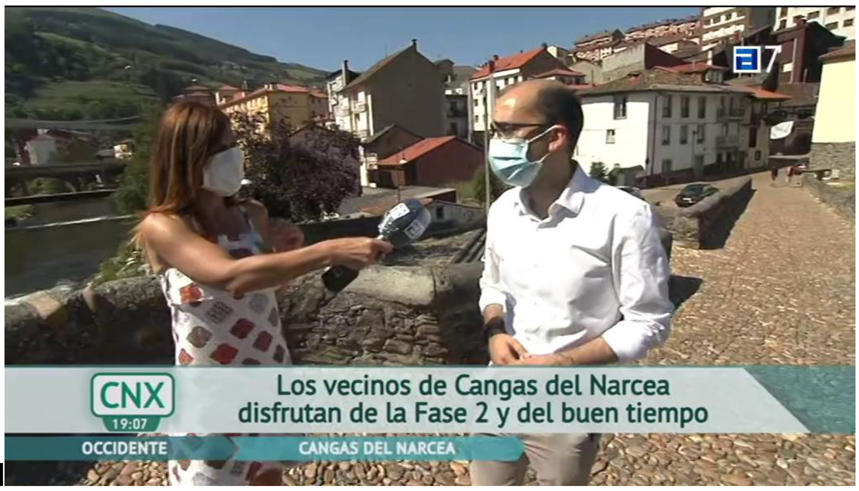

CNX
19:07
Los vecinos de Cangas del Narcea
disfrutan de la Fase 2 y del buen tiempo
OCCIDENTE
CANGAS DEL NARCEA

Fig A3.3 Rep_Interview frame

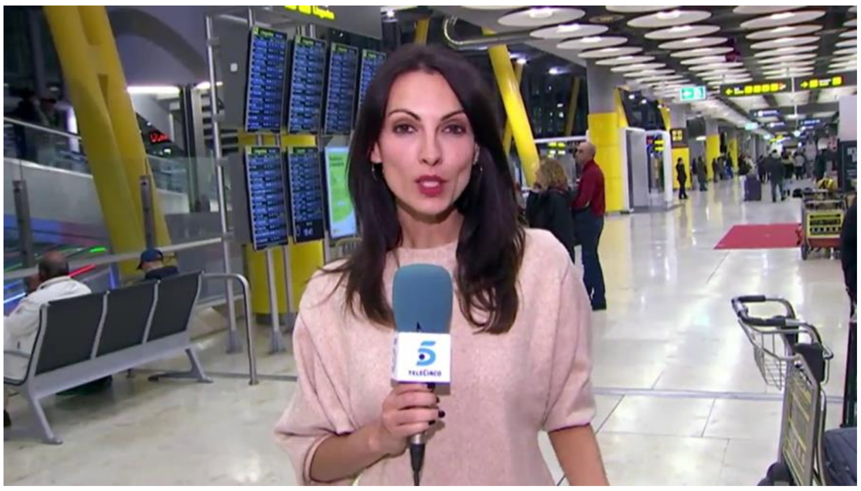


Fig A3.4 Reporter frame

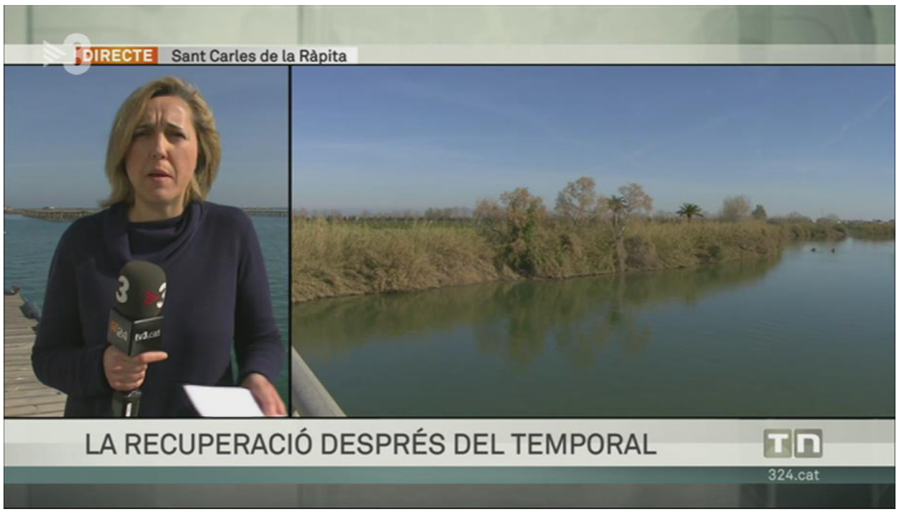


Fig A3.5 Rep_New frame

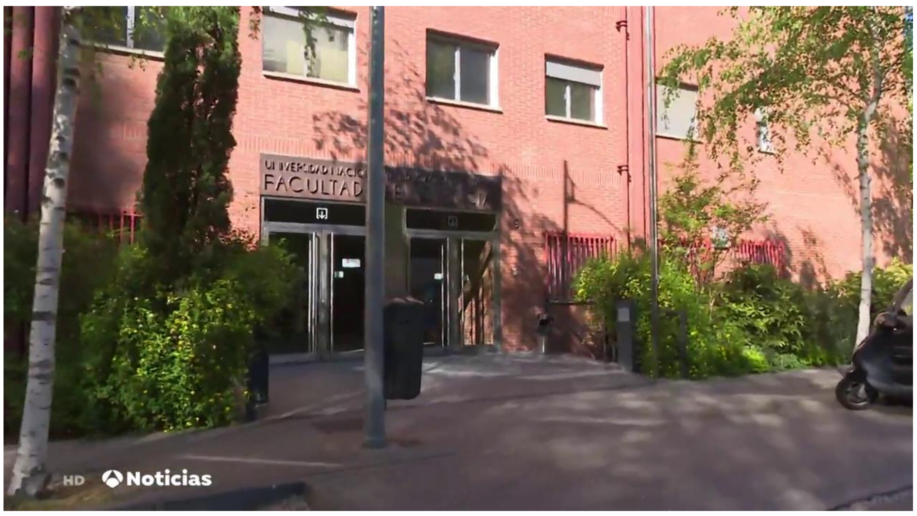
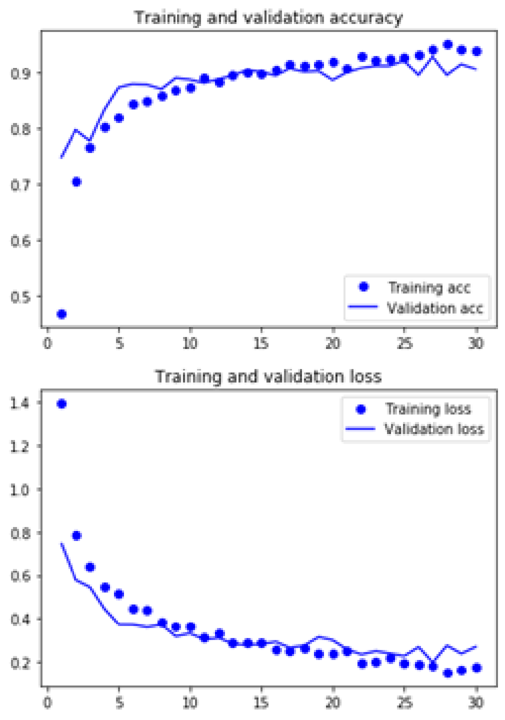


Fig A3.6 Video_New frame

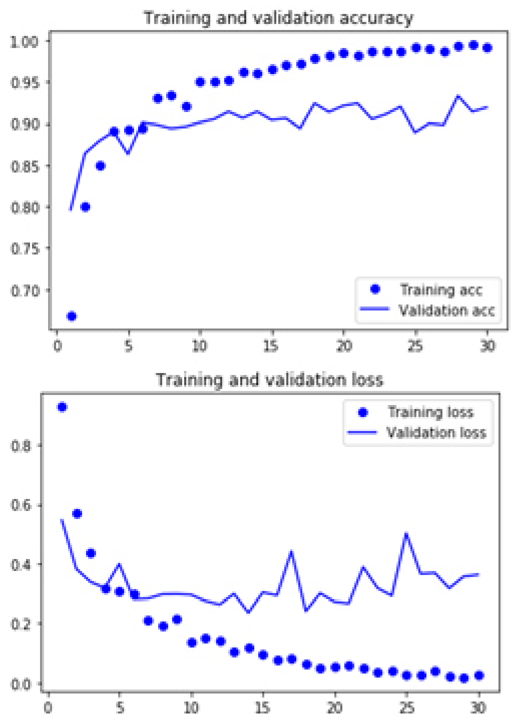
A4. Results from all architectures



```
                Confusion Matrix
  Graph_Intro [[890  11   1   1   8  96]
    Intro_New [ 14 790  23  30 130  19]
Rep_Interview [  5  89 779   6 122   4]
      Rep_New [  1   7  10 936  34  13]
     Reporter [  0  60  92   4 835  13]
    Video_New [ 42  13   3   6   7 934]]
```

```
Classification Report
               precision    recall  f1-score   support

  Graph_Intro       0.93      0.88      0.91      1007
    Intro_New       0.81      0.79      0.80      1006
Rep_Interview       0.86      0.78      0.81      1005
      Rep_New       0.95      0.94      0.94      1001
     Reporter       0.74      0.83      0.78      1004
    Video_New       0.87      0.93      0.90      1005

     accuracy                           0.86      6028
    macro avg       0.86      0.86      0.86      6028
 weighted avg       0.86      0.86      0.86      6028
```

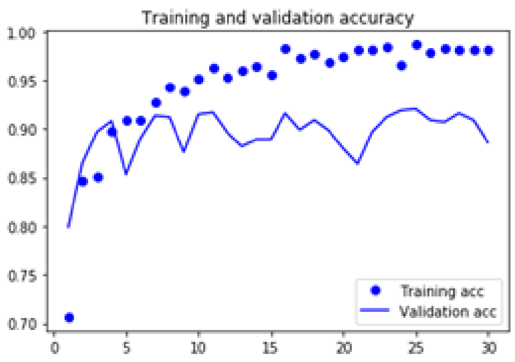
Fig A4.1 VGG16 results – SGD

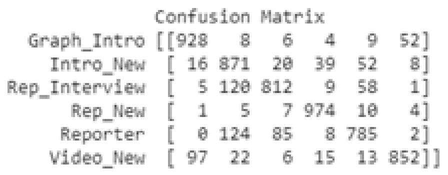


```
                Confusion Matrix
  Graph_Intro [[946  11   0   1   4  45]
    Intro_New [ 30 843  12  17  93  11]
Rep_Interview [ 17 102 761   6 115   4]
      Rep_New [  1   8   7 951  16  18]
     Reporter [  4  72  73   7 836  12]
    Video_New [ 94  14   1   8   7 881]]
```

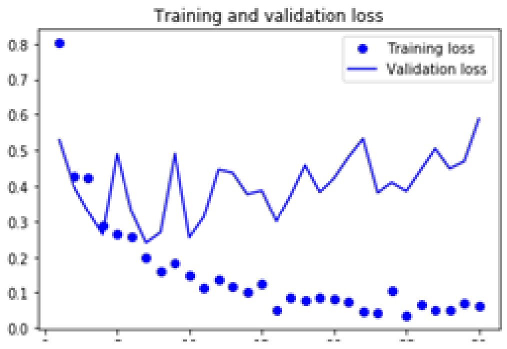
```
Classification Report
               precision    recall  f1-score   support

  Graph_Intro       0.87      0.94      0.90      1007
    Intro_New       0.80      0.84      0.82      1006
Rep_Interview       0.89      0.76      0.82      1005
      Rep_New       0.96      0.95      0.96      1001
     Reporter       0.78      0.83      0.81      1004
    Video_New       0.91      0.88      0.89      1005

     accuracy                           0.87      6028
    macro avg       0.87      0.87      0.87      6028
 weighted avg       0.87      0.87      0.87      6028
```

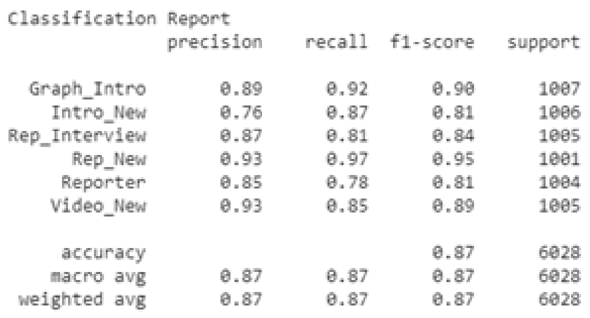
Fig A4.2 VGG16 results – RMSprop





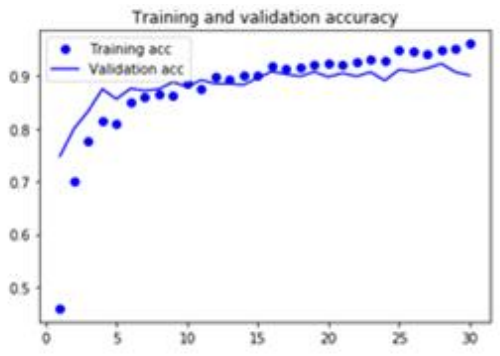
```
             Confusion Matrix
Graph_Intro [[928   8   6   4   9  52]
  Intro_New  [ 16 871  20  39  52   8]
Rep_Interview [  5 120 812   9  58   1]
    Rep_New  [  1   5   7 974  10   4]
   Reporter  [  0 124  85   8 785   2]
  Video_New  [ 97  22   6  15  13 852]]
```

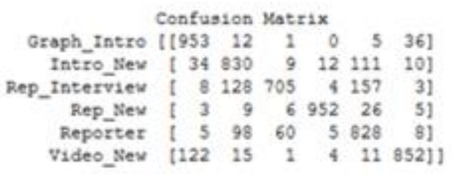




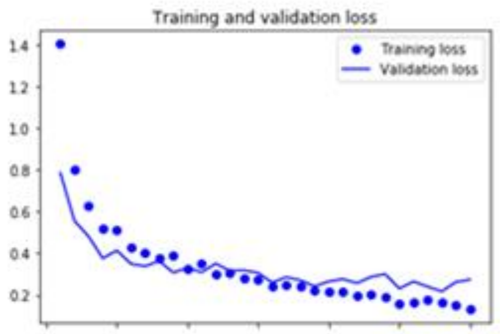
```
Classification Report
               precision    recall  f1-score   support

  Graph_Intro       0.89      0.92      0.90      1007
    Intro_New       0.76      0.87      0.81      1006
Rep_Interview       0.87      0.81      0.84      1005
      Rep_New       0.93      0.97      0.95      1001
     Reporter       0.85      0.78      0.81      1004
    Video_New       0.93      0.85      0.89      1005

     accuracy                           0.87      6028
    macro avg       0.87      0.87      0.87      6028
 weighted avg       0.87      0.87      0.87      6028
```

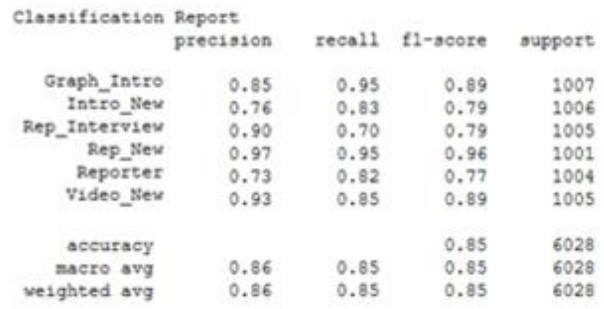
Fig A4.3 VGG16 results – ADAM





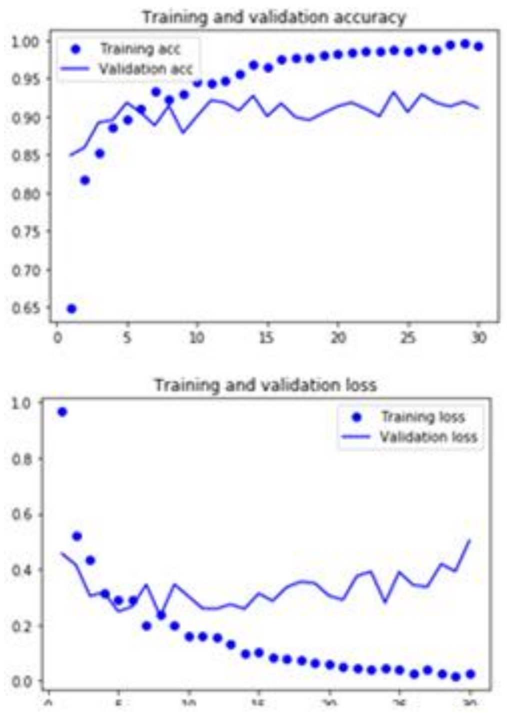
```
             Confusion Matrix
Graph_Intro [[953  12   1   0   5  36]
  Intro_New  [ 34 830   9  12 111  10]
Rep_Interview [  8 128 705   4 157   3]
    Rep_New  [  3   9   6 952  26   5]
   Reporter  [  5  98  60   5 828   8]
  Video_New  [122  15   1   4  11 852]]
```





```
Classification Report
               precision    recall  f1-score   support

  Graph_Intro       0.85      0.95      0.89      1007
    Intro_New       0.76      0.83      0.79      1006
Rep_Interview       0.90      0.70      0.79      1005
      Rep_New       0.97      0.95      0.96      1001
     Reporter       0.73      0.82      0.77      1004
    Video_New       0.93      0.85      0.89      1005

     accuracy                           0.85      6028
    macro avg       0.86      0.85      0.85      6028
 weighted avg       0.86      0.85      0.85      6028
```

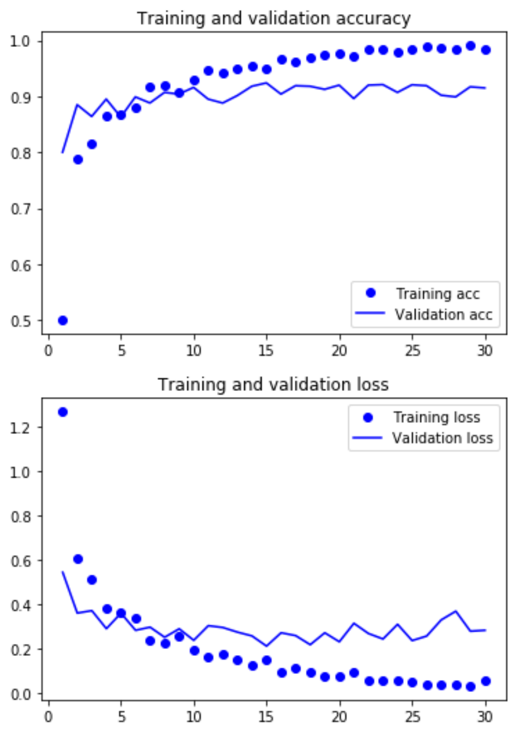
Fig A4.4 VGG19 results – SGD



```
                  Confusion Matrix
   Graph_Intro [[854  15   2   3   5 128]
    Intro_New  [ 15 884  12  22  43  30]
Rep_Interview  [  5 140 769   4  76  11]
      Rep_New  [  2  10  13 947  11  18]
     Reporter  [  0 154  88   7 741  14]
    Video_New  [ 25   8   1   8   2 961]]
```

```
Classification Report
               precision    recall  f1-score   support

  Graph_Intro       0.95      0.85      0.90      1007
    Intro_New       0.73      0.88      0.80      1006
Rep_Interview       0.87      0.77      0.81      1005
      Rep_New       0.96      0.95      0.95      1001
     Reporter       0.84      0.74      0.79      1004
    Video_New       0.83      0.96      0.89      1005

     accuracy                           0.86      6028
    macro avg       0.86      0.86      0.86      6028
 weighted avg       0.86      0.86      0.86      6028
```

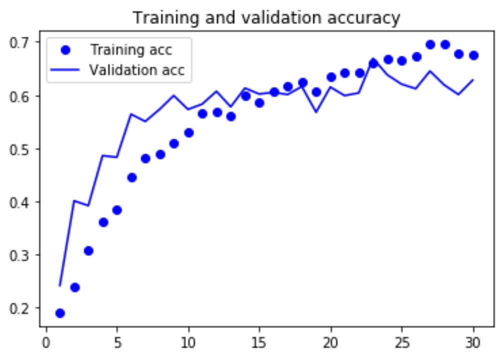
Fig A4.5 VGG19 results – RMSprop



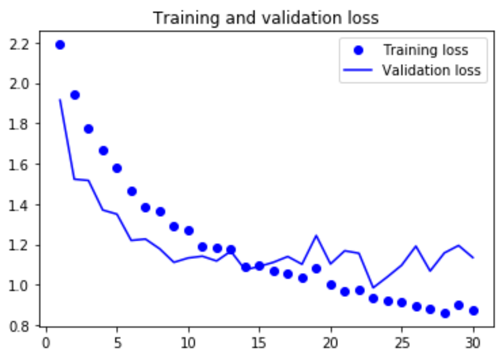
```
                  Confusion Matrix
   Graph_Intro [[918  11   1   2   6  69]
    Intro_New  [ 18 839  11  26  95  17]
Rep_Interview  [  9 120 711   7 155   3]
      Rep_New  [  2   7   4 958  18  12]
     Reporter  [  1  76  48   4 868   7]
    Video_New  [ 60   7   0   7  12 919]]
```

```
Classification Report
               precision    recall  f1-score   support

  Graph_Intro       0.91      0.91      0.91      1007
    Intro_New       0.79      0.83      0.81      1006
Rep_Interview       0.92      0.71      0.80      1005
      Rep_New       0.95      0.96      0.96      1001
     Reporter       0.75      0.86      0.80      1004
    Video_New       0.89      0.91      0.90      1005

     accuracy                           0.86      6028
    macro avg       0.87      0.86      0.86      6028
 weighted avg       0.87      0.86      0.86      6028
```

Fig A4.6 VGG19 results – ADAM

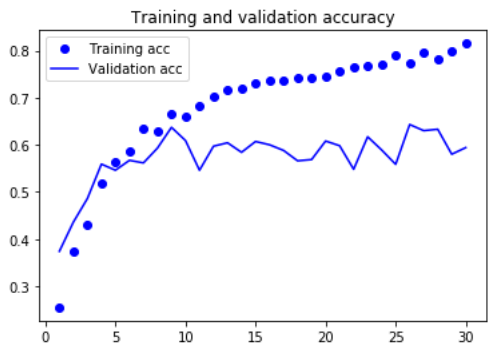


```
              Confusion Matrix
Graph_Intro [[401 158   4 426  12   6]
  Intro_New  [  2 564   8 348  84   0]
Rep_Interview [  0 136 519 195 155   0]
    Rep_New  [  0   5   2 990   4   0]
   Reporter  [  0 113  59 158 674   0]
  Video_New  [ 18  70  15 616  65 221]]
```

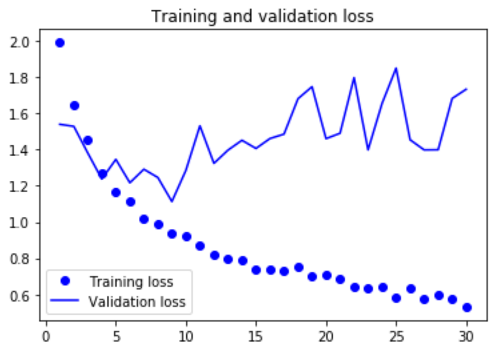


```
Classification Report
              precision    recall  f1-score   support

  Graph_Intro       0.95      0.40      0.56      1007
    Intro_New       0.54      0.56      0.55      1006
Rep_Interview       0.86      0.52      0.64      1005
      Rep_New       0.36      0.99      0.53      1001
     Reporter       0.68      0.67      0.67      1004
    Video_New       0.97      0.22      0.36      1005

     accuracy                           0.56      6028
    macro avg       0.73      0.56      0.55      6028
 weighted avg       0.73      0.56      0.55      6028
```

Fig A4.7 Inception V3 results – SGD

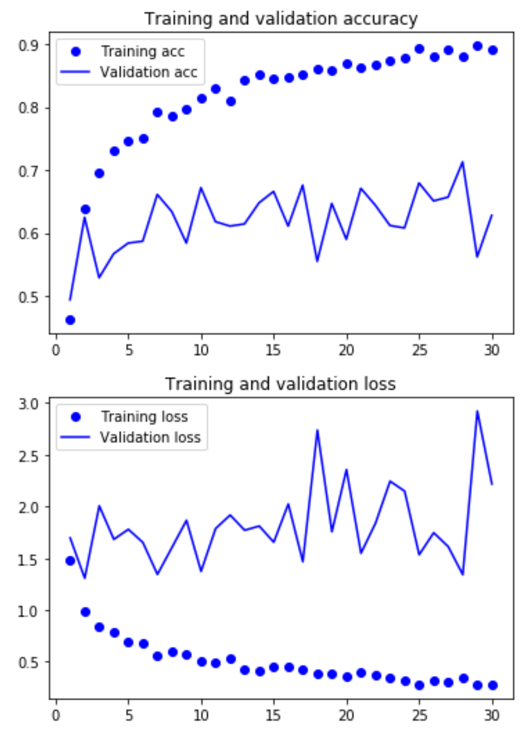


```
              Confusion Matrix
  Graph_Intro [[343 146  14 484  13   7]
    Intro_New  [  3 519   8 420  55   1]
Rep_Interview  [  0 122 471 276 136   0]
      Rep_New  [  0   3   1 995   2   0]
     Reporter  [  0 115  64 218 607   0]
    Video_New  [  8  48  17 608  52 272]]
```



```
Classification Report
              precision    recall  f1-score   support

  Graph_Intro       0.97      0.34      0.50      1007
    Intro_New       0.54      0.52      0.53      1006
Rep_Interview       0.82      0.47      0.60      1005
      Rep_New       0.33      0.99      0.50      1001
     Reporter       0.70      0.60      0.65      1004
    Video_New       0.97      0.27      0.42      1005

     accuracy                           0.53      6028
    macro avg       0.72      0.53      0.53      6028
 weighted avg       0.72      0.53      0.53      6028
```

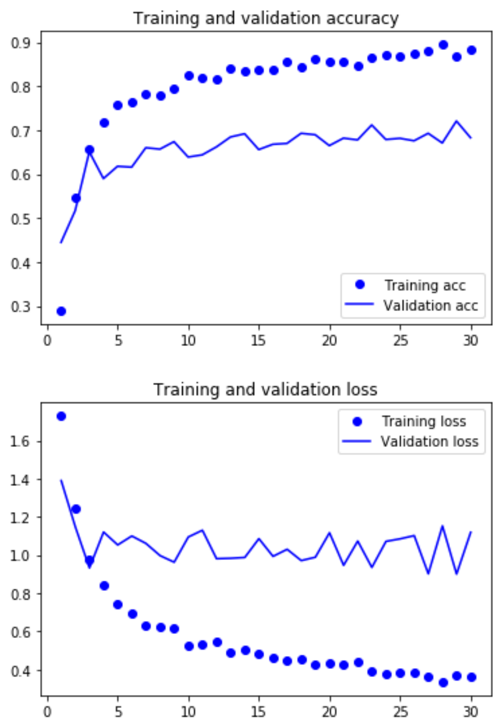
Fig A4.8 Inception V3 results – RMSprop



```
                     Confusion Matrix
 Graph_Intro [[419 164  14 390  10  10]
   Intro_New  [  6 562  11 333  92   2]
Rep_Interview [  0 119 516 187 183   0]
     Rep_New  [  0   2   0 997   2   0]
    Reporter  [  0  98  47 152 707   0]
   Video_New  [ 17  59  12 487  95 335]]
```

```
Classification Report
               precision    recall  f1-score   support

  Graph_Intro       0.95      0.42      0.58      1007
    Intro_New       0.56      0.56      0.56      1006
Rep_Interview       0.86      0.51      0.64      1005
      Rep_New       0.39      1.00      0.56      1001
     Reporter       0.65      0.70      0.68      1004
    Video_New       0.97      0.33      0.50      1005

     accuracy                           0.59      6028
    macro avg       0.73      0.59      0.59      6028
 weighted avg       0.73      0.59      0.59      6028
```

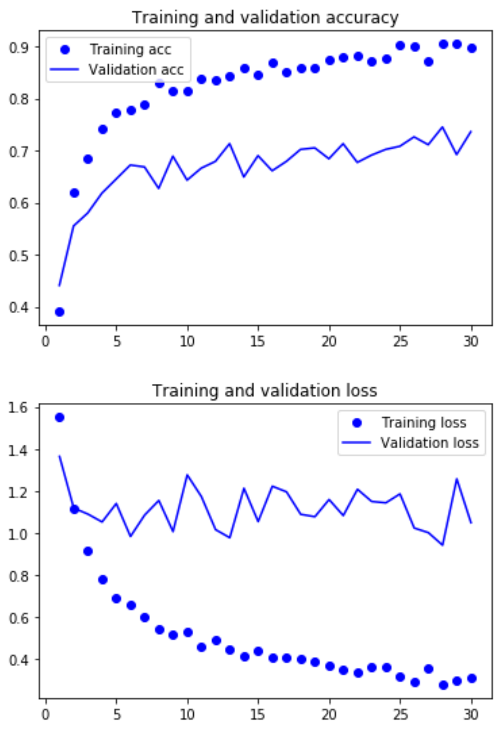
Fig A4.9 Inception V3 results – ADAM



```
                     Confusion Matrix
 Graph_Intro [[615 109   3 249  29   2]
   Intro_New  [  1 580  27 321  77   0]
Rep_Interview [  0 113 599 161 132   0]
     Rep_New  [  0  15  24 939  23   0]
    Reporter  [  0  67 151 121 665   0]
   Video_New  [ 10  34  19 410 163 369]]
```

```
Classification Report
               precision    recall  f1-score   support

  Graph_Intro       0.98      0.61      0.75      1007
    Intro_New       0.63      0.58      0.60      1006
Rep_Interview       0.73      0.60      0.66      1005
      Rep_New       0.43      0.94      0.59      1001
     Reporter       0.61      0.66      0.64      1004
    Video_New       0.99      0.37      0.54      1005

     accuracy                           0.62      6028
    macro avg       0.73      0.63      0.63      6028
 weighted avg       0.73      0.62      0.63      6028
```

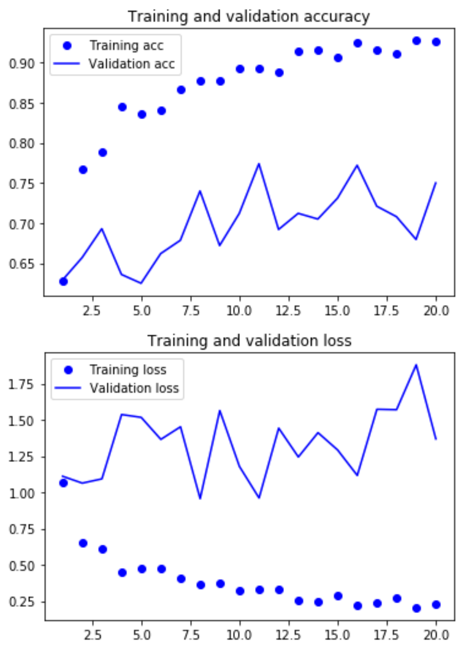
Fig A4.10 Xception results – SGD



```
                 Confusion Matrix
 Graph_Intro [[567 129   3 296  11   1]
   Intro_New  [  3 622   9 328  44   0]
Rep_Interview [  0 144 541 211 109   0]
     Rep_New  [  0   7  13 974   7   0]
    Reporter  [  0 104 130 167 603   0]
   Video_New  [ 19  56  15 465 103 347]]
```

```
Classification Report
               precision    recall  f1-score   support

  Graph_Intro       0.96      0.56      0.71      1007
    Intro_New       0.59      0.62      0.60      1006
Rep_Interview       0.76      0.54      0.63      1005
      Rep_New       0.40      0.97      0.57      1001
     Reporter       0.69      0.60      0.64      1004
    Video_New       1.00      0.35      0.51      1005

     accuracy                           0.61      6028
    macro avg       0.73      0.61      0.61      6028
 weighted avg       0.73      0.61      0.61      6028
```

Fig A4.11 Xception results – RMSprop



```
                 Confusion Matrix
 Graph_Intro [[611 147   3 221   8  17]
   Intro_New  [  3 692  18 267  25   1]
Rep_Interview [  0 219 586 141  59   0]
     Rep_New  [  1  25  26 934  15   0]
    Reporter  [  0 159 196 125 524   0]
   Video_New  [  6  29  20 339  60 551]]
```

```
Classification Report
               precision    recall  f1-score   support

  Graph_Intro       0.98      0.61      0.75      1007
    Intro_New       0.54      0.69      0.61      1006
Rep_Interview       0.69      0.58      0.63      1005
      Rep_New       0.46      0.93      0.62      1001
     Reporter       0.76      0.52      0.62      1004
    Video_New       0.97      0.55      0.70      1005

     accuracy                           0.65      6028
    macro avg       0.73      0.65      0.65      6028
 weighted avg       0.73      0.65      0.65      6028
```

Fig A4.12 Xception results – ADAM