|  |  |  |
| --- | --- | --- |
|  | **Софийски университет „Св. Кл. Охридски”**  Факултет по математика и информатика |  |

**Курсов Проект**

на тема: „Explainable ML: Explainable Deep NN Models”

Студент: **Тервел Радославов Вълков** Ф.Н. **9MI3400058**

Курс: „1“, Учебна година: 2021/22

Преподаватели: **проф. Иван Койчев, Борис Величков**

=================================

Декларация за липса плагиатство:

* Плагиатство е да използваш, идеи, мнение или работа на друг, като претендираш, че са твои. Това е форма на преписване.
* Тази курсова работа е моя, като всички изречения, илюстрации и програми от други хора са изрично цитирани.
* Тази курсова работа или нейна версия не са представени в друг университет или друга учебна институция.
* Разбирам, че ако се установи плагиатство в работата ми ще получа оценка “Слаб”.

8.7.22 г. Подпис на студента:

**Съдържание**

[1 Увод 2](#_Toc536441759)

[2 Преглед на обяснимостта 2](#_Toc536441760)

[2.1 Обяснимост на различни типове модели 2](#_Toc536441763)

[2.2 Подходи 3](#_Toc536441764)

[3 Проектиране 6](#_Toc536441761)

[4 Реализация и резултати 6](#_Toc536441762)

[4.1 Използвани технологии, платформи и библиотеки 6](#_Toc536441763)

[4.2 Реализация 8](#_Toc536441764)

[5 Заключение 14](#_Toc536441765)

[6 Използвана литература 15](#_Toc536441766)

# Увод

С увеличаващата се популярност на алгоритмите за машинно самообучение, започват да възникват въпроси за надеждността и коректността им - за да се доверим на даден алгоритъм и да предприемаме действия според резултатите му, трябва да разбираме защо всъщност е стигнал до конкретно решение. Това е особено важно в области, при които ефектът от такива решения може да бъде доста сериозен, като например в медицината.[1]

Поради тази нужда е въведен терминът „*обяснимост*“ (или „*интерпретируемост*“) на машинното самообучение, който да обозначава доколко за различни модели считаме че техните резултати са разбираеми от хората. Няма общоприета формална дефиниция, но неформално може да смятаме че говорим за разбирането на степента, до която дадена причина (като промяна на атрибути, на параметри на модела, и др.) може да въздейства на предсказанията на модела. Или с други думи – степента, до която ние като разработчици на модела може да предскажем какво ще се случи при дадена промяна на входа или параметрите на алгоритъма.[2]

Освен за крайния потребител, обяснимостта може да бъде полезна и по време на самата разработка – при debug-ване на грешки и друго неочаквано поведение, възможността за вникване по-дълбоко в работата на модела може да ни изясни по-бързо защо той прави съответните грешки, и така ние по-лесно да открием и коригираме съответните проблеми. Обяснимостта може също така да ни помогне за идентифициране на проблеми при данните (като data skew или drift) и при верифициране на съответствието на модела с дадени закони и регулации.[4]

Целта на проекта е да бъде проведен експеримент, при който биват приложени избрани методи за обяснимост и демонстрирани и интерпретирани техните резултати.

# Преглед на обяснимостта

## Обяснимост на различни типове модели

За някои модели за машинно самообучение се счита, че те самите са директно интерпретируеми от хората, и съответно за тях е по-малко нужно да се прилагат специфични методи за обяснимост. Други модели обаче спадат към категорията на така наречените “*black-box*” („черна кутия“) модели, които са прекалено сложни за човешка интерпретация.

Няма прецизно-дефинирана граница между тези две категории, но обикновено модели като линейните, класификационните дървета и множествата от правила се считат за лесни за разбиране, докато дълбоките невронни мрежи са класически пример за *black-box* модели[3]. Тяхната сложност идва от структурата им – като дълбоки мрежи, те имат множество слоеве, като всеки от тези слоеве има множество неврони, всеки със свои връзки и техни тегла. Това може да доведе до огромен брой параметри – например една от известните модерни архитектури за конволюционни мрежи, *VGG16*, има над 130 милиона[5]. При тази сложна структура, дори и да е ясно как работи един отделен неврон, става много по-трудно да се изясни как всичките неврони работят съвместно и си взаимодействат за да стигнат до крайното решение.

Ако всички модели бяха еднакво мощни, то очевидно бихме предпочели да ползваме тези които вече са обясними сами по себе си, но за съжаление това не е така – дълбоките невронни мрежи са едни от най-мощните методи за машинно самообучение, но същевременно и най-трудните за разбиране, точно защото за постигането на тази мощност се изисква тяхната изключително сложна структура. Поради този факт, дълбоките мрежи са една от областите в които обяснимостта бива разработвана най-активно.

## Подходи

Изборът на подход зависи от множество различни фактори, като типа на модела, дали имаме директен достъп до него или не, изчислителната сложност на подхода и съответно търсения компромис между точност и скорост, дали се фокусираме върху глобални или колани свойства, съобразяване с регулации, и др.

Самите типове обяснения могат да бъдат категоризирани по различен начин, като например:

* дали те са *специфични за конкретен модел* или са *моделно-агностични* и могат да се прилагат към всеки модел
* дали са *глобални*, даващи цялостно обяснение на поведението на модела, или *локални*, произвеждащи индивидуални обяснения за отделни примери и атрибути
* дали биват предоставени чрез *текст*, чрез различни видове *визуализации*, чрез *правила*, и др.
* дали използват *същите входни атрибути като модела*, или използват *междинни атрибути* (например на междинните слоеве на конволюционни невронни мрежи)
* и други класификации

Някои конкретни подходи са[6]:

* глобални моделно-агностични
  + *Partial Dependence Plot* (*PDP*) – прост подход, показващ ефекта върху изхода на 1 или 2 атрибута, чрез което може да видим дали отношението между целта и дадения атрибут е линейно, монотонно, или по-сложно
  + *Feature Interaction* чрез *Friedman’s H-statistic* – измерва силата на взаимодействието между различни атрибути
  + *Functional Decomposition* – създава декомпозиция на многомерна функция и я изразява чрез сума на индивидуални ефекти на атрибути и взаимодействия, които могат да бъдат визуализирани
  + *Permutation Feature Importance* – измерва увеличението в грешката при предсказване на модела след пермутиране на стойностите на даден атрибут (при което се нарушава взаимоотношението между атрибута и изхода)
  + *Global Surrogate* – създаваме по-прост и интерпретируем модел (т.нар. „*глобален сурогат*“), който бива трениран да приближава предсказанията на по-сложния *black-box* модел който се опитваме да обясним. По-простият модел може да интерпретираме директно, и чрез анализа на неговите предсказания правим заключения за оригиналния по-сложен модел.
  + *MMD-critic* – използваме *прототипи* (инстанции които са представителни за целите данни) и *критицизми* (инстанции които не са представени добре от множеството прототипи) и анализираме техните разпределения и предсказанията върху тях
* локални моделно-агностични:
  + *Individual Conditional Expectation* (*ICE*) – начертава на графика по една линия за всяка инстанция, която показва как предсказването на класа на дадената инстанция се променя с промяната на атрибут. Локалния вариант на *PDP*.
  + *Local Surrogate* чрез *Local interpretable model-agnostic explanations* (*LIME*) – както при глобалния сурогат, създаваме прост интерпретируем модел който да приближава оригиналния, но тук се фокусираме върху индивидуални предсказания. *LIME* постига това като генерира нови данни от извадка на оригиналните, чрез въвеждане на вариации (като разбъркване или добавяне на шум), и тренира интерпретируем модел върху тях, претеглен по близостта им с оригиналните данни
  + *Counterfactual Explanations* – използваме идеята на алтернативни сценарии, представящи причинно-следствени ситуации от вида „Ако X не се беше случило, щеше да се случи Y“. На практика прилагаме това като променяме стойностите на атрибути на дадена инстанция и проверяваме как това променя предсказанието
  + *Scoped Rules (Anchors)* – обяснява предсказание като намира класификационно правило което „закотвя“ предсказанието достатъчно добре, тоест при промени на другите атрибути (тези които не са част от правилото) предсказанието си остава същото
  + *Shapley Values* – разглеждаме модела като игра, в която всеки атрибут е играч, а предсказанието е наградата. Така може да приложим метод от Теория на Игрите, с който да намерим как справедливо да разпределим наградата, което в този случай всъщност ни казва доколко всеки атрибут е допринесъл за предсказанието
    - тъй като изчисляването на тези *Shapley* стойности е тежко и бавно, са създадени и методи които по-ефективно намират техни приближения, предназначени за различни типове модели - *TreeSHAP*, *KernelSHAP*, *DeepSHAP*, и др.
* специфични за невронни мрежи:
  + *Learned Features* чрез *Feature Visualization* или *Network Dissection* – конволюционните мрежи научават абстрактни характеристики и концепти от изображения, и анализирането им може да ни помогне да разберем работата на цялата мрежа. Например при *визуализация на атрибутите* опитваме да визуализираме тези научени характеристики като максимизираме активизацията на дадени неврони
  + *Pixel Attribution (Saliency Maps)* – това е група методи, при които подчертаваме пикселите на изображението които са били от значение за дадена класификация. Категорията включва множество методи, които може да се разделят на две под-категории:
    - *Occlusion- or perturbation-based* – манипулират части от изображението за да генерират моделно-агностични обяснения. Към тях спадат *SHAP* и *LIME* (когато биват прилагани към класификация на изображения)
    - *Gradient-based* – много различни методи, които изчисляват по различни начини градиенти на предсказанията относно входните атрибути. Примери са *Vanilla Gradients*, *GradCAM*, *SmoothGrad*, *Integrated Gradients*, и др.
  + *DeepLIFT* – използва *backpropagation* за да сравни активацията на всеки неврон с референтна такава, чрез което открива тези неврони и тегла, които са имали значителен ефект върху изхода
  + *Detecting Concepts, Adversarial Examples, Influential Instances* и др.

# Проектиране

За целите на проекта са нужни както метод за обяснение, така и модел който всъщност да обясним. Проектът ще се фокусира върху обяснението на класификация на изображения, извършена чрез невронни мрежи, като за избягване на нуждата за дълго обучение ще бъдат използвани предварително тренирани модели или сравнително просто мрежи които се тренират бързо.

За демонстрация на методите за обяснимост ще бъдат използвани подходящи изображения, които ще трябва да минат през нужната предварителна обработка за да бъдат подадени на мрежите в очаквания формат. Това включва преоразмеряване (тъй като конволюционните мрежи обикновено работят с изображения с еднакъв размер, като всеки модел има свои изисквания за размера), конвертиране към правилната цветова схема, към нужния тип на данни и към нужната размерност.

След като имаме готови модели и изображения, създаваме обекти за избраните видове обяснения, като всеки от тях е асоцииран с конкретен модел и функция за обработка на изображенията, а в някои случаи и със специфичен слой на модела. Извикваме функциите за обяснения на тези обекти, като им подаваме избрано изображение и след като те извършват съответния анализ, ние като резултат получаваме нови изображения, състоящи се от оригиналните изображения с приложени отгоре им специфични оцветявания, подчертаващи дадени (групи) пиксели – тези, които използваният метод за обяснимост е маркирал като играли важна роля при класификацията.

# Реализация и резултати

## Използвани технологии, платформи и библиотеки

Използват се някои функции на библиотеката *numpy*, както и библиотеката *Pillow* за работа с изображения. За реализация на невронните мрежи за класифициране на изображения се използва библиотеката *TensorFlow*[7], като в проекта се използват две мрежи:

* *ResNet50*, която е 50-слойна конволюционна невронна мрежа, тренирана върху множеството данни *ImageNet*[10], съдържащо огромен брой снимки на обекти от най-различни типове. Конкретно се използва предварително трениран модел на *ResNet50* от *Keras Applications*[8].
* проста 7-слойна конволюционна мрежа[9], тренирана върху множеството данни *MNIST*[12], съдържащо изображения на ръкописни цифри. За дефинирането и тренирането ѝ се използват функционалностите на *Keras*, като за данните от *MNIST* се използва тяхна предварително обработена и форматирана версия от *Keras Datasets*[13]

За примерни обяснения на класификациите на първата мрежа се използват няколко избрани изображения от извадка на *ImageNet*[11], а за втората мрежа се използват избрани изображения от тестовото множество на *MNIST*.

За извършване на самите обяснения има множество различни библиотеки:

* *ELI5*, която имплементира различни методи за моделите от няколко библиотеки за машинно самообучение, като *scikit-learn*, *XGBoost*, *Keras* и др.
* *Captum,* която имплементира множество методи, но само за модели от *PyTorch*
* *DeepLIFT*
* *SHAP*, имплементираща упоменатото намиране на приближения на *Shapley* стойности, и методи като *TreeSHAP*, *KernelSHAP*, *DeepSHAP* и др.
* *tf-explain*, създадена за *TensorFlow 2.0* и имплементираща *GradCAM*, *SmoothGrad*, *Integrated Gradients* и др.
* *OmniXAI*, имплементираща множество различни методи, за разнообразни типове данни (таблични, изображения, NLP и времеви серии), и поддържаща модели от *scikit-learn* и невронни мрежи както от *TensorFlow* така и от *PyTorch*

Първоначално за проекта бяха избрани библиотеките *SHAP* и *tf-explain*, но за съжаление и при двете бяха срещнати технически проблеми, както и липса на детайлна документация. Беше направен опит за прилагане на *ELI5*, но се оказа че тя не може да работи с *TensorFlow* модели. Финално беше избрана *OmniXAI*[14], с която имах повече успех, като в допълнение беше използван и *DeepExplainer* от *SHAP* (всъщност *SHAP* модула на *OmniXAI* използва точно него, но по малко по-различен начин и с различна визуализация на резултатите).

OmniXAI поддържа голям брой различни методи, но за целите на проекта бяха избрани следните:

* *Gradient-weighted Class Activation Map* (*GradCAM*)[16] – подход базиран на градиенти, който както повечето други подходи от тази категория присвоява на всеки неврон ниво на релевантност към дадена класификация. Целта на *GradCAM* е да разбере за даден конволюционен слой към кои части от изображението той „гледа“ за да стигне до конкретна класификация, тоест на базата на кои региони от пиксели моделът решава че изображението принадлежи на един клас или друг. За да постигне това, алгоритъмът използва специфичната за класовете информация от градиента стигаща до финалния конволюционен слой.
* *Integrated Gradients* – прост, широко-приложим подход, който намира доколко всеки атрибут (в случая на изображения, всеки пиксел) влияе на изхода чрез изчисляване и интегриране на градиенти
* *SHAP* – използва метода *DeepSHAP*, който е базиран на *DeepLIFT*, но с изчисления на приближения на *Shapley* стойности
* *LIME* – конкретно при изображения, стандартният *LIME* подход с въвеждането на разбъркване на атрибутите не е подходящ, тъй като тук атрибутите са огромен брой пиксели, и принадлежността към даден клас бива определена от повече от един пиксел. Поради това при прилагането на *LIME* към изображения се използва сегментиране на изображението на „суперпиксели“ (т.е. цели региони от пиксели, свързани чрез сходство на цветове), и изключването и включването на избрани такива суперпиксели.

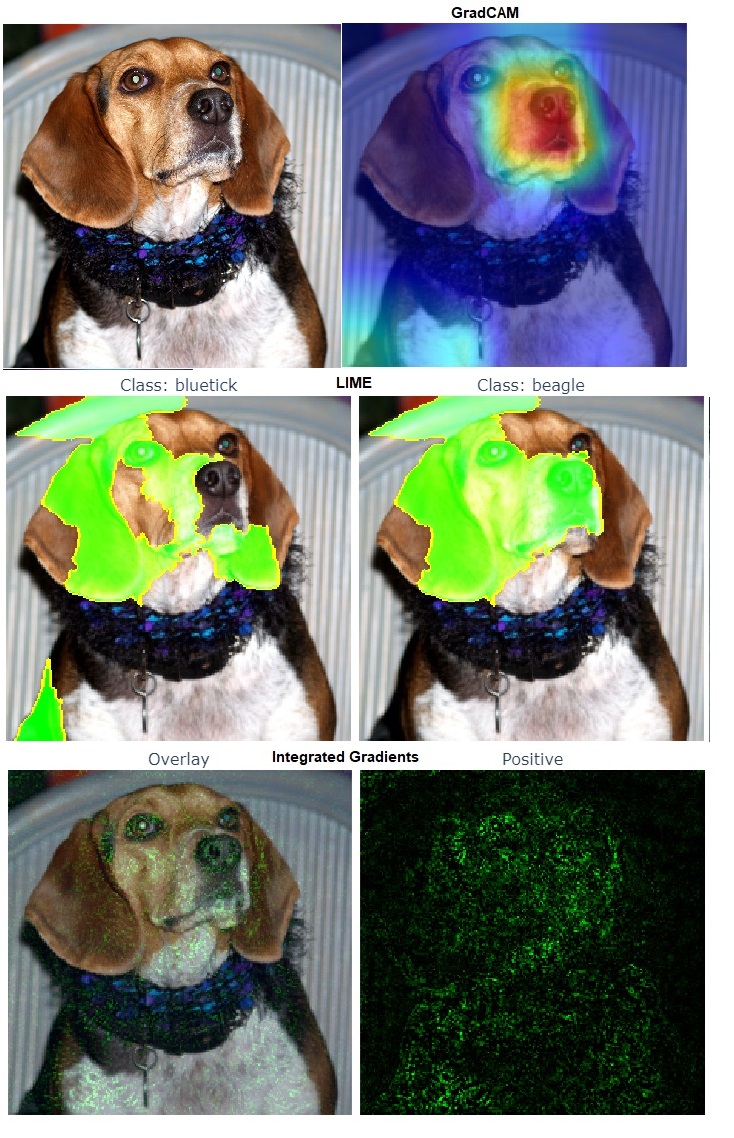
## Реализация

Невронните мрежи са *TensorFlow* модели, като *ResNet50* бива заредена директно като предварително трениран модел, а за мрежата за *MNIST* се извършва отделно трениране в *mnist\_model.py* след което тренираният модел бива записван на диска, за да може след това да бъде директно зареден.

Използват се функции на *Pillow*, *OmniXAI* и *Keras* за обработка на изображения, с които след зареждането им от диска те биват трансформирани в нужния формат за подаването им на невронните мрежи. За *ResNet50* се скалират до размер 224х224, а за *MNIST* мрежата до размер 28x28, след което се извършват и допълнителни операции като конвертиране до правилната цветова схема и разширяване до нужната размерност. За обясненията при *ResNet50* мрежата също така се налага да заредим етикетите на класовете от *ImageNet* в JSON формат.

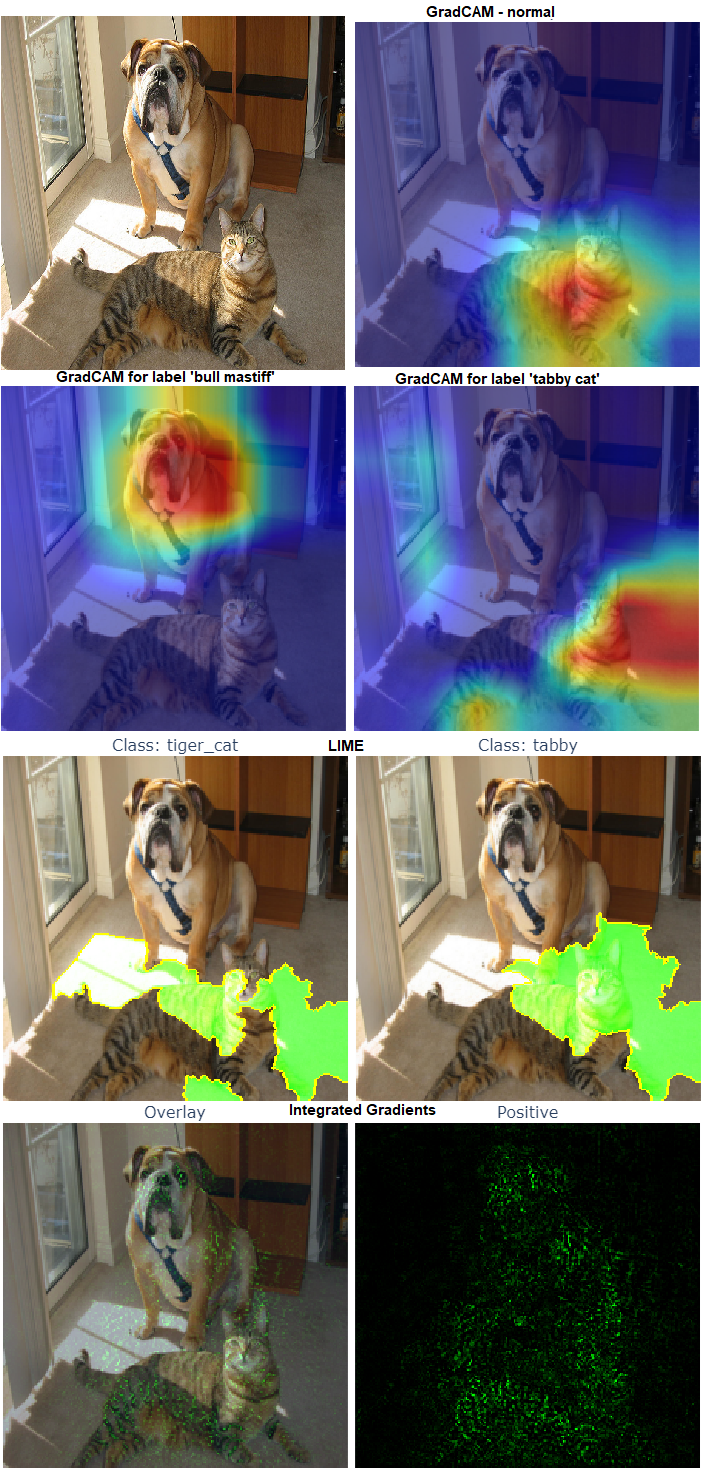
Първите три избрани метода имат подобен интерфейс за ползване – инициализира се обект на съответния клас чрез модел и функция за предварителна обработка, след което се извиква негова функция с подадено изображение, чиято класификация искаме да обясним. При *LIME* вместо това при инициализиране се подава само функция, връщаща вероятности за принадлежност към различните класове.

Поради технически проблеми с *SHAP*, обясненията му не работят за *ResNet50*, така че те са приложени само за *MNIST* мрежата. Другите три метода са приложени за *ResNet50* и няколко различни изображения.



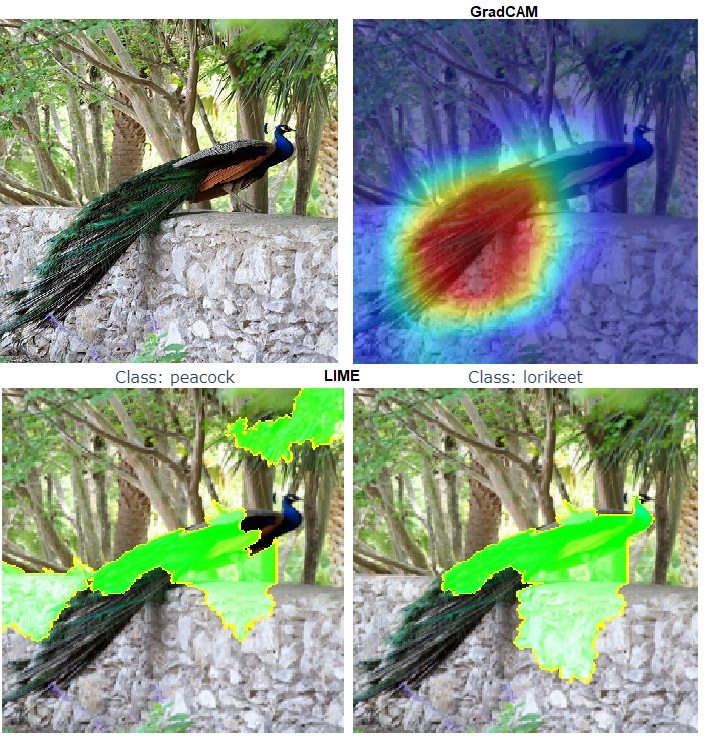
Фигура - Куче бийгъл

Може да видим че в случая на *GradCAM*, фокусът е бил върху муцуната на кучето, предимно носа и устата. При *LIME* виждаме двата най-вероятно класа, което ни показва колебанието между бийгъл и друга порода куче. При Integrated Gradients също муцуната е важна, но виждаме и други части от изображението които са били отчетени.



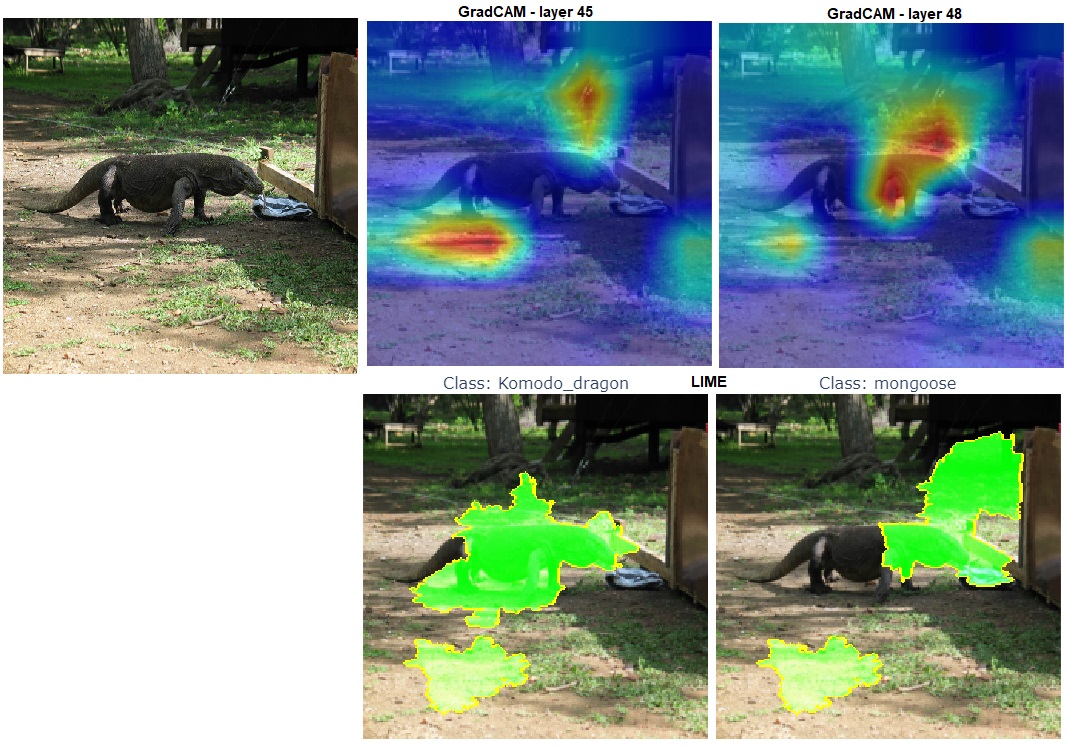
Фигура - Куче (булмастиф) и котка (най-вероятно таби)

Тук е демонстрирана и функционалността на GradCAM да даде обяснение за конкретен клас, което е полезно в изображения с няколко обекта като тук.



Фигура 3 - Паун

Тук може да видим разлика в обясненията – докато *GradCAM* се фокусира върху опашката на пауна, локалният модел на *LIME* по-скоро се фокусира върху тялото.



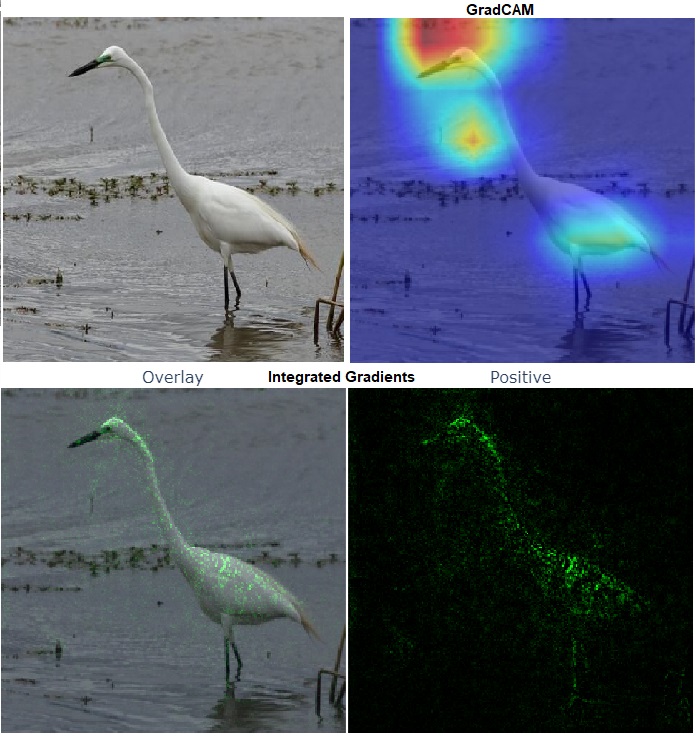
Фигура 4 - Комодски варан

Тук при проверката на 45ти слой с GradCAM изглеждаше че мрежата се фокусира върху напълно различни неща, но при минаване няколко слоя напред вече фокусът е преместен правилно върху животното.

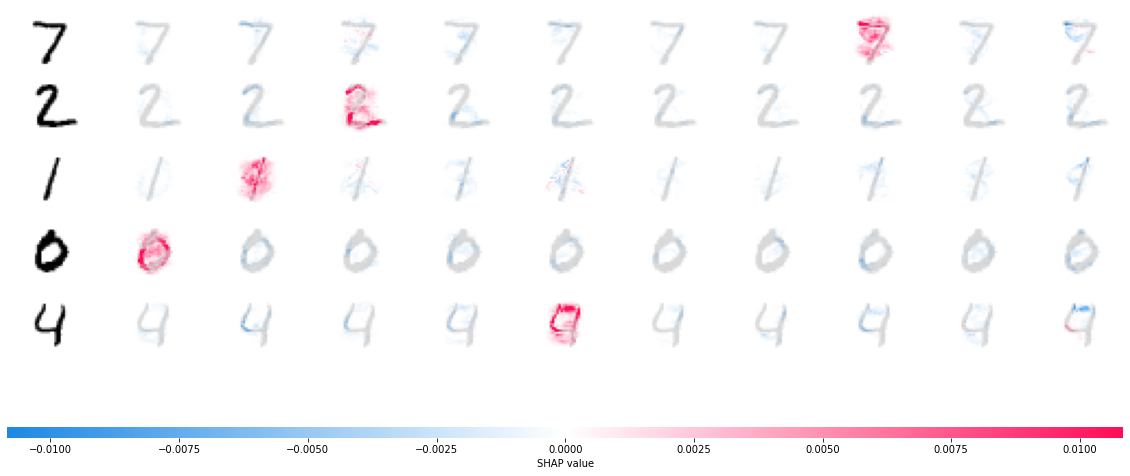


Фигура 5 - Toy terrier

Тук изображението има както куче, така и човек, като двете най-вероятно класификации са за куче (грешна порода, но все пак правилния тип животно) и газ маска, което звучи доста странно, но с LIME виждаме че това най-вероятно е тъй като лицето на човека не е напълно видимо и той носи очила



Фигура 6 - Американска чапла



Фигура 7 - Класификация на цифри с SHAP

Тук имаме 10 колони, съответно за класовете 0, 1, 2, и т.н до 9, като тъмно розовите пиксели са тези които потвърждават тази класификация, а сините които я отхвърлят. Например нека разгледаме последният ред (за цифрата 4) – в последната колона (за класа 9), може да видим че липсата на свързваща линия в горния края е нещото което спомага за отхвърляне на класификация като цифрата 9.

# Заключение

В проекта бяха приложени няколко метода за обяснение върху модели класифициращи изображения. За съжаление, голяма част от времето за разработката на проекта беше изгубена в неуспешни опити за боравене с няколко различни библиотеки, като *tf-explain*, *SHAP* и *ELI5*, докато евентуално не беше успешно приложена библиотеката *OmniXAI*, с която имаше по-малко проблеми. Изглежда че тъй като обяснимостта е сравнително нова, все още развиващата се област в машинното самообучение, все още няма толкова много добре развити и надеждни инструменти за прилагане ѝ.

Резултатите са интересни, но в някои случаи трудни за интерпретиране, и също така като цяло трудни за оценяване обективно, тъй като и самата обяснимост е субективно понятие, което не е лесно да бъде измерено и оценено математически.

В проекта се използва основно една невронна мрежа и малко множество изображения, поради ограниченото време, но разбира се той може да се разшири с експерименти върху други модели (като други варианти на *ResNet*, и други архитектури като *VGG16/19*, *Inception*, и т.н.) и повече изображения. Така например при изображения за които някои от мрежите бъркат, бихме могли чрез различните методи за обяснимост да се опитаме да разберем защо всъщност го правят. Освен това, проектът се фокусира върху класификацията на изображения, но разбира се има и много други области на машинното самообучение в които могат да се приложат методи за обяснимост, както други типове невронни мрежи, така и други категории модели. При бъдещо развитие на проекта може той да бъде разширен с допълнителни демонстрации за някои от тези случаи.

# Използвана литература

1. Aparna Dhinakaran, “What Are the Prevailing Explainability Methods?”, *Towards Data Science*, 22 December 2021

[<https://towardsdatascience.com/what-are-the-prevailing-explainability-methods-3bc1a44f94df>]

1. Ejiro Onose, “Explainability and Auditability in ML: Definitions, Techniques, and Tools”, *neptune.ai*, 13 December 2021

[<https://neptune.ai/blog/explainability-auditability-ml-definitions-techniques-tools>]

1. Florian Perteneder, “Understanding Black-Box ML Models with Explainable AI”, *Dynatrace Engineering*, 29 Apr 2021

[<https://engineering.dynatrace.com/blog/understanding-black-box-ml-models-with-explainable-ai/>]

1. Doug Kelly, “Introduction to Explainable AI (ML Tech Talks)”, *TensorFlow*, 15 Jul 2021

[<https://www.youtube.com/watch?v=6xePkn3-LME>]

1. Aqeel Anwar, “Difference between AlexNet, VGGNet, ResNet, and Inception”, *Towards Data Science*, 7 Jun 2019

[<https://towardsdatascience.com/the-w3h-of-alexnet-vggnet-resnet-and-inception-7baaaecccc96>]

1. Christoph Molnar, “Interpretable Machine Learning: A Guide for Making Black Box Models Explainable”, *Self-published*, 28 Feb 2022

[<https://christophm.github.io/interpretable-ml-book/>]

1. *TensorFlow*

[<https://www.tensorflow.org/>]

1. pre-trained deep learning models*, Keras Applications*

[<https://keras.io/api/applications/>]

1. Simple MNIST convent, *Keras Code Examples*

[<https://keras.io/examples/vision/mnist_convnet/>]

1. *ImageNet dataset*

[<https://image-net.org/>]

1. *ImageNet sample images*

[<https://github.com/EliSchwartz/imagenet-sample-images>]

1. *MNIST dataset of handwritten digits*

[<http://yann.lecun.com/exdb/mnist/>]

1. MNIST digits classification dataset, *Keras Datasets*

[<https://keras.io/api/datasets/mnist/>]

1. OmniXAI: A Library for eXplainable AI, *GitHub*

[<https://github.com/salesforce/OmniXAI>]

1. OmniXAI Documentation

[<https://opensource.salesforce.com/OmniXAI/latest/index.html>]

1. Grad-CAM: Gradient-weighted Class Activation Mapping, *CloudCV*

[<http://gradcam.cloudcv.org/>]

Фигури

1. Куче бийгъл
2. Куче (булмастиф) и котка (най-вероятно таби)
3. Паун
4. Комодски варан
5. Toy terrier
6. Американска чапла
7. Класификация на цифри с SHAP