uc3m | Universidad Carlos III de Madrid

Degree in Computer Science Engineering
2020-2021

Bachelor Thesis

"Classification of Noise Sources in LIGO Gravitational Wave Observations using Deep Neural Networks and Model Explainability Analysis"

Alfredo María Núñez Herrero

Miguel Ángel Patricio Guisado
Antonio Berlanga de Jesús
Madrid, Spain, February 15th 2021

This work is licensed under Creative Commons **Attribution – Non Commercial – Non Derivatives**

SUMMARY

This document uses state of the art image classification techniques to automate the detection of noise sources in gravitational wave observations from the Laser Interferometer Gravitational-Wave Observatory (LIGO). Additionally, it analyzes the resulting model using post-hoc local interpretation methodologies with the intent of defining possible paths of improvement.

Keywords: XAI, explainable AI, deep learning, deep neural networks, LIGO, astronomy, gravitational waves, saliency maps and post-hoc local interpretation.

DEDICATION

This project is dedicated to all the fantastic Computer Science Engineering professors at the Universidad Carlos III de Madrid, specially to both my mentors Antonio Berlanga de Jesús and Miguel Ángel Patricio Guisado. To my professional mentors Dr. Blerina Gkotse, David Tow, Dr. Federico Ravotti, Michael Ritchson and Dr. Raheem Beyah. Lastly, to my friends and family who have been an endless source of support while pursuing my degree, specially Daniel Sarmiento, Luis Muñoz del Pozo, María Nieto Petinal and Nicolás Salomone Pérez the fearless and adventurous companions I met as a first year student in 2013. Thank you for all your help.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1. Contextualization

In 2015, the Laser Interferometer Gavitational-Wave Observatory (LIGO) physically sensed the undulations in space time caused by gravitational waves generated by two merging black holes 1.3 billion light-years away. A physical measurement to one of Albert Eintein's general theory of relativity predictions. One of humanity's greatest scientific achievements [1].

In order to make such an accomplishment, scientists needed an incredibly sensitive system. The processes that generate gravitational waves can be extremely violent and destructive, but by the time waves reach Earth, they are thousands of billions of times smaller. As an example, the amount of space-time wobbling generated by the first gravitational waves detected at LIGO, was 1000 times smaller than the nucleus of an atom [2].

With such a sensitive system, noise can easily be confused with gravitational-wave signals. In order to be able to make valuable observations, it is important to identify and differentiate noise sources. One of the best ways to do so is still the human eye. Therefore, a citizen science project called Gravity Spy was started to help in this task. In the project, citizens have the opportunity to learn to characterize and classify potentially new sources of noise in the LIGO system [3].

The classifications made by citizens, are later processed and used to train a machine learning algorithm integrated as part of the LIGO system [4] as shown in Figure 1.1.

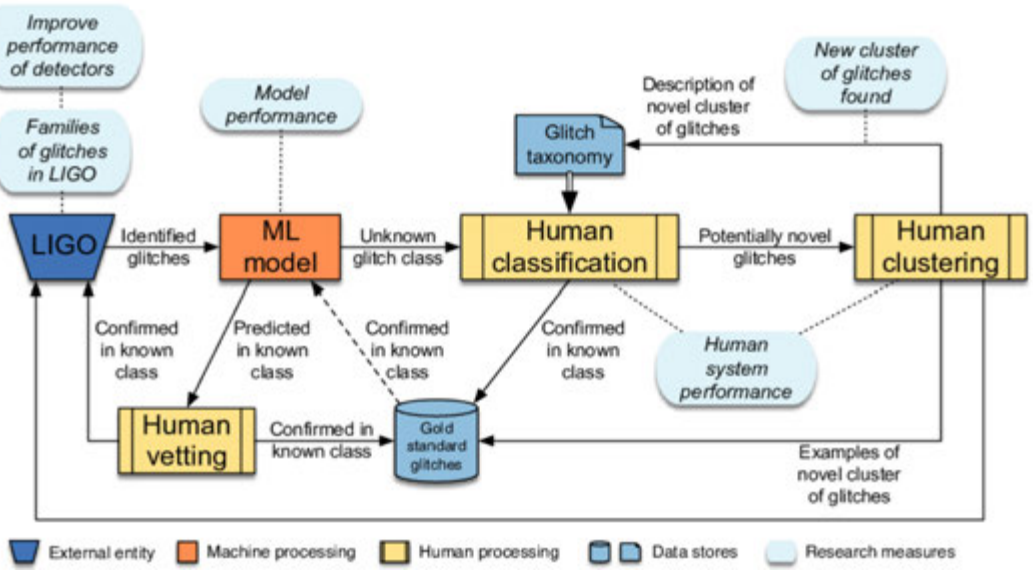

Fig. 1.1. Gravity Spy Architecture. Taken from [4].

1.2. Objectives

The objectives of this project are two: First, develop a machine learning algorithm capable of correctly classifying known types of noise sources in gravitational-wave signal observations. Second, to analyze the prediction capabilities of the model using post-hoc local interpretation methods and define possible paths of improvement for the deep learning black box model.

1.3. Legal Framework

The legal framework for this project includes software tools, data privacy, intellectual property and legal implications of the implementation of the project.

1.3.1. Software

The software tools and platforms used in the development of the project are listed below:

- **Google Cloud Platform**: a suite of cloud computing services, operated by Google LLC, that runs on the same infrastructure that Google uses internally for its end-user products.
- **Kaggle Kernels**: a cloud-based workbench for data science and machine learning. Operated by Kaggle, a subsidiary of Google LLC.
- **TensorFlow**: a free and open-source software library for machine learning developed by the Google Brain Team. Released under the Apache License 2.0.
- **Keras**: an open-source software library that provides a Python interface for artificial neural networks. Initially developed by François Chollet and now maintained by the open-source project contributors. Released under the MIT license.
- **Tf-keras-vis**: an open-source software library in Python used to visualize mdoels developed using TensorFlow and Keras. Initially developed by Yasuhiro Kubota and now maintained by the open-source project contributors. Released under the MIT license.
- **Python**: an interpreted high-level general-purpose programming language. Designed by Guido van Rossum and developed and maintained by the Python Software Foundation. Released under the Python Software Foundation License.

All the software tools used are appropriate for non-commercial research use and therefore for this project.

1.3.2. Intellectual Property

The intellectual property of this project belongs to the author, since it has been developed individually without using information belonging to a private entity and without the intervention of a public or private entity. Anyhow, it is encouraged to replicate and modify the work of this project for non-commercial applications with the appropriate recognition to the author.

1.3.3. Data Privacy

The dataset used in this project was made public in November 2018 [5] in the platform Zenondo under the Creative Commons Attribution 4.0 International license.

1.4. Socio-Economic Environment

The immediate application of the techniques presented in this document, is to improve the architecture of the Gravity Spy project as previously shown in Figure 1.1. The use of post-hoc local methods could help them find why their image recognition model is under performing an use the information in future development iterations.

Besides astrophysics research, the techniques used in this project can be applied in multiple other areas. One of the areas is the judiciary system. As an example, in the U.S., black-box algorithms are used in the judiciary system to evaluate the like hood of recidivism. An example of this kind of systems is COMPAS [6]. These systems aim to help judges evaluate cases. Post-hoc methods could help explaining the outputs produced by the algorithm, increasing the trustworthiness and acceptance by the general public.

Another application is automated transportation systems. A clear example is autonomous driving. Image recognition is an essential task for autonomous vehicles. The methods used in this project could help assure correct decisions are being taken by the vehicle.

Medical imaging is another area where interpretability methods are greatly valued. Image processing and analysis are involved in diagnostic processes. The impact of algorithm outputs in this area is inherently high, and the use of interpretability methods can help lower the risk of performing an incorrect diagnosis.

Form the ethical point of view, Deep Neural Networks are being increasingly used across all sectors in industry. Their use is automating processes and increasing organization's efficiencies. As a consequence, deep neural networks are involved in decision-making processes of progressively increasing risk. Their obscure logic and our incapacity of examining their rationale, is increasing the public's concern. This sentiment of distrust is currently reflected in regulations such as the GDPR. The GDPR's Article 15, requires "meaningful information about the logic involved" in automated decision-making [7].

This means any organization using automated systems to make decisions, needs to also provide a method capable of providing meaningful information about the logic involved in making the decision, making interpretability necessary.

Overall interpretation methods can play a key role in the development of trustworthy automated systems for high-stake applications.

1.5. State of the Art

1.5.1. Previous Work

The first objective of the project, classification of gravitational wave noise sources using machine learning, has previously been attempted before using various machine learning methodologies [8] [9] [10] [11] [12] [13]. As part of the first objective, the machine learning algorithm developed should have a competitive accuracy compared to latest attempts in literature. But for the second objective, using visualization techniques to understand the outputs of the model, no previous related work has been found applied to gravitational waves. Therefore, in this case, no comparisons will be available to evaluate the methodology presented in this project.

1.5.2. Methodologies

For the first objective, convolutional neural networks have been chosen to solve the image recognition problem. For the second objective, Vanilla Saliency, SmoothGrad and Grad-CAM have been chosen to carry the explainability analysis.

In order to explain the reasoning behind the selection of methodologies for this project, it is necessary to give a contextual background of the problems attempted: image recognition and post-hoc local interpretation.

Image Recognition Contextual Background

In 1959, David Hubel and Torsten Wiesel published a paper [14] which would set the foundations of image recognition algorithms. In their publication, they demonstrated the existance of simple and complex neurons in the primary visual cortex of a cat's brain. They also showed how visual processing begins with the recognition of simple structures such as edges.

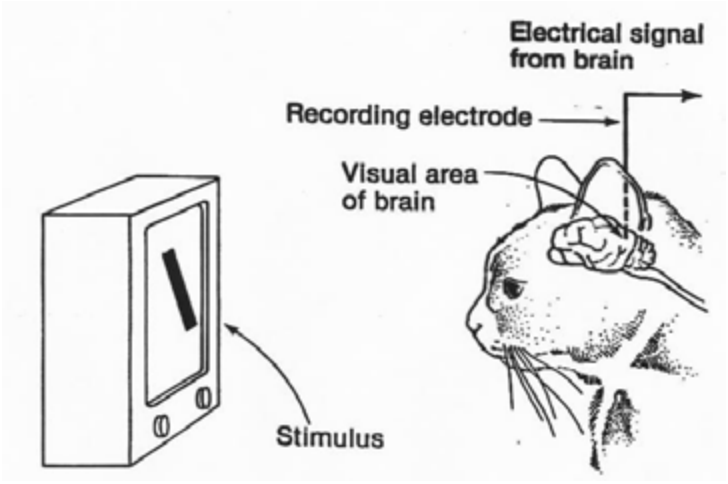

Fig. 1.2. Illustration of Hubel and Wiesel's experiment. Taken from [15].

Their work inspired future scientists to develop some of the first computer vision and deep learning algorithms making use of convolutional filters, such as Neocognitron [16] in 1980 and LeNet-5 [17] in 1989.

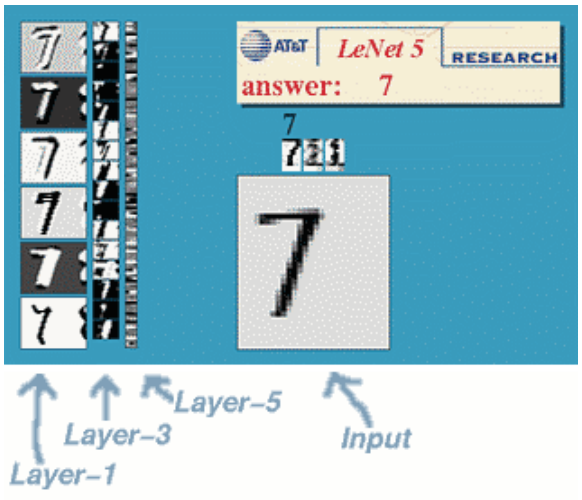

Fig. 1.3. Example of LeNet-5 in action. Taken from [18]

In the following years, scientists attempted harder problems such as object and face recognition with promising results. As the popularity of these problems grew, a way to compare the performance of different computer vision techniques became necessary, and a benchmark image dataset was developed to provide the same source of information for different image recognition techniques.

In 2005 the PASCAL VOC challenge offered a dataset containing approximately 20,000 images and 20 object classes. Annual competitions were scheduled to study which where the best performing techniques. Following the footsteps of the PASCAL VOC chal-

lenge, in 2010 the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [19] was established. The ILSVRC contained over a million images of 1000 different classes.

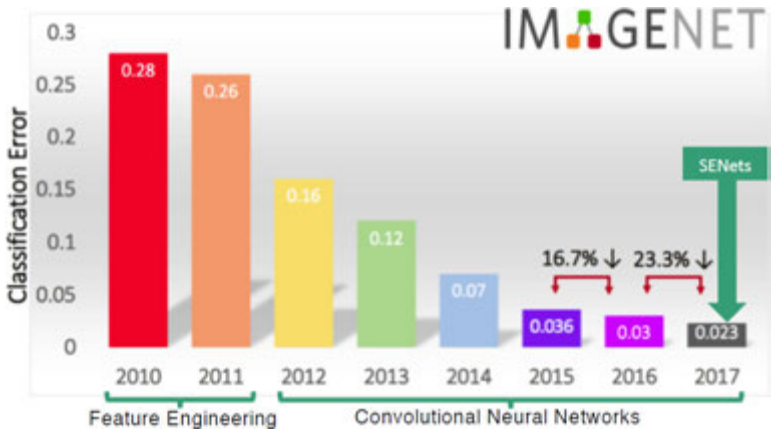

Fig. 1.4. Best error rates and neural network types for ILSVRC competitions. Taken from [20].

In 2012 convolutional neural networks became the image classification technique of choice after a team from the University of Toronto achieved an error rate of 16% at ILSVRC, a drastic improvement compared to the rest of competitors. The next best competitor's error rate was of 26% [21]. After 2012, error rates were reduced to a few percent and ILSVRC winners have always been convolutional neural networks as shown in Figure 1.4.

Post-hoc local interpretation using pixel attribution

In the last decade, deep neural networks have revolutionized the world of business and technology. With the expansion of deep learning, one of the greatest challenges has been to address the problem of interpretability. High-risk environments require proper understanding about how decisions are made by opaque deep neural networks. With the intent to help address this problem, scientist have developed multiple techniques.

Interpretability techniques can be distinguished in two main categories: model-based and post-hoc. Within post-hoc techniques, most of them can be categorized into prediction-level and dataset-level interpretations, which can be referred to as local and global interpretations respectively [22]. This document focuses on local interpretation methods, specifically pixel attribution methods for convolutional neural networks.

In 2013, two publications presented the use of gradient-based pixel attribution methods in convolutional neural networks [23] [24]. As stated in [24], they presented "a novel way to visualize the activity within the model. This reveals the features to be far from random, uninterpretable patterns. Rather, they show many intuitively desirable properties such as compositionality, increasing invariance and class discrimination as we ascend the layers." Also, the authors showed how the visualizations can "be used to debug prob-

lems with the model to obtain better results, for example improving on Krizhevsky et al. (Krizhevskyet al., 2012) impressive ImageNet 2012 result.". Seeing what visualization methods could do to extract information about the inner structure of convolutional neural networks, scientists started to build on top of the work published by M.D. Zeiler, R. Fergus, K. Simonyan, A. Vedaldi and A. Zisserman.

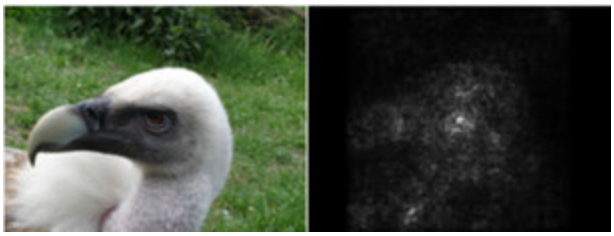

Fig. 1.5. Vanilla saliency. Taken from [24].

In 2014, a publication by Springenberg et. al [25] presented an improved back-propagation method "GuidedBackprop" for gradient-based pixel attribution methods like Vanilla Saliency [24]. It avoided the propagation of negative gradients, reducing the noise in the resulting map.

In 2016, a publication by Selvaraju et. al from the Georgia Institute of Technology [26] presented a new visualization method called Gradient-weighted Class Activation Mapping (Grad-CAM). The method solved multiple important problems previous visualization methods were having. First, the method solved problems faced by the use of backpropagation in gradient-based methods like Vanilla Saliency. By using the activation map of the last convolutional layer, it didn't have to propagate gradients all the way to the input layer. Second, as a consequence of the first point, the method can be used for any convolutional neural network model. As mentioned in the paper "In this work, we proposed a novel class-discriminative localization technique for making any CNN-based model more transparent by producing visual explanations.". Third, it performed well with images containing multiple classes as shown in Figure 1.6.

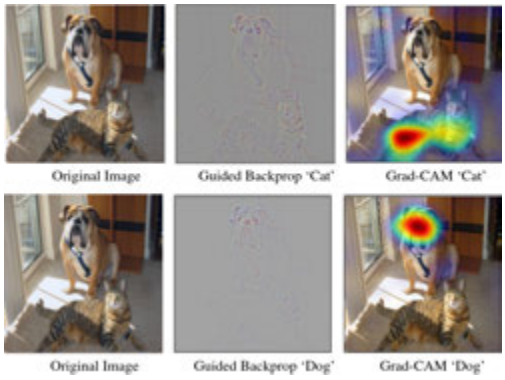

Fig. 1.6. Grad-CAM used in images containing multiple classes. Taken from [26].

In 2017, a publication by Smilkov et. al [27] showed it is possible to compute sharper saliency maps in two ways. From the paper "First, averaging maps made from many small perturbations of a given image seems to have a significant smoothing effect. Second, that effect can be enhanced further by training on data that has been perturbed with random noise". By adding noise to the data, the method improves the resulting map at a higher computational cost.

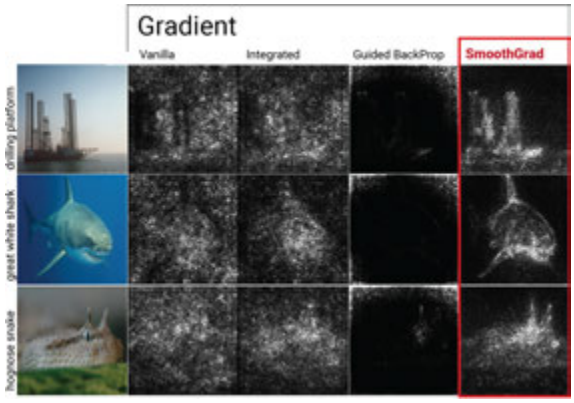

Fig. 1.7. Comparison between SmoothGrad and Vanilla saliency. Taken from [27].

Multiple additional methods were developed in following years, but this project is focusing on the methods mentioned previously, Vanilla Saliency, SmoothGrad and Grad-CAM. The methods have been chosen due to their extensive popularity and reasonable computational cost.

1.6. Document Structure

This document is structured in four main areas: introduction, research and development, results and conclusions and future work. First, the introduction provides all the contextual information necessary to understand in detail the project. Second, the research and development explains the journey from the beginning of the project until the final results. It is divided in two areas, model development and explainability analysis. Third, the results show which are the models chosen for analysis and their performance, as well as the results of using post-hoc pixel attribution methods to analyze the models. Finally, the last section summarizes the main lessons learned in the project and also sets possible paths to continue and improve this project.

2. RESEARCH AND DEVELOPMENT

2.1. Data

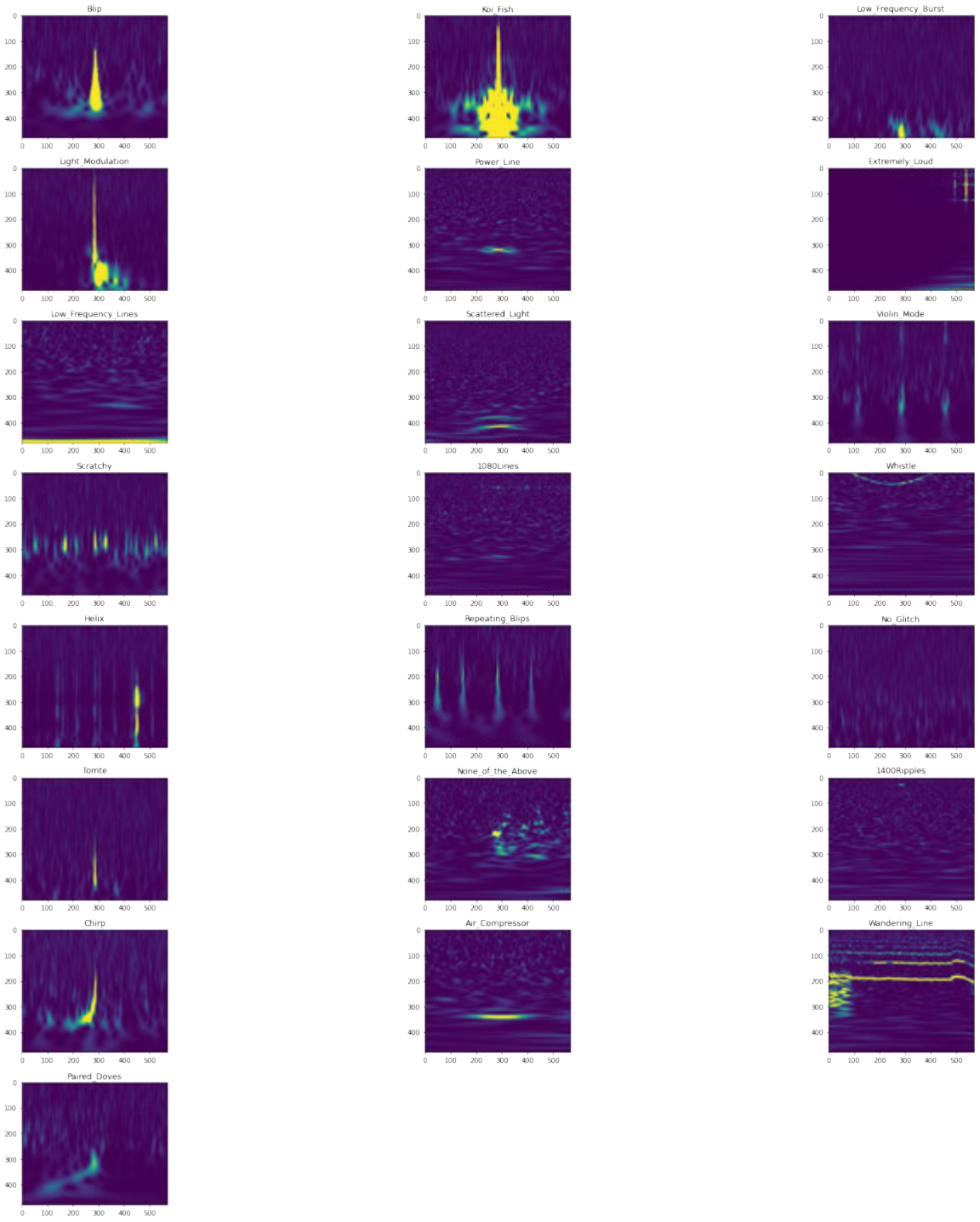

Fig. 2.1. Sample images for each class in the dataset.

The original dataset was published in Zenodo [5] at the same time as the publication by Zevin et. al [4] which presented for the first time the Gravity Spy project.

TABLE 2.1. AVERAGE CLASS SET PROPORTIONS.

Set Name	Average size
Train	14.69
Test	70.21
Validation	15.09

Table 5.1 shows the total number of images is 7966 across 22 different classes. The proportion images per set and class are displayed in 5.2 5.3. Table 5.2 shoes the difference between class image proportions in the dataset. The difference is considerable, with a minimum of 0.53% of total images for the Wandering_Line class and a maximum of 22.86% of images for the Blip class. This disproportion is expected to be reflected in the model's performance.

Table 2.1 contains information about set proportions. Train, test and validation sets are balanced across the whole dataset and also individually in each class. Table 2.1 shows the average proportions for all classes is approximately 15% for test and validation sets and 70% for the train set.

The images from the original dataset have been modified to be used in the model development. Original images include three dimensional information. Figure 2.2 shows differences between an original image and the images used to train the model. The title, axis and legend are removed from the image.

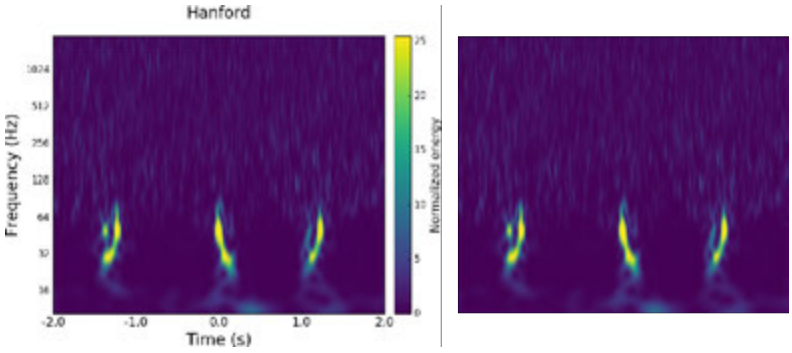

Fig. 2.2. Comparison between image from original dataset and image used for model development.

The dataset contains four versions of gravitational wave observations. The versions differ only in the observation time from 0.5 to 4 seconds. Figure 2.3 shows an example of each of the four versions for the same gravitational wave observation. An example of image name is "L1_TbHPjqgaIB_spectrogram_4.0.png". The observation time is included as part of the name.

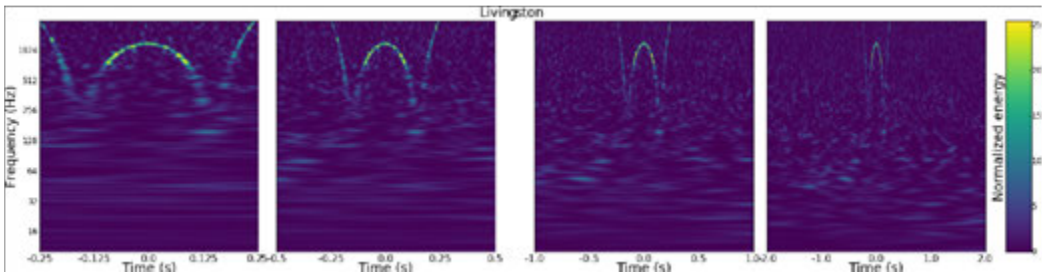

Fig. 2.3. Example of gravitational wave observation in different time spans.

2.2. Classification

As mentioned previously, convolutional neural networks are the type of Artificial Neural Network chosen to carry the classification task. Image classification can be performed with traditional neural networks, but the computational costs are usually extremely high. convolutional neural networks change this, by offering a significantly more computationally friendly alternative. Convolutional neural networks are more efficient than traditional neural networks (e.g. Multilayer Perceptron) in image classification tasks, because image's pixels aren't directly connected to dense layers, but first undergo a processing stage. The processing stage is part of the convolutional neural network and it is usually composed by two types of layers, the convolutional and pooling layers.

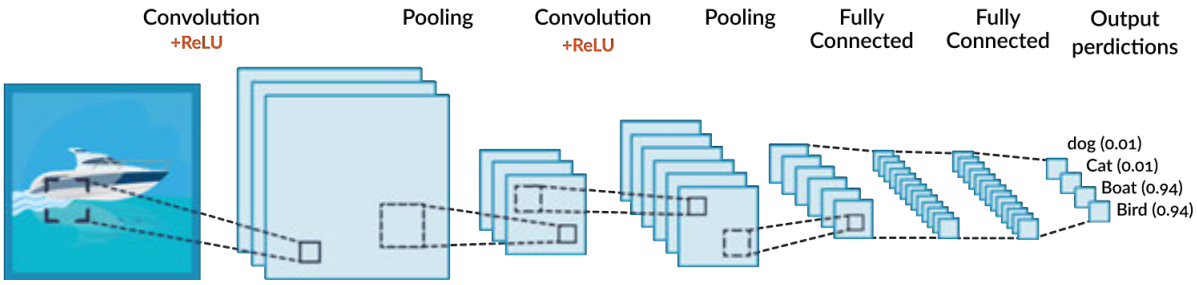

Fig. 2.4. Typical structure of convolutional neural network. Taken from [28].

Figure 2.4 shows a typical structure for a convolutional neural network. The convolutional layers, by applying matrix operations to pixels, reduce the number of inputs later used in dense layers. The pooling layers also reduce the information pushed forward, by calculating a single value from the output returned by convolutional filters. Together, they reduce the number of inputs for dense layers, which are finally trained to perform the classification task.

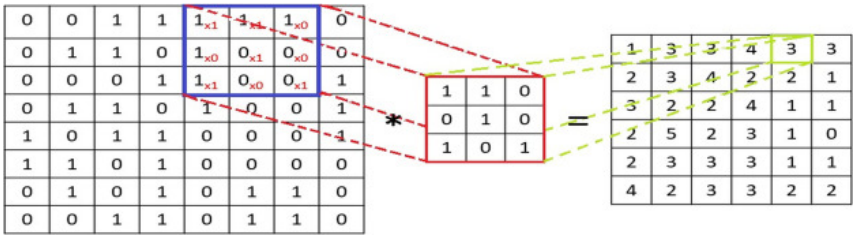

Fig. 2.5. Simplified example of convolutional filter operation. Taken from [29].

Figure 2.5 shows a simplified example of how convolutional filters slide across the image and return feature maps. The calculation performed is represented in the following equation.

$$ActivationMap = Input * Filter = \sum_{y=0}^{columns} \sum_{x=0}^{rows} Input_{x-p,y-q} * Filter_{x,y} \tag{2.1}$$

After the convolutional layer comes the pooling layer. Figure 2.6 shows an example of the two most common operation performed in the pooling layer. The concept is similar, the operation is performed across the matrix producing a smaller matrix in dimension.

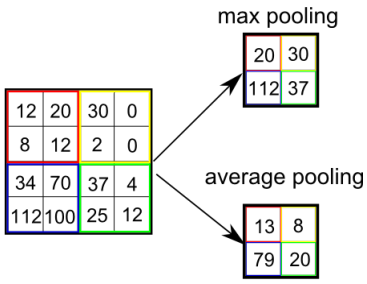

Fig. 2.6. Simplified example of pooling operations. Taken from [30]

Finally, after a variable number of convolution and pooling layers, the result is flattened into a one dimensional array, which is fed into the fully connected layers as described in Figure 2.4.

Now having gone over the characteristic details of convolutional neural networks' structure, the focus will shift towards the development of the model for this project.

Initially, at the beginning of the iterative process, the model was designed without following any concrete guidelines besides the structure mentioned above. Parameters such as the number of convolutional and pooling layers, padding methods, the number of filters per layer, the stride of kernels, activation functions and the pooling operations were undefined. Some of the parameters, like the stride of kernels, activation functions and

padding methods, have remained constant during the iterative development of the project. First, strides were set to 1 for convolutional kernels and 2 for pooling kernels in the three dimensions (height, width and channels) for all layers. Second, pooling kernel sizes are set to 2 for all dimensions. Third, the padding strategy is set to zero padding. Fourth, ReLU activation functions are used in convolution and dense layers to not propagate negative values. Fifth and last, at the end of the network, two fully connected layers. The first one followed by a ReLU activation function and the second one followed by a softmax activation function for normalization.

With these base parameters defined and constant, the iterative process started with the variables number of convolutional and pooling layers, number of convolution kernels per layer, and convolutional kernel sizes per layer. Table 2.2 shows the most relevant iterations of the process.

TABLE 2.2. MOST RELEVANT MODEL ITERATIONS.

iter_num[a]	num_c_p_lay[b]	num_conv_k[c]	conv_k_size[d]	t_acc[e]
1	2	(5,10)	(20,20)	0.2314
2	2	(25,50)	(20,20)	0.2314
3	2	(25,50)	(20,40)	0.2314
4	3	(25,50,100)	(20,20,20)	0.2314
5	3	(25,50,100)	(20,30,40)	0.2314
6	3	(25,50,100)	(21,31,41)	0.8964
7	3	(25,50,100)	(9,15,21)	0.9117
8	3	(5,5,5)	(3,3,3)	0.8970
9	2	(5,5)	(3,3)	0.9201
10	1	(5)	(3)	0.9242

[a] Iteration number.

[b] Number of convolution and pooling layers.

[c] Number of convolution kernels per layer. Displayed inside parenthesis in order.

[d] Convolution kernel sizes. Displayed inside parenthesis in order. Same size for both dimensions.

[e] Test accuracy.

Initially, as shown in Table 2.2, the network was composed of two convolutional and pooling layers, with a convolutional kernel for both of 20x20. The number of filters doubled from 5 to 10 between layers. The accuracy of the model was low but above a random response. Being 22 classes, a random model should have an accuracy of 4.55%, and the model achieved an accuracy of 23.14%, which is 5 times higher. Therefore, the model was learning to discriminate between classes.

In an attempt to improve the model, future iterations tried increasing the complexity of the model by progressively increasing the number of filters per layer, the size of convolutional kernels and the number of layers. Curiously, for these attempts, not only did

the test accuracy not increase, but it also remained constant. It is still unclear the reason behind achieving exactly the same accuracy using different models.

The turning point during development, came when using odd sizes for convolutional filters. Note the similarity between iteration number 5 and 6. Slightly modifying the sizes of convolutional kernels resulted in an increase of 66 percentage points. For comparison, Figure 2.2 and 2.2 show the accuracy and loss per epoch for iterations 5 and 7 of Table 2.2.

The use of even kernel sizes in convolutional layers is not recommended, because it goes against the purpose of convolutions. Convolutions are supposed to extract information from a source pixel and its neighbours. Therefore, the size of the kernel must be odd to maintain a symmetrical number of pixels in each direction from the source pixel. This was the cause of the model's poor performance.

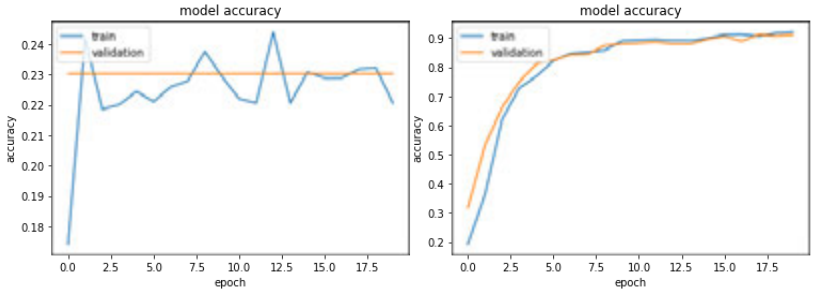

Fig. 2.7. Train and validation accuracy for iterations 5 and 7 of Table 2.2.

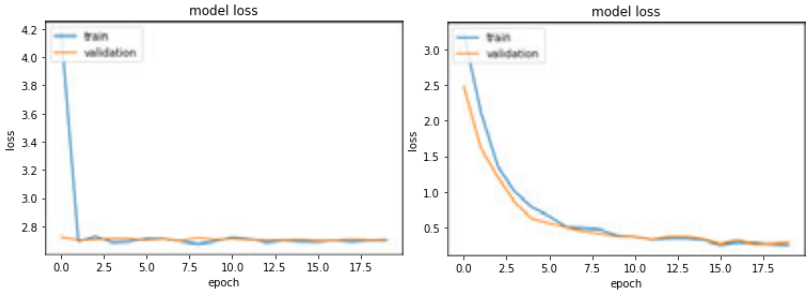

Fig. 2.8. Train and validation loss for iterations 5 and 7 of Table 2.2.

From there on, further modifications increasing the complexity of the model didn't produce significant performance improvements. But it was possible to improve in the opposite direction. Reducing the pattern complexity recognizable by the model. Using smaller convolutional kernels and less layers proved to be more effective.

This makes sense, since the geometric complexity of gravitational wave observations isn't great, and as a result, smaller geometric patterns recognized in shallow layers of the network, play a greater role in this classification problem than higher complex patters learnt in deeper layers. Overall this classification problem is easier than the typical (e.g. facial recognition). It lacks factors which other more complicated problems have. Factors

like highly complex geometrical elements (e.g. human faces), perspective, lighting, poor image quality, similar backgrounds, etc.

2.3. Post-hoc local interpretation

Once the classification model was developed, the development of the model explainability analysis comes next. Due to the nature of convolutional neural networks, the use of model-based methods isn't possible. Therefore, the focus is on post-hoc methods, specifically pixel attribution methods. They can be categorized as: model agnostic and non model agnostic methods. Model agnostic methods are methods applicable to any model independently their structure, and non model agnostic methods are only applicable to models with a specific structure. The methods used in this project are non-model agnostic, and are displayed in Table 2.3.

TABLE 2.3. POST-HOC LOCAL METHODS USED IN ANALYSIS.

Name	Type
Vanilla Saliency	Gradient-based saliency map
SmoothGrad	Gradient-based saliency map
Grad-CAM	Class activation map

The idea is to visualize saliency maps using the methods shown in Table 2.3 for some of the classes with the lowest and highest error rates. Their comparison should give some insight on how well is the model performing, and help define ways to improve it. A detailed explanation on how the methods work is provided below.

First, Vanilla saliency. It is one of the first and most simple methods. The method, developed by Simonyan et. al [23], makes use of vanilla backpropagation, which is normally used for adjusting network weights by calculating the gradients of the loss function w.r.t to weights. In Vanilla Saliency is different, the gradient of the output class w.r.t. input pixels is calculated. In other words, the rate in which pixels change the probability of the output class.

The mathematical expressions below show how vanilla backpropagation propagates gradients through Rectified Linear Units (ReLus). Equation 2.2 is the forward pass through a ReLU function. It turns all negative value to zeros. Equation 2.3 is the backward pass in vanilla backpropagation. It takes into account the values previously passed forward, and the places zeros where negative values used to be in the forward pass.

$$f_i^{l+1} = relu(f_i^l) = max(f_i^l, 0) \tag{2.2}$$

$$R_i^l = (f_i^l > 0) \cdot R_i^{l+1}, where\ R_i^{l+1} = \frac{\partial f^{out}}{\partial f_i^{l+1}} \tag{2.3}$$

Using Equation 2.3, Vanilla Saliency propagates the gradients of the classification score all the way to the input features.

Second, SmoothGrad. It is a method of reducing noise in saliency map calculations, by averaging out multiple saliency calculations of the same image with slight random changes in pixel values. The averaged result of the individual saliency calculations, produces a visualization which highlights better the most influential pixels of the input image. Since the operation is an average of multiple saliency maps, the calculation is the same as vanilla saliency.

Third, Grad-CAM. To understand how Grad-CAM works, first a previously proposed method by Zhou et. al [31] called CAM needs to be explained. Grad-CAM is a generalization of CAM.

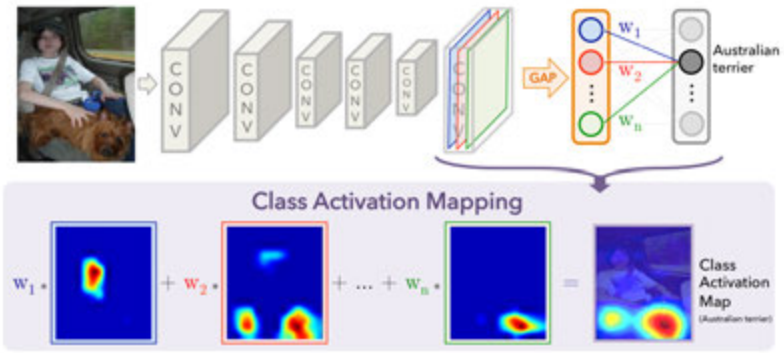

Fig. 2.9. CAM architecture. Taken from [31].

The paper authors first explain how convolutional layers have a remarkable ability to localize objects, and how that precious information is lost when the last convolutional layer is flattened and connected to a fully connected layer. In order to take advantage of this localization information for visualization purposes, the authors use a modified network architecture as shown in Figure 2.9. This architecture, instead of connecting the last convolutional layer into a fully connected layer, it performs Global Average Pooling (GAP) and afterwards a simple linear weighted operation. As Equation 2.4 shows, GAP is the average of all the values in a feature map.

Once the weights of the last fully connected layer are trained, they represent the influence of each feature map in the classification. Lastly, the values in feature maps are multiplied by the trained weights as shown Equation 2.6, producing a heatmap maintaining the localization information stored in feature maps. Equation 2.5 shows the network's score function, which is the same as the one used to calculate the heat map in Equation 2.6.

$$F_k = \sum_{x,y} f_k(x, y) \tag{2.4}$$

$$S_c = \sum_k w_k^c F_k = \sum_{x,y} \sum_k w_k^c f_k(x,y) \tag{2.5}$$

$$M_c(x,y) = \sum_k w_k^c f_k(x,y) \tag{2.6}$$

One of the drawbacks of CAM is the use of a different architecture, which is less accurate than standard convolutional neural network architectures. Grad-CAM was designed to fix this problem. The proposed method by Selvaraju et. al [26] is a generalization of CAM to any convolutional neural network architecture. The authors realized the information contained in the weights of the final layer in CAM is equivalent to the gradients of the output class w.r.t the last feature map weights, which can be calculated using back-propagation. The Grad-CAM architecture is as shown in Figure 2.9.

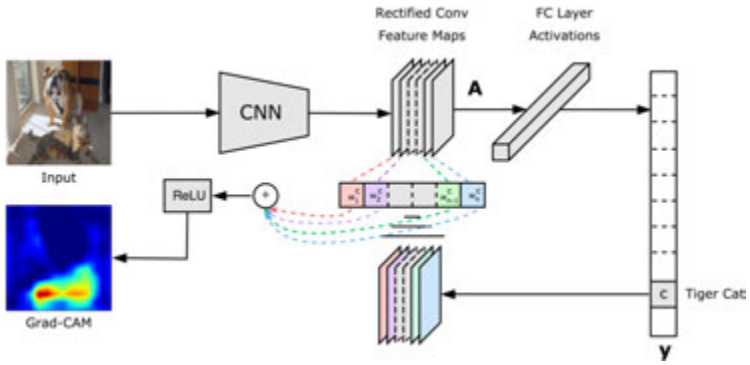

Fig. 2.10. Grad-CAM architecture. Taken from [26].

To calculate a saliency map with Grad-CAM, firstly the class activations before the softmax function have to be set to zero besides the target class. Secodly, the gradients of the class w.r.t the target rectified feature maps have to be calculated. It doesn't necessarily have to be the last convolutional layer. Thirdly, the gradients are global average pooled to obtain weights as shown in Equation 2.7. Lastly, the weighted combination of feature maps is calculated and passed through a ReLU to only display the positive contributions as shown in Equation 2.8.

$$\alpha_k^c = \frac{1}{Z} \sum_i \sum_j \frac{\partial y^c}{\partial A_{ij}^k} \tag{2.7}$$

$$L_{Grad-CAM}^c = ReLU(\sum_k \alpha_k^c A^k) \tag{2.8}$$

Altogether, the methods explained in this section will help analyze the positive and negative classifications in the network using the Gravity Spy dataset and identify points of improvement.

3. RESULTS

3.1. Classification

As seen in Table 2.2 the best performing model is the model with a single layer, three filters of size 5x5. The complete structure of the model is shown in Figure 5.1. Although this is the best performing model, the explainability analysis will be run with three different models Model_0, Model_1 and Model_2 to study how model depth affects visualizations. Therefore this section will explain the results for the three models. The models correspond to the iterations 8, 9 and 10 shown in Table 2.2.

Figures 3.1 3.2 and 3.3 show the weight optimization process during training. All the models converge successfully and the validation and test accuracies remain considerably similar, which is a sign the model is not overfitting.

Initially the training epochs were set to 20, to allow sufficient time for models to reach their maximum accuracies and even overfit. After using odd size convolutional filters, models were converging faster than previous iterations and the training epochs were reduced to the half. Reducing the training epochs avoided overfitting.

Note the use of dropout layers is common to regulate the network and prevent overfitting, but in this case limiting the number of training epochs was sufficient. Also, since the use of dropout layers didn't affect the performance of the model significantly, in the end they weren't included.

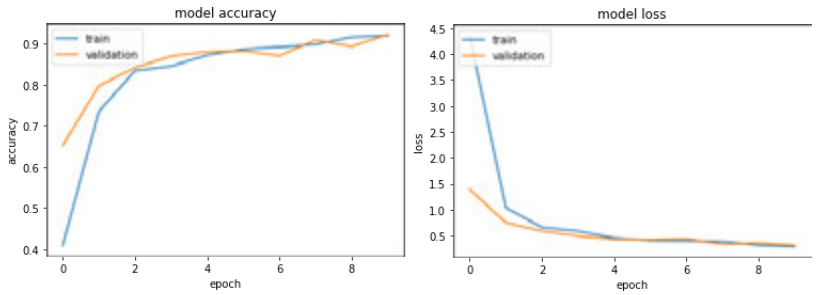

Fig. 3.1. Model_0 accuracy and loss per epoch.

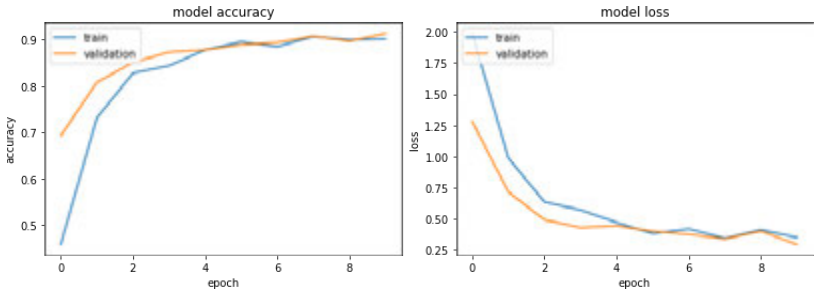

Fig. 3.2. Model_1 accuracy and loss per epoch.

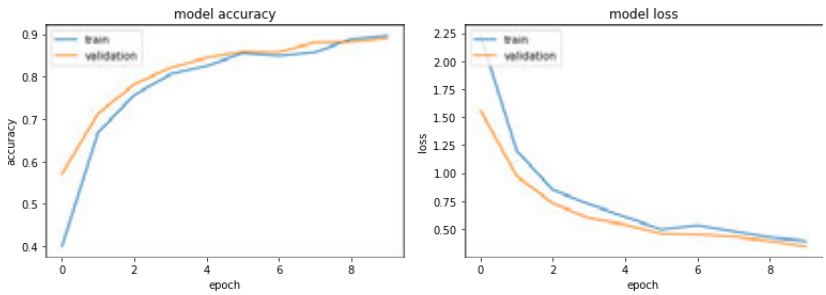

Fig. 3.3. Model_2 accuracy and loss per epoch.

Regarding image classes, Tables 5.4, 5.5 and 5.6 show the accuracy of the model per class. Some of the worst predicted classes for the best performing model, Model_0, are the class Paired_Doves and Wandering_Line with an accuracy of 13% and 39% respectively. Also, Table 5.1 shows how the classes Wandering_Line and Paired_Doves have the least number of records in the dataset. Together they only sum 69 records, which is less than the number of records of 90% of the remaining classes in the database and 0.086% of the overall records in the dataset. There is small margin for improvement if the model doesn't have enough images to train.

Looking in the opposite direction, we find classes such as Blip and Koi_Fish. Together they contain more than 30% of all the dataset records. Curiously they don't hold the highest positions regarding accuracy. Table 5.4 show us they are in 6th and 5th position, which is the same as their average position across all three models. Contrarily to the Wandering_Line and Paired_Doves classes, due to the high number of images, the accuracies of Blip and Koi_Fish could be improved by processing the data. Processing the data might make the task easier for the models and improve their accuracies, but it doesn't necessarily mean it will improve its performance in a real case scenario.

Regarding the architecture of the model, using a smaller number of layers has proved to be more effective than deeper networks. As previously mentioned, not having to deal with factors like perspective, complex patterns, similar backgrounds and lighting variations in this problem probably allows the model to discriminate between classes using simple patterns, which can be learned in early convolutional layers.

Note although the best performing model only has one convolutional and pooling layer, it doesn't less computationally costly. Comparing Figures 5.1, 5.2 and 5.3 the number of parameter in Model_0 is more than 4 times larger than Model_1 and more than 18 times larger than Model_2. Performing less convolutions, requires a higher number of inputs in the fully connected layer. Therefore, if computational costs are included in the performance measure, Model_1 would be a better performing model than Model_0.

Overall the development of the convolutional neural network has been a success. The most accurate model has a testing accuracy of 92.42%, which is 88 percentage points more accurate than random selection for 22 classes.

3.2. Explainability Analysis

3.2.1. Visualizations

Model_0

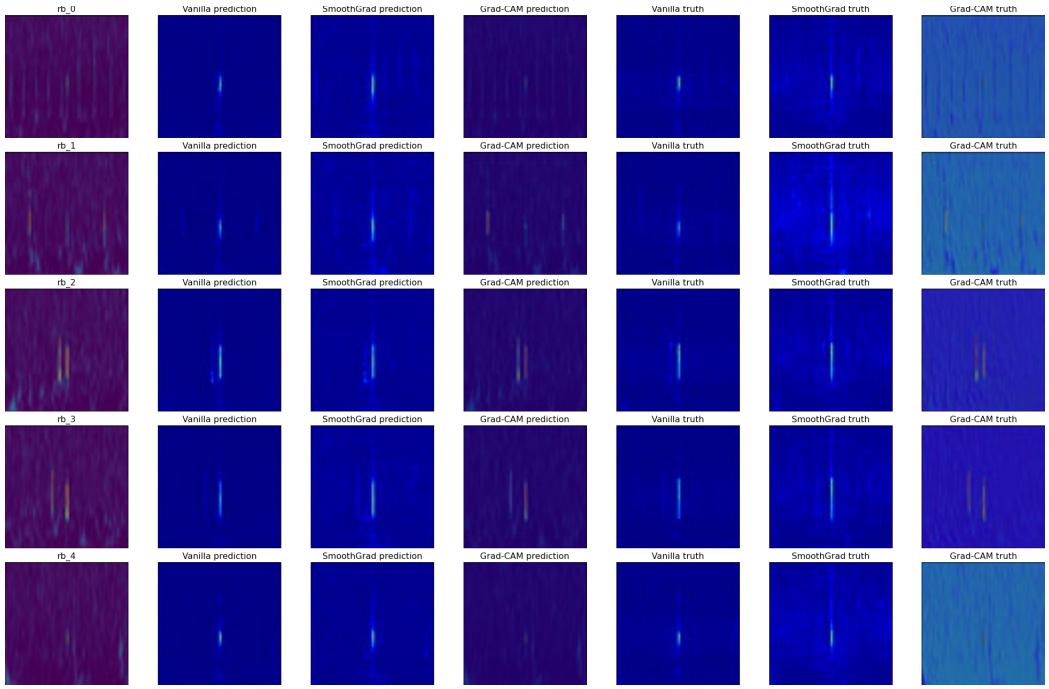

Fig. 3.4. Visualizations of incorrect predictions for class Repeating_Blips and Model_0. Predicted class is Blip.

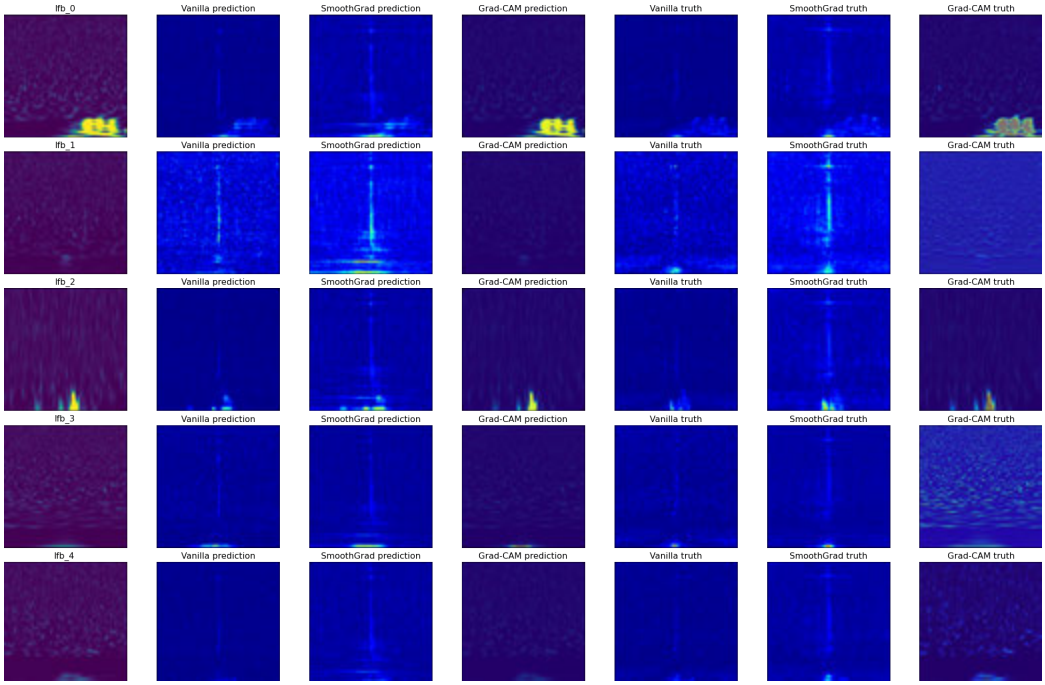

Fig. 3.5. Visualizations of incorrect predictions for class Low_Frequency_Burst and Model_0. Predicted class is Low_Frequency_Lines.

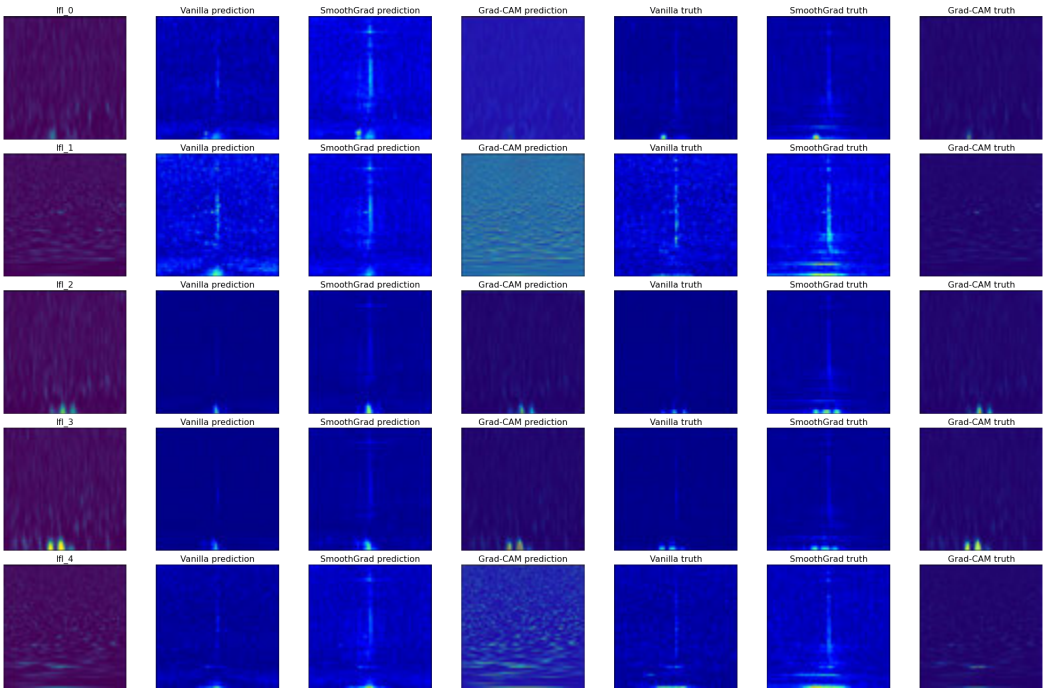

Fig. 3.6. Visualizations of incorrect predictions for class Low_Frequency_Lines and Model_0. Predicted class is Low_Frequency_Burst.

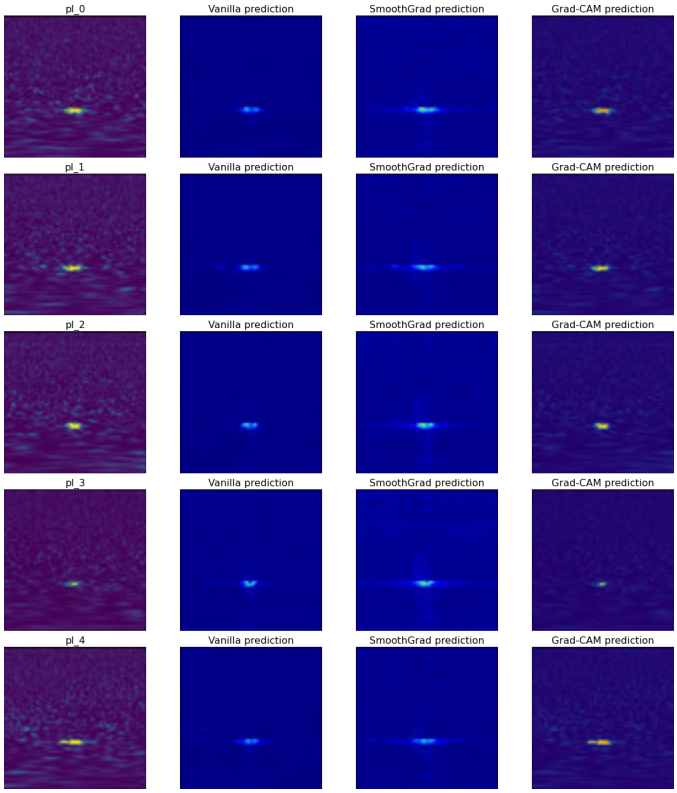

Fig. 3.7. Visualizations of correct predictions for class Power_Line and Model_0.

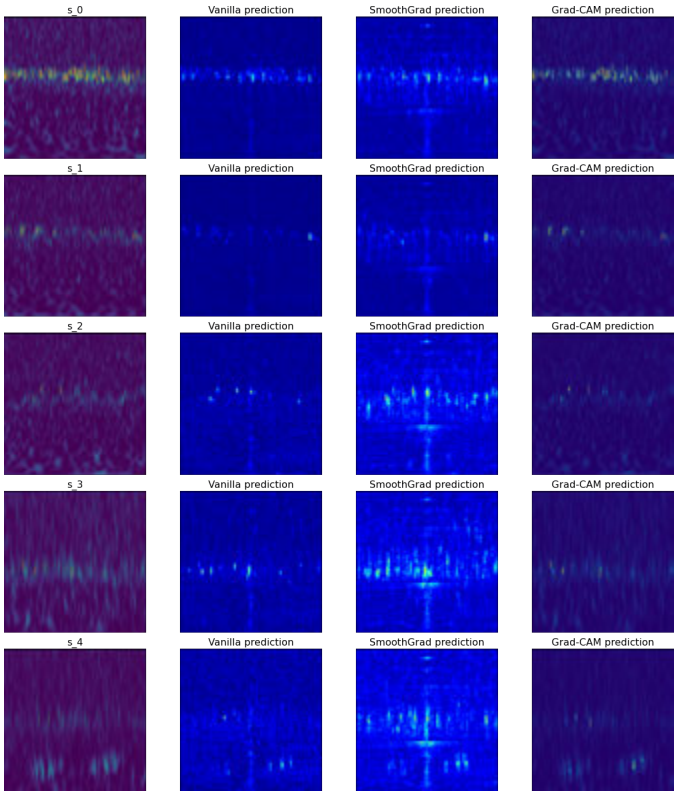

Fig. 3.8. Visualizations of correct predictions for class Scratchy and Model_0.

Model_1

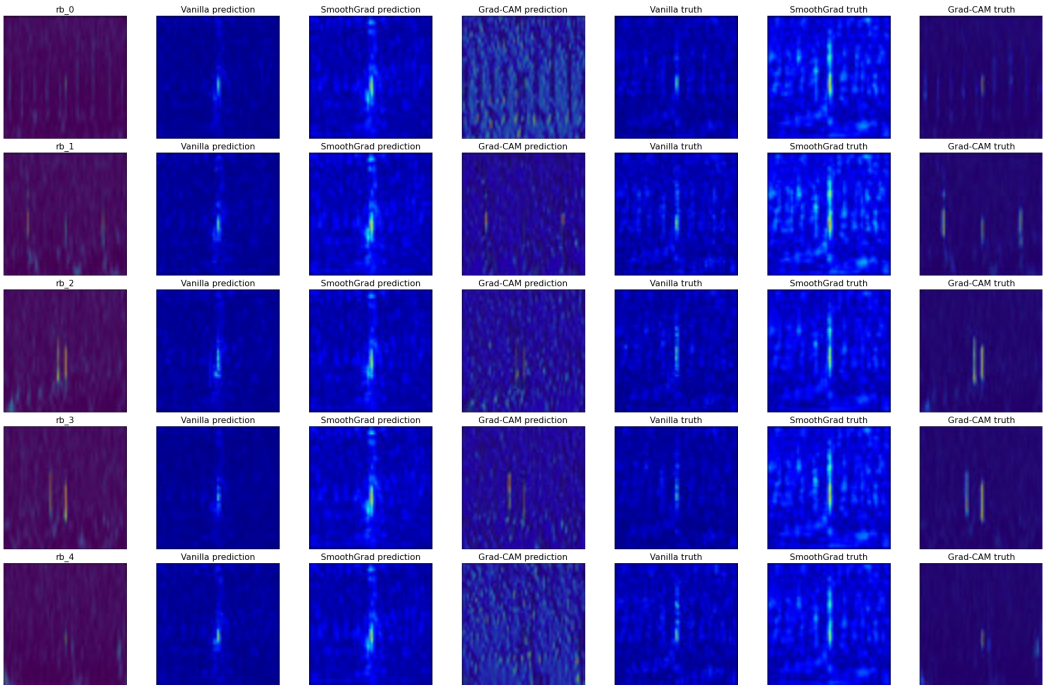

Fig. 3.9. Visualizations of incorrect predictions for class Repeating_Blips and Model_1. Predicted class is Blip.

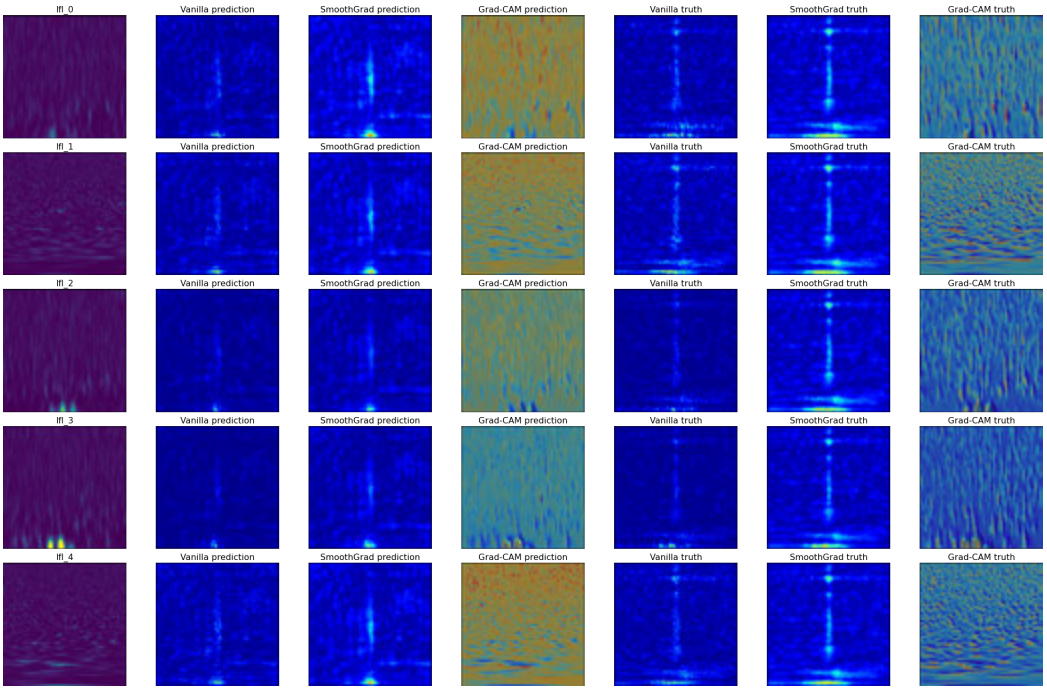

Fig. 3.10. Visualizations of incorrect predictions for class Low_Frequency_Lines and Model_1. Predicted class is Low_Frequency_Burst.

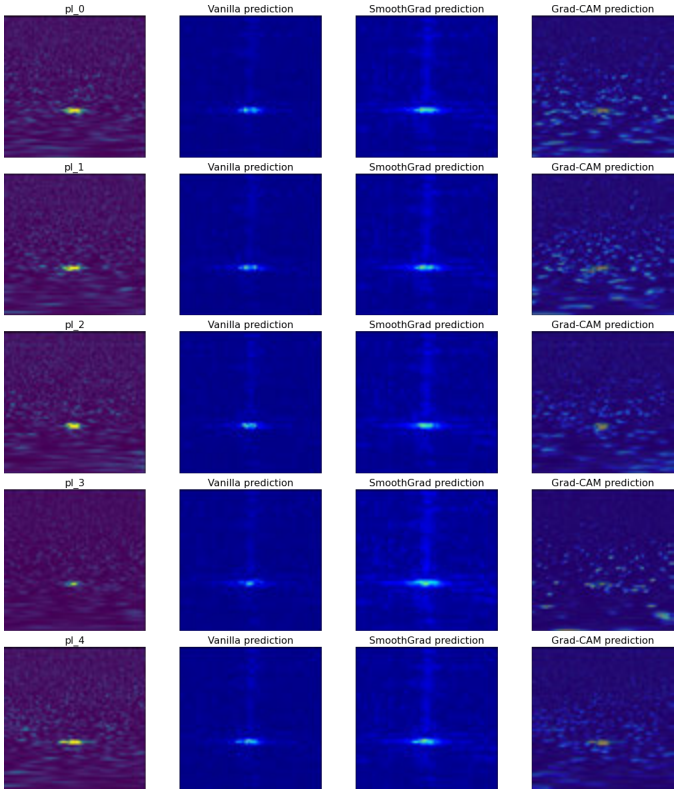

Fig. 3.11. Visualizations of correct predictions for class Power_Line and Model_1.

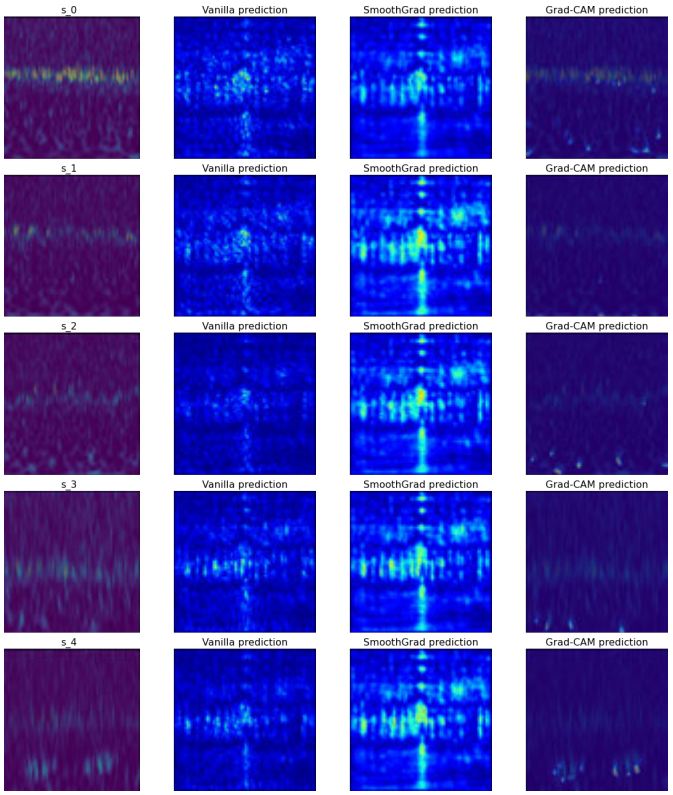

Fig. 3.12. Visualizations of correct predictions for class Scratchy and Model_1.

Model_2

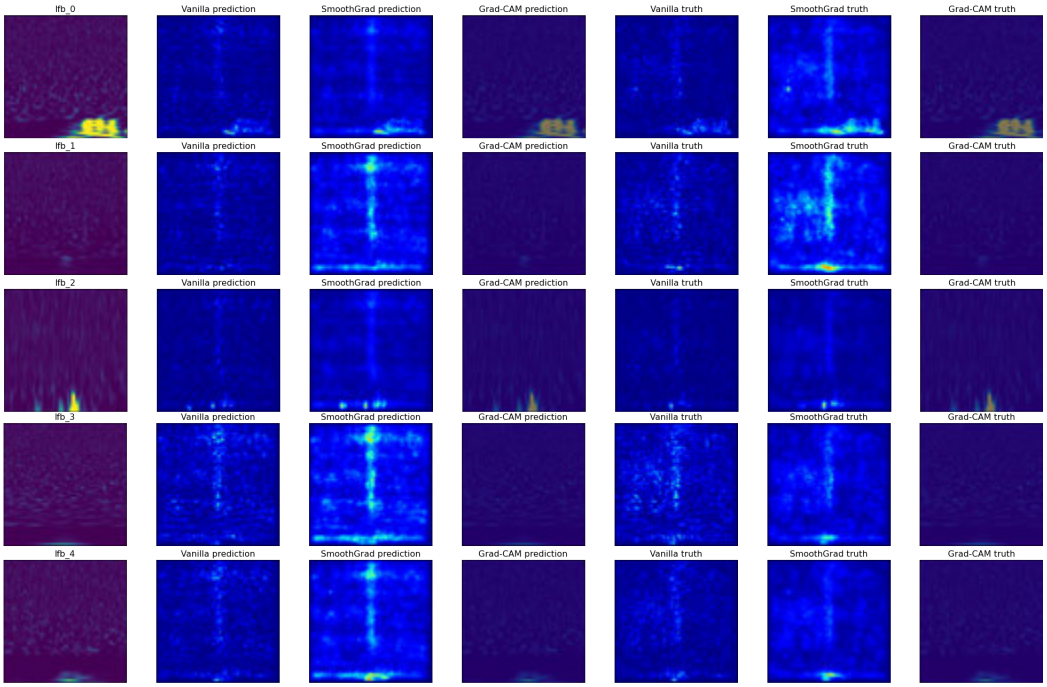

Fig. 3.13. Visualizations of incorrect predictions for class Low_Frequency_Burst and Model_2. Predicted class is Low_Frequency_Lines.

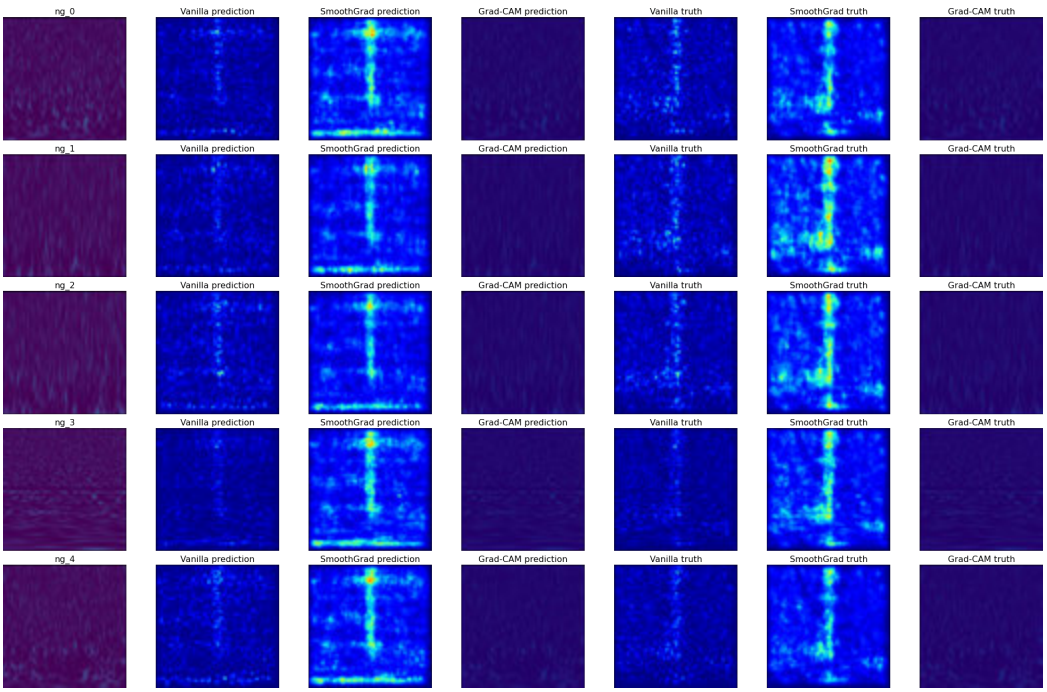

Fig. 3.14. Visualizations of incorrect predictions for class No_Glitch and Model_2. Predicted class is Low_Frequency_Lines.

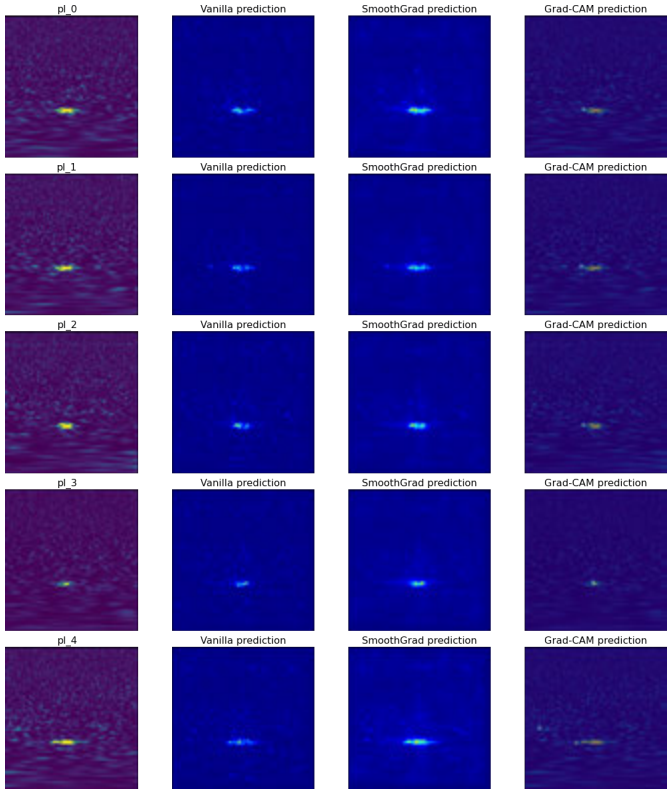

Fig. 3.15. Visualizations of correct predictions for class Power_Line and Model_2.

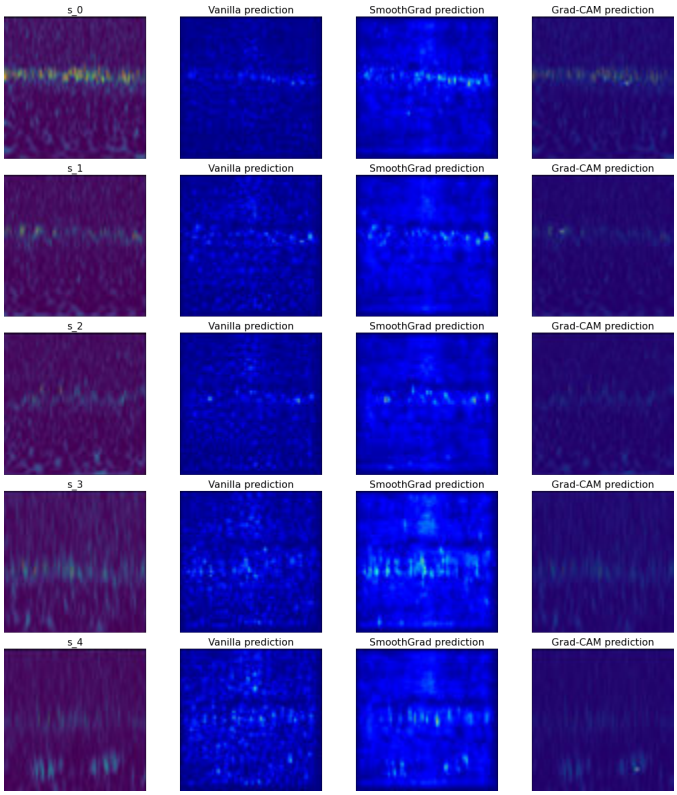

Fig. 3.16. Visualizations of correct predictions for class Scratchy and Model_2.

3.2.2. Analysis

As mentioned before, this section is going to analyze three convolutional neural networks with a similar structure but different depth. The reason behind this decision, is deeper structures should be able to recognize patterns of higher complexity and might produce different visualizations.

For each of the models, some of the top most confused classes will be analyzed using pixel attribution methods. Table 3.1 shows a summary of the target classes for each of the models. For each of the models and confused classes, five images are selected where the model made the same mistake. The analysis will start with the incorrect cases for each model and afterwards continue with the correct cases for all models.

TABLE 3.1. CLASS PAIRS WITH HIGHEST NUMBER OF ERRORS PER MODEL.

Model Name	Truth	Prediction	Frequency
Model_0	Repeating_Blips	Blip	35
Model_0	Low_Frequency_Burst	Low_Frequency_Lines	18
Model_1	Repeating_Blips	Blip	31
Model_1	Low_Frequency_Lines	Low_Frequency_Burst	23
Model_2	No_Glitch	Low_Frequency_Lines	47
Model_2	Low_Frequency_Burst	Low_Frequency_Lines	29

For the best performing models, the errors seem to be consistent between the classes Repeating_Blips and Blips and Low_Frequency_Lines and Low_Frequency_Burst. The latter seems to be a bidirectional relation, with the model confusing the classes interchangeably.

For the opposite scenario, the best performing classes, the selection has been the same for the three models for consistency and to appreciate differences regarding network depth and pattern recognition complexity. As shown in Table 3.2, the accuracy for the selected classes remains high across models, which makes them proper candidates for analysis.

TABLE 3.2. CLASS NAMES AND ACCURACIES OF TOP PERFORMING CLASSES SELECTED FOR ANALYSIS PER MODEL.

Model Name	Class Name	Accuracy
Model_0	Power_Line	1
Model_0	Scratchy	1
Model_1	Power_Line	1
Model_1	Scratchy	0.98
Model_2	Power_Line	1
Model_2	Scratchy	0.93

Notice how the classes with the highest amount of errors are very similar between Model_0 and Model_1. This similarity should be useful during analysis when making comparisons.

Figures 3.4 to 3.16 contain five rows and four columns or five rows and seven columns depending on the type of image. Each row contains visualizations for a single image. For the incorrectly predicted images, visualizations are calculated for both the correct and incorrectly predicted classes. Contrarily, for the correctly predicted images the visualizations are calculated for only one class. The prediction errors are the ones shown in Table 3.2.

Figure 3.4 shows visualizations calculated for Repeating_Blips class images incorrectly predicted as Blip. Without looking at the visualizations, it is easy to suspect which could be the reason behind the errors, since the class in question is a repetition of the Blip class, which is the one being incorrectly predicted. The model is having a hard time to differentiate between an image with a single pattern and an image with repeated patterns. Some images contain patters more easily recognizable than others. An example would be image rb_4 and rb_2. The former image's signal is much weaker than the latter. Clearly, the strength of the signal is a limiting factor to be taken into account. The weaker the signal is, the higher the probability of the model getting confused. Especially, considering how a lack of signal would belong to the class No_Glitch. Therefore, for 21 out of the 22 classes, the model is exposed to an additional risk, the confusion between weak signals and instances of the class No_Glitch.

Note how the Grad-CAM visualizations differ greatly between the predicted and true classes. In the the true class, the gradients of background pixels are higher. The true class isn't the one selected by the model, therefore it has a lower confidence level than the predicted class. Additionally, this case is stronger in the images rb_0, rb_1 and rb_4, and not in rb_2 and rb_3. One aspect these images have in common, is the signal strength. Images rb_0, rb_1 and rb_4 have a weaker signal than rb_2 and rb_3. Both observations could be indicating the model needs a stronger signal strength to identify multiple blips, than single blips. Which is problematic, knowing a single Blip is contained in a multiple Blip instance. Also, image rb_2 is interesting. Although it contains two blips of similar dimensions, only one of them is reflected in the gradients.

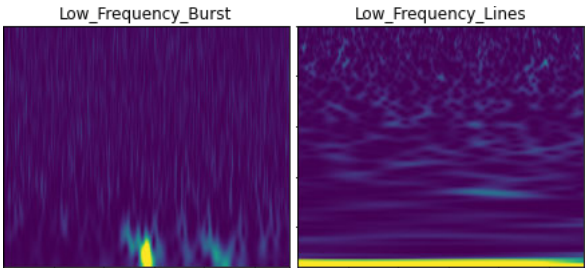

Fig. 3.17. Comparison between instances of Low_Frequency_Burst and Low_Frequency_Lines.

For comparison, Figure 3.9 shows visualizations for the same images but for Model_1 instead of Model_0. Model_1 also had this class pair as the most confusing. The difference in errors for this pair of classes between models is low, and the overall accuracy for this class in Model_1 is only 1% higher than in Model_0. Comparing Figures 3.9 and 3.4, the main difference is in SmoothGrad visualizations. In general, background pixels have higher gradients. Focusing on the background pixels doesn't seem to have negative effects for the Repeating_Blips class predictions, but rather the opposite, since both the class pair and the overall class accuracy increased.

Figure 3.5 contains visualizations for instances of the class Low_Frequency_Burst, which have been incorrectly classified as the class Low_Frequency_Lines by Model_0. Figure 3.17 shows a comparison between two instances of both classes. Both classes have a similar structure, and it is expected to find this pair between the ones the model finds more confusing.

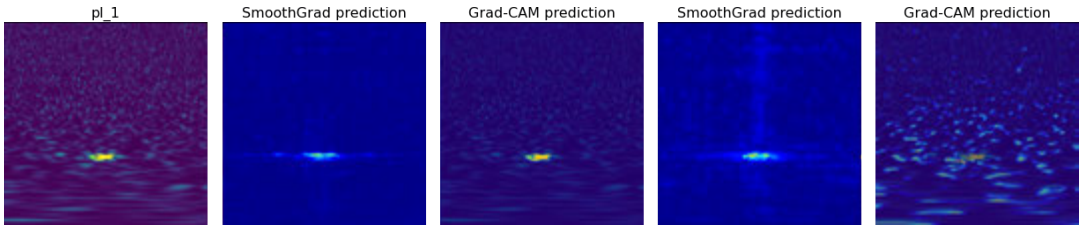

Fig. 3.18. Example of model focusing on background information. SmoothGrad and Grad-CAM visualizations for pl_1 in Model_0 and Model_1.

Figure 3.5 Grad-CAM visualizations show a similar behavior to Figure 3.4. The instances with a weak signal seem to set a higher focus on background information. Figure 3.18 shows an example. The model could be deriving the prediction information from the background of the image. It is probable this behavior is characteristic of images containing weak signals and not of specific classes. Also, the model isn't behaving as expected, because it is classifying images with weak signals/low information as Low_Frequency_Lines, when it should classify them as No_Glitch, the specific class for images without signals. The cause of this problem might be in the data. As an example, in Figure 3.5, lfb_1 should belong to the No_Glitch class instead of the Low_Frequency_Burst class. Since Gravity Spy is a citizen science project, images could be miss classified by collaborators and the model might be learning to accept No_Glitch instances as part of other classes. Cleaning the data before training the model would be a way to see if this is indeed a problem.

All images, but specially images containing weak signals like lfb_1, for the Vanilla and SmoothGrad visualizations have higher gradients at the middle of the image across the vertical axis. This could be a result of the imbalance of class images in the dataset. When an image doesn't contain obvious information, high gradients are measured from pixels which usually would contain signals in classes like Blip and Koi_Fish. As mentioned before, these classes contain more than 30% of the images in the dataset. There-

fore the model might be biased towards those classes. Anyhow, in case of existing a bias towards these classes, it doesn't seem to be a major drive in the predictions of Low_Frequency_Burst instances. It is also curious how these possible "biases" are not visible in the maps calculated using Grad-CAM. Since this technique is based not only on gradients but also on class activations, the results "look" at the model from a different perspective. From these two images, Grad-CAM seems to offer a better insight on why the model might be making an error. Both Vanilla and SmoothGrad maps are very similar with the gradients being slightly higher in the predicted class map. But Grad-CAM shows a clearer difference. The activations for the true class are on the edges of the signal, while for the predicted class cover the signal completely.

Figure 3.10 contains visualizations for the second most confusing class pair in Model_1, Low_Frequency_Lines. The class is confused with the class Low_Frequency_Burst. Previously, for Model_0 the same class pair but with the opposite relation was analyzed. Although it initially wasn't between Model_0's class pairs for analysis, in order to make a comparison, Figure 3.6 contains Model_0's visualizations for instances of the Low_Frequency_Lines class. Three characteristics, previously mentioned in the Low_Frequency_Burst analysis are also common for this class pair. First, gradient-based maps display high values in the vertical central region of the image. Second, for Model_0, Grad-CAM visualizations show high activation values for background information, when no strong signals are present. Third, for Model_1, both Vanilla and SmoothGrad visualizations show an increase in values for background information. Adding an additional layer could be leading the model to focus more on the background information, independently of the signal strength.

Figure 3.15 shows visualizations for Low_Frequency_Burst instances classified as Low_Frequency_Lines by Model_2. Compared to both visualizations of the previous models, the only difference detected is the lack of activation in Grad-CAM visualizations, as if the model didn't detect any patterns.

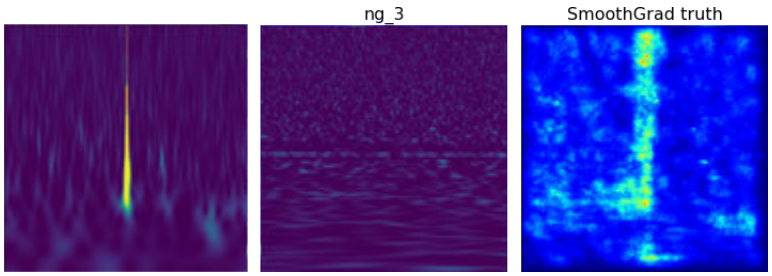

Fig. 3.19. Example of Smoothgrad visualization displaying high gradients in regions similar to Blip class for No_Glitch instance. Model_2. Blip instance (left), No_Glitch instance (middle), SmoothGrad visualization (right).

Figure 3.14 and 3.19 shows visualizations for the class No_Glitch in Model_2. This class is used to represent observations without a signal. The model confuses this class

with the class Low_Frequency_Lines, which visually is similar. Gradient-based saliency maps have the shape of the most predominant class in the dataset, Blip. This might be an indication of how the large number of images of the class Blip imbalance the dataset and consequently the model's predictions. Although no signals are present in the instance, Vanilla and Smoothgrad maps show high gradient values for both the predicted and true classes in the center of images.

Lastly, the analysis of the correctly predicted classes Scratchy and Power_Lines for all the models. The results of visualizations are displayed in Figures 3.7 and 3.8 for Model_0, 3.11 and 3.12 for Model_1 and 3.15 and 3.16 for Model_2. First of all, an important note is how both classes are unique in their signal shapes. One one hand, typical Power_Line instance signals have an elliptical shape almost at the center of the image. It is a single occurrence and has a fixed location. On the other hand, Scratchy signals usually have the same length as the image, are located in similar frequency ranges, and are made of numerous small vertical lines. Both these classes are very characteristic and no other classes are similar. An opposite example are Low_Frequency_Lines and Low_Frequency_Bursts, which are very similar and the model doesn't differentiate them well.

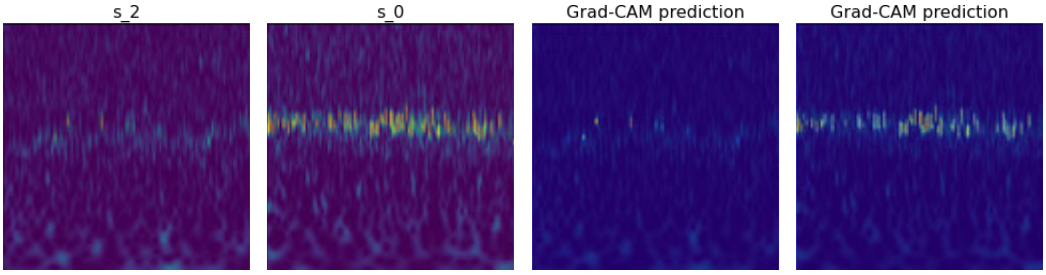

Fig. 3.20. Scratchy weak vs strong signal and Grad-CAM visualizations for Model_0.

For Model_0, in Figures 3.7 and 3.8, it is visible how all the maps reflect portions of the Scratchy patterns. Usually with better results for instances with strong signals, like s_0. In Power_Line instances, visualizations show how the model is doing a good job at detecting the signal. The maps usually don't reflect only portions of the signal, but the complete signal. For Model_1 and Model_2 in Figures 3.11, 3.12, 3.15 and 3.16 visualization maps behave differently. Similarly to incorrectly predicted examples, background information in the image seems to gain relevance. Also from Model_1 to Model_2 gradients increase while activations from Grad-CAM are smaller. Image s_0 is a good example, it is possible to see how Vanilla and SmoothGrad gradients increase, specially from Model_0 to Model_1 and Grad_CAM activations decrease, specially in from Model_1 to Model_2.

Overall, Grad_CAM seems to align better with models accuracies than Vanilla and SmoothGrad. For example for Scratchy instances, the higher number of convolutional layers, the worse is the test class accuracy and less activated pixels are in Grad-CAM visualizations. The visualizations match the performance of the model. With Vanilla and

SmoothGrad, it is harder to make an evaluation. This could be due to how Grad-CAM, since it uses a ReLU function, it only displays positive contribution to classes, while Vanilla and Smoothgrad, display more than positive contributions. Nevertheless, Vanilla and SmoothGrad are still useful to "look" at the model from a different perspective.

4. CONCLUSIONS AND FUTURE WORK

4.1. Conclusions

All project objectives have been met. An image recognition model has been developed using convolutional neural networks, the most accurate model has been analyzed using gradient-based saliency maps and class activation maps varying the number of convolutional and pooling layers, and possible paths of improvement have been defined for following iterations. The main takeaways of the project are summarized below.

Firstly, during the development of the model, parameters such as the size of convolutional kernels, and the number of convolutional and pooling layers have proven to be crucial for achieving the best performance. The use of odd sized filters allows to encode information regarding a pixel symmetrically during convolution operations. Reducing the number of convolutional layers helps detect simple patterns, which seem to be more efficient for the model to differentiate between classes. Also shallow networks require a higher computational cost when training the model, since the number of trainable parameters is larger.

Secondly, in the explainability analysis, multiple probable reasons have been found that could be affecting the model's performance. First, there seems to be a bias in the data since the number of images per class has a high variance. The classes with most images, in this case Blip, is reflected in Vanilla and SmoothGrad visualizations for images containing weak signals or not signal at all such as in the No_Glitch class. This predisposition of the model for predominant classes could be limiting the model's maximum achievable accuracy. Second, the number of layers in the model seem to increase the focus of the model in background information. A pattern observable in all the post-hoc local methods used. This observation could be related to why the model is more accurate as a shallow network. Third, Grad-CAM was the technique most aligned with model class accuracies while Vanilla and SmoothGrad were harder to interpret. Fourth, SmoothGrad consistently produced more reliable results than Vanilla, but also had a higher computational cost, since it requires multiple backpropagation passes. Fifth, the use of a single interpretation technique might not be sufficient to analyze model results. Using more than one technique of varying types is better than a single technique.

Lastly, it is important to note the interpretation methods used in this project are not sufficient to confirm the logic followed by models in prediction tasks. Visualizations are open to interpretation and help define tentative paths of improvement, but don't guarantee success.

4.2. Future work

Both the model and explainability analysis are improvable. The use of different explainability techniques could be used to design a better analysis of the model. In this project the focus has been set solely on post-hoc local methods, but besides existing multiple additional gradient-based saliency map and class activation map methods, other methods such as feature attribution methods could also be used to analyze an image recognition model such as SHAP, LIME and DeepLIFT. Also, the techniques used are mainly local interpretations, explaining single case scenarios. It would be beneficial to include techniques capable of global interpretations to understand what global relationships has the model learned.

Repeating the analysis with a cleaned database would also be interesting. Balancing the number of images per class could help confirm if classes with a larger presence in the data like Koi_Fish and Blip are negatively influencing the classification of other class instances. Also waiting for the Gravity Spy project to publish a larger dataset, so the number of instances in classes like Paired_Doves and Wandering_Line is larger and the model has the necessary information to classify them correctly.

It would also be interesting to perform the analysis including the confidence of the model for the top x classes in predictions. In this project it would had been used multiple times, to have a better picture of which classes does the model consider as best candidates for an specific image.

The authors of Grad-CAM at the time of publication also proposed another method called Guided Grad-CAM, which combines the use of Guided Backpropagation and Grad-CAM. In their publication they show how this method achieves better results than Grad-CAM. A future task would be to use this method and compare its results to the results obtained with Grad-CAM in this project.

Other publications, like Shrikumar et. al [32] have shown how the gradient-based saliency maps used in this project are limited by saturation and discontinuity problems. Other techniques as the one proposed in [32] claim to overcome this limitations. Therefore another future work task would be to include methods such as DeepLIFT to analyze how avoiding these limitations affects the analysis.

Lastly, varying more the architecture of models. In this analysis, the only variable parameter was the number of convolutional and pooling layers. Applying the analysis to a larger variation of models, by modifying other parameters like activation functions, kernel sizes and strides, pooling functions, higher number of layers, etc could help understand which interpretation methods are more adequate in different scenarios.

5. BIBLIOGRAPHY

[1] B. P. Abbott *et al.*, "Observation of gravitational waves from a binary black hole merger," *Physical Review Letters*, vol. 116, no. 6, Feb. 2016. DOI: 10.1103/physrevlett.116.061102. [Online]. Available: http://dx.doi.org/10.1103/PhysRevLett.116.061102.

[2] L. I. G.-W. Observatory. (). "What are gravitational waves?" [Online]. Available: https://www.ligo.caltech.edu/page/what-are-gw.

[3] U. G. S. Administration. (). "Gravity spy," [Online]. Available: https://www.citizenscience.gov/catalog/368/#.

[4] M. Zevin *et al.*, "Gravity spy: Integrating advanced ligo detector characterization, machine learning, and citizen science," *Classical and Quantum Gravity*, vol. 34, no. 6, p. 064 003, Feb. 2017. DOI: 10.1088/1361-6382/aa5cea. [Online]. Available: http://dx.doi.org/10.1088/1361-6382/aa5cea.

[5] S. Coughlin, *Updated gravity spy data set*, version v1.1.0, Zenodo, Nov. 2018. DOI: 10.5281/zenodo.1476551. [Online]. Available: https://doi.org/10.5281/zenodo.1476551.

[6] R. A. Smith. (). "Opening the lid on criminal sentencing software," [Online]. Available: https://today.duke.edu/2017/07/opening-lid-criminal-sentencing-software.

[7] (). "Gdpr article 15," [Online]. Available: https://gdpr-info.eu/art-15-gdpr/.

[8] S. Mukherjee, R. Obaid, and B. Matkarimov, "Classification of glitch waveforms in gravitational wave detector characterization," English, *Journal of Physics: Conference Series*, vol. 243, 2010, Copyright: Copyright 2018 Elsevier B.V., All rights reserved. DOI: 10.1088/1742-6596/243/1/012006.

[9] S. RAMPONE, V. PIERRO, L. TROIANO, and I. M. PINTO, "Neural network aided glitch-burst discrimination and glitch classification," *International Journal of Modern Physics C*, vol. 24, no. 11, p. 1 350 084, Oct. 2013. DOI: 10.1142/s0129183113500848. [Online]. Available: http://dx.doi.org/10.1142/S0129183113500848.

[10] J. Powell, D. Trifirò, E. Cuoco, I. S. Heng, and M. Cavaglià, "Classification methods for noise transients in advanced gravitational-wave detectors," *Classical and Quantum Gravity*, vol. 32, no. 21, p. 215 012, Oct. 2015. DOI: 10.1088/0264-9381/32/21/215012. [Online]. Available: http://dx.doi.org/10.1088/0264-9381/32/21/215012.

[11] J. Powell *et al.*, "Classification methods for noise transients in advanced gravitational-wave detectors ii: Performance tests on advanced ligo data," *Classical and Quantum Gravity*, vol. 34, no. 3, p. 034 002, Jan. 2017. DOI: 10.1088/1361-6382/34/3/034002. [Online]. Available: http://dx.doi.org/10.1088/1361-6382/34/3/034002.

[12] N. Mukund, S. Abraham, S. Kandhasamy, S. Mitra, and N. S. Philip, "Transient classification in ligo data using difference boosting neural network," *Physical Review D*, vol. 95, no. 10, May 2017. DOI: 10.1103/physrevd.95.104059. [Online]. Available: http://dx.doi.org/10.1103/PhysRevD.95.104059.

[13] M. Razzano and E. Cuoco, "Image-based deep learning for classification of noise transients in gravitational wave detectors," *Classical and Quantum Gravity*, vol. 35, no. 9, p. 095 016, Apr. 2018. DOI: 10.1088/1361-6382/aab793. [Online]. Available: http://dx.doi.org/10.1088/1361-6382/aab793.

[14] D. H. Hubel and T. N. Wiesel, "Receptive fields of single neurones in the cat's striate cortex," *The Journal of physiology*, Oct. 1959. DOI: 10.1113/jphysiol.1959.sp006308.

[15] D. Macêdo, "Enhancing deep learning performance using displaced rectifier linear unit," Ph.D. dissertation, Jul. 2017. DOI: 10.13140/RG.2.2.23893.88807.

[16] F. K., "Neocognitron: A self organizing neural network model for a mechanism of pattern recognition unaffected by shift in position.," *Biological cybernetics*, Jan. 1980. DOI: 10.1007/BF00344251.

[17] Y. "LeCun *et al.*, ""handwritten digit recognition: Applications of neural net chips and automatic learning"," *"IEEE Communication"*, "41–46", Nov. 1989.

[18] (). "Lenet-5, convolutional neural networks.," [Online]. Available: http://yann.lecun.com/exdb/lenet/.

[19] O. Russakovsky *et al.*, *Imagenet large scale visual recognition challenge*, 2015. arXiv: 1409.0575 [cs.CV].

[20] (). "Imagenet site," [Online]. Available: https://image-net.org/.

[21] L. S. V. R. Challenge. (). "Large scale visual recognition challenge 2012 (ilsvrc2012)," [Online]. Available: https://www.image-net.org/challenges/LSVRC/2012/results.html.

[22] W. J. Murdoch, C. Singh, K. Kumbier, R. Abbasi-Asl, and B. Yu, "Definitions, methods, and applications in interpretable machine learning.," *Proceedings of the National Academy of Sciences of the United States of America*, 2019. DOI: 10.1073/pnas.1900654116.

[23] K. Simonyan, A. Vedaldi, and A. Zisserman, *Deep inside convolutional networks: Visualising image classification models and saliency maps*, 2014. arXiv: 1312.6034 [cs.CV].

[24] M. D. Zeiler and R. Fergus, *Visualizing and understanding convolutional networks*, 2013. arXiv: 1311.2901 [cs.CV].

[25] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, *Striving for simplicity: The all convolutional net*, 2015. arXiv: 1412.6806 [cs.LG].

[26] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-cam: Visual explanations from deep networks via gradient-based localization," *International Journal of Computer Vision*, vol. 128, no. 2, pp. 336–359, Oct. 2019. DOI: 10.1007/s11263-019-01228-7. [Online]. Available: http://dx.doi.org/10.1007/s11263-019-01228-7.

[27] D. Smilkov, N. Thorat, B. Kim, F. Viégas, and M. Wattenberg, *Smoothgrad: Removing noise by adding noise*, 2017. arXiv: 1706.03825 [cs.LG].

[28] (). "Introduction to neural networks.," [Online]. Available: https://edgeaiguru.com/Introduction-to-Neural-Networks.

[29] (). "Convolutional neural networks (cnns) in tensorflow.," [Online]. Available: https://nasirml.wordpress.com/2019/01/08/convnet-in-tensorflow.

[30] (). "Convolutional neural network demystified for a comprehensive learning with industrial application, dynamic data assimilation - beating the uncertainties.," [Online]. Available: https://www.intechopen.com/chapters/72480.

[31] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, *Learning deep features for discriminative localization*, 2015. arXiv: 1512.04150 [cs.CV].

[32] A. Shrikumar, P. Greenside, and A. Kundaje, *Learning important features through propagating activation differences*, 2019. arXiv: 1704.02685 [cs.CV].

APPENDIX I: SUPPORT FIGURES AND TABLES

TABLE 5.1. DATASET IMAGES PER CLASS PER SET.

Description	**Train**	**Test**	**Validation**	**Total**
1080Lines	50	229	49	**328**
1400Ripples	9	59	13	**81**
Air_Compressor	9	41	8	**58**
Blip	272	1274	275	**1821**
Chirp	10	41	9	**60**
Extremely_Loud	64	316	67	**447**
Helix	42	195	42	**279**
Koi_Fish	102	498	106	**706**
Light_Modulation	78	361	73	**512**
Low_Frequency_Burst	90	437	94	**621**
Low_Frequency_Lines	66	315	66	**447**
No_Glitch	21	107	22	**150**
None_of_the_Above	11	57	13	**81**
Paired_Doves	4	19	4	**27**
Power_Line	68	314	67	**449**
Repeating_Blips	37	185	41	**263**
Scattered_Light	67	308	68	**443**
Scratchy	50	237	50	**337**
Tomte	13	73	17	**103**
Violin_Mode	64	284	64	**412**
Wandering_Line	7	29	6	**42**
Whistle	45	208	46	**299**
Total	**5587**	**1179**	**1200**	**7966**

TABLE 5.2. DATASET TEST, TRAIN AND VALIDATION PROPORTIONS PER SET (%).

Description	**Train**	**Test**	**Validation**	**Total**
1080Lines	4.24	4.1	4.08	**4.12**
1400Ripples	0.76	1.06	1.08	**1.02**
Air_Compressor	0.76	0.73	0.67	**0.73**
Blip	23.07	22.8	22.92	**22.86**
Chirp	0.85	0.73	0.75	**0.75**
Extremely_Loud	5.43	5.66	5.58	**5.61**
Helix	3.56	3.49	3.5	**3.5**
Koi_Fish	8.65	8.91	8.83	**8.86**
Light_Modulation	6.62	6.46	6.08	**6.43**
Low_Frequency_Burst	7.63	7.82	7.83	**7.8**
Low_Frequency_Lines	5.6	5.64	5.5	**5.61**
No_Glitch	1.78	1.92	1.83	**1.88**
None_of_the_Above	0.93	1.02	1.08	**1.02**
Paired_Doves	0.34	0.34	0.33	**0.34**
Power_Line	5.77	5.62	5.58	**5.64**
Repeating_Blips	3.14	3.31	3.42	**3.3**
Scattered_Light	5.68	5.51	5.67	**5.56**
Scratchy	4.24	4.24	4.17	**4.23**
Tomte	1.1	1.31	1.42	**1.29**
Violin_Mode	5.43	5.08	5.33	**5.17**
Wandering_Line	0.59	0.52	0.5	**0.53**
Whistle	3.82	3.72	3.83	**3.75**
Total	**100**	**100**	**100**	**100**

TABLE 5.3. DATASET TEST, TRAIN AND VALIDATION PROPORTIONS PER CLASS (%).

Description	**Train**	**Test**	**Validation**	**Total**
1080Lines	15.24	69.82	14.94	**100**
1400Ripples	11.11	72.84	16.05	**100**
Air_Compressor	15.52	70.69	13.79	**100**
Blip	14.94	69.96	15.1	**100**
Chirp	16.67	68.33	15	**100**
Extremely_Loud	14.32	70.69	14.99	**100**
Helix	15.05	69.89	15.05	**100**
Koi_Fish	14.45	70.54	15.01	**100**
Light_Modulation	15.23	70.51	14.26	**100**
Low_Frequency_Burst	14.49	70.37	15.14	**100**
Low_Frequency_Lines	14.77	70.47	14.77	**100**
No_Glitch	14	71.33	14.67	**100**
None_of_the_Above	13.58	70.37	16.05	**100**
Paired_Doves	14.81	70.37	14.81	**100**
Power_Line	15.14	69.93	14.92	**100**
Repeating_Blips	14.07	70.34	15.59	**100**
Scattered_Light	15.12	69.53	15.35	**100**
Scratchy	14.84	70.33	14.84	**100**
Tomte	12.62	70.87	16.5	**100**
Violin_Mode	15.53	68.93	15.53	**100**
Wandering_Line	16.67	69.05	14.29	**100**
Whistle	15.05	69.57	15.38	**100**
Total	**14.8**	**70.14**	**15.06**	**100**

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 248, 248, 5)       140
_________________________________________________________________
activation (Activation)      (None, 248, 248, 5)       0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 124, 124, 5)       0
_________________________________________________________________
flatten (Flatten)            (None, 76880)             0
_________________________________________________________________
dense (Dense)                (None, 250)               19220250
_________________________________________________________________
dense_1 (Dense)              (None, 22)                5522
=================================================================
Total params: 19,225,912
Trainable params: 19,225,912
Non-trainable params: 0
_________________________________________________________________
```

Fig. 5.1. Model_0 architecture.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 248, 248, 5)       140
_________________________________________________________________
activation (Activation)      (None, 248, 248, 5)       0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 124, 124, 5)       0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 122, 122, 5)       230
_________________________________________________________________
activation_1 (Activation)    (None, 122, 122, 5)       0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 61, 61, 5)         0
_________________________________________________________________
flatten (Flatten)            (None, 18605)             0
_________________________________________________________________
dense (Dense)                (None, 250)               4651500
_________________________________________________________________
dense_1 (Dense)              (None, 22)                5522
=================================================================
Total params: 4,657,392
Trainable params: 4,657,392
Non-trainable params: 0
_________________________________________________________________
```

Fig. 5.2. Model_1 architecture.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 248, 248, 5)       140
_________________________________________________________________
activation (Activation)      (None, 248, 248, 5)       0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 124, 124, 5)       0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 122, 122, 5)       230
_________________________________________________________________
activation_1 (Activation)    (None, 122, 122, 5)       0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 61, 61, 5)         0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 59, 59, 5)         230
_________________________________________________________________
activation_2 (Activation)    (None, 59, 59, 5)         0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 29, 29, 5)         0
_________________________________________________________________
flatten (Flatten)            (None, 4205)              0
_________________________________________________________________
dense (Dense)                (None, 250)               1051500
_________________________________________________________________
dense_1 (Dense)              (None, 22)                5522
=================================================================
Total params: 1,057,622
Trainable params: 1,057,622
Non-trainable params: 0
_________________________________________________________________
```

Fig. 5.3. Model_2 architecture.

TABLE 5.4. MODEL_0 ACCURACIES PER CLASS.

Class Name	Incorrect	Correct	Total	Accuracy
1080Lines	13	187	200	**0.94**
1400Ripples	2	34	36	**0.94**
Air_Compressor	5	31	36	**0.86**
Blip	44	1048	1092	**0.96**
Chirp	2	38	40	**0.95**
Extremely_Loud	3	253	256	**0.99**
Helix	2	166	168	**0.99**
Koi_Fish	14	394	408	**0.97**
Light_Modulation	32	280	312	**0.9**
Low_Frequency_Burst	45	315	360	**0.88**
Low_Frequency_Lines	25	239	264	**0.91**
No_Glitch	10	74	84	**0.88**
None_of_the_Above	22	22	44	**0.5**
Paired_Doves	14	2	16	**0.13**
Power_Line	1	271	272	**1**
Repeating_Blips	51	97	148	**0.66**
Scattered_Light	19	249	268	**0.93**
Scratchy	1	199	200	**1**
Tomte	5	47	52	**0.9**
Violin_Mode	13	243	256	**0.95**
Wandering_Line	17	11	28	**0.39**
Whistle	18	162	180	**0.9**
Total	**358**	**4362**	**4720**	**0.9242**

TABLE 5.5. MODEL_1 ACCURACIES PER CLASS.

Class Name	Incorrect	Correct	Total	Accuracy
1080Lines	13	187	200	**0.94**
1400Ripples	1	35	36	**0.97**
Air_Compressor	4	32	36	**0.89**
Blip	29	1063	1092	**0.97**
Chirp	1	39	40	**0.98**
Extremely_Loud	6	250	256	**0.98**
Helix	1	167	168	**0.99**
Koi_Fish	23	385	408	**0.94**
Light_Modulation	41	271	312	**0.87**
Low_Frequency_Burst	38	322	360	**0.89**
Low_Frequency_Lines	27	237	264	**0.9**
No_Glitch	28	56	84	**0.67**
None_of_the_Above	16	28	44	**0.64**
Paired_Doves	10	6	16	**0.38**
Power_Line	1	271	272	**1**
Repeating_Blips	48	100	148	**0.68**
Scattered_Light	22	246	268	**0.92**
Scratchy	5	195	200	**0.98**
Tomte	8	44	52	**0.85**
Violin_Mode	17	239	256	**0.93**
Wandering_Line	16	12	28	**0.43**
Whistle	22	158	180	**0.88**
Total	**377**	**4343**	**4720**	**0.9201**

TABLE 5.6. MODEL_2 ACCURACIES PER CLASS.

Class Name	Incorrect	Correct	Total	Accuracy
1080Lines	15	185	200	**0.93**
1400Ripples	4	32	36	**0.89**
Air_Compressor	4	32	36	**0.89**
Blip	66	1026	1092	**0.94**
Chirp	8	32	40	**0.8**
Extremely_Loud	50	206	256	**0.8**
Helix	7	161	168	**0.96**
Koi_Fish	11	397	408	**0.97**
Light_Modulation	18	294	312	**0.94**
Low_Frequency_Burst	63	297	360	**0.83**
Low_Frequency_Lines	17	247	264	**0.94**
No_Glitch	59	25	84	**0.3**
None_of_the_Above	24	20	44	**0.45**
Paired_Doves	16	0	16	**0**
Power_Line	1	271	272	**1**
Repeating_Blips	32	116	148	**0.78**
Scattered_Light	16	252	268	**0.94**
Scratchy	15	185	200	**0.93**
Tomte	6	46	52	**0.88**
Violin_Mode	14	242	256	**0.95**
Wandering_Line	12	16	28	**0.57**
Whistle	28	152	180	**0.84**
Total	**486**	**4234**	**4720**	**0.8970**

APPENDIX II: PROJECT BUDGET

This section consists of a detailed breakdown of the budget required to carry out the project. For this project, the costs can be subdivided into three categories: software, hardware and human resources. Note the final calculation is an estimation since concepts such as the time value of money, costs associated to aid offered by mentors and multiple exchange rates are not being taken into consideration.

To calculate the associated amortization costs, the formula displayed below is used. The table below displays a summary of all project costs.

$$AmortizationCost = PurchasePrice * \frac{UsagePeriod}{AmortizationPeriod} \tag{5.1}$$

TABLE 5.7. OVERALL COSTS

Category	Cost
Hardware	305.55€
Software	252.00€
Human Resources	5802.00€
Total	**6359.55€**

Hardware

The hardware costs have been mainly the use of a personal laptop. The table below displays the total hardware costs.

TABLE 5.8. HARDWARE COSTS

Description	Purchase Price	Usage Period (m)	Amortization Period (m)	Cost
Laptop	1099.99€	10	36	305.55€
Total				**305.55€**

Software

Since most of the software used is open-source, the software-related costs are the smallest.

TABLE 5.9. SOFTWARE COSTS

Description	**Purchase Price**	**Usage Period (m)**	**Amortization Period (m)**	**Cost**
Google Cloud	252€	10	10	252€
Kaggle Kernel	0€	10	10	0€
Open-source libraries	0€	10	10	0€
Linux OS	0€	10	36	0€
Total				**252.00€**

Human Resources

Human resources are the largest expenditure. The latest salary is used as a reference to calculate the hourly rate.

TABLE 5.10. HUMAN RESOURCES COSTS

Hours per week	**Number of Weeks**	**Hourly rate**	**Cost**
7.5	40	19.34€	5802.00€
Total			**5802.00€**