UNIVERSITAT DE BARCELONA

Facultat de Matemàtiques
i Informàtica

GRAU DE MATEMÀTIQUES
GRAU D'ENGINYERIA INFORMÀTICA

Treball final de grau

Analyzing state-of-the-art CNN's explainability focusing on food classification

---

Autor: Joan Bergadà Salsen

Directora: Dra. Petia Radeva

Tutor: Javier Ródenas Cumplido

Barcelona, 21 de juny de 2020

Agraïments

Voldria agrair a la meva tutora Petia Radeva i al meu cotutor Javier Ródenas el seu ajut durant tota la realització d'aquest treball, tant en l'elecció de tema com en el seu desenvolupament. Sense ells la realització d'aquest Treball de Fi de Grau hauria estat molt més costosa.

També voldria agrair a Javier Pérez, professor de Matemàtiques de l'INS Manuel de Pedrolo, haver incentivat la meva curiositat per les ciències i, en concret, per les matemàtiques. Sense els seus anys d'ensenyança possiblement aquest Treball de Fi de Grau mai s'hauria realitzat.

Abstract

Convolutional Neural Networks (CNNs) are Deep Learning algorithms that can be applied to a wide range of environments with a high performance. We will study all the elements that form the CNNs, learn how they work, how to train them and how to diagnose its performance. We will also work the state-of-the-art visual explainability algorithms developing how they create their heatmaps. In the practical part we will use CNNs in order to classify food images into their classes and we will apply the studied explainability algorithms to understand the predictions made by the neural networks. Finally, we will perform a both qualitative and quantitative comparison between the explanations given by the applied algorithms.

Resum

Les xarxes neuronals convolucionals són algorismes de *Deep Learning* que es poden utilitzar en àmbits molt diversos amb molts bons resultats. En aquest treball estudiarem tots els seus elements en profunditat, aprendrem el seu funcionament, com entrenar-les i com diagnosticar-les. Paral·lelament, treballarem els algorismes d'explicabilitat visual de l'estat de l'art i detallarem com creen els seus *heatmaps*. A la part pràctica, utilitzarem xarxes neuronals convolucionals per a classificar imatges de menjar en diferents classes i aplicarem els algorismes d'explicabilitat estudiats per a entendre les decisions de classificació preses per les xarxes neuronals. Finalment, realitzarem una comparativa qualitativa i quantitativa entre les diferents explicacions donades pels algorismes d'explicabilitat.

2010 Mathematics Subject Classification. 68T05, 68T10, 68T45.

Contents

1 Introducció **1**

1.1 Motivació 4

1.2 Objectius 4

2 Estat de l'art **5**

2.1 Conceptes previs 5

2.2 Primera aproximació 6

2.2.1 Score function 7

2.2.2 Loss function 8

2.3 Xarxes neuronals 10

2.4 Entrenament d'una xarxa neuronal 13

2.5 Xarxes neuronals convolucionals 17

2.5.1 Definició 17

2.5.2 Capes d'una xarxa neuronal convolucional 17

2.6 Hiperparàmetres 22

2.7 Optimitzadors 25

2.8 Problemes a l'entrenar una xarxa neuronal 27

2.9 Algorismes d'explicabilitat 32

3 Planificació i metodologia **34**

4 Implementació **35**

4.1 Programari utilitzat 35

4.2 Detalls dels algorismes 36
4.2.1 CAM 36
4.2.2 Guided backpropagation 38
4.2.3 Grad-CAM i Guided Grad-CAM 39
4.2.4 Grad-CAM++ 41
4.2.5 Score-CAM 43
4.3 Datasets 44
4.4 Proves realitzades 46
4.4.1 Transer Learning de Cifar10 46
4.4.2 Realització de tests d'algorismes d'explicabilitat sobre Imagenet . . 46
4.4.3 Proves d'explicabilitat amb Food101 47
4.4.4 Proves d'explicabilitat amb multilabel 47

5 Resultats **47**

5.1 Transfer Learning de Cifar10 47
5.2 Explicabilitat sobre Imagenet 49
5.3 Transfer Learning i Explicabilitat sobre Food101 50
5.4 Explicabilitat sobre Food201 51

6 Conclusions i futurs treballs **52**

Annexos **54**

A Estructures de les xarxes neuronals convolucionals creades **54**

B Imatges de mostra **57**

B.1 Food101 57

B.2 Food201 66

Bibliografia **69**

Taula d'imatges

1 Extreta de la col·lecció pública Food101. Modificada per a marcar el valor d'un píxel. 5

2 Extreta de [28], lectura 4. Estructura d'una xarxa neuronal de dues capes ocultes amb les seves neurones. 11

3 Extreta de [28], lectura 4. Funcions d'activació. 12

4 Extreta de [28], lectura 4 i adaptada. Graf de la funció $f(x, y, z) = (x + y) * z$ amb els valors del *forward pass* damunt i els gradients del *backward pass* sota les fletxes. 16

5 Captura d'un entrenament d'una xarxa neuronal, època 4, on podem veure la pèrdua i la precisió de l'entrenament i les seves anàlogues de validació. La quantitat de dades d'entrenament és de 2250. 16

6 Extreta de [28], lectura 5. Estructura bàsica d'una xarxa neuronal convolucional. 17

7 Extreta de [24], Capes d'una xarxa neuronal convolucional. 18

8 Extreta de [28]. Aplicació d'una capa densa a una entrada per a obtenir una sortida de 10 puntuacions. 18

9 Creació pròpia. A la primera fila, les 4 primeres aplicacions d'un filtre 3x3 a una entrada 7x7 amb pas 1. A la segona fila, el mateix amb pas 2. 19

10 Extreta de [28]. Podem veure com un input de 32x32 passa per una capa convolucional amb 6 filtres de 5x5, generant 6 mapes d'activació. 19

11 Extreta de `https://www.jeremyjordan.me/content/images/2018/02/Screen-Shot-2018-02-24-at-11.47.09-AM.png`. Casos segons el learning rate. 22

12 Extreta de `https://miro.medium.com/max/1146/1*vwICNxW03oZxdFsXntfg2g.png`. Problema de l'SGD. 25

13 Extreta de [28], lectura 8. Esquema de funcionament de SGD+*momentum*. 26

14 Extreta de [15]. Gràfic de les funcions de pèrdua segons els problemes amb el learning rate. . . 29

15 Extreta de [15]. Diagnosi d'overfitting segons les precisions de la xarxa neuronal. 29

16 Extreta de `https://www.researchgate.net/profile/Geok_Ng/publication/327203363/-accuracy-and-interpretability.png`. Explicabilitat vs precisió. 31

17 Etapes de la realització del Treball de Fi de Grau. 35

18 Extreta de [3]. Funcionament de CAM. 36

19 Extreta de [3]. Resultats de CAM. 37

20 Extreta de [16]. Funcionament de Guided Backpropagation, fusionant Backpropagation i backward 'convnet'. 38

21 Extreta de [16]. Resultats de Guided Backpropagation. 39

22 Extreta de [22]. Resultats de Grad-CAM. 40

23 Extreta de [22]. Funcionament de Grad-CAM: realitzar *backpropagation* per a obtenir les jacobianes, calcular les importàncies i, finalment, fer la suma ponderada dels mapes d'activació, passant el resultat per una ReLu. 40

24 Extreta de [1]. Funcionament de Grad-CAM++. 41

25 Extreta de [1]. Comparativa de Grad-CAM++ i Grad-CAM. 42

26 Extreta de [11]. Comparativa de Grad-CAM++, Grad-CAM i Score-CAM. 43

27 Extreta de [11]. Procediment d'Score-CAM. 44

28 Extreta de [21]. Transfer learning en una VGG16. 46

29 Extreta de `https://neurohive.io/en/popular-networks/vgg16/`. xarxa neuronal VGG16. . . 48

30 Resultats Transfer Learning. Predit amb un 99% i un 84% de seguretat, respectivament. . . . 48

31 Resultats explainability a Imagenet. 50

32 Procés d'entrenament per a Food101. 51

33 Explicabilitat per a patates i hamburguesa. 52

34 Xarxa neuronal adaptada per a classificar Cifar10 a partir d'una VGG16. 54

35 Xarxa neuronal adaptada per a classificar Food101 a partir d'una VGG16. 55

36 Xarxa neuronal adaptada per a classificar Food201 a partir d'una VGG16. 56

37 Imatges de mostra d'explicabilitat sobre Food101. 65

38 Imatges de Food201 amb explicabilitat sobre patates (dalt) i hamburgueses (baix). Amb puntuacions per classe. 66

39 Imatges de Food201 amb explicabilitat sobre patates (dalt) i hamburgueses (baix). Amb puntuacions per classe. 67

40 Imatges de Food201 amb explicabilitat sobre patates (dalt) i hamburgueses (baix). Amb puntuacions per classe. 68

1 Introducció

La intel·ligència artificial és una de les branques de la informàtica que més s'ha desenvolupat en els darrers anys. Es considera que aquesta es va establir com a disciplina l'any 1956, al *Dartmouth Summer Research Project on Artificial Intelligence*, on també es va acordar el nom pel que se la coneix. No obstant, ja des dels anys 30 s'havien fet descobriments que durien inevitablement a la consolidació d'aquesta branca: estudis en el camp de la neurologia van demostrar que el cervell es basa en una xarxa de neurones que envia impulsos elèctrics de forma binària. Això, juntament amb els avenços fets en paral·lel en el camp de la computació, obliguen a formular la pregunta de si es pot crear un ''cervell digital".

L'any 1950, Turing presenta el seu famós Test de Turing i posa sobre la taula una definició de màquina intel·ligent: si aquesta és capaç de prendre les mateixes decisions i actuar com ho faria un ésser humà, llavors es pot considerar que està pensant. Turing arriba a aquesta definició a partir del cas concret en el que una persona no és capaç de discernir si està parlant, mitjançant interacció escrita, amb una màquina o amb un altre humà.

Des del seu assentament, la intel·ligència artificial ha sofert un desenvolupament irregular amb dos períodes coneguts com els dos hiverns de la IA (1974-1980 i 1987-1993). Aquests períodes de decadència foren causats per la gran expectació vers aquest camp, el seu ràpid creixement i unes promeses d'obtenir grans resultats a curt termini que van resultar impossibles de complir donada la limitació tècnica que suposava la capacitat d'emmagatzematge i computació dels ordinadors d'aquella època.

No obstant, fora d'aquests períodes la IA es va desenvolupar amb força, aconseguint moltes aplicacions pràctiques en camps variats com poden ser la resolució de jocs competitius, la classificació d'elements o l'aprenentatge de llengües. Concretament, a partir de l'any 2011, la intel·ligència artificial ha efectuat passos agegantats dins les àrees de *Machine Learning* i de *BigData*.

El mateix any 2011, la intel·ligència artificial Watson, d'IBM, va competir i guanyar en un concurs de preguntes i respostes d'Estats Units. Watson és una IA dissenyada per a analitzar preguntes formulades en llenguatge natural i donar una resposta adequada a

elles. El 2013 es va començar a utilitzar en l'àmbit mèdic per a prendre decisions sobre l'autorització de tractaments de càncer de pulmó per la sanitat estatunidenca.

El 2015, la IA AlphaGo, de DeepMind (Google), venç al campió europeu de Go i el 2017 una versió millorada derrota al millor jugador de Go del món en tres ocasions diferents.

El 2018, Alibaba, una IA de processament de llenguatge demostra una capacitat de comprensió lectora millor que la humana en un test de lectura i comprensió de la Universitat d'Stanford.

Aquest "boom" va associat tant a la millora exponencial de la tecnologia, que proporciona cada cop més capacitat de càlcul, com als esforços fets als anys 80 per a aconseguir que la IA sigui una rama que comparteix coneixements i rigor amb altres àrees científiques com són les matemàtiques, fent possible la creació i la millora d'algorismes que tracten una gran quantitat de dades i necessiten realitzar una quantitat d'operacions que uns anys abans seria impensable poder computar i optimitzar.

Dins el camp del *Machine Learning* i, més concretament, a l'àmbit del *Deep Learning*, aquest desenvolupament de la intel·ligència artificial es tradueix en el perfeccionament de les xarxes neuronals, millorant les recurrents (RNN, de l'anglès *Recurrent Neural Network*) i les convolucionals (CNN, de l'anglès *Convolutional Neural Network*). Tot i que aquests tipus de xarxes no eren conceptes nous, van poder perfeccionar-se tècnicament de forma efectiva als voltants del 2010 i permeten crear algorismes molt potents d'aprenentatge en camps molt variats com poden ser la classificació d'imatges, la detecció d'objectes o l'anàlisi de texts en el cas de les CNN; i el reconeixement i la classificació de veu i so en el cas de les RNN.

No obstant, aquesta explosió d'algorismes amb gran capacitat d'aprenentatge ha comportat que, tot i fer increïbles avenços i crear aplicacions potents i altament efectives per a la realització de diferents tasques d'anàlisi i/o classificació, no s'hagi parat atenció al control dels possibles errors en les decisions preses per aquestes. Això provoca que si una xarxa neuronal ja entrenada s'equivoca amb la seva decisió sobre un element, l'usuari és incapaç de veure per què és errònia aquesta sortida. És més, encara que la xarxa doni una sortida correcta, l'usuari no té forma de quantificar la veracitat d'aquesta i això li impedeix dipositar la seva confiança, restant utilitat a la mateixa xarxa.

Per exemple, creure cegament una xarxa neuronal que detecta càncers en imatges clíniques que no és capaç d'explicar les seves decisions pot conduir a situacions fatals si dona diagnòstics erronis. A causa d'aquest gran problema, les xarxes neuronals han estat considerades des de la seva creació com caixes negres a les que hom hi introdueix unes dades i se n'obté una resposta sense cap mena justificació, impedint en gran mesura la seva utilització comercial.

Actualment ja s'han creat alguns algorismes per a buscar solucions a aquest contratemps i ajudar a enfortir el pont de confiança entre usuaris i xarxes neuronals, encara que la feina que queda per fer en aquest àmbit és gran. Alguns d'aquests algorismes es centren en mostrar com la xarxa neuronal pren una decisió a partir de l'estat d'alguna de les seves capes intermèdies. És el cas dels algorismes desenvolupats a partir de CAM com són Grad-CAM, Grad-CAM++ i Guided Grad-CAM, entre d'altres, basats en la idea de mostrar, en el cas de la classificació d'imatges, un *heatmap* sobre la imatge tractada que mostra en quines parts d'aquesta s'ha basat l'algorisme per a prendre una decisió respecte a quina classe pertany. Així, l'usuari pot verificar que la sortida que està donant l'aplicació és correcta: si l'algorisme dona que la imatge és d'un gos i indica que ha detectat el pèl, les orelles i el morro sobre la pròpia imatge i, per tant, comunica que ho ha utilitzat per a prendre la seva decisió, ajuda a crear confiança i fiabilitat per als usuaris de la xarxa neuronal, ja que veuen que les decisions no són aleatòries, sinó que són ben fundades. És més, aquests algorismes de justificació de la sortida són molt útils també per a realitzar tests durant el desenvolupament d'aquestes aplicacions: al crear una aplicació, interessa comprovar abans de posar-la al mercat si aquesta pren decisions de forma correcta i considerant les parts de la imatge que hauria d'estar considerant en els diferents casos experimentals.

Vegem-ho amb un exemple: hom podria crear una xarxa neuronal que detecti a la perfecció doctors en una imatge. No obstant, la decisió de notificar que hi ha un doctor s'hauria de basar en la presència de la bata blanca, un fonendoscopi... I no pas en com porta el pèl el doctor, el seu color de pell o el seu sexe. Detectar a temps si aquesta xarxa neuronal es fixa en atributs no inherents a "ser un doctor" pot permetre corregir-ho i, per tant, donar un producte de qualitat i de confiança.

Aquest treball de fi de grau, com especificarem a l'apartat Objectius, tracta d'entendre aquests algorismes i experimentar amb ells, veient com reaccionen en diverses situacions

i analitzant si són ja prou efectius com per a ajudar a netejar el mal renom que tenen les xarxes neuronals en termes d'explicabilitat.

1.1 Motivació

Durant la meva etapa com a estudiant no m'ha estat possible treballar en cap moment amb xarxes neuronals. Si bé tant a Intel·ligència Artificial com a Visió Artificial el terme ha sortit en algunes ocasions (en la segona assignatura hi vam dedicar unes sessions per a veure'l de forma resumida), no s'ha profunditzat sobre ell en el temari ja que les assignatures tenien altres objectius. No obstant, les xarxes neuronals en totes les seves variacions tenen una gran importància i tot sembla indicar que aquesta anirà en augment durant els pròxims anys. Per això, i donat que semblen una eina molt útil dins el camp de *Machine Learning* , em va atraure la idea de realitzar un Treball de Fi de Grau en aquest camp. A més, em podria servir com a introducció a un possible màster i/o especialització orientats en aquest àmbit.

Entre les diferents opcions de Treballs de Fi de Grau orientats a les xarxes neuronals hi havia estudiar no només el funcionament de les xarxes en sí si no també el que s'anomena, en anglès, *explainability*. No és difícil adonar-se que l'explicabilitat d'una xarxa neuronal és tan important com el seu correcte funcionament i, ja que un dels objectius principals del meu Treball de Fi de Grau era adquirir tots els coneixements, tan teòrics com pràctics, sobre les xarxes neuronals que pogués, em va semblar molt interessant profunditzar també en com es poden justificar davant d'un usuari les decisions preses per una xarxa neuronal ja entrenada, ja que al cap i a la fi un programa tan complex que dona una sortida a unes dades sense cap mena de justificació és complicat que aconsegueixi la popularitat suficient per a ser competitiu.

1.2 Objectius

Els objectius d'aquest treball, com s'ha explicat a la secció anterior, giren al voltant d'augmentar el meu coneixement en aquesta àrea. Així, el que ens proposem és:

- Aprendre en profunditat el funcionament teòric de les xarxes neuronals.

- Familiaritzar-nos amb els conceptes tècnics que s'utilitzen per parlar de xarxes neuronals i la seva explicabilitat.
- Posar en pràctica els conceptes i algorismes apresos en els punts anteriors per a entrenar una xarxa neuronal.
- Entendre i aplicar els principals algorismes de l'estat de l'art que utilitzen les eines d'explicabilitat visual per a xarxes neuronals.
- Realitzar experiments i tests sobre aquests algorismes d'explicabilitat per a veure el seu rendiment, utilitzant *datasets* relacionats amb el menjar.

2 Estat de l'art

En aquesta secció sintetitzarem de forma ordenada tots els coneixements teòrics sobre les xarxes neuronals i la seva explicabilitat que, d'alguna forma, permetran realitzar la part pràctica d'aquest treball que exposarem en els apartats posteriors. També inclourem en aquest apartat, per exemple, mètodes d'explicabilitat que no hem treballat de forma pràctica però són prou significants dins l'àmbit de les xarxes neuronals com per ser explicats de forma teòrica.

2.1 Conceptes previs

Imatge 1: Extreta de la col·lecció pública Food101. Modificada per a marcar el valor d'un píxel.

Abans d'endinsar-nos en el complex funcionament de les xarxes neuronals i estudiar com fan front al problema de la classificació d'imatges, és convenient una secció introductòria sobre quin és el format de les dades amb les que treballen i on començarem a introduir conceptes que utilitzen, com veurem més endavant, les xarxes neuronals.

Com es pot apreciar a la imatge 1, treballarem amb imatges a color formades per píxels. Cada píxel es pot considerar una llista ordenada de 3 elements, indicant els 3 canals RGB (vermell, verd i blau, respectivament) que formen el color que nosaltres percebem amb els nostres ulls. Aquests tres valors solen estar compresos

entre 0 i 255 segons la intensitat del color corresponent, i solen ser enters. No obstant, és usual trobar-se'ls reescalats entre 0 i 1, ja que treballar amb valors petits ajuda a que els pesos de les xarxes neuronals també ho siguin i funcionin millor les capes d'activació, concepte que definirem més endavant. Així, tornant a l'exemple, podem veure que un píxel de la imatge té els valors (x,y,z), equivalents a (x/255,y/255,z/255) entre 0 i 1. Suposant que la imatge és de 32x32, tenim que es pot representar mitjançant 32x32x3 valors.

Val la pena fer menció al terme de classe a la que pertany una imatge. De forma òbvia, una imatge pertany a la classe X si hi és present aquest X. Posant un exemple, la imatge 1 pertany clarament a la classe de les amanides.

Així, el problema que volem resoldre és com podem fer que donada una imatge se'ns digui a quina classe pertany. Per a resoldre'l és usual tenir una col·lecció d'imatges ja associades amb la seva classe correcta per a tal d'aprendre a partir d'ella. Durant l'etapa d'entrenament, es disposa d'una separació d'aquesta col·lecció entre dades d'entrenament i dades de test. Llavors, la xarxa neuronal es mira les dades d'entrenament, aprèn d'elles i un cop les ha treballat totes valida el seu coneixement sobre les de test. Aquest procediment es repeteix fins que la xarxa neuronal ha acabat d'entrenar. És important la separació de les dades, ja que si utilitzés les mateixes dades per entrenar que per validar hom no sabria si ha estat capaç d'extreure les característiques de les classes i, per tant, generalitzar com es requereix.

Els algorismes que utilitzen dades per a entrenar de forma satisfactòria s'anomenen d'aprenentatge supervisat. Dins d'aquest treball, dedicat a les xarxes neuronals convolucionals, només estudiarem, per tant, algorismes d'aquest tipus [25].

En contrast, existeixen els algorismes no supervisats, que es basen en buscar patrons o similituds en unes dades d'entrada no identificades, creant ells mateixos les classes de semblança adequades i extraient les característiques essencials de cadascuna, sent capaços de classificar una nova dada dins d'una d'aquestes agrupacions.

2.2 Primera aproximació

Per a realitzar una aproximació suficientment bona que ens dugui a poder classificar correctament una imatge dins la seva classe són necessaris dos elements. Primer, una

funció que, donada una imatge formada per píxels, ens proporcioni les puntuacions de les classes associades a la imatge per tal de poder seleccionar la classe amb la puntuació més alta. Segon, una funció que ens digui com de "bona" és aquesta puntuació respecte a la classe real a la que pertany cada imatge. Aquestes funcions s'anomenen funció de puntuació i funció de pèrdua (en anglès, *score function* i *loss function*), respectivament. ([10], capítol 6).

2.2.1 Score function

Utilitzarem com a exemple una funció de puntuació bàsica que després extrapolarem al funcionament general d'una xarxa neuronal. Aquesta rep el nom de classificador lineal (*linear classifier*, en anglès).

Sigui x una imatge amb tots els seus píxels "aplanats" formant un vector de dimensió K i sigui D el nombre de classes possibles a les que pot pertànyer x. Definim la funció *score* s de la següent forma:

$$s = f(W, x, b) = Wx + b \tag{1}$$

on W és una matriu de dimensió DxK i b és un vector de dimensió D. Per tant, s té dimensió D i està formada per les puntuacions de cada classe respecte la imatge x.

La matriu W rep el nom de matriu de pesos i el vector b rep el nom de vector de biaix (*weights matrix* i *bias vector*, respectivament, en anglès). Cal mencionar que és comú associar al terme pesos o paràmetres d'una xarxa neuronal tant les matrius de pesos com els vectors de biaix.

Un cop definida la funció de puntuació, el problema de classificar imatges es redueix a crear W i b utilitzant la col·lecció d'imatges etiquetades donades per a entrenar. Un cop obtinguts, les imatges ja no són necessàries i es poden descartar, fent que classificar una imatge amb el mètode obtingut sigui molt poc costós, a diferència d'altres mètodes molt menys eficients i precisos com podria ser el k-Nearest Neighbor [12], [26].

A la pràctica, és comú expandir x en una dimensió amb valor 1, de tal forma que es pot posar b com una columna de W i simplificar la funció de puntuació, quedant

$$s = f(W, x, b) = W'x' \tag{2}$$

2.2.2 Loss function

La funció de pèrdua ([28], lectura 3) s'encarrega d'expressar com de conformes estem amb els resultats que ens donen un pes i un biaix. Idealment, serà 0 si l'algorisme funciona a la perfecció i no dubta amb cap imatge del conjunt d'imatges d'entrenament i anirà en augment com més disti el resultat obtingut de l'òptim. A la pràctica s'utilitzen dues funcions de pèrdua, obtenint resultats molt similars.

La primera d'elles és la funció *Multiclass Support Vector Machine*, SVM. Aquesta funció es defineix utilitzant un marge Δ. Sigui x_i una imatge pertanyent a la classe y_i. Es defineix la funció de pèrdua SVM de la següent forma:

$$L_i = \sum_{j \neq y_i} max(0, s_j - s_{y_i} + \Delta) \tag{3}$$

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i \tag{4}$$

on s_j indica el valor de la funció *score* de la imatge x_i en la classe y_j (classe a la que pertany x_i). Per tant, la funció SVM serà menor com més gran sigui la puntuació de la classe bona respecte les restants.

La segona és l'anomenada funció *Softmax*. Ve donada per

$$L_i = -log(P(Y = y_i | X = x_i)) \tag{5}$$

$$P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}} \tag{6}$$

És a dir, la funció de pèrdua *Softmax* busca interpretar les puntuacions de la funció score com probabilitats. És notori veure que si $s_k \in R$, $P(Y = k|X = x_i) \in [0, 1]$ i, per

tant, L_i será més gran com més disti d'1 la probabilitat calculada de que la imatge x_i pertanyi a la classe y_i.

Aquestes definicions de funció de pèrdua tenen, no obstant, un problema. Suposem que aconseguim uns pesos i un biaix tals que L és 0. Per exemple, tenim que si doblem els valors de W i b també tenim $L = 0$ en el cas de la funció SVM. Llavors, la pregunta que hom s'ha de formular és com saber elegir entre tots els pesos que minimitzen la funció de pèrdua.

Per a resoldre-ho, afegim a la fórmula una funció de regulació (*regularization function*) [23], ([10], capítol 7) $R(W)$ a la funció de pèrdua, quedant

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i + \lambda R(W) \tag{7}$$

on λ és un hiperparàmetre (*hyperparameter*) que controla com de forta és aquesta regulació. Un hiperparàmetre és un valor que hom fixa i per al qual ha de buscar el valor adient per a cada cas. En general, la inicialització recomanada per defecte que funciona bastant bé. Parlarem dels hiperparàmetres de les xarxes neuronals més endavant en aquest treball.

Hi ha diferents tipus de funcions de regulació. Entre elles, destacar les funcions de regulació $L2$ i $L1$, que tenen les formes següents

$$L1 = \sum_i \sum_j |W_i, j| \tag{8}$$

$$L2 = \sum_i \sum_j W_{i,j}^2 \tag{9}$$

on els sub-índexos i, j indiquen l'element de la fila i, columna j de la matriu W.

Notem que ambdues regulacions prefereixen els pesos de la xarxa neuronal petits en contra dels grans. Això és a causa de que en general ens interessa tenir valors petits, ja que ajuden a prevenir l'*Overfitting*, un problema de les xarxes neuronals que tractarem durant el penúltim apartat de l'estat de l'art.

2.3 Xarxes neuronals

En aquesta secció explicarem per què les xarxes neuronals reben tal nom i utilitzarem els conceptes donats en la secció anterior per a extrapolar l'estructura d'una xarxa neuronal.

Prenem primer el classificador lineal vist a l'apartat anterior. Si n'encadenem dos, posant els biaixos respectius dins la matriu W, obtenim

$$f(x) = W_1(W_2x) \tag{10}$$

Notem que si x té dimensió k ($k+1$ si incloem la dimensió afegida pel biaix), W_2 ha de tenir dimensió $(d+1)$ x $(k+1)$ i, per tant, W_1 té dimensió n x $(d+1)$, on k és la quantitat de píxels en una imatge i n el nombre de classes d'imatge a considerar per la classificació. Notem, però, que prenent $W = W_1W_2$ obtenim un classificador lineal altre cop.

Ara, considerem la següent funció de puntuació

$$f(x) = W_1max(0, W_2x) \tag{11}$$

on $max(0, W)$ indica que es fa el màxim entre cada posició de la matriu W i 0.

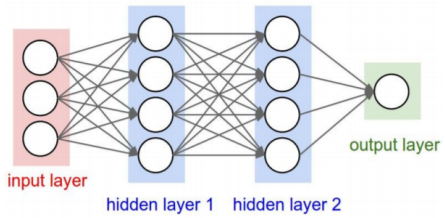
Concatenar aquesta funció al primer classificador lineal i abans d'aplicar el segon fa que la linealitat es perdi, però es mantenen les dimensions descrites anteriorment. Aquesta operació que s'intercala entre les aplicacions dels pesos i biaixos rep el nom de funció d'activació (*activation function*, en anglès). A l'exemple, s'ha utilitzat una de les més famoses i àmpliament utilitzades: la ReLu (*Rectified Linear Unit*). [6]

Una concatenació d'n classificadors lineals intercalant entre ells funcions d'activació és el que s'anomena xarxa neuronal. Així, l'exemple posat és una xarxa neuronal de dues capes, o bé d'una capa oculta (*hidden layer*), donat que entre l'aplicació de W_2, que s'anomena capa de sortida (*output layer*), i la capa d'entrada (*input layer*) que rep les dades per passar-les per la xarxa neuronal, només hi ha una capa: l'aplicació de W_1 amb la seva activació. A l'acció de passar una imatge per la xarxa neuronal per a obtenir les seves puntuacions se l'anomena fer el *forward pass* de la imatge x per la xarxa neuronal.

A aquestes alçades de la secció hom ja pot entendre la relació de les xarxes neuronals i les neurones del cervell. Una xarxa neuronal està dividida en capes, ja siguin classificadors lineals o capes convolucionals com les que veurem en el cas de les xarxes neuronals convolucionals. Aquestes capes prenen un *input* i el transformen, passant-lo a la capa d'activació que envia els valors a la següent capa de la xarxa. No és difícil deduir que cada valor passat a la següent capa és resultat d'uns valors concrets de la capa anterior. Per exemple, en el cas d'un classificador lineal es té que el valor i de la sortida és la multiplicació de la fila i dels pesos per l'entrada més l'element i del biaix passant per l'activació:

$$y_i = \sigma(W_i * x + b_i) \tag{12}$$

on σ és l'activació.



Imatge 2: Extreta de [28], lectura 4. Estructura d'una xarxa neuronal de dues capes ocultes amb les seves neurones.

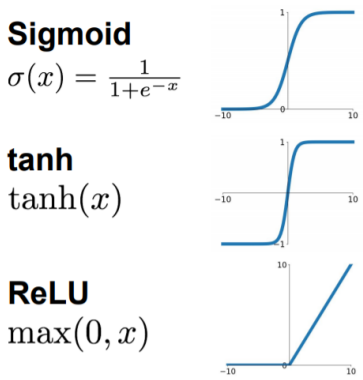
Al conjunt d'elements d'una capa que permeten la sortida d'un element cap a la següent se l'anomena una neurona de la xarxa neuronal, ja que actua com una neurona del cervell: rep el senyal enviat per les neurones anteriors, el processa i el propaga cap a les neurones on està connectada. Per tant, és usual entendre les capes de les xarxes neuronals com conjunts de neurones. Podem veure una xarxa neuronal amb les seves neurones a la imatge 2.

Abans de passar a la següent secció, on explicarem com s'entrenen els pesos de la xarxa neuronal, exposarem algunes idees i conceptes sobre les funcions d'activació.

Les funcions d'activació ajuden en molts aspectes a l'aprenentatge de les xarxes neuronals. Per una banda, afegeixen no-linealitat, permetent a la xarxa neuronal aproximar funcions més complexes. L'objectiu d'una xarxa neuronal en el cas que tractem és trobar una funció $f : A \to B$ del conjunt d'imatges A cap a $B \subset R^n$ tal que, si $x \in A$ i x pertany a la classe $k <= n$, llavors $f(x) = b$ on b és un vector de puntuacions que indica la classe k com la correcta i fa que la funció de pèrdua sigui 0.

No obstant, trobar una funció així pot ser una tasca impossible de realitzar. Així, el que es busca realment és aproximar-la el millor possible i, per tant, és necessari que la xarxa neuronal pugui adaptar-se el màxim de bé a la funció desitjada, que serà complicada amb quasi total seguretat.

Per altra banda, les funcions d'activació ajuden a controlar els valors que passen per la xarxa neuronal. Per exemple, la ReLu només deixa passar positius, mentre que una activació tanh els deixa entre -1 i 1. És molt útil també l'activació *Softmax*, sobretot a l'aplicar-la a la capa de sortida, permetent rebre puntuacions ja entre 0 i 1, facilitant moltíssim veure quina és la classe predita de forma immediata i facilitant calcular el valor de la funció de pèrdua. A la imatge 3 s'il·lustren algunes funcions d'activació conegudes.



Imatge 3: Extreta de [28], lectura 4. Funcions d'activació.

Òbviament, les funcions d'activació també tenen inconvenients. El primer problema que podem observar és que tendeixen a "saturar" els valors de la xarxa neuronal. Si prenem una xarxa neuronal que utilitza funcions d'activació Sigmoid, tots els valors grans col·lapsen a 1, mentre que els petits ho fan a 0. En el cas de la ReLu, es pot donar el cas de que arriben valors negatius importants que no han de ser posats a 0 (notem que en el cas de la ReLu no és ben bé un problema de saturació).

Això és un gran impediment, ja que es perd la diferència i importància entre els valors, fent que només es mantinguin sense saturar els valors propers al 0 i, per tant, es perdin, potser, característiques importants extretes per la xarxa neuronal. Afortunadament, es pot resoldre fàcilment aplicant, per exemple, normalització o escalat en les dades. Per exemple, per la funció de ReLu seria suficient centrar els valors al voltant del 0 i en el cas de la Sigmoid podríem primer centrar les dades i després escalar-les per a evitar saturacions.

El segon problema té a veure amb la següent secció que exposarem. D'aquí unes línies veurem necessari calcular el gradient al llarg de la xarxa neuronal. La majoria de funcions d'activació tenen un gradient menor que 1, fent que, per xarxes neuronals amb moltes capes (i, per tant, moltes funcions d'activació), a l'aplicar la regla de la cadena per moure el gradient endarrere per la xarxa 1neuronal, aquest es redueixi de forma excessiva, fent

difícil l'aprenentatge. Potser aquest paràgraf a vista del lector ara mateix no tingui el sentit que hauria de tenir, però seria convenient tenir-lo en ment durant la lectura del següent apartat.

A nivell pràctic, el que més s'utilitza són ReLu per a les capes intermèdies i una *Softmax* com a activació final en les puntuacions, donant bons resultats tant en xarxes neuronals poc profundes com en xarxes neuronals amb una quantitat considerable de capes. Aquesta decisió és fàcilment justificable. L'activació *Softmax* com a capa d'activació final, com hem vist unes línies més amunt, dona una sortida molt intuïtiva i fàcil d'interpretar, convertint-la en la ideal. En quant a les ReLu, presenten tres avantatges respecte els altres tipus de funcions d'activació.

Primer, les ReLu, encara que presenten el problema d'eliminar certs valors que potser eren necessaris per a l'assignació d'una classe, amb una normalització de les dades d'entrada ja aconsegueixen eliminar només els valors que realment no interessen a la xarxa neuronal i no saturar els valors positius que s'obtenen. En canvi, les altres funcions d'activació necessiten més processament i no aconsegueixen eliminar del tot la saturació, donant un rendiment pitjor que les ReLu.

Segon, tenen la capacitat de posar totalment a 0 un dels valors que passen per la xarxa neuronal, sent de les poques que poden aconseguir un 0 vertader amb facilitat. Aquesta és una bona propietat si se'n fa un bon ús, ja que ajuda a que l'aprenentatge de la xarxa neuronal contingui menys càlculs i, per tant, sigui més eficient.

Tercer, el gradient d'una ReLu és 1 si el valor és major que 0 i 0 si el valor és negatiu. Per tant, ens soluciona el problema que he mencionat uns paràgrafs més amunt sobre l'extinció del gradient i a més facilita molt els càlculs per a l'entrenament respecte a les seves homòlogues.

2.4 Entrenament d'una xarxa neuronal

En aquesta secció tractarem com minimitzar la funció de pèrdua actualitzant els paràmetres de la xarxa neuronal. Per a fer-ho, introduirem el procés de *backpropagation* [20], [14], ([10],capítol 8), ([28], lectura 4) per a calcular els gradients al llarg de la xarxa neuronal. Utilitzarem aquests gradients per a actualitzar els pesos de la xarxa neuronal i així reduir

el valor de la funció de pèrdua.

Comencem amb un exemple bàsic. Sigui $f : R^{nxm}$ x R^{nxm} x $R^{nxm} \rightarrow R^{nxm}$ tal que, donats $x, y, z \in R^{nxm}$, $f(x, y) = zx + y$. Deixem x fixat i prenem uns valors inicials de y, z. Volem veure per quins y, z la funció f(x,y,z) és mínima. Donat que la funció és un exemple i en el cas real no tindrem ni la capacitat ni la potència per derivar-la amb facilitat, procedirem a introduir el concepte de graf de computació. Donada una funció anomenarem graf de computació de la funció a la seva descomposició en funcions bàsiques: sumes, restes, logaritmes, exponencials, etcètera. Així, el graf de computació d'aquesta funció quedaria:

$$x, y, z \tag{13}$$

$$u(x, z) = zx \tag{14}$$

$$f(u, y) = u + y \tag{15}$$

Notem que al dividir la funció en elements bàsics es pot calcular amb facilitat la derivada de cada element respecte els seus paràmetres.

És convenient explicar una mica per què és interessant el graf computacional d'una funció. A la pràctica, f serà equivalent a la nostra funció de pèrdua. Suposem que tenim una funció de pèrdua $L = L(g(x, W_1, W_2, W_3, ..., W_n))$ on g és la nostra xarxa neuronal (obviem la funció de regulació associada a la funció de pèrdua, ja que en general la seva matriu jacobiana respecte els pesos serà fàcilment recuperable a posteriori). Si podem calcular les matrius jacobianes $\frac{\partial L}{\partial W_1}, \frac{\partial L}{\partial W_2}, ..., \frac{\partial L}{\partial W_n}$ evaluades en els valors $x, W_1, W_2, W_3, ..., W_n$ (i sumant les jacobianes de la funció de regulació respecte cada pes a aquests valors), podrem utilitzar-les per a actualitzar els pesos per a que L sigui menor i, per tant, millorar el rendiment de la xarxa neuronal. Notem que les matrius tindran les mateixes dimensions que $W_1, W_2, ..., W_n$, respectivament. El procediment serà

$$W_1 = W_1 - lr * \frac{\partial L}{\partial W_1} \tag{16}$$

$$W_2 = W_2 - lr * \frac{\partial L}{\partial W_2} \tag{17}$$

...

$$W_n = W_n - lr * \frac{\partial L}{\partial W_n} \tag{18}$$

on lr és un hiperparàmetre petit anomenat ràtio d'aprenentatge de la xarxa neuronal (*learning rate*, en anglès) del qual parlarem més endavant en la secció corresponent.

Així, el que volem és calcular aquestes matrius jacobianes. Al dividir la funció de pèrdua i les capes de la xarxa neuronal en funcions més simples podrem aconseguir els gradients desitjats utilitzant la regla de la cadena, com veurem en breus.

Tornem altre cop a l'exemple il·lustratiu. Tenim inicialitzats y i z a uns certs valors i fem un *forward pass* de x, obtenint $f_0 = f(x, y, z)$ i guardant els valors x_0, y_0, z_0 i u_0. Ara, utilitzem el graf computacional per arribar a les jacobianes desitjades $\frac{\partial f}{\partial y}$ i $\frac{\partial f}{\partial z}$ (*backward pass*). Tenim

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial f} * \frac{\partial f}{\partial y} = I * I = I \tag{19}$$

$$\frac{\partial f}{\partial u} = \frac{\partial f}{\partial f} * \frac{\partial f}{\partial u} = I * I = I \tag{20}$$

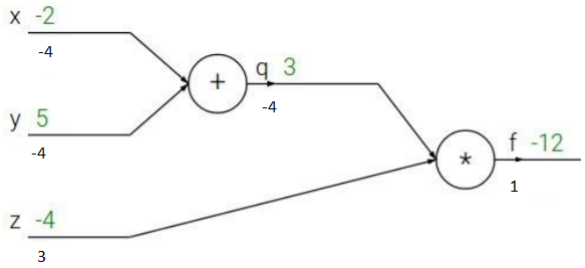
$$\frac{\partial f}{\partial z} = \frac{\partial f}{\partial u} * \frac{\partial f}{\partial u} = I * x = x \tag{21}$$

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial u} * \frac{\partial f}{\partial x} = I * z = z \tag{22}$$

on notem que x tindrà valor x_0, aconseguit en el *forward pass* i guardat per a ser utilitzat posteriorment (el mateix passa amb el valor de z). El gradient de x ha estat calculat a mode il·lustratiu, ja que com només volem minimitzar la funció modificant

els valors y i z no és necessari, tot i que veurem que s'utilitza en alguns algorismes d'explicabilitat.

Durant el *forward pass* es guarden totes les variables intermèdies generades pel graf, així com els valors inicials dels paràmetres d'entrada i el valor de la sortida, per si s'han d'utilitzar en el *backward pass*.



Imatge 4: Extreta de [28], lectura 4 i adaptada. Graf de la funció $f(x, y, z) = (x + y) * z$ amb els valors del *forward pass* damunt i els gradients del *backward pass* sota les fletxes.

Un cop obtingudes les jacobianes, podem realitzar l'actualització dels paràmetres com hem vist unes línies més amunt. Amb això queda vist un cicle complet de l'entrenament de la xarxa neuronal. Es pot veure a la imatge 4 un altre exemple, on es calculen per *backpropagation* tots els gradients.

Una xarxa neuronal utilitza una gran quantitat d'imatges per a optimitzar els seus paràmetres. El procediment per a entrenar la xarxa neuronal consisteix en passar com a paràmetres de la xarxa neuronal les imatges, realitzar el *forward pass* guardant en variables tots els càlculs intermedis, calcular la funció de pèrdua, realitzar el *backward pass* per a obtenir les jacobianes d'aquesta respecte els pesos i biaixos i actualitzar-los. Usualment, en comptes de passar totes les imatges a la vegada per fer una actualització es fan agrupacions d'imatges, anomenats *image batches* i s'entrena la xarxa neuronal per cada *batch*. Quan totes les imatges han passat per la xarxa neuronal un cop, es diu que hem avançat una època o *epoch* de l'entrenament.

Un entrenament complet consta de varies èpoques o passades completes de la col·lecció d'imatges. Al final de cada època és usual i interessant saber quin és el valor de la funció de pèrdua i quin percentatge d'encert té la xarxa neuronal en aquell moment, tal i com es pot veure a la imatge 5.

```
Epoch 00004: LearningRateScheduler setting learning rate to 0.018.
2250/2250 [==============================] - 31s 14ms/step - loss: 0.2747 - acc: 0.9080 - val_loss: 0.4062 - val_acc: 0.9027
```

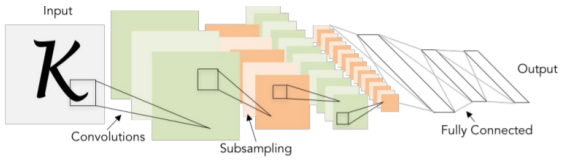
Imatge 5: Captura d'un entrenament d'una xarxa neuronal, època 4, on podem veure la pèrdua i la precisió de l'entrenament i les seves anàlogues de validació. La quantitat de dades d'entrenament és de 2250.

Tant la mida dels *image batch* com el número d'èpoques són hiperparàmetres que veurem en la secció corresponent.

2.5 Xarxes neuronals convolucionals

En aquesta secció veurem tots els termes relacionats amb aquests tipus de xarxes neuronals, que són les que utilitzarem a la part de desenvolupament. Donarem la definició de xarxa neuronal convolucional i parlarem dels tipus de capes que podem trobar en aquesta xarxa neuronal i el seu funcionament.

2.5.1 Definició



Imatge 6: Extreta de [28], lectura 5. Estructura bàsica d'una xarxa neuronal convolucional.

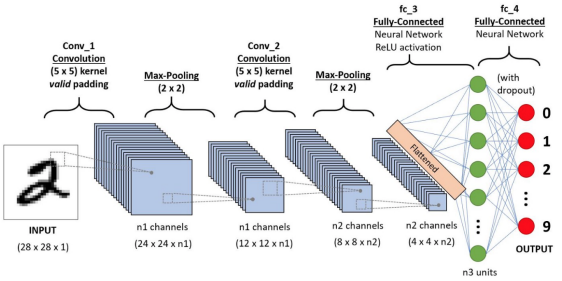
Una xarxa neuronal convolucional es diferencia dels altres tipus de xarxes neuronals per contenir capes convolucionals. Més endavant en aquesta secció veurem el funcionament d'aquestes, però en essència el que fan és aplicar una convolució a l'*input*. Per tant, una xarxa neuronal convolucional consistirà en una sèrie de capes convolucionals, intercalades amb capes d'activació i de *pooling* i, finalment, una sèrie de capes denses (també amb les seves activacions) encarregades de donar la puntuació final de la imatge respecte cada classe. Un esquema bàsic és el que es pot trobar a la imatge 6.

Les xarxes neuronals convolucionals permeten una millor precisió al classificar imatges respecte les xarxes neuronals convencionals, ja que les capes convolucionals permeten extreure les característiques de la imatge tractada i les capes denses utilitzaran aquestes característiques per a calcular la puntuació de cada classe. A més, les capes convolucionals contenen relativament menys paràmetres, fent que entrenar sigui més ràpid i eficient.

2.5.2 Capes d'una xarxa neuronal convolucional

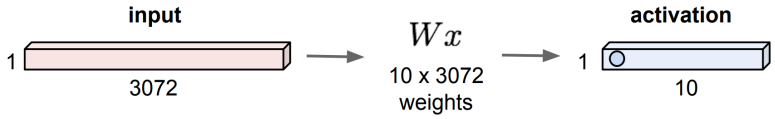
En aquest apartat descriurem els tipus de capes més útils i usuals que poden formar una xarxa neuronal convolucional ([10], capítol 9), [24], ([28], lectura 5). Podem veure un

esquema de la composició per capes d'una xarxa neuronal a la imatge 7.



Imatge 7: Extreta de [24], Capes d'una xarxa neuronal convolucional.

La primera capa que comentarem és la capa d'activació, ja vista en la secció de xarxes neuronals. La capa d'activació és una capa que aplica un canvi sobre les dades de la xarxa neuronal, afegint no-linealitat de la mateixa forma i amb les mateixes conseqüències que en el cas de les xarxes neuronals. Aquí, les més utilitzades altre cop són les ReLu, després de les capes convolucionals, i l'activació *Softmax* a l'última capa densa o *fully-connected* que té com a *output* les puntuacions de la xarxa neuronal.



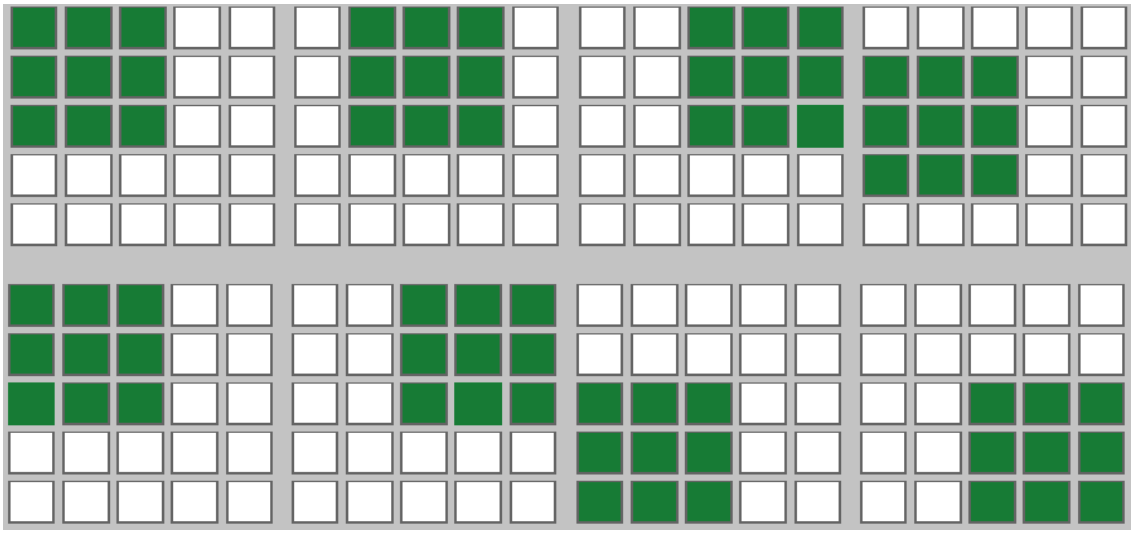
Imatge 8: Extreta de [28]. Aplicació d'una capa densa a una entrada per a obtenir una sortida de 10 puntuacions.

Una capa densa o *fully-connected* és una capa que té associada una matriu de pesos W que s'aplica a l'entrada x, la qual té dimensions de vector (s'aplica un *flattening* abans d'aquestes capes, com es pot veure a la imatge 7, a l'inici de la secció). Els pesos d'W són valors entrenables de la xarxa neuronal. Aquest tipus de capa té com a funció l'obtenció de les puntuacions de les classes. Podem veure el seu funcionament a la imatge 8.

Una capa convolucional té associada una sèrie de filtres, en general més petits en dimensions que l'entrada x. Cada filtre consta d'una matriu de pesos W i un biaix b. El filtre s'aplica a la imatge pas a pas, utilitzant un valor anomenat pas o *stride*. Per exemple, suposem que tenim un *input* de la capa de $5x5$ i un filtre de $3x3$. Començaríem aplicant el filtre per dalt a l'esquerra, i obtindríem els 4 primers passos mostrats a la imatge 9 , segons si posem pas 1 o 2.

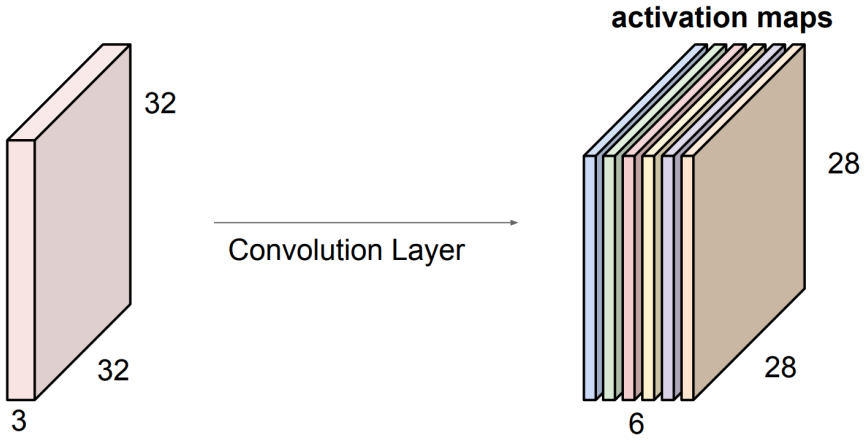
A cada pas del filtre, s'aplica el producte de Hadamard, (valor a valor) entre els elements del filtre i els seleccionats de l'*input*, es sumen tots els resultats i s'hi afegeix el biaix, de forma que es genera un element de la sortida:

$$Y = \sum_j W_j * x_j + b \tag{23}$$



Imatge 9: Creació pròpia. A la primera fila, les 4 primeres aplicacions d'un filtre 3x3 a una entrada 7x7 amb pas 1. A la segona fila, el mateix amb pas 2.

Quan un filtre ha passat per tota la imatge, posem els resultats en forma de matriu i aquesta rep el nom de mapa d'activació (*activation map*, en anglès) de x pel filtre i. Per tant, una capa convolucional amb 5 filtres de $5x5x3$ crearà 5 mapes d'activació, que passaran com a entrada de la següent capa de la xarxa neuronal. Podem veure la creació dels mapes d'activació al passar per una capa convolucional a la imatge 10.



Imatge 10: Extreta de [28]. Podem veure com un input de 32x32 passa per una capa convolucional amb 6 filtres de 5x5, generant 6 mapes d'activació.

Notem que, depenent del pas elegit, el filtre no podrà cabre correctament per a tal de recórrer tot x. El que es fa llavors és aplicar un 0-*padding*: s'afegeixen zeros a les vores de x per tal que les dimensions siguin compatibles.

Les dimensions del mapa d'activació resultant d'aplicar un filtre n x n x m a un *input* de dimensions k x k x m amb un pas p i un 0-*padding* de f són v x v on

$$v = (k + 2f - n)/p + 1 \tag{24}$$

Així, aplicant tres filtres de $5x5x3$ a un *input* $20x20x3$ obtindrem tres mapes d'activació de dimensions v x v amb $v = (20 + 2 * 0 - 5)/1 + 1 = 16$.

Les dimensions de la sortida són menors que les de l'entrada (si s'utilitzen filtres de dimensió més gran que 1), reduint els paràmetres necessaris per a les capes posteriors i, per tant, millorant la eficiència de la xarxa neuronal.

La capa convolucional és la capa encarregada d'extreure característiques de la imatge per a que les capes denses puguin assignar puntuacions a partir d'aquestes.

La capa de batchNormalization [13], ([28], lectura 7) s'utilitza en certes situacions entre les capes convolucionals o *fully-connected* i les d'activació. Aquesta capa pren la sortida de l'anterior i aplica una normalització a les dades per tal d'evitar la pèrdua de valors importants al passar per la capa d'activació. En concret, aquesta capa aconsegueix que les columnes de la seva entrada tinguin variància unitària i 0 de mitjana.
Aplica, per cada columna x,

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x_i \tag{25}$$

$$\sigma^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu)^2 \tag{26}$$

$$\hat{x_i} = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}} \tag{27}$$

$$y_i = \gamma\hat{x_i} + \beta \tag{28}$$

on a (25) es realitza la mitjana, a (26) es calcula la variància, a (27) aplica un ϵ petit per guanyar estabilitat numèrica i a l'última equació s'utilitzen dos paràmetres entrenables que afegeixen marge per optimitzar aquesta normalització si millora els resultats de la xarxa neuronal. Aquests paràmetres són diferents per a cada columna tractada.

Les equacions utilitzades són fàcilment derivables [18] i, per tant, es pot utilitzar *backpropagation* per a actualitzar aquests dos paràmetres i propagar el gradient cap a les capes superiors. Cal recordar que l'entrenament es fa per grups d'imatges i, per tant, al fer *backpropagation* necessitarem uns valors genèrics de μ i σ. Per això, la capa manté una mitjana dels valors del *batch* d'imatges que passa per la capa durant el *forward pass* i s'utilitza aquest valor en el *backward pass*.

La capa de *pooling* es sol situar després de la capa d'activació i abans de la següent capa convolucional. El seu principal objectiu és reduir la mida de les dades per a poder entrenar més fàcilment la xarxa neuronal sense perdre massa precisió. La més famosa d'aquest tipus és el *max pooling*, que donat un pas i dimensions del filtre, segueix el mateix procediment que una capa convolucional i fa el màxim de la part on s'està aplicant el filtre, reduint així la mida de les dades com es desitja.

L'última capa a comentar és la de *dropout* [5]. Aquesta és una capa molt circumstancial. Donat un hiperparàmetre α, impedeix que algunes de les dades que es passen per la xarxa neuronal no s'utilitzin per a l'aprenentatge d'aquesta. Per exemple, amb $\alpha = 0.3$, s'apagaran un 30% de les neurones de la xarxa neuronal al passar per la capa. Cal notar que les que s'apaguen al passar pel *dropout* en cada època de l'entrenament no són les mateixes, però sí que dins una mateixa època, tant al *forward pass* com al *backward pass*, es desactiven les mateixes. Utilitzar aquesta capa serà una de les possibles solucions per al problema d'*Overfitting*, com veurem més endavant.

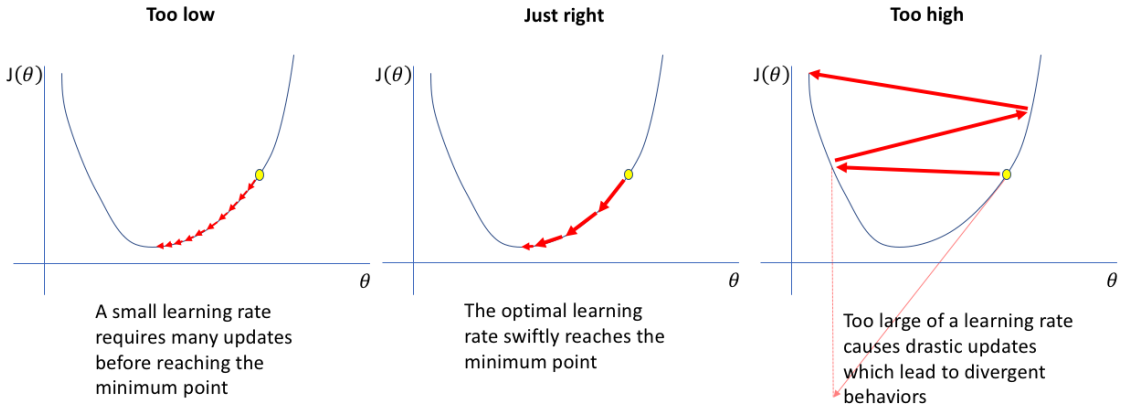
Durant l'entrenament, la capa de *dropout* actua diferent segons si estem entrenant o estem validant. Al validar no hem de posar les neurones a 0, només hem d'escalar per α les dades. És usual utilitzar el *dropout* invers en comptes del normal, on durant l'entrenament s'escala per $\frac{1}{\alpha}$ les dades a més d'apagar les pertinents i, per tant, a la validació ja no fa falta fer res.

2.6 Hiperparàmetres

Aquesta és una de les 3 seccions que contenen els termes teòrics restants que hem anat mencionant al llarg de tots els conceptes estudiats. Els hiperparàmetres d'una xarxa neuronal ([28], lectura 8) són paràmetres que tenim a la nostra disposició per a posar en un valor adequat al nostre problema. Un cop assignats, aquests paràmetres no poden ser modificats altre cop durant l'entrenament, exceptuant alguns casos especials. En aquest apartat explicarem els més rellevants i utilitzats.

L'hiperparàmetre λ, que controla com afecta la regulació a la funció de pèrdua és un paràmetre entre 0 i 1, generalment al voltant de 0.1, encara que és usual utilitzar-lo amb valors de l'ordre de 0.01. És un paràmetre necessari, ja que volem que el pes de la funció de pèrdua provingui del propi anàlisi de les puntuacions i només s'utilitzi el valor de la regulació per a decidir entre pèrdues molt similars i, per tant, no introdueixi un error prou gran com per a fer que la funció de pèrdua deixi de poder-se utilitzar per a comparar quins paràmetres de la xarxa neuronal donen millors resultats.

El paràmetre de *learning rate*, lr, és probablement el més important d'una xarxa neuronal (igualat només pel *batch size* i pel total d'èpoques). És el que s'utilitza per a realitzar l'actualització dels paràmetres a les equacions (13.x) (o, equivalentment, en l'optimitzador que decidim utilitzar, com veurem més endavant) i l'elecció d'aquest paràmetre té unes grans conseqüències sobre la precisió de la xarxa neuronal.



Imatge 11: Extreta de `https://www.jeremyjordan.me/content/images/2018/02/Screen-Shot-2018-02-24-at-11.47.09-AM.png`. Casos segons el learning rate.

D'una banda, un *learning rate* elevat pot ajudar a convergir més ràpid, si és massa

gran, però, no aconseguirà arribar al mínim de la funció de pèrdua ja que els "passos" seran massa grans, com il·lustra la imatge 11. D'altra banda, un *learning rate* massa baix podria comportar que entrenar fos computacionalment poc pràctic. Un valor adient per al *learning rate* és 0.01; en funció del problema, però, pot funcionar bé fins a 0.1. És important analitzar bé el problema a tractar ja que la seva complexitat i el seu tipus influiran molt en el valor d'aquest paràmetre.

Hi ha molts mètodes implementats per a poder modificar de forma dinàmica aquest paràmetre segons el valor de la funció de pèrdua o del número d'èpoques que hem entrenat fins aquell moment, fent que al començar a entrenar es pugui posar un valor "elevat" i que es vagi reduint amb el temps, maximitzant la precisió que es pot arribar a obtenir alhora que es manté una convergència ràpida.

El paràmetre que marca el total d'èpoques indica quants cops la xarxa neuronal veurà la col·lecció de dades durant el seu entrenament. Un cop han passat totes les dades de nou es considera una nova època. Quants cops han de passar les dades per la xarxa neuronal és una decisió de considerable importància i de variables resultats. Per exemple, en un problema amb poques classes o bé si partim d'un model ja entrenat que només hem d'adaptar, amb poques èpoques podem obtenir uns resultats suficientment bons.

No hi ha un nombre d'èpoques ideal. Com a idea general, el procediment és entrenar durant el temps que tinguem disponible, mentre la funció de pèrdua no sigui prou baixa (o la precisió no sigui prou alta). Hi ha mètodes implementats per aturar l'entrenament si la funció de pèrdua no millora durant un cert número d'èpoques.

Com hem comentat anteriorment, l'actualització dels paràmetres de la xarxa neuronal durant l'entrenament es realitza tenint en compte entrades de grups d'imatges per a fer més eficient l'entrenament en comptes d'actualitzar per cada imatge passada. Això s'aconsegueix utilitzant l'hiperparàmetre de *batch size*, per indicar el volum dels grups. No hi ha una condició ideal d'aquest paràmetre, però és preferible que

$$N\%bs = 0 \tag{29}$$

on N és el nombre total d'imatges a l'entrenar, bs és el *batch size* i % indica el residu de la divisió.

Si el residu no és 0, l'últim *batch* passarà incomplet. Tot i que no suposa un gran problema, aquesta última passada contindrà menys imatges que les anteriors, creant una desigualtat.

El ràtio de *dropout* és l'hiperparàmetre de les capes de *dropout*. Indica quin percentatge de les neurones s'apagarà al passar per aquesta capa. Un bon valor per defecte és 0.5, indicant que al passar per la capa només tindran valor la meitat de les neurones de l'entrada.

Les capes que utilitzen aquest hiperparàmetre són molt circumstancials, fent que no siguin necessàries en molts casos. La situació més usual on aplicar *dropout* es dona quan la xarxa neuronal entrena molt bé sobre les dades d'entrenament però no és capaç de generalitzar a les dades de validació, com veurem més endavant quan parlem d'*Overfitting*.

Per acabar aquesta secció, tot i no ser pròpiament hiperparàmetres, comentarem una mica a gran escala com inicialitzar de forma eficient els pesos d'una xarxa neuronal [8]. Hom podria pensar que generar-los de forma aleatòria utilitzant, per exemple, una gaussiana de mitjana 0 i desviació molt baixa seria suficient. No obstant, en xarxes neuronals amb moltes capes el gradient en les últimes pràcticament desapareix o es dispara si la desviació és massa gran.

El problema resideix en que la variància de la sortida d'una neurona inicialitzada de forma aleatòria creix segons el nombre d'*inputs* de la neurona. Per exemple, donada una neurona d'una capa densa es té, sense l'activació:

$$s = \sum_{i=1}^{n} W_i x_i \tag{30}$$

$$Var(s) = Var(\sum_{i=1}^{n} W_i x_i) = \tag{31}$$

$$= \sum_{i=1}^{n} E(W_i)^2 * Var(x_i) + E(x_i)^2 * Var(W_i) + Var(W_i) * Var(x_i) = \tag{32}$$

$$= \sum_{i=1}^{n} Var(W_i) * Var(x_i) = \tag{33}$$

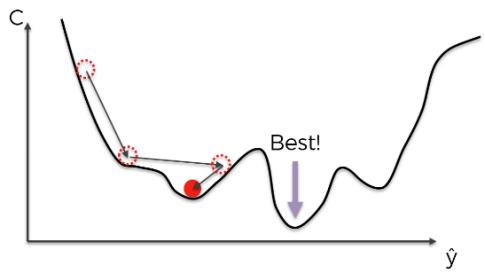
$$= n * Var(w) * Var(x) \tag{34}$$

on hem assumit dades d'entrada i pesos amb mitjana 0 i equitativament distribuïts i hem utilitzat propietats de la variància.

La inicialització de Xavier [9] diu que inicialitzant els pesos amb una distribució aleatòria i escalant-los per $\frac{1}{\sqrt{n}}$, on n és el nombre d'*inputs* es pot aconseguir una variància unitària en la sortida, fent que hi hagi gradient tractable a totes les capes de la xarxa i, per tant, es pugui entrenar de forma satisfactòria.

A [17] podem veure un anàlisi semblant que porta a la forma d'inicialitzar els pesos en cas de que hi hagi capes d'activació ReLu, fet que es dona sovint. L'anomenada inicialització de Kaiming per a xarxes neuronals amb capes ReLu proposa inicialitzar els pesos de forma aleatòria escalant-los per $\frac{2}{\sqrt{n}}$, on n és la quantitat de dades d'entrada. Aquesta inicialització és la utilitzada actualment i de forma general en aquests casos.

2.7 Optimitzadors



Imatge 12: Extreta de `https://miro.medium.com/max/1146/1*vwICNxW03oZxdFsXntfg2g.png`. Problema de l'SGD.

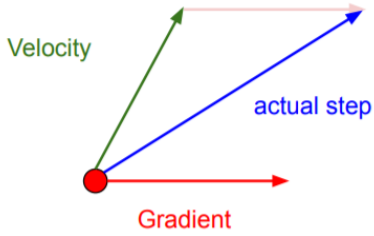
En aquest apartat parlarem sobre els optimitzadors [19], ([28], lectura 8), utilitzats per a substituir l'actualització dels paràmetres vista anteriorment i que busquen solucionar els seus múltiples problemes. Primer, convé tornar a fer una ullada a les equacions 16-18. Aquesta actualització és la més bàsica i rep el nom d'SGD (*Stochastic Gradient Descent*). Notem que, si trobem un mínim local utilitzant aquestes equacions, és improbable sortir-ne i, per tant, perdem la possibilitat de millorar la precisió de la nostra xarxa neuronal. Un exemple bàsic d'aquest problema es pot trobar a la imatge 12.

La següent millora aconsegueix superar aquest problema. Consisteix en afegir una velocitat a l'algorisme, de tal forma que si la xarxa neuronal es troba en un mínim local o en un lloc amb poc gradient en pugui sortir per a seguir minimitzant la pèrdua. Aquest

algorisme rep el nom de SGD+*momentum* i les seves equacions serien, de forma intuïtiva

$$v = \rho * v - lr * \frac{\partial L}{\partial W} \tag{35}$$

$$W = W + v \tag{36}$$



Imatge 13: Extreta de [28], lectura 8. Esquema de funcionament de SGD+*momentum*.

on a la primera època $v = 0$ i ρ és el fregament i té valors propers a 1. A la imatge 13 es pot veure el funcionament d'aquest mètode.

El següent optimitzador permet adaptar el *learning rate* en funció de la suma dels gradients obtinguts fins ara, fent-lo més petit a mesura que iterem. Aquest algorisme s'anomena Adagrad, però presenta poc control sobre com decaurà en el temps el ràtio d'aprenentatge. Les equacions que maneja són les següents:

$$ag = ag + (\frac{\partial L}{\partial W})^2 \tag{37}$$

$$W = W - lr * \frac{\frac{\partial L}{\partial W}}{\sqrt{ag} + \epsilon} \tag{38}$$

on ag, el gradient acumulat, és inicialment 0 i ϵ s'utilitza per a donar estabilitat numèrica. Notem com, a mesura que augmentem les èpoques i, per tant, augmenta ag, disminueix el *learning rate*.

L'algorisme RMSProp millora l'Adagrad introduint un ràtio de caiguda a les equacions, permetent un major control sobre el descens del ràtio d'aprenentatge:

$$ag = dc * ag + (1 - dc) * (\frac{\partial L}{\partial W})^2 \tag{39}$$

$$W = W - lr * \frac{\frac{\partial L}{\partial W}}{\sqrt{ag} + \epsilon} \tag{40}$$

on *dc* és el ràtio de caiguda (*decay rate*) entre 0 i 1.

Finalment, l'últim algorisme a presentar és l'aplicació de l'RMSProp juntament amb el factor de velocitat vist unes línies més amunt. Rep el nom d'Adam i és l'algorisme per defecte quan es vol entrenar una xarxa neuronal. Tot i així, val la pena provar els altres, ja que poden donar millors resultats en funció del problema. Adam ve donat per les següents equacions:

$$fm = \beta 1 * fm + (1 - \beta 1) * (\frac{\partial L}{\partial W}) \tag{41}$$

$$sm = \beta 2 * sm + (1 - \beta 1) * (\frac{\partial L}{\partial W})^2 \tag{42}$$

$$uf = \frac{fm}{1 - \beta 1^{\#epochs}} \tag{43}$$

$$us = \frac{sm}{1 - \beta 2^{\#epochs}} \tag{44}$$

$$W = W - lr * \frac{uf}{\sqrt{us} + \epsilon} \tag{45}$$

on $\beta 1$ i $\beta 2$ són hiperparàmetres generalment propers a 1, fm i sm (*first momentum* i *second momentum*) són inicialment 0 i aporten una velocitat i un *decay rate* respectivament; i $\#epoch$ indica el número d'època actual. Les equacions (43) i (44) s'utilitzen per a compensar que l'inicialització de fm i sm és 0, donant estabilitat numèrica durant les primeres èpoques.

2.8 Problemes a l'entrenar una xarxa neuronal

Aquesta és l'última secció abans de concloure el mapatge teòric de les xarxes neuronals d'aquesta primera part de l'estat de l'art. Un cop acabada, podrem començar a tractar els algorismes d'explicabilitat a la segona part.

Les causes per les quals una xarxa neuronal no classifica prou bé poden ser diverses [15]. Els principals problemes que combatem quan entrenem una xarxa neuronal tenen a veure amb el gradient (*vanishing/exploding gradients*) o la precisió resultant (*Overfitting/Underfitting*).

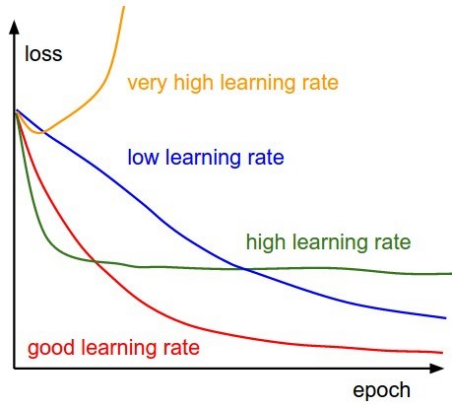
Pel que fa als problemes de gradient, és problemàtic tant si s'obtenen gradients molt grans com gradients molt petits. Els valors grans són obtinguts al propagar el gradient durant el *backward pass* per capes amb el gradient més gran que 1, fent que creixi. Si el gradient augmenta molt deixa de ser útil per a realitzar l'actualització de paràmetres i fins i tot es poden donar casos en el que l'algorisme no el pugui comptabilitzar i es posi a NaN (*not a number*). Aquest problema rep el nom d'*exploding gradient*. Passa el mateix si el gradient és molt petit. Un gradient que es propaga a través de capes amb gradient menor que 1 es va reduint, fent que al final l'actualització dels paràmetres sigui ínfima i no s'aconsegueixi entrenar la xarxa neuronal. Anàlogament al cas anterior, el gradient pot arribar fins i tot a desaparèixer del tot en xarxes amb moltes capes. Aquest segon problema rep el nom tècnic de *vanishing gradient*.

En l'actualitat és molt difícil trobar-se en la situació d'haver de fer front a aquests problemes, ja que la majoria de models estan preparats per a solucionar-los. Tot i així, les recomanacions per a evitar dificultats relacionades amb la propagació del gradient són:

- Inicialitzar de forma adequada els paràmetres segons el tipus de capa, és a dir, aconseguir que les neurones tinguin una variància el més pròxim possible a la unitària;
- Utilitzar capes d'activació ReLu, ja que al passar el gradient durant la *backpropagation* el multipliquen per 1 o per 0, evitant augmentar-lo o disminuir-lo;
- Utilitzar regulació a la funció de pèrdua, permetent prioritzar pesos baixos que evitaran un gradient massa elevat;
- Controlar el nombre de capes de la xarxa per a evitar un *backward pass* massa llarg on puguin produir-se càlculs inestables.

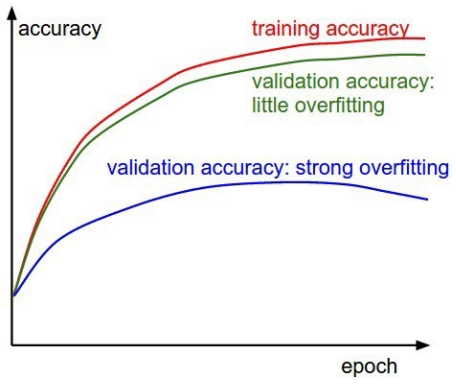
En quant als problemes de precisió, el seu origen pot ser molt variat. El primer que s'ha de tenir en compte per trobar-ne les causes és la gràfica de la funció de pèrdua al llarg de les èpoques. Per a un entrenament satisfactori, la funció de pèrdua ha de disminuir

moltíssim les primeres èpoques i després seguir decreixent però a menor ritme. Que no decreixi prou ràpid al principi podria significar que es té un *learning rate* massa baix pel problema que s'està intentant resoldre. Si decreix massa ràpid després del començament de l'entrenament, potser s'hauria de configurar el ràtio de caiguda de l'optimitzador o del *callback* per a que sigui més pronunciat. La imatge 14 mostra de forma simple com hauria de comportar-se la funció de pèrdua al llarg dels *epochs* i com s'hauria de modificar el *learning rate* per a que la funció de pèrdua evolucioni de forma favorable.



Imatge 14: Extreta de [15]. Gràfic de les funcions de pèrdua segons els problemes amb el learning rate.

Si la funció de pèrdua evoluciona correctament i tot i així la xarxa neuronal no aprèn de la forma esperada es podria donar el cas que les dimensions de la xarxa siguin massa petites o massa grans per al problema que estem intentant resoldre. Per tant, si es té una gran quantitat de classes o un problema complex és important considerar augmentar tant les capes convolucionals per a poder diferenciar millor les característiques entre les classes com les denses per a poder utilitzar tota la informació que haurem obtingut a les convolucionals. En general, però, no és una bona idea anar afegint capes fins a l'infinit, ja que llavors la xarxa perd la capacitat de generalitzar la classificació a noves imatges.



Imatge 15: Extreta de [15]. Diagnosi d'overfitting segons les precisions de la xarxa neuronal.

Si manipulant una mica la xarxa neuronal no s'aconsegueixen millores en els resultats és molt probable que es tingui un problema o bé d'*Overfitting* o bé d'*Underfitting*. Per a detectar visualment un d'aquests dos problemes, es pot analitzar el gràfic de la precisió de validació respecte la d'entrenament al llarg de les èpoques, com s'indica a la imatge 15: si la precisió de validació és molt més baixa que la de l'entrenament, tindrem el primer cas; si la precisió de l'entrenament no millora correctament, tindrem el segon.

Es possible que la xarxa presenti un problema

del primer tipus si durant l'entrenament la precisió en les dades d'entrenament millora però no s'aconsegueix passar aquest coneixement per a classificar les dades de validació. Aquest comportament és deu a que la xarxa neuronal aprèn massa sobre les imatges i, per tant, es queda amb característiques que molt probablement no siguin inherents a aquella classe.

En canvi, es considera que es té un problema del segon tipus quan no s'aconsegueix aprendre la classificació de l'entrenament ja que o bé les nostres dades o bé la nostra xarxa són insuficients.

Una manera d'evitar problemes de precisió és processar les imatges abans de passar-les per la xarxa. . El processament de les imatges pot consistir en normalitzar-les o escalar-les, en restar la mitjana global de totes les imatges a cada píxel per cada canal RGB, etcètera. Això ajuda a la xarxa a rebre dades normalitzades que podrà reconèixer millor.

També és útil barrejar les imatges abans de passar-les per la xarxa, de tal forma que a cada *batch* hi hagi imatges de diferents classes per a entrenar.En canvi, si es passen les imatges ordenades per classes, la xarxa neuronal actualitzarà primer els pesos per a classificar les primeres classes que vegi i després "els adaptarà" per a les següents, fent que perdi precisió en les darreres.

Per al problema d'*Underfitting*, una de les possibles solucions és veure si es pot augmentar el nombre de capes de la xarxa neuronal. Si es considera que l'estructura de la xarxa és ja adequada al problema, s'hauria d'intentar augmentar el nombre d'imatges per a entrenar de cada classe, ja que és probable que la xarxa no tingui prou informació sobre cada grup d'imatges per a classificar nous elements. L'augment es pot realitzar tant aconseguint noves imatges de cada classe com realitzant *data augmentation*. El procés de *data augmentation* permet fer reflexions, focus, retalls, rotacions... sobre un conjunt d'imatges per a generar-ne de "noves". És una eina molt popular i àmpliament utilitzada, ja que fins i tot està integrada dins de la llibreria Keras.

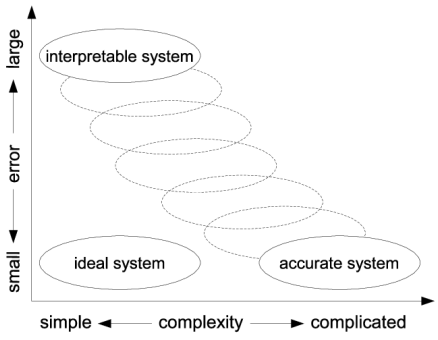
En quant a l'*Overfitting*, es pot aplicar *batch normalization* (encara que és recomanat tenir-ne ja per defecte) i *dropout*. Les capes de *dropout* donen com a resultat que a cada època la xarxa neuronal actua diferent i considera unes certes característiques de cada classe i pesos de les capes per les quals passen. Això provoca una petita reducció de la precisió de la xarxa però ajudarà a que les característiques no inherents es deixin

d'utilitzar i, per tant, l'aprenentatge millori la precisió en la validació.

També és possible que aquest problema sigui causat per una xarxa neuronal amb massa paràmetres, però ajustar el nombre de capes al problema sempre és complicat i no sempre és una opció viable. Es pot intentar ajustar el *learning rate* per a donar passos més suaus i augmentar el temps d'entrenament per a facilitar l'aplicació del que s'aprèn a la validació. També seria prudent augmentar la quantitat de dades que té la xarxa neuronal per a entrenar, com en el problema d'*Underfitting*.

Una xarxa sobreentrenada també pot caure en *Overfitting* a partir d'una certa època. Per a evitar-ho, cal reduir el nombre total d'*epochs* de la xarxa neuronal o utilitzar un *callback* per a aturar l'entrenament quan la funció de pèrdua ja no millori.

Per acabar aquesta secció, comentarem un problema que ens sorgeix a l'intentar explicar les decisions que pren la xarxa neuronal. Al mirar els càlculs d'una xarxa neuronal molt simple i amb poques capes, o fins i tot d'un classificador lineal, podem veure fàcilment i de forma intuïtiva com s'ha pres una decisió.



Imatge 16: Extreta de `https://www.researchgate.net/profile/Geok_Ng/publication/327203363/-accuracy-and-interpretability.png`. Explicabilitat vs precisió.

No obstant, les xarxes neuronals amb poques capes no són capaces d'assolir una alta precisió al resoldre un problema. Això és a causa de que, com hem vist, no són capaces d'aproximar prou bé la funció de classificació buscada: necessiten més llibertat de paràmetres. Per altra banda, les xarxes complexes poden assolir molt millor una precisió elevada, però explicar les seves decisions es converteix en una odissea ja que les contribucions de cada part de la imatge i cada paràmetre es perden en el mar dels càlculs. Per això, s'ha de buscar un balanç entre precisió i explicabilitat, de forma que es pugui aconseguir un bon resultat en ambdues característiques, com es pot veure a la imatge 16.

2.9 Algorismes d'explicabilitat

Amb l'última secció ja hem acabat d'exposar tot el material teòric pertanyent a les xarxes neuronals, a la seva construcció i al seu entrenament satisfactori. En aquesta última part sintetitzarem els algorismes d'explicabilitat de xarxes neuronals convolucionals més coneguts, els quals provarem a la part pràctica del treball.

En aquest apartat farem èmfasi a tot el context que necessitem per aplicar aquests algorismes a la decisió de classificació presa per la xarxa neuronal respecte una imatge.

Durant tota aquesta secció considerarem que ja tenim una xarxa neuronal entrenada per a classificar en N classes i que disposem d'una imatge per a classificar. Per tant, podem realitzar tant el *forward pass* de la imatge per a obtenir les puntuacions d'aquesta com el *backward pass* per a obtenir els gradients.

L'objectiu d'aquests algorismes és mostrar visualment en quines àrees o característiques de la imatge es fixa la xarxa neuronal per a prendre una decisió. Per tant, la sortida dels algorismes serà una imatge com la d'entrada, però amb les regions importants marcades. És el que s'anomena un *heatmap*.

A continuació enumerarem els més rellevants, els quals veurem en detall a l'apartat d'implementació.

El primer algorisme és el CAM. Va ser publicat el 2015 a [3] i ha estat millorat al llarg del temps amb versions com el Grad-CAM i posteriors, les quals també treballarem. Tot i no ser un algorisme versàtil, val la pena incloure'l ja que és el pare de tots els algorismes CAM.

L'algorisme consisteix en generar el que s'anomena un *Class Activation Map*, o CAM, a partir d'introduir una capa GAP (*Global Average Pooling*) entre les capes convolucionals i la capa densa de puntuació. Aquesta capa fa la mitjana per cada profunditat de l'entrada, de forma que una entrada m x n x k queda reduïda a un vector de k dimensions. Amb això s'aconsegueixen les importàncies dels *activation map* i es poden utilitzar per a generar la suma ponderada dels mapes d'activació.

L'algorisme de Guided Backpropagation [16] va ser ideat el 2014. Utilitza el mètode de backpropagation descrit anteriorment per a aconseguir el gradient d'un dels mapes

d'activació d'una capa convolucional (normalment l'última) respecte la imatge d'entrada, permetent veure les característiques importants apreses d'aquesta imatge per la capa convolucional en molt bon detall, a diferència de CAM que dona només la importància de les zones. No obstant, aquest algorisme no és capaç d'identificar les característiques importants d'una classe concreta.

L'algorisme Grad-CAM és una millora de l'anterior CAM, publicada a [22] el 2016. Igual que Guided Backpropagation, Grad-CAM és un algorisme orientat al càlcul del gradient. Utilitza *backpropagation* per a obtenir els gradients de la puntuació de la classe c abans de l'activació respecte cada *activation map* d'una capa convolucional per a crear les importàncies de cada mapa d'activació i generar el *heatmap* fent una suma ponderada. Es pot veure que és una generalització de CAM.

L'algorisme Guided Grad-CAM, inclòs en [22], és el producte escalar del *heatmap* generat per Grad-CAM amb la sortida de Guided Backpropagation, fent que és generi una imatge amb les característiques importants marcades i alhora s'obtingui el detall d'aquestes.

L'algorisme Grad-CAM++, publicat a [1] el 2017 és el més recent entre els que utilitzen el gradient per a donar l'explicabilitat de la xarxa neuronal. Grad-CAM presenta dos principals problemes que s'intenten adreçar en aquesta nova iteració: la incapacitat de detectar múltiples ocurrències d'un objecte pertanyent a una classe en una imatge i el fet de que els *heatmaps* generats per aquest mètode cobreixen petites parts de l'objecte que volen remarcar, fent que perdin fiabilitat.

Aquest algorisme utilitza una reformulació de la importància de cada mapa d'activació de la capa convolucional fent que si hi ha parts d'un objecte o múltiples ocurrències que activen diferents mapes d'activació no desapareguin del tot les que s'activen amb menys força i, per tant, totes les característiques tindran representació en el *heatmap* final. De forma anàloga a Grad-CAM, és possible crear Guided Grad-CAM++ fusionant la sortida de Grad-CAM++ amb Guided Backpropagation.

A juny de 2020 podem dir que l'Score-CAM [11] és un algorisme molt recent, ja que l'última iteració és de l'abril de 2020. Aquest algorisme trenca el monopoli de la utilització del gradient per a explicabilitat i passa a utilitzar el que s'anomena la contribució d'un mapa d'activació respecte una puntuació. L'algorisme utilitzarà aquesta contribució per

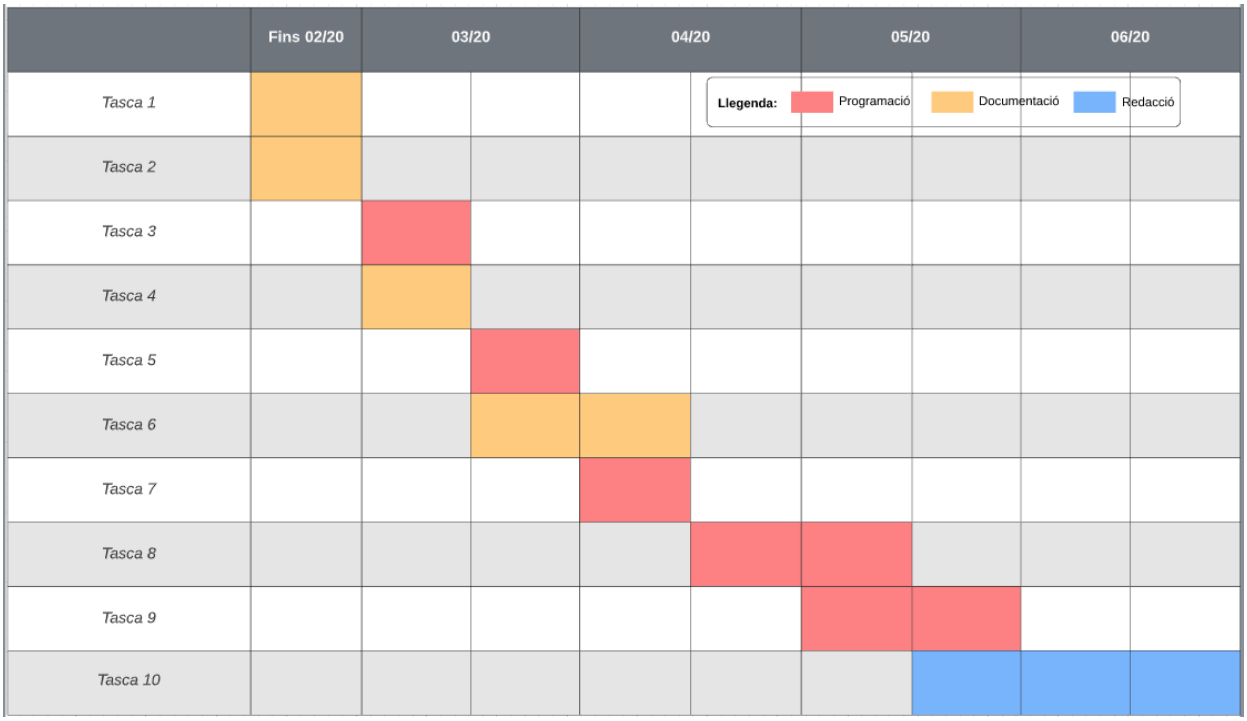
a decidir la importància de cada mapa d'activació, creant el *heatmap* fent una suma ponderada.

L'Score-CAM és l'últim algorisme CAM que explicarem i és, també, un dels últims algorismes de l'estat de l'art d'explicabilitat mitjançant *heatmaps* que s'han publicat fins al moment, competint amb rendiment amb l'anterior Grad-CAM++ però aportant més estabilitat.

3 Planificació i metodologia

La feina realitzada en aquest Treball de Fi de Grau es pot dividir en la documentació i l'assimilació de conceptes teòrics, la programació d'aplicacions pràctiques dels conceptes anteriors i, finalment, la redacció de la memòria. Seguidament es pot trobar una taula amb cada tasca realitzada i un diagrama de Gantt amb la durada de cadascuna, així com el tipus de la tasca: documentar, programar o redactar. Aquest treball es va començar a plantejar a mitjans de 2019.

Etapa	Objectiu
Tasca 1	Documentació sobre xarxes neuronals i xarxes neuronals convolucionals.
Tasca 2	Decisió sobre quin programari s'utilitzarà i quines llibreries. Experimentació amb ells.
Tasca 3	Transfer-Learning de Cifar-10.
Tasca 4	Documentar i entendre l'algorisme de Grad-CAM++.
Tasca 5	Entrenar models amb Food101.
Tasca 6	Buscar, documentar i entendre altres algorismes d'explicabilitat.
Tasca 7	Obtenir resultats amb Grad-CAM++. Realitzar proves amb el dataset de l'article de Grad-CAM++ per comprovar funcionalitat real.
Tasca 8	Comparar Grad-CAM, Grad-CAM++ i Score-CAM i obtenir Guided Back-propagation, tant amb Food101 com amb Imagenet.
Tasca 9	Executar prova multilabel amb food201. Preparar mètrica quantitativa.
Tasca 10	Redacció de la memòria, organització de codi, generar imatges de mostra.



Imatge 17: Etapes de la realització del Treball de Fi de Grau.

4 Implementació

4.1 Programari utilitzat

El programari que utilitzat està orientat al *machine learning* i, per tant, permet un desenvolupament eficient de xarxes neuronals, fet que facilitarà realitzar una gran quantitat d'experiments amb diferents models.

Per a realitzar les proves i diversos experiments utilitzarem Google Colab (`https://colab.research.google.com/`). Google Colab és un editor de *notebooks* de Python que permet executar el codi que hom creï a les màquines sobrants de Google de forma gratuïta. Té disponibles entorns de TPU i GPU, a més de memòria per a guardar totes les dades per a entrenar les xarxes neuronals. La versió gratuïta té els inconvenients de desconnectar-se al cap d'un temps d'inactivitat de l'usuari, encara que el codi s'estigui executant, i de tenir menys prioritat per les TPU disponibles que els usuaris *premium*. A més, permet la connexió amb Google Drive per a carregar les imatges a processar i guardar els models entrenats de forma àgil.

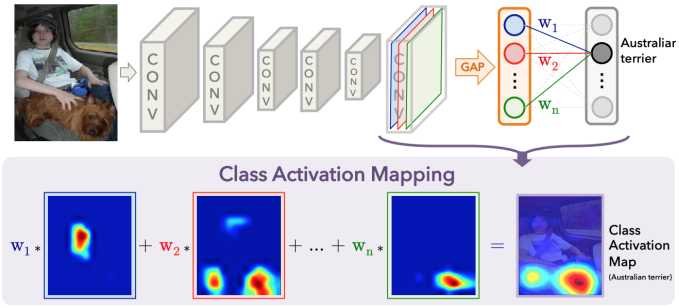
En quant a la realització de codi, la principal llibreria que utilitzarem és Keras amb tensorflow, una llibreria de Python per a entrenar xarxes neuronals. N'existeix una altra, PyTorch, que també seria adequada per a la realització del TFG. No obstant, és a més baix nivell i aprendre-la requereix molt més temps que Keras, tenint molta menys documentació i exemples disponibles.

Finalment, utilitzarem Weights and Biases (`https://www.wandb.com/`), Wandb, una pàgina web en la que es poden registrar gràfics d'entrenament. Wandb es pot associar a un codi de Python per a autogenerar i tenir control de com evoluciona, per exemple, la precisió de la xarxa neuronal a mesura que l'anem entrenant.

4.2 Detalls dels algorismes

4.2.1 CAM

La idea darrera aquest algorisme [3], com hem comentat abans, és introduir una capa GAP abans de donar la puntuació de les classes, entre l'última capa convolucional i la capa densa (que, per exemple, pot tenir una funció d'activació *Softmax*).



Imatge 18: Extreta de [3]. Funcionament de CAM.

Una capa GAP (*Global Average Pooling*) pren les dades d'entrada i en fa la mitjana per cada profunditat, de forma que, si la capa convolucional anterior té N filtres i, per tant, la seva sortida té dimensions XxYxN o, equivalentment, N mapes d'activació de mida XxY, la GAP agafarà cada mapa d'activació i farà la mitjana de tots els seus píxels. La sortida de la GAP serà un vector de dimensió N on els seus elements corresponen a la mitjanes dels mapes d'activació.

Suposem el *forward pass* d'una imatge. Sigui $f_k(x, y)$ el mapa d'activació k en la posició (x, y). Al passar per la GAP, tenim que la posició k de la sortida és $\frac{1}{Z}\sum_{x,y} f_k(x, y)$ amb Z el número d'elements del mapa d'activació.

Ara, tenim que al passar per la capa densa s'obtenen les puntuacions de les classes abans d'aplicar la capa d'activació *Softmax*. La puntuació per a la classe c, S_c, ve donada per

$$S_c = \frac{1}{Z} \sum_k \sum_{x,y} w_k^c * f_k(x,y) = \frac{1}{Z} \sum_{x,y} \sum_k w_k^c * f_k(x,y) \tag{46}$$

on els w_k^c són els pesos de la capa densa a la fila c.

Llavors, definim el mapa d'activació de la classe c (*Class Activation Map* o CAM, d'on pren el nom l'algorisme) a la posició (x, y) com

$$M_c(x,y) = \sum_k w_k^c * f_k(x,y) \tag{47}$$

Podent reescriure la puntuació, quedant

$$S_c = \frac{1}{Z} \sum_{x,y} M_c(x,y) \tag{48}$$

Imatge 19: Extreta de [3]. Resultats de CAM.

Per tant, la puntuació de la classe c ve donada per la potència dels píxels de M_c: s'obtindrà més puntuació cap a aquella classe com més píxels activats i amb més potència tingui el CAM. Si ara reescalem M_c a la mida de la imatge d'entrada obtenim resultats com els de la imatge 19, on no només obtenim les zones on es fixa la xarxa neuronal, sinó també la localització d'objectes en les imatges.

A la imatge 18 podem veure un esquema del procediment descrit. Podem apreciar en colors els mapes d'activació. Les esferes són les respectives sortides de la GAP per a cada mapa i $W_1, ..., W_n$ són els pesos de la fila 2 de la matriu de pesos per a donar la puntuació de la segona classe al multiplicar-los per la sortida de la GAP. Així, el CAM per a la

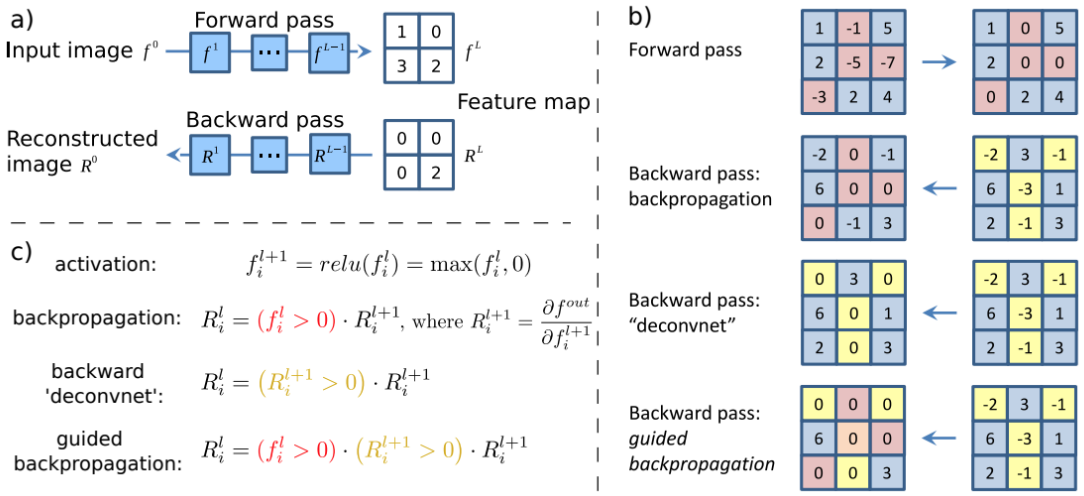
classe 2 es genera fent la suma ponderada dels mapes d'activació amb els pesos de la fila 2, seguint les fórmules.

CAM està en desús per un motiu obvi: obliga a afegir una GAP a qualsevol xarxa neuronal a la que es vulgui aplicar i, per tant, limita l'estructura i disminueix la precisió de la classificació.

4.2.2 Guided backpropagation

Aquest algorisme [16] utilitza *backpropagation* per a generar el *heatmap*, tot i que la seva sortida és més aviat un relleu de les característiques que ha après una capa convolucional i no pas un mapa de calor com és comú en els algorismes visuals d'explicabilitat.

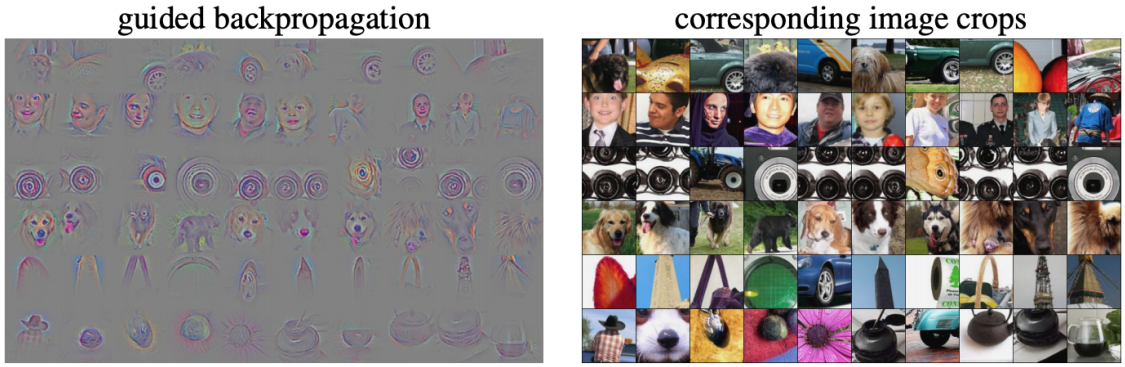
Guided Backpropagation pren com a entrada una de les capes convolucionals de la xarxa neuronal i agafa l'*activation map* que té els valors més alts un cop fet el *forward pass* d'una imatge. Idealment, aquesta capa és l'última convolucional de la xarxa, ja que és on més informació sobre les característiques de la imatge hi haurà. Llavors, posa a 0 les altres activacions i realitza *backpropagation* fins a tenir el gradient de la sortida del filtre elegit respecte la imatge d'entrada.



Imatge 20: Extreta de [16]. Funcionament de Guided Backpropagation, fusionant Backpropagation i backward 'convnet'.

Ho fa, però amb una peculiaritat: al passar per una ReLu, en comptes de posar a 0 les posicions dels elements que en el *forward pass* eren negatius, també posa a 0 les posicions

en que el propi gradient és negatiu. Podem veure les equacions amb un exemple pràctic a la imatge 20.



Imatge 21: Extreta de [16]. Resultats de Guided Backpropagation.

Aquest procediment és la suma de dos mètodes: el mètode de *backward pass* i l'algorisme *backward 'convnet'*. En el *backward pass*, quan el gradient passa per la ReLu, es tenen en compte els valors del *forward pass*. Si durant el *forward pass* l'element era positiu, el gradient es manté; en cas contrari, el gradient s'anul·la. En l'algorisme *backward 'convnet'* es tracta la derivada de la ReLu com a una ReLu inversa i, per tant, quan el gradient passa per una ReLu durant realitzant *backpropagation*, s'eliminen els valors de gradient negatius.

A la imatge 21 podem veure alguns resultats de l'aplicació d'aquest algorisme, on es pot apreciar com, en comptes d'obtenir un *heatmap*, s'obté el relleu de les característiques de la imatge.

4.2.3 Grad-CAM i Guided Grad-CAM

L'algorisme Grad-CAM [22], considerat una generalització de CAM, pren com a paràmetre una capa convolucional de la xarxa neuronal i calcula totes les jacobianes de la puntuació de la classe c abans de l'activació Softmax Y^c respecte cada mapa d'activació A^k, $\frac{\partial Y_c}{\partial A^k}$.

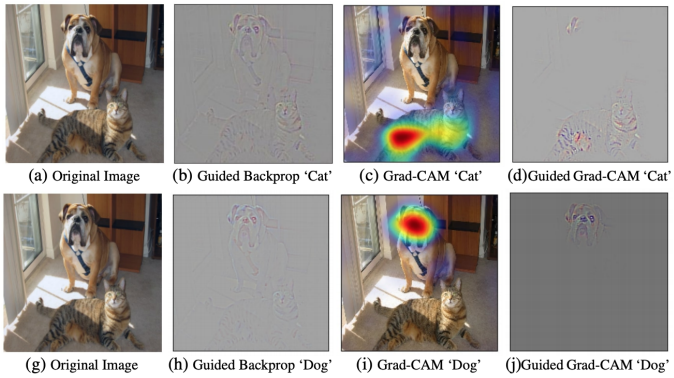
Llavors, definim la importància d'A_k per la classe c com

$$w_k^c = \frac{1}{Z} \sum_i \sum_j \frac{\partial Y_c}{\partial A^k}_{i,j} \tag{49}$$

on el sufix i, j indica la fila i i la columna j de la jacobiana i Z és el número de píxels (elements) d'aquesta. Per tant, la importància és la mitjana dels valors de la jacobiana.

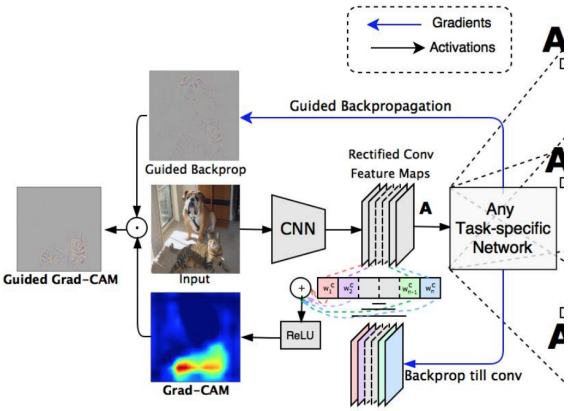
Finalment, definim el mapa de localització de la classe c a la capa convolucional elegida (*class-discriminative localization map Grad-CAM*), $L^c_{Grad-CAM}$ com

$$L^c_{Grad-CAM} = ReLu(\sum_k w^c_k * A^k) \tag{50}$$



Imatge 22: Extreta de [22]. Resultats de Grad-CAM.

La ReLu ens ajuda a eliminar els valors negatius, ja que no ens interessa visualitzar quines regions fan que disminueixi la puntuació de la classe (aquestes contribuiran, molt probablement, a la decisió d'una altra classe). A la imatge 23 tenim un esquema gràfic de com funciona Grad-CAM.



Imatge 23: Extreta de [22]. Funcionament de Grad-CAM: realitzar *backpropagation* per a obtenir les jacobianes, calcular les importàncies i, finalment, fer la suma ponderada dels mapes d'activació, passant el resultat per una ReLu.

Reescalant a la imatge de sortida obtenim el *heatmap* que mostrarà quines regions de la imatge influeixen més en la puntuació per aquella classe, com podem veure a la imatge 22.

Intuïtivament, calcular les importàncies segons la classe ens permet veure quines parts de la imatge d'entrada són importants per a cada classe (és a dir, quines característiques de cada mapa d'activació influeixen més una classe). Així, si una imatge té dos elements, un de la classe X i un de la classe Y, podem veure com al demanar el *heatmap* de la classe X es remarca el primer objecte i, si ho repetim amb la classe Y, com es remarca el segon.

A més, com hem comentat a l'estat de l'art, és possible utilitzar l'algorisme de Guided Backpropagation i fusionar-lo amb el *heatmap* de Grad-CAM utilitzant un producte escalar, creant Guided Grad-CAM i obtenint les característiques importants de la imatge per a pertànyer a aquella classe alhora que s'extrau la textura d'aquestes característiques. Aquesta fusió es podrà realitzar també de la mateixa forma amb Grad-CAM++ i Score-CAM.

4.2.4 Grad-CAM++

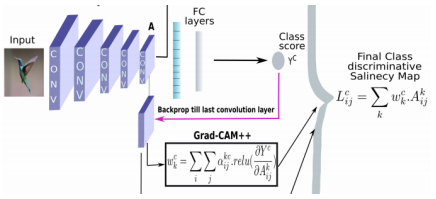
L'algorisme de Grad-CAM++ [1] proposa una modificació als pesos d'importància de Grad-CAM, equilibrant l'aportació dels diferents *activation maps*. Tenim, inicialment, les equacions de Grad-CAM 49 i 50. Ara, modifiquem l'equació 49, quedant

$$w_k^c = \sum_i \sum_j \alpha_{ij}^{kc} * ReLu(\frac{\partial Y^c}{\partial A_{ij}^k}) \tag{51}$$

on α_{ij}^{kc} són coeficients que depenen del mapa d'activació A^k, la classe c i la posició del píxel ij, pels quals ara deduirem una fórmula.

Sigui Y_c la puntuació de la imatge d'entrada per la classe c, abans de passar per l'activació *Softmax*. Substituint l'equació 51 a 46 queda

$$Y^c = \frac{1}{Z} \sum_k ([\sum_x \sum_y \alpha_{xy}^{kc} * ReLu(\frac{\partial Y^c}{\partial A_{xy}^k})] * [\sum_i \sum_j A_{ij}^k]) \tag{52}$$



Imatge 24: Extreta de [1]. Funcionament de Grad-CAM++.

Sense pèrdua de generalitat podem treure la ReLu, ja que només serveix per impedir que certs valors es mantinguin. A l'equació, i, j i x, y iteren sobre el número de píxels del mapa d'activació k.

Derivant respecte A_{ij}^k i arreglant els índexs, obtenim

$$\frac{\partial Y^c}{\partial A_{ij}^k} = \frac{1}{Z} * (\sum_x \sum_y \alpha_{xy}^{kc} * \frac{\partial Y^c}{\partial A_{xy}^k} + \sum_x \sum_y A_{xy}^k * \alpha_{ij}^{kc} * \frac{\partial^2 Y^c}{\partial A_{ij}^{k^2}}) \tag{53}$$

Tornant a derivar respecte A_{ij}^k es té

$$\frac{\partial^2 Y^c}{\partial A_{ij}^{k^2}} = \frac{1}{Z} * (2 * \alpha_{ij}^{kc} * \frac{\partial^2 Y^c}{\partial A_{ij}^{k^2}} + \sum_x \sum_y A_{xy}^k * \alpha_{ij}^{kc} * \frac{\partial^3 Y^c}{\partial A_{ij}^{k^3}}) \tag{54}$$

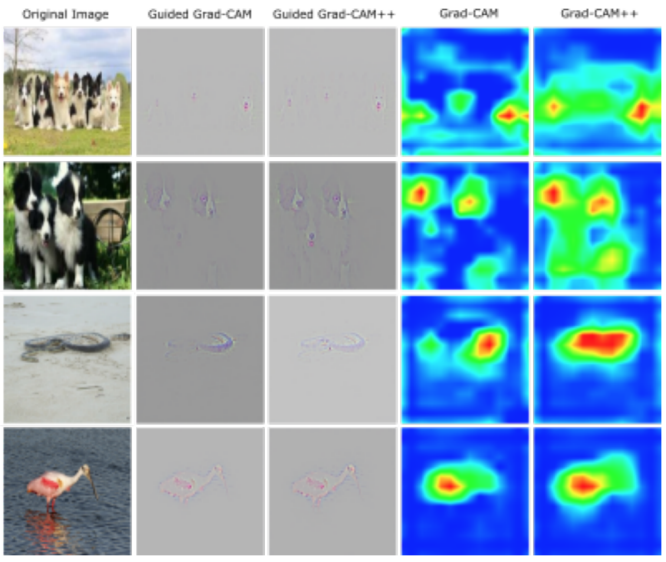
i aïllant α_{xy}^{kc}

$$\alpha_{ij}^{kc} = Z * \frac{\frac{\partial^2 Y^c}{\partial A_{ij}^{k^2}}}{2 * \frac{\partial^2 Y^c}{\partial A_{ij}^{k^2}} + \sum_x \sum_y A_{xy}^k * \frac{\partial^3 Y^c}{\partial A_{ij}^{k^3}}} \tag{55}$$

Substituint 55 a 51 obtenim la importància de cada mapa d'activació, donant per acabada la fórmula de Grad-CAM++. Un esquema del funcionament de Grad-CAM++ està il·lustrat a la imatge 24.

Per a trobar el *heatmap* de Grad-CAM++ el que queda calcular és

$$L_{Grad-CAm++}^c = ReLu(\sum_k w_k^c * A_k) \tag{56}$$



Imatge 25: Extreta de [1]. Comparativa de Grad-CAM++ i Grad-CAM.

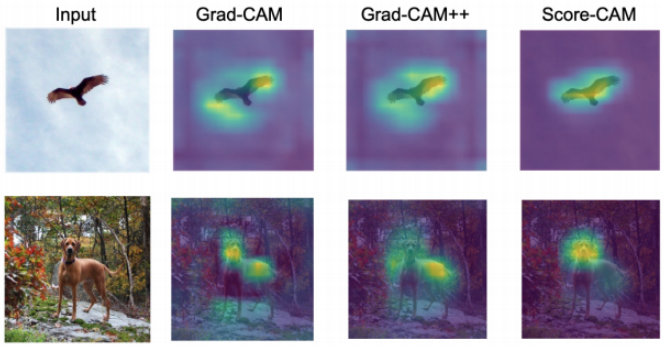
que escalant-lo a la mida de la imatge d'entrada ens donarà les regions més importants per a la decisió de la puntuació de la classe.

L'algorisme Grad-Cam++ ha estat considerat fins l'actualitat l'algorisme d'estat de l'art per excel·lència a causa dels seus bons resultats, com podem veure a la imatge 25. Un dels problemes més grans que presenta és la seva inestabilitat i la sensibilitat al soroll generada pels càlculs amb el gradient. El següent algorisme intenta solucionar aquest inconvenient.

4.2.5 Score-CAM

L'algorisme d'Score-CAM [11], com hem comentat a l'estat de l'art, suposa un canvi respecte als estàndards dels últims CAM. Amb un rendiment que supera fins i tot l'explicabilitat de Grad-CAM++, aconsegueix allunyar-se del càlcul de gradients per a utilitzar càlculs molt més estables.

Un dels punts dèbils de Grad-CAM++ és precisament estar basat fortament en el gradient, que pot ser sorollós després de propagar-se per la xarxa neuronal i fins i tot pot deixar de ser significatiu per a generar *heatmaps*. L'algorisme d'Score-CAM utilitza el que s'anomena la contribució d'un *activation map*, que com veurem a continuació no conté en el seu càlcul cap tipus de gradient, fent que s'elimini tot l'error que provoca en el *heatmap* treballar amb el gradient calculat amb backpropagation.



Imatge 26: Extreta de [11]. Comparativa de Grad-CAM++, Grad-CAM i Score-CAM.

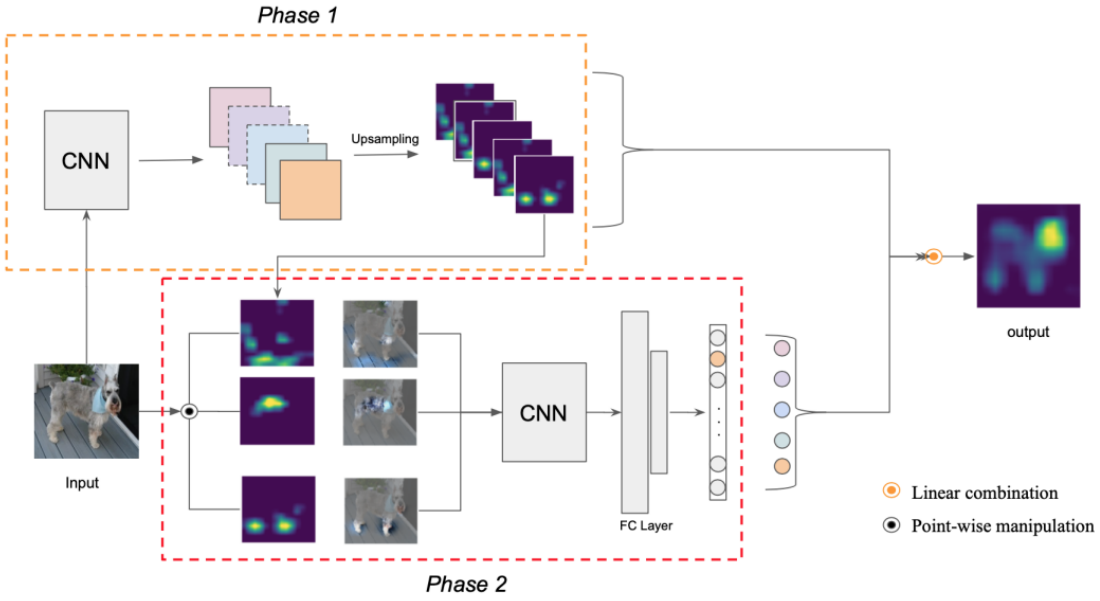
Així, donada una capa convolucional i els mapes d'activació d'un *forward pass* d'una imatge X, obtenint puntuació Y, definim la contribució del mapa d'activació A_k respecte X cap a la puntuació Y_c (puntuació de la classe c) com

$$C(A_k) = f(X \circ H_k) - f(X) \quad (57)$$

on $\circ$ indica el producte de Hadamard (element a element), f(X) realitza el *forward pass* de X per la xarxa neuronal, obtenint la puntuació per la classe c i

$$H_k = s(Up(A_k)) \quad (58)$$

on s(X) posa els valors de X entre 0 i 1 i Up(Y) reescala Y a les dimensions de la imatge inicial.



Imatge 27: Extreta de [11]. Procediment d'Score-CAM.

Per tant, a l'equació 57 el que fem és emmascarar la imatge X utilitzant les característiques apreses per A_k, realitzar el *forward pass* i veure com canvia respecte la imatge original.

El *heatmap* d'Score-CAM ve generat, com és usual, per

$$L_{sc} = ReLu(\sum_k C(A_k) * A_k) \tag{59}$$

on $\sum_k C(A_k) = 1$.

Finalment, es reescala a la mida de la imatge original per a poder veure les parts marcades correctament. A la imatge 26 podem veure una comparativa de resultats amb els algorismes previs i a 27 l'estructura del procediment que hem anat seguint. En general dona un rendiment superior i més estable que Grad-CAM++, podent ser una millora en l'estat de l'art d'explicabilitat mitjançant *heatmaps*.

4.3 Datasets

En aquesta secció comentarem els *datasets* o conjunts d'imatges que utilitzarem durant els diferents codis que realitzem.

El primer i més important és Imagenet. Imagenet (`image-net.org/`) és un dels *datasets* més grans d'imatges disponibles actualment amb més de 100.000 classes disponibles. Els models que s'ofereixen ja preentrenats ho estan amb aquesta col·lecció d'imatges, donada la increïble versatilitat i diversitat que presenta.

El següent *dataset* a comentar és una col·lecció bàsica utilitzada per a aprendre a entrenar xarxes neuronals. Es tracta de Cifar10 (https://www.cs.toronto.edu/ kriz/cifar.html), una col·lecció de 10 classes amb 10000 imatges a cada una, ideal per a començar a practicar. El nostre primer contacte amb el codi d'una xarxa neuronal serà per a aconseguir classificar decentment aquest *dataset*.

Els últims dos conjunts que treballarem i amb els que volem comprovar els algorismes d'explicabilitat estan relacionats amb el menjar. És convenient fer èmfasi en el perquè s'ha elegit aquest àmbit. Tant la detecció de menjar com la classificació de menjar en classes són temes punters a causa de la preocupació emergent de la societat per saber quins ingredients conté i com de sa és allò que menja, i el gran avenç en el control de dietes per causes mèdiques que suposaria tenir la capacitat d'automatitzar la identificació de menjar. No obstant, analitzar menjar suposa un gran repte donada la gran varietat que existeix, en milions de formats i mides de racions d'un mateix menjar. Per això, els algorismes d'explicabilitat aplicats a aquest àmbit poden ser clau per a millorar les aplicacions comercials d'anàlisi de menjar.

El *dataset* de Food101 [4] és el principal objecte d'estudi i en el que centrarem els nostres esforços d'explicabilitat. Aquest *dataset* conté 101 classes dedicades al menjar, amb 1000 imatges a cada classe. Actualment és la col·lecció d'imatges de menjar ja classificades més gran que hi ha disponible de forma pública.

Finalment, Food201 [2] utilitza les imatges de Food101 i enumera els elements extra en cada imatge. Per exemple, si en una imatge de la classe "amanida cèsar" hi apareix un taco, aquesta imatge serà etiquetada com a pertanyent, també, a aquesta última classe. Ens serà útil per a treballar el *multilabeling* i comprovar identificació de diferents objectes en una sola imatge.

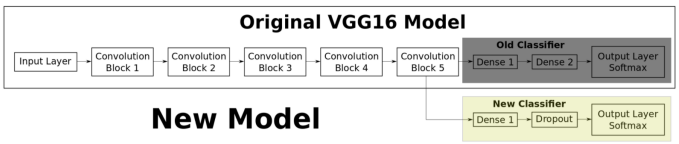
4.4 Proves realitzades

En aquesta secció detallarem les proves realitzades durant les diferents etapes del Treball de Fi de Grau. Els resultats obtinguts s'analitzaran a l'apartat Resultats.

4.4.1 Transer Learning de Cifar10

Com hem comentat anteriorment, les capes convolucionals de la xarxa neuronal convolucional extreuen les característiques principals d'una imatge i les capes denses utilitzen aquestes característiques per a obtenir les puntuacions de les N classes. Per tant, si agafem una xarxa neuronal convolucional entrenada en un *dataset* molt general com és Imagenet, les capes convolucionals podran extreure de forma acceptable les característiques d'imatges d'una col·lecció més específica, com en el nostre cas serà Cifar10

Utilitzant aquestes capes convolucionals ja entrenades en Imagenet i afegint noves capes denses per a entrenar en comptes de les ja entrenades, el que ens plantegem és aconseguir una xarxa neuronal que classifiqui imatges de Cifar10.



Imatge 28: Extreta de [21]. Transfer learning en una VGG16.

Aquest procediment es coneix com a *Transfer Learning* [7] [21] d'una xarxa neuronal. En aquesta operació es col·loquen les capes convolucionals ja entrenades dins una altra xarxa neuronal i aquesta s'entrena deixant congelats els paràmetres de les capes convolucionals. D'aquesta forma, fem l'entrenament molt més ràpid i, tot no aconseguir, en general, resultats excel·lents, permet aconseguir una xarxa decent en molt poc temps i capacitat de computació. Un esquema d'aquest procés s'il·lustra a la imatge 28, on es canvien les últimes capes denses per a adaptar la xarxa neuronal a un altre problema de classificació.

4.4.2 Realització de tests d'algorismes d'explicabilitat sobre Imagenet

En aquest codi comprovarem la funcionalitat dels algorismes d'explicabilitat Grad-CAM, Guided Backpropagation, Grad-CAM++ i Score-CAM donat el *dataset* d'Imagenet. Aque-

sta comprovació és necessària abans de les següents proves amb menjar, ja que posteriorment aplicarem *Transfer Learning* per a aconseguir una xarxa neuronal que classifiqui Food-101 i aplicar explicabilitat en ella i, per tant, ens servirà com a base de veritat tenir proves realitzades en una xarxa perfectament entrenada en Imagenet. També intentarem definir una mètrica per a donar quantitativament quin algorisme dona millors resultats.

4.4.3 Proves d'explicabilitat amb Food101

El que ens plantegem en aquesta secció és utilitzar *Transfer learning* per a obtenir una xarxa neuronal convolucional que ens permeti classificar imatges d'un subconjunt de Food-101. Es realitzaran iteracions sobre aquesta xarxa per anar millorant el seu rendiment, explicitant la millor obtinguda a l'apartat Resultats.

Paral·lelament, aplicarem els mateixos algorismes d'explicabilitat a aquesta nova xarxa i utilitzarem la mètrica ideada a l'apartat anterior i una mostra visual per a realitzar una comparativa quantitativa i qualitativa dels *heatmaps* generats.

4.4.4 Proves d'explicabilitat amb multilabel

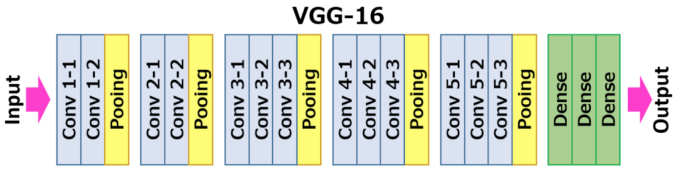
Com a una de les proves finals, intentarem treballar d'alguna forma amb Food201 per a comprovar si en una imatge en la que hi ha continguts varis elements de les classes som capaços de generar *heatmaps* que expliquin (i remarquin) cadascun d'ells de forma diferenciada. Com tots els algorismes que estan sent comprovats presenten la capacitat de donar l'explicabilitat per a cada classe, s'utilitzarà només Grad-CAM per a aquest codi, ja que el seu objectiu no és realitzar una comparativa sinó explorar la capacitat de diferenciar dos classes en una imatge.

5 Resultats

5.1 Transfer Learning de Cifar10

L'objectiu d'aquesta primera part pràctica del codi era obtenir una xarxa neuronal, mitjançant Transfer Learning, per a classificar imatges del *dataset* de Cifar10.

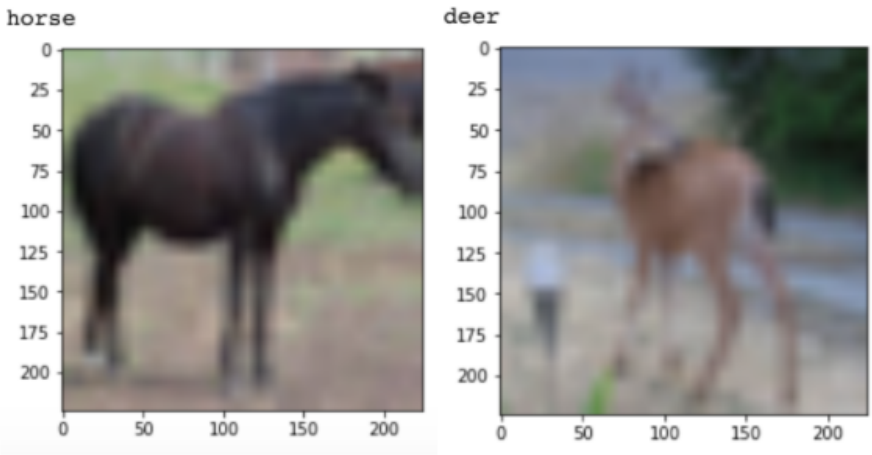
La xarxa elegida per a realitzar el Transfer Learning ha estat una VGG16 (imatge 29) que accepta com a entrada imatges de 32x32x3 píxels, ja entrenada en Imagenet.



Imatge 29: Extreta de `https://neurohive.io/en/popular-networks/vgg16/`. xarxa neuronal VGG16.

Llavors, prenem només la part convolucional de la xarxa neuronal, seguint el procediment de *Transfer Learning*, i per tant, eliminem les 3 capes denses del final que es troben a partir de l'últim *max pooling*. Finalment, afegim una capa de *Flattening* per a aplanar la sortida de les capes convolucionals i poder-la passar a les noves denses, una capa densa de 100 neurones i una capa densa amb *Softmax* per a obtenir ja les puntuacions. Intercalem capes de *batchNormalization* per a aconseguir més estabilitat numèrica.

L'estructura de la xarxa neuronal resultant es pot veure a la imatge 34, a l'apèndix. Notem que l'última capa densa té 10 neurones ja que volem classificar Cifar10, que té 10 classes i, per tant, la sortida de 10 neurones és un vector de 10 elements.



Imatge 30: Resultats Transfer Learning. Predit amb un 99% i un 84% de seguretat, respectivament.

Per a entrenar la xarxa congelem els paràmetres de les capes exceptuant els de les noves denses i els de l'última capa convolucional per a permetre adaptabilitat al model. Després de 7 èpoques utilitzant un *learning rate* de 0.03, RMSprop com a optimizador i un *batch size* de 50 obtenim un 79% de precisió, un percentatge prou bo tenint en compte l'objectiu de l'exercici. A la imatge 30 es mostren algunes de les imatges classificades.

5.2 Explicabilitat sobre Imagenet

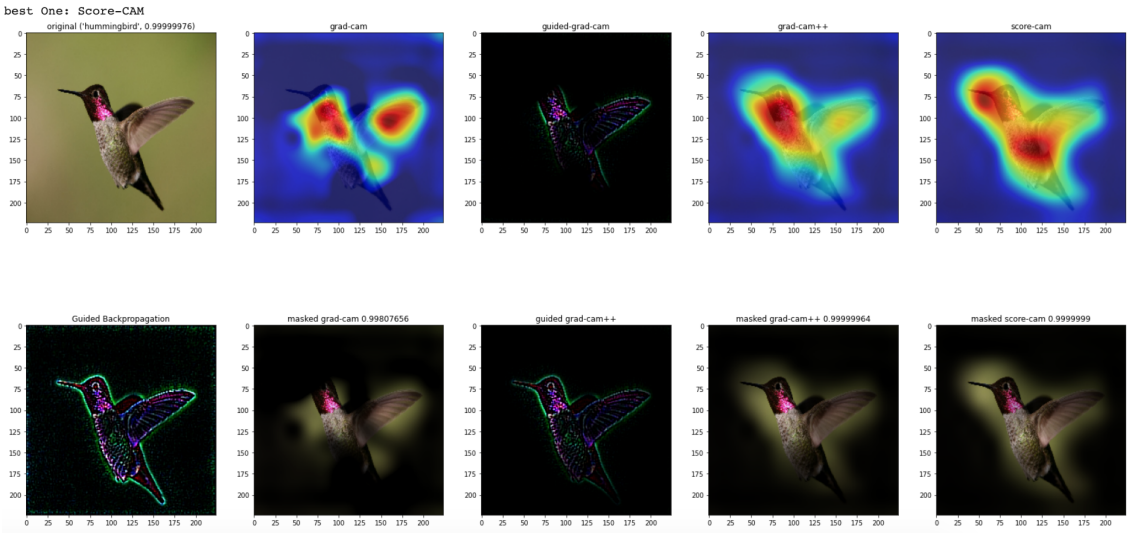
En aquest apartat busquem treballar amb una xarxa neuronal entrenada amb Imagenet i volem obtenir resultats amb els algorismes d'explicabilitat mencionats a l'estat de l'art. Amb aquest propòsit, es pren el codi de Grad-CAM, Grad-CAM++, Guided Backpropagation i Score-CAM implementats en Keras de [29] i [27], modificats lleugerament, i els apliquem sobre VGG16 entrenada ja per Imagenet.

A més, ens plantegem definir una mètrica que ens permeti comparar els diversos algorismes d'explicabilitat entre si i la presentada és la següent:

- Es classifica la imatge X, obtenint les puntuacions de cada classe i, per tant, assignant una classe a X.
- S'apliquen els algorismes d'explicabilitat, obtenint el seu *heatmap* respectiu.
- Es crea una imatge emmascarada utilitzant com a màscara el *heatmap* de cada algorisme de forma que les àrees no marcades com a importants s'eliminen de la imatge original.
- Es passa la imatge emmascarada per la xarxa neuronal i es mira la puntuació de la classe que ha predit la xarxa per la imatge original.
- Es mira quin algorisme genera una màscara amb puntuació més alta, significant que les característiques de la imatge que senyala tenen més importància per la xarxa.

Així, els resultats d'aquest codi són del format de la imatge 31. Primer, s'anuncia quin algorisme ha obtingut millor puntuació. Després, es mostra una graella amb dos files d'imatges. D'esquerra a dreta i de dalt a baix: imatge original, Grad-CAM, Guided Grad-CAM, Grad-CAM++, Score-CAM, Guided Backpropagation, imatge emmascarada per Grad-CAM, Guided Grad-CAM++, imatge emmascarada per Grad-CAM++ i imatge emmascarada per Score-CAM.

Cadascuna de les imatges emmascarades té associada la puntuació que ha tret sobre la classe a la qual s'ha assignat la imatge original (*hummingbird*, en aquest cas). Sobre la imatge original veiem la classe a la que ha estat assignada i amb quina puntuació.



Imatge 31: Resultats explainability a Imagenet.

Aquesta informació és suficient per a fer estudis comparatius. Donada una sèrie d'imatges, es pot veure quin algorisme és millor mirant quin dona millor puntuació que els altres en més imatges o bé utilitzant un sistema de puntuació.

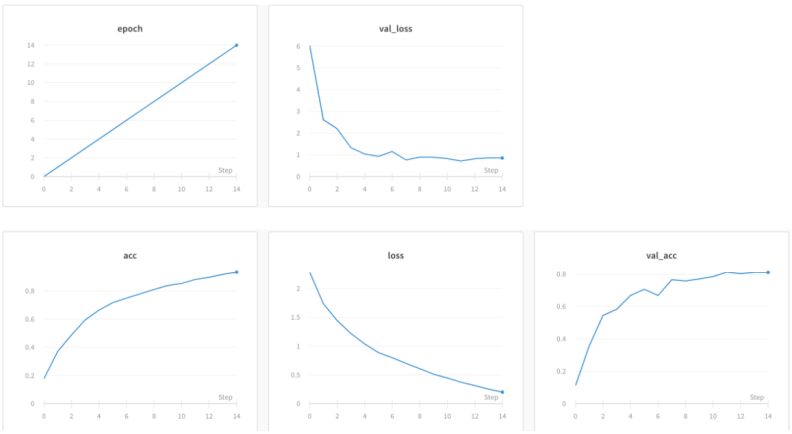
Per exemple, cada cop que un algorisme millora la puntuació que ha obtingut una imatge guanya un punt i si l'empitjora el perd. Es pot afegir un marge d'error en el que, encara que l'algorisme doni pitjor puntuació, es compta com a error raonable i segueix sumant el punt. Així, $-n$ és el pitjor resultat possible i n el millor.

5.3 Transfer Learning i Explicabilitat sobre Food101

En aquest codi el nostre objectiu és aconseguir una xarxa neuronal que classifiqui amb una precisió acceptable el *dataset* de Food101. En concret, hem seleccionat per a classificar les imatges de 10 classes, a saber: *caesar salad, chicken curry, chicken quesadilla, filet mignon, fish and chips, fried calamari, fried rice, grilled salmon, hamburger i huevos rancheros.*

Amb el procés de *Transfer Learning* i després de diverses iteracions hem obtingut una xarxa neuronal amb un 81% de precisió en la classificació d'aquest *dataset*. Podem trobar l'estructura de la xarxa neuronal obtinguda a l'annex A. A la imatge 32 podem veure els gràfics d'entrenament extrets de Wandb.

Utilitzant aquesta xarxa, hem aplicat els mateixos algorismes que en el codi anterior, obtenint imatges com les que es poden consultar a l'annex B.



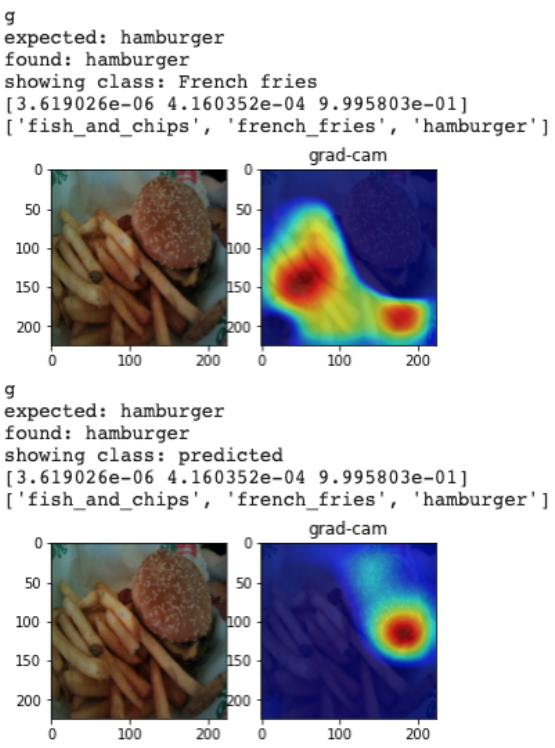
Imatge 32: Procés d'entrenament per a Food101.

A més, hem executat les mètriques descrites en l'apartat anterior per a treure una valoració numèrica de l'explicabilitat dels mètodes. Així, hem obtingut que per 100 imatges elegides aleatòriament entre tot el *dataset* es té que Score-CAM (42) ha tingut millor puntuació que els altres més vegades, seguit de Grad-CAM++ (37) i per últim Grad-CAM (21); i que Score-CAM i Grad-CAM++ (-20) tenen un ràtio de millora de puntuació respecte la imatge original més alt que Grad-CAM (-42), encara que tots tres han acabat, en general, empitjorant la puntuació més vegades que millorant-la. Aquests resultats coincideixen amb l'anàlisi qualitatiu de les imatges: els algorismes Score-CAM i Grad-CAM++ donen millors *heatmaps* per a explicar les decisions de la xarxa neuronal.

No obstant, Score-CAM té uns resultats lleugerament millors i, sumant el fet de ser més estable que Grad-Cam++ al no dependre de gradients, el converteix en un molt bon candidat per a ser el nou algorisme d'estat de l'art.

5.4 Explicabilitat sobre Food201

L'objectiu d'aquesta part final és explorar la capacitat dels algorismes d'explicabilitat (en concret, de Grad-CAM), per a identificar objectes en una mateixa imatge. Per a fer-ho, tenim el *dataset* Food201, que ens aporta una llista formada per les imatges de Food101 on cada imatge té associada elements d'altres classes que apareixen en ella. Per exemple, ens podria dir que la imatge 3 de les amanides cèsar de Food101 conté també macarrons.



Imatge 33: Explicabilitat per a patates i hamburguesa.

Utilitzant aquesta llista obtenim les imatges d'hamburgueses en les que apareixen, també, patates fregides. El nostre objectiu serà veure si podem diferenciar les hamburgueses de les patates en les imatges, donada una xarxa neuronal entrenada en classificar hamburgueses, patates fregides i, per aportar diversitat a la classificació, fish and chips.

La xarxa neuronal ha estat entrenada de forma similar a la dels anteriors codis, obtenint un 94% de precisió. L'estructura de xarxa neuronal creada es pot consultar a l'annex A. Els resultats són visualment molt satisfactoris en la gran majoria dels casos. Podem veure a la imatge 33 com distingeix la zona d'hamburguesa de la zona de patata quan es canvia la classe a explicar demanada a l'algorisme. A l'annex B hi ha annexades més imatges per a il·lustrar els resultats obtinguts.

6 Conclusions i futurs treballs

Per acabar el treball, valorarem el compliment dels objectius anunciats a l'inici d'aquest. També analitzarem els futurs treballs i les futures aplicacions a partir del que hem aconseguit.

Per una banda, hem realitzat i documentat un anàlisi exhaustiu sobre les xarxes neuronals i, en concret, sobre les xarxes neuronals convolucionals. Hem vist la seva estructura, el seu aprenentatge i el seu funcionament, de forma tant teòrica com pràctica, i hem entrenat i aplicat de forma satisfactòria diverses xarxes neuronals convolucionals a diferents *datasets*.

Per altra banda, hem documentat els diferents algorismes CAM per a explicabilitat

visual i l'algorisme Guided Backpropagation per a anàlisi de les textures i relleus i els hem aplicat de forma satisfactòria a les xarxes neuronals construïdes.

Finalment, hem realitzat un anàlisi dels resultats d'aquests algorismes, arribant a que Score-CAM és lleugerament millor que Grad-CAM++ en termes d'explicabilitat i els dos són clarament superiors a CAM.

En quant a futurs treballs i aplicacions, es podria utilitzar la identificació d'objectes amb *heatmaps* obtinguda mitjançant Food201 per a entrenar un *clustering* dins de cada classe, o utilitzar els mapes d'explicabilitat per a realitzar segmentació per a la identificació d'objectes en imatges o vídeos.

Com a reflexió personal, aquest treball m'ha permès comprendre i endinsar-me dins el món de les xarxes neuronals. He comprès tot l'estat de l'art al voltant d'aquestes i he experimentat amb els algorismes actuals d'explicabilitat, veient el perfeccionament que falta en aquest àmbit. En el futur, m'agradaria cursar un màster que em permeti contribuir dins l'àrea de *Deep Learning*, aprendre a utilitzar PyTorch, estudiar altres algorismes d'explicabilitat que s'han quedat fora de l'objectiu del treball per falta de temps i espai i, qui sap, contribuir en fer les xarxes neuronals més explicables.

Annexos

A Estructures de les xarxes neuronals convolucionals creades

Layer (type)	Output Shape
input_10 (InputLayer)	(None, 32, 32, 3)
block1_conv1 (Conv2D)	(None, 32, 32, 64)
block1_conv2 (Conv2D)	(None, 32, 32, 64)
block1_pool (MaxPooling2D)	(None, 16, 16, 64)
block2_conv1 (Conv2D)	(None, 16, 16, 128)
block2_conv2 (Conv2D)	(None, 16, 16, 128)
block2_pool (MaxPooling2D)	(None, 8, 8, 128)
block3_conv1 (Conv2D)	(None, 8, 8, 256)
block3_conv2 (Conv2D)	(None, 8, 8, 256)
block3_conv3 (Conv2D)	(None, 8, 8, 256)
block3_pool (MaxPooling2D)	(None, 4, 4, 256)
block4_conv1 (Conv2D)	(None, 4, 4, 512)
block4_conv2 (Conv2D)	(None, 4, 4, 512)
block4_conv3 (Conv2D)	(None, 4, 4, 512)
block4_pool (MaxPooling2D)	(None, 2, 2, 512)
block5_conv1 (Conv2D)	(None, 2, 2, 512)
block5_conv2 (Conv2D)	(None, 2, 2, 512)
block5_conv3 (Conv2D)	(None, 2, 2, 512)
block5_pool (MaxPooling2D)	(None, 1, 1, 512)
flatten_9 (Flatten)	(None, 512)
batch_normalization_16 (Batc	(None, 512)
dense_16 (Dense)	(None, 100)
batch_normalization_17 (Batc	(None, 100)
dense_17 (Dense)	(None, 10)

Imatge 34: Xarxa neuronal adaptada per a classificar Cifar10 a partir d'una VGG16.

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten_1 (Flatten)	(None, 25088)	0
batch_normalization_1 (Batch	(None, 25088)	100352
dense_1 (Dense)	(None, 100)	2508900
dropout_1 (Dropout)	(None, 100)	0
batch_normalization_2 (Batch	(None, 100)	400
dense_2 (Dense)	(None, 10)	1010

Total params: 17,325,350
Trainable params: 9,639,710
Non-trainable params: 7,685,640

Imatge 35: Xarxa neuronal adaptada per a classificar Food101 a partir d'una VGG16.

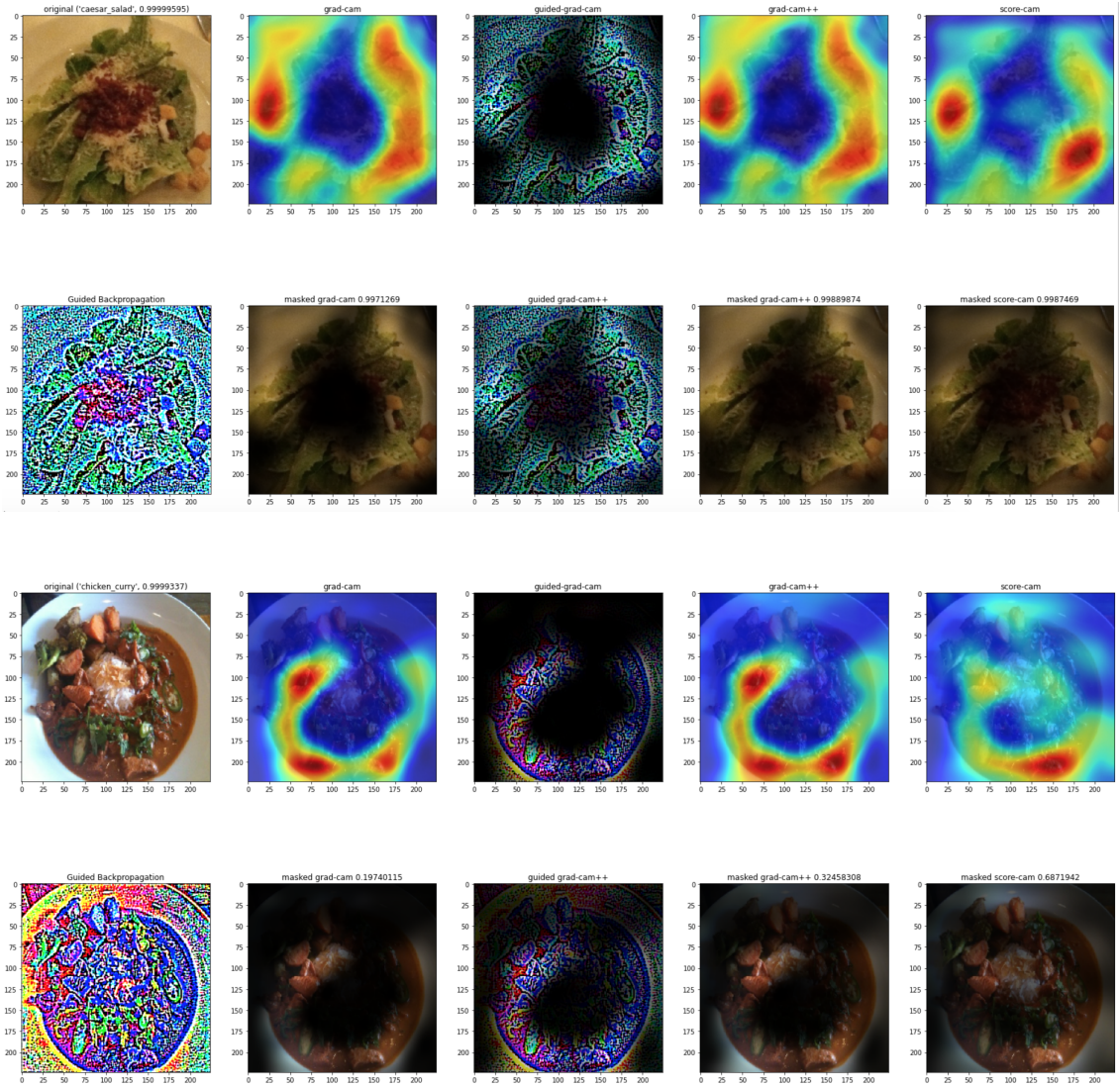
Layer (type)	Output Shape	Param #
input_8 (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten_6 (Flatten)	(None, 25088)	0
batch_normalization_11 (Batc	(None, 25088)	100352
dense_11 (Dense)	(None, 150)	3763350
batch_normalization_12 (Batc	(None, 150)	600
dense_12 (Dense)	(None, 3)	453

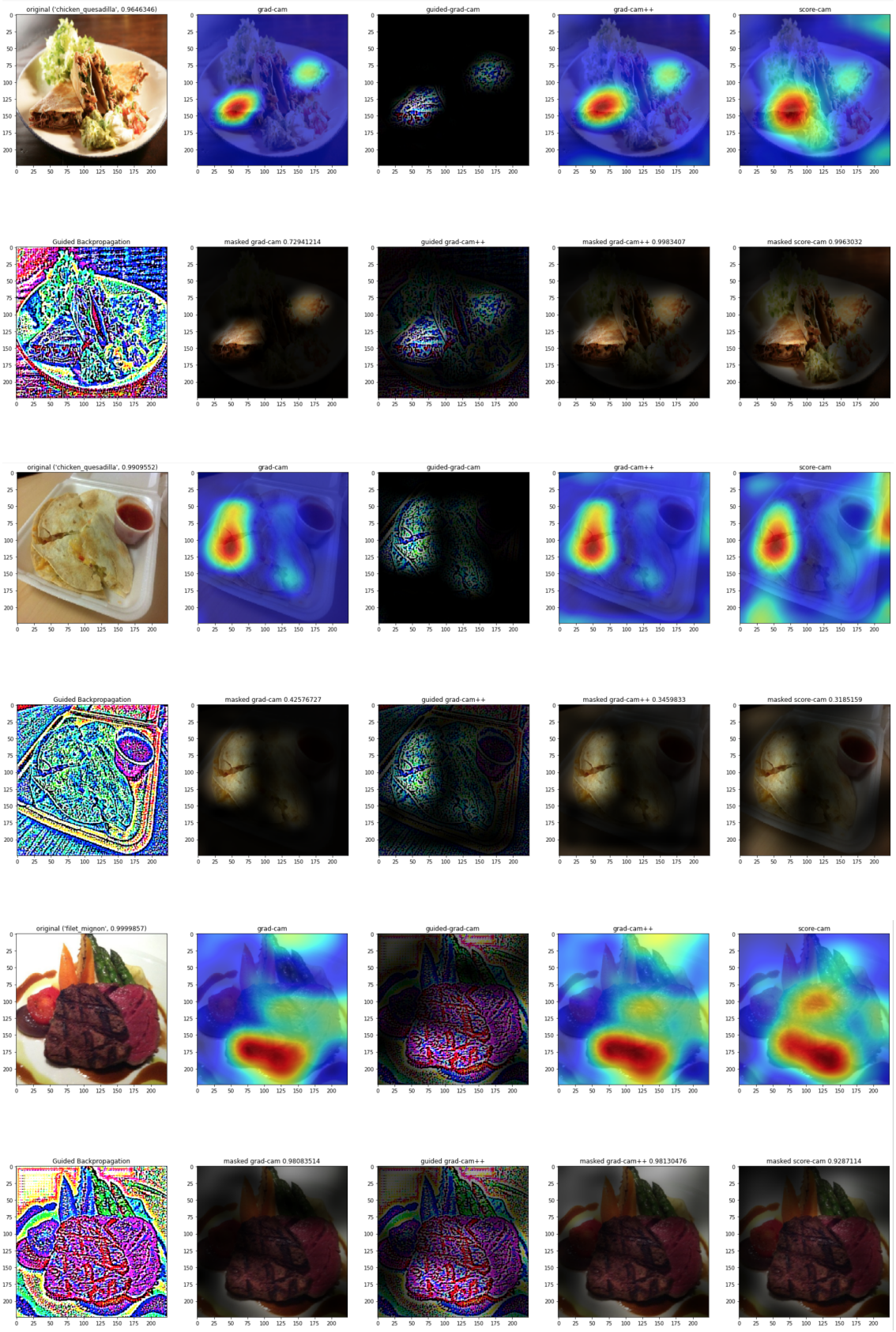
Total params: 18,579,443
Trainable params: 10,893,703
Non-trainable params: 7,685,740

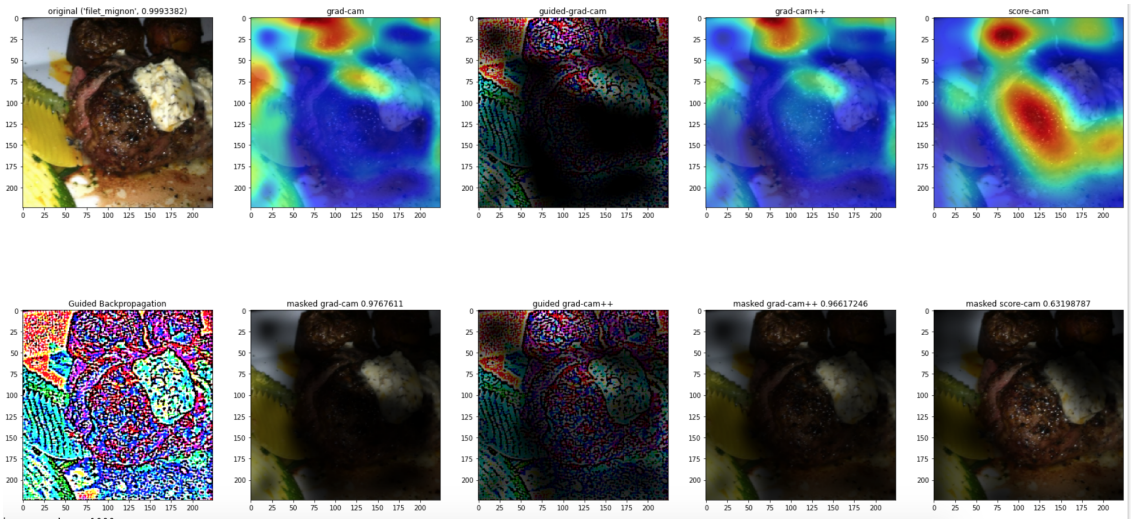
Imatge 36: Xarxa neuronal adaptada per a classificar Food201 a partir d'una VGG16.

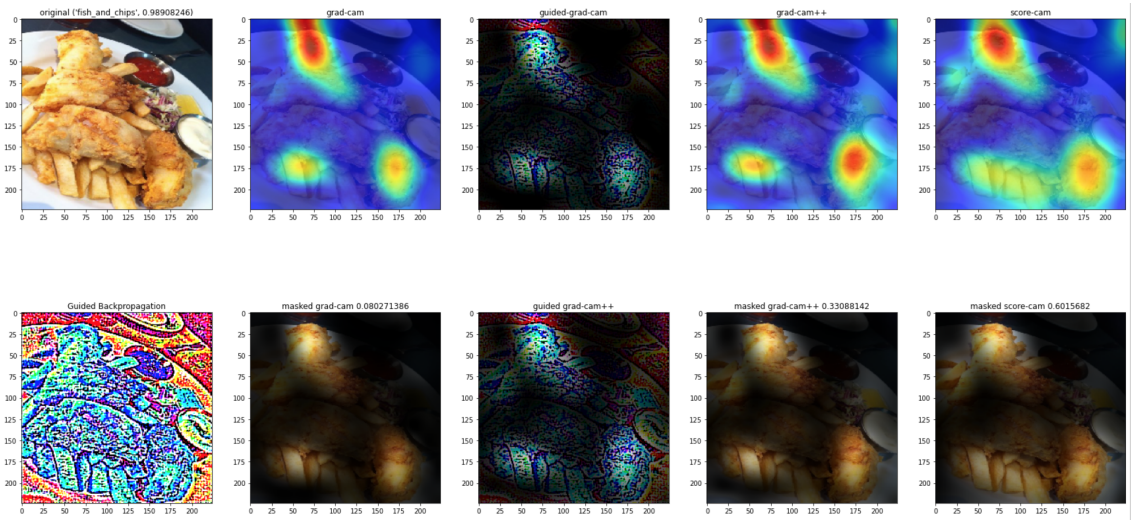
B Imatges de mostra

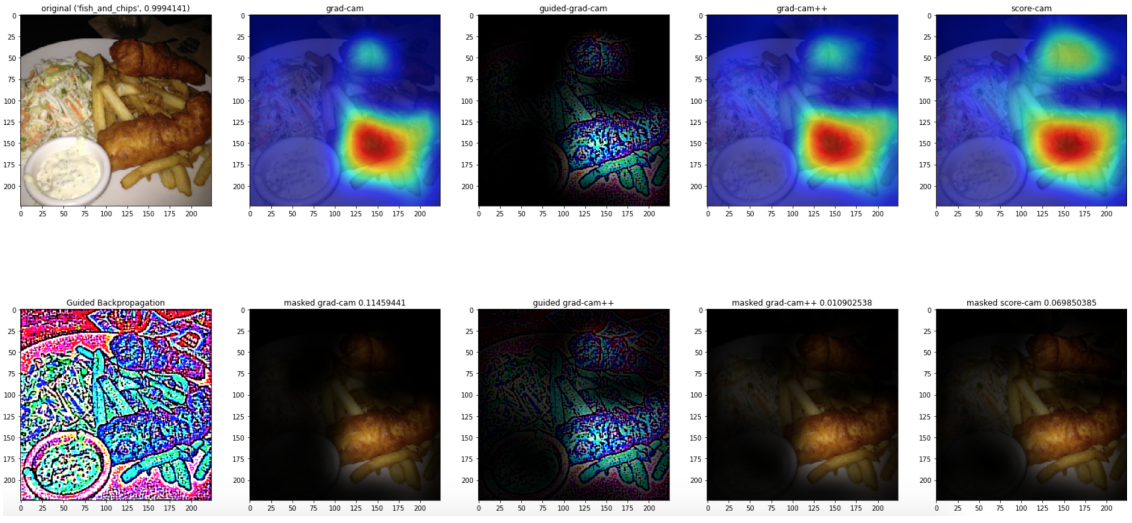
B.1 Food101

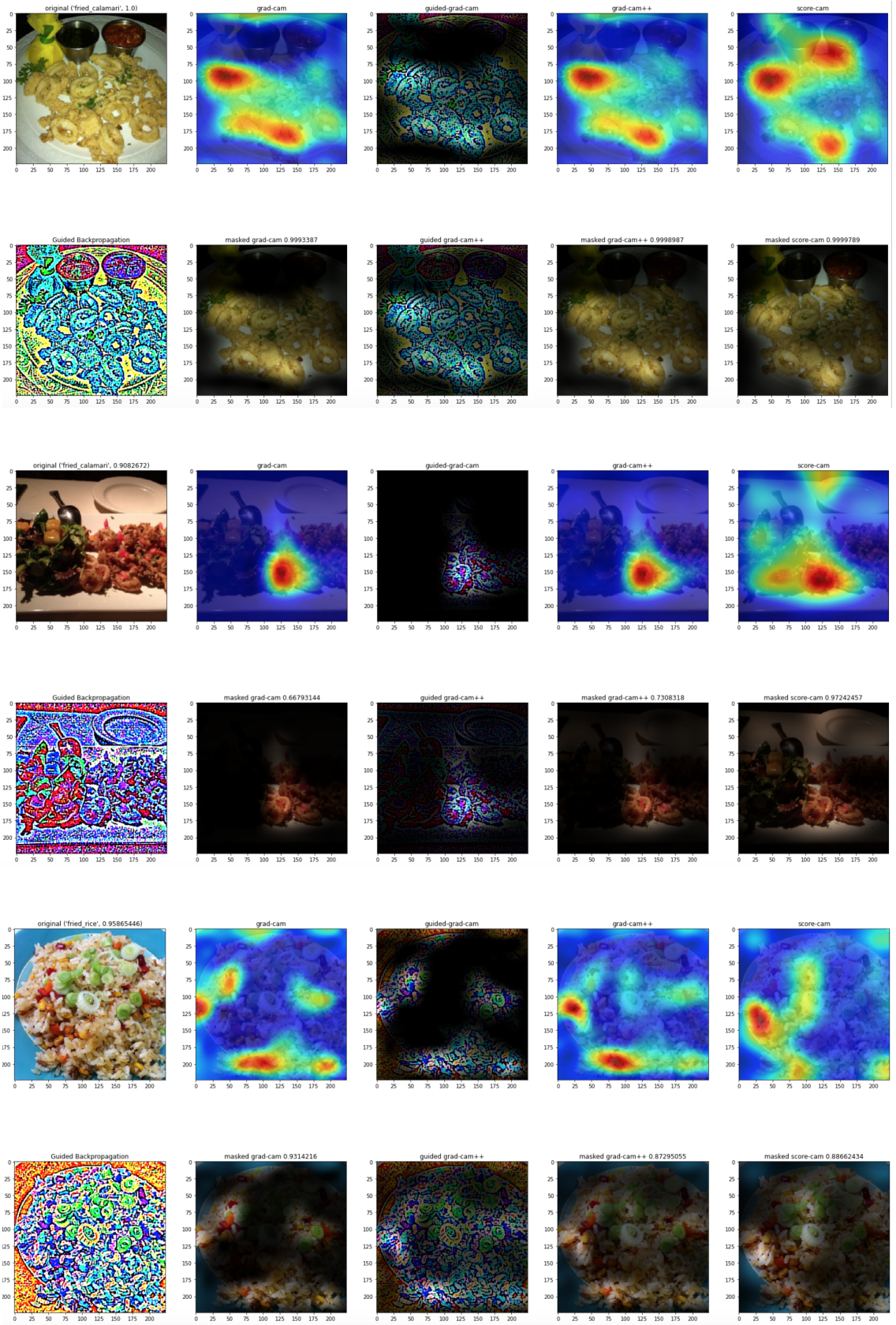


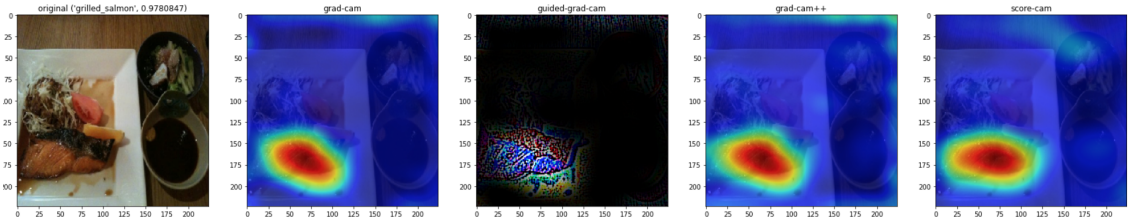

original ('chicken_quesadilla', 0.9646346)
grad-cam
guided-grad-cam
grad-cam++
score-cam
Guided Backpropagation
masked grad-cam 0.72941214
guided grad-cam++
masked grad-cam++ 0.9983407
masked score-cam 0.9963032
original ('chicken_quesadilla', 0.9909552)
grad-cam
guided-grad-cam
grad-cam++
score-cam
Guided Backpropagation
masked grad-cam 0.42576727
guided grad-cam++
masked grad-cam++ 0.3459833
masked score-cam 0.3185159
original ('filet_mignon', 0.9999857)
grad-cam
guided-grad-cam
grad-cam++
score-cam
Guided Backpropagation
masked grad-cam 0.98083514
guided grad-cam++
masked grad-cam++ 0.98130476
masked score-cam 0.9287114

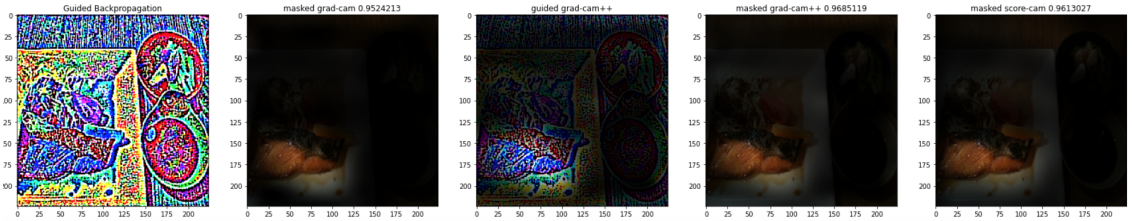

original ('filet_mignon', 0.9993382)
grad-cam
guided-grad-cam
grad-cam++
score-cam
Guided Backpropagation
masked grad-cam 0.9767611
guided grad-cam++
masked grad-cam++ 0.96617246
masked score-cam 0.63198787

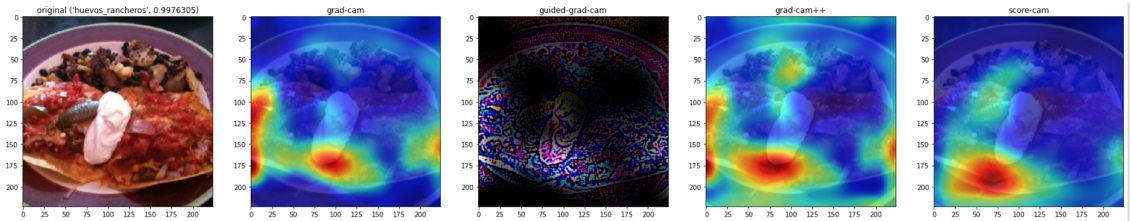

original ('fish_and_chips', 0.98908246)
grad-cam
guided-grad-cam
grad-cam++
score-cam
Guided Backpropagation
masked grad-cam 0.080271386
guided grad-cam++
masked grad-cam++ 0.33088142
masked score-cam 0.6015682

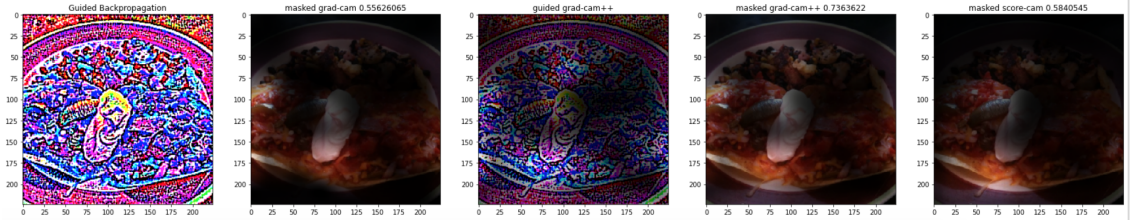

original ('fish_and_chips', 0.9994141)
grad-cam
guided-grad-cam
grad-cam++
score-cam
Guided Backpropagation
masked grad-cam 0.11459441
guided grad-cam++
masked grad-cam++ 0.010902538
masked score-cam 0.069850385


original ('fried_calamari', 1.0)
grad-cam
guided-grad-cam
grad-cam++
score-cam
Guided Backpropagation
masked grad-cam 0.9993387
guided grad-cam++
masked grad-cam++ 0.9998987
masked score-cam 0.9999789
original ('fried_calamari', 0.9082672)
grad-cam
guided-grad-cam
grad-cam++
score-cam
Guided Backpropagation
masked grad-cam 0.66793144
guided grad-cam++
masked grad-cam++ 0.7308318
masked score-cam 0.97242457
original ('fried_rice', 0.95865446)
grad-cam
guided-grad-cam
grad-cam++
score-cam
Guided Backpropagation
masked grad-cam 0.9314216
guided grad-cam++
masked grad-cam++ 0.87295055
masked score-cam 0.88662434

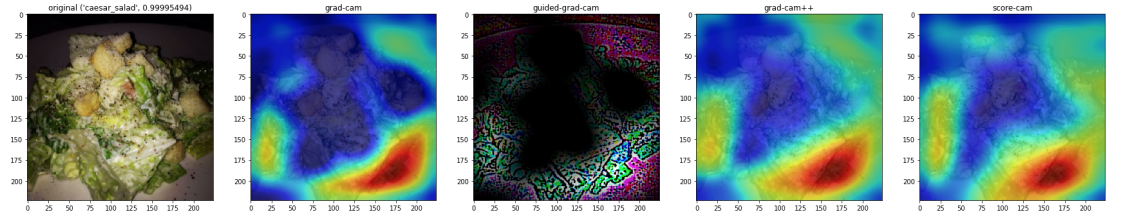


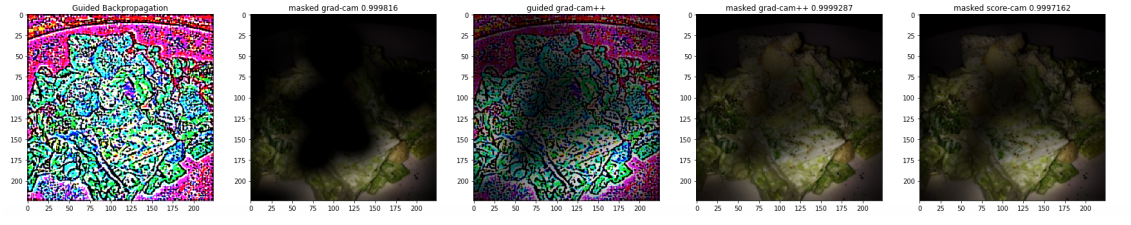


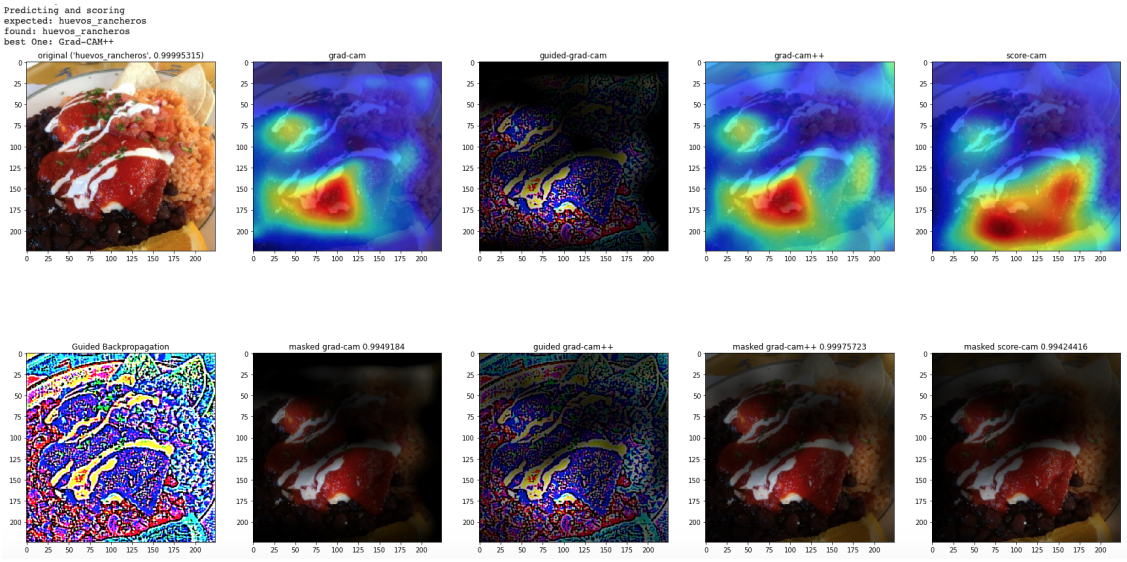


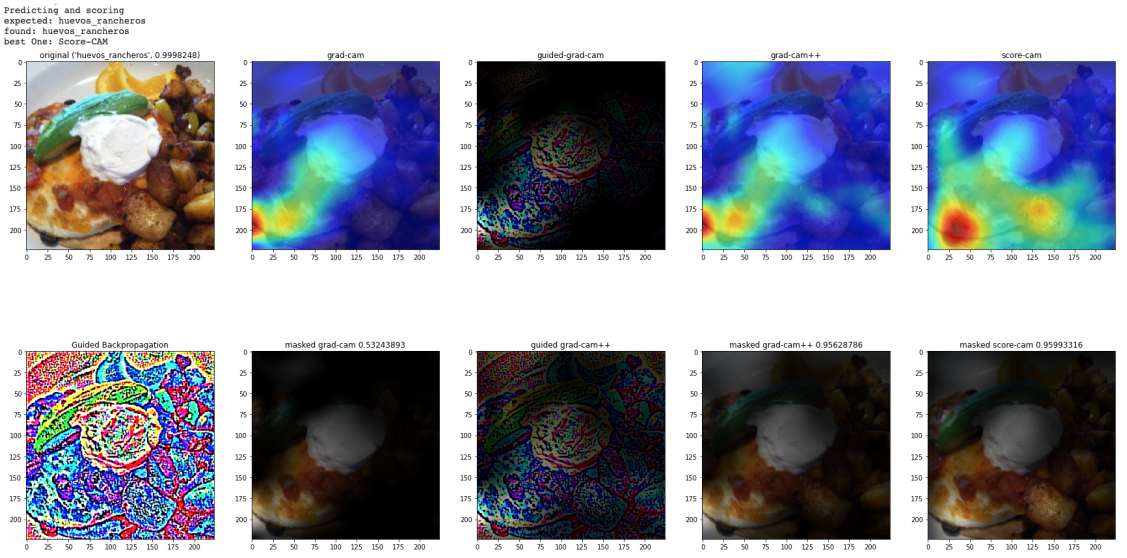


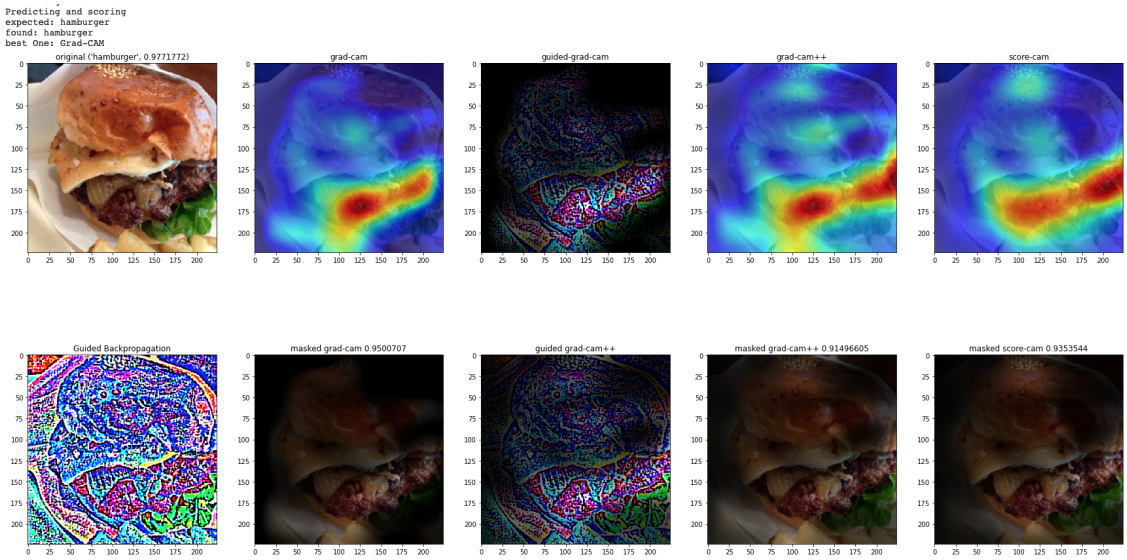

```
Predicting and scoring
expected: caesar_salad
found: caesar_salad
best One: Grad-CAM++
```

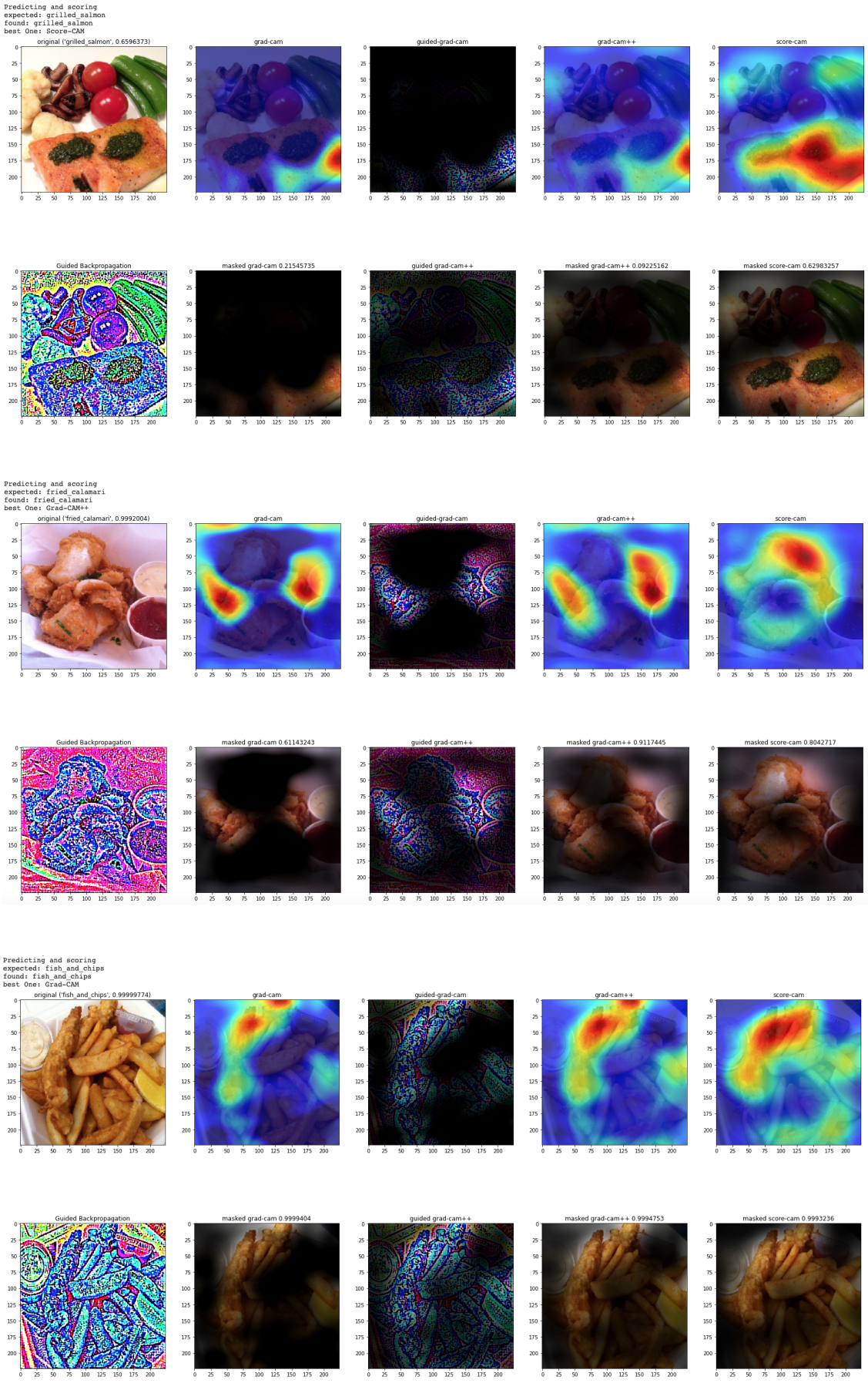


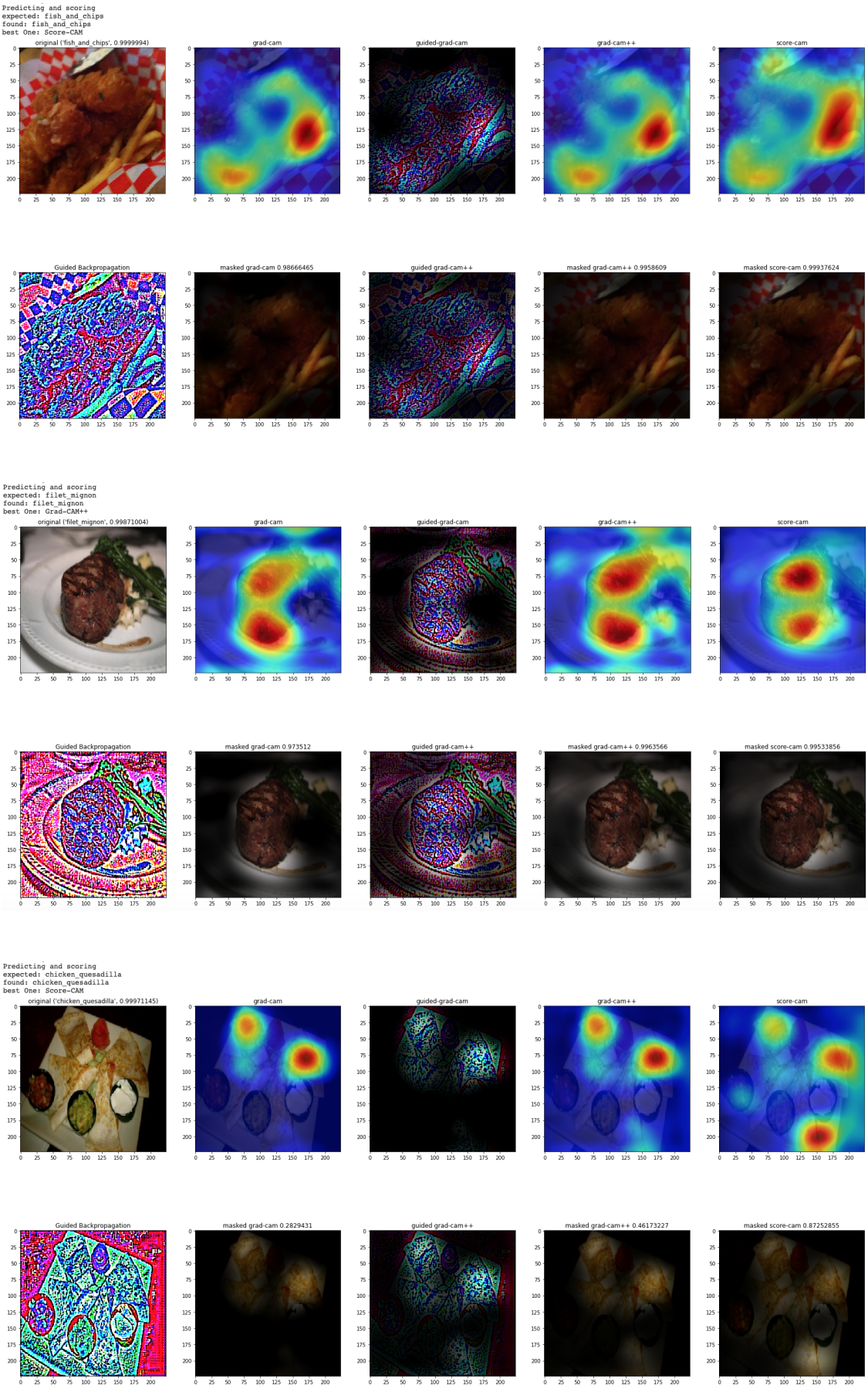


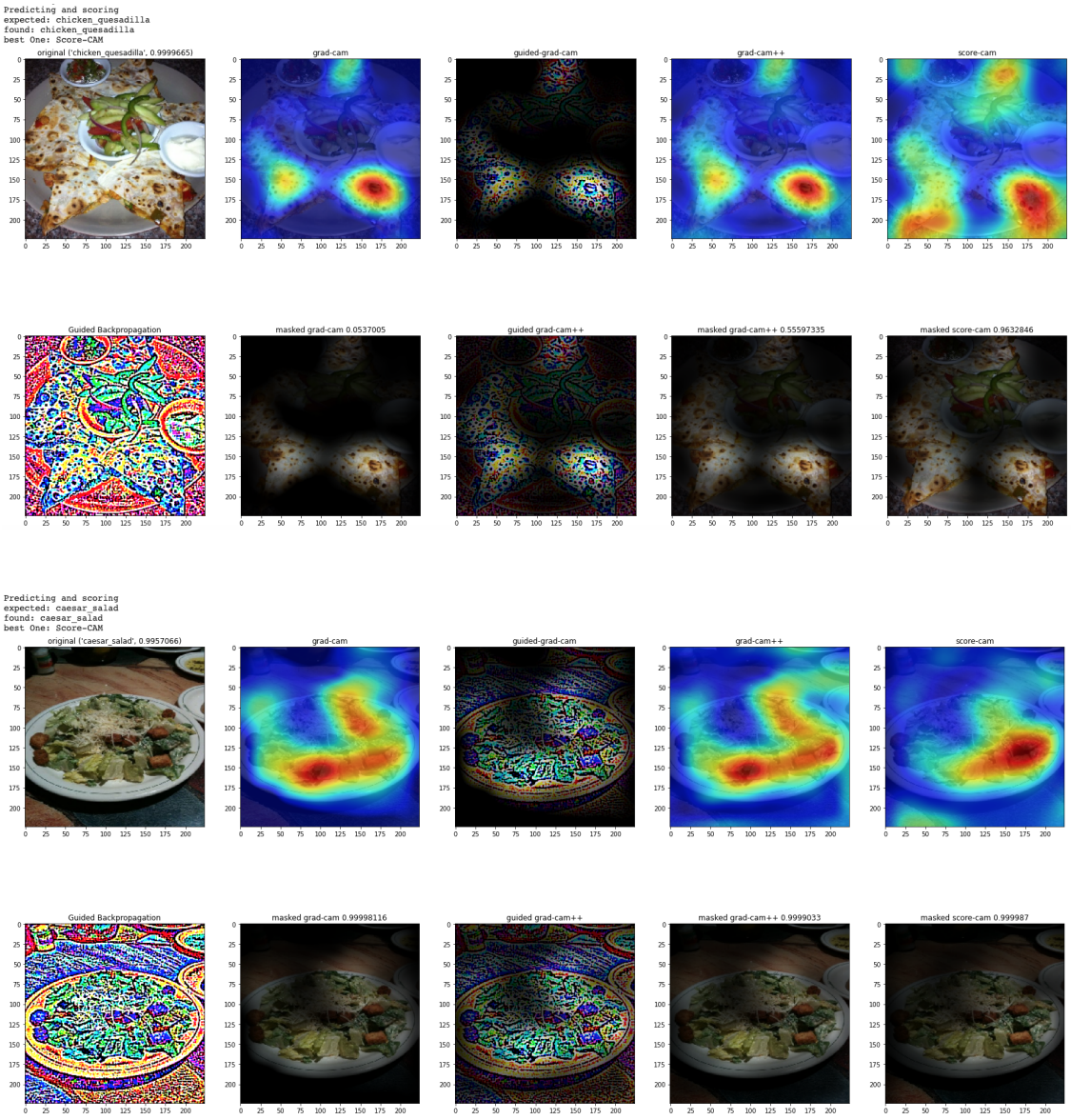

Predicting and scoring
expected: huevos_rancheros
found: huevos_rancheros
best One: Grad-CAM++
original ('huevos_rancheros', 0.99995315)
grad-cam
guided-grad-cam
grad-cam++
score-cam
Guided Backpropagation
masked grad-cam 0.9949184
guided grad-cam++
masked grad-cam++ 0.99975723
masked score-cam 0.99424416


Predicting and scoring
expected: huevos_rancheros
found: huevos_rancheros
best One: Score-CAM
original ('huevos_rancheros', 0.9998248)
grad-cam
guided-grad-cam
grad-cam++
score-cam
Guided Backpropagation
masked grad-cam 0.53243893
guided grad-cam++
masked grad-cam++ 0.95628786
masked score-cam 0.95993316


Predicting and scoring
expected: hamburger
found: hamburger
best One: Grad-CAM
original ('hamburger', 0.9771772)
grad-cam
guided-grad-cam
grad-cam++
score-cam
Guided Backpropagation
masked grad-cam 0.9500707
guided grad-cam++
masked grad-cam++ 0.91496605
masked score-cam 0.9353544

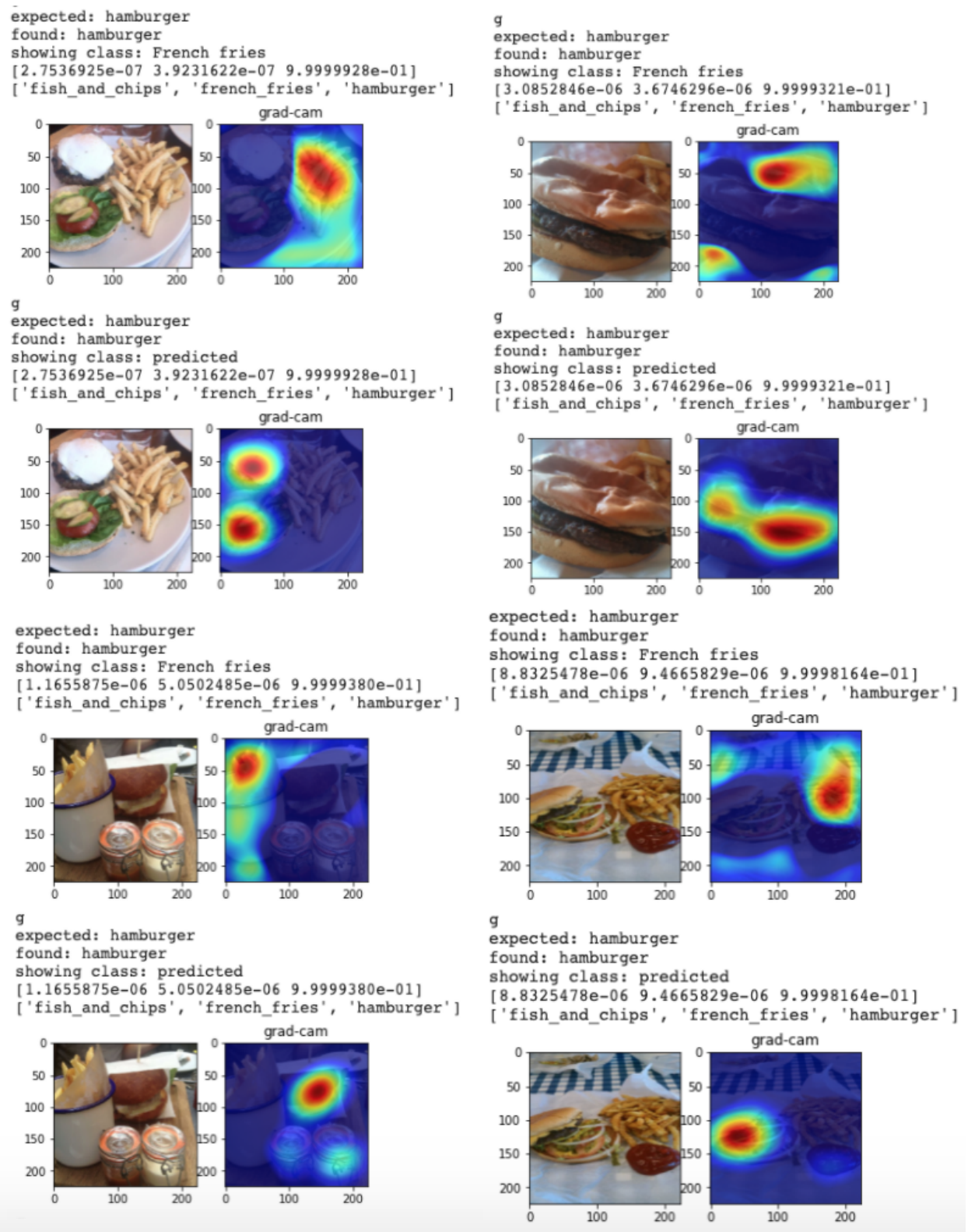

Predicting and scoring
expected: grilled_salmon
found: grilled_salmon
best One: Score-CAM
original ('grilled_salmon', 0.6596373)
grad-cam
guided-grad-cam
grad-cam++
score-cam
Guided Backpropagation
masked grad-cam 0.21545735
guided grad-cam++
masked grad-cam++ 0.09225162
masked score-cam 0.62983257
Predicting and scoring
expected: fried_calamari
found: fried_calamari
best One: Grad-CAM++
original ('fried_calamari', 0.9992004)
grad-cam
guided-grad-cam
grad-cam++
score-cam
Guided Backpropagation
masked grad-cam 0.61143243
guided grad-cam++
masked grad-cam++ 0.9117445
masked score-cam 0.8042717
Predicting and scoring
expected: fish_and_chips
found: fish_and_chips
best One: Grad-CAM
original ('fish_and_chips', 0.99999774)
grad-cam
guided-grad-cam
grad-cam++
score-cam
Guided Backpropagation
masked grad-cam 0.9999404
guided grad-cam++
masked grad-cam++ 0.9994753
masked score-cam 0.9993236


Predicting and scoring
expected: fish_and_chips
found: fish_and_chips
best One: Score-CAM
original ('fish_and_chips', 0.9999994)
grad-cam
guided-grad-cam
grad-cam++
score-cam
Guided Backpropagation
masked grad-cam 0.98666465
guided grad-cam++
masked grad-cam++ 0.9958609
masked score-cam 0.99937624
Predicting and scoring
expected: filet_mignon
found: filet_mignon
best One: Grad-CAM++
original ('filet_mignon', 0.99871004)
grad-cam
guided-grad-cam
grad-cam++
score-cam
Guided Backpropagation
masked grad-cam 0.973512
guided grad-cam++
masked grad-cam++ 0.9963566
masked score-cam 0.99533856
Predicting and scoring
expected: chicken_quesadilla
found: chicken_quesadilla
best One: Score-CAM
original ('chicken_quesadilla', 0.99971145)
grad-cam
guided-grad-cam
grad-cam++
score-cam
Guided Backpropagation
masked grad-cam 0.2829431
guided grad-cam++
masked grad-cam++ 0.46173227
masked score-cam 0.87252855

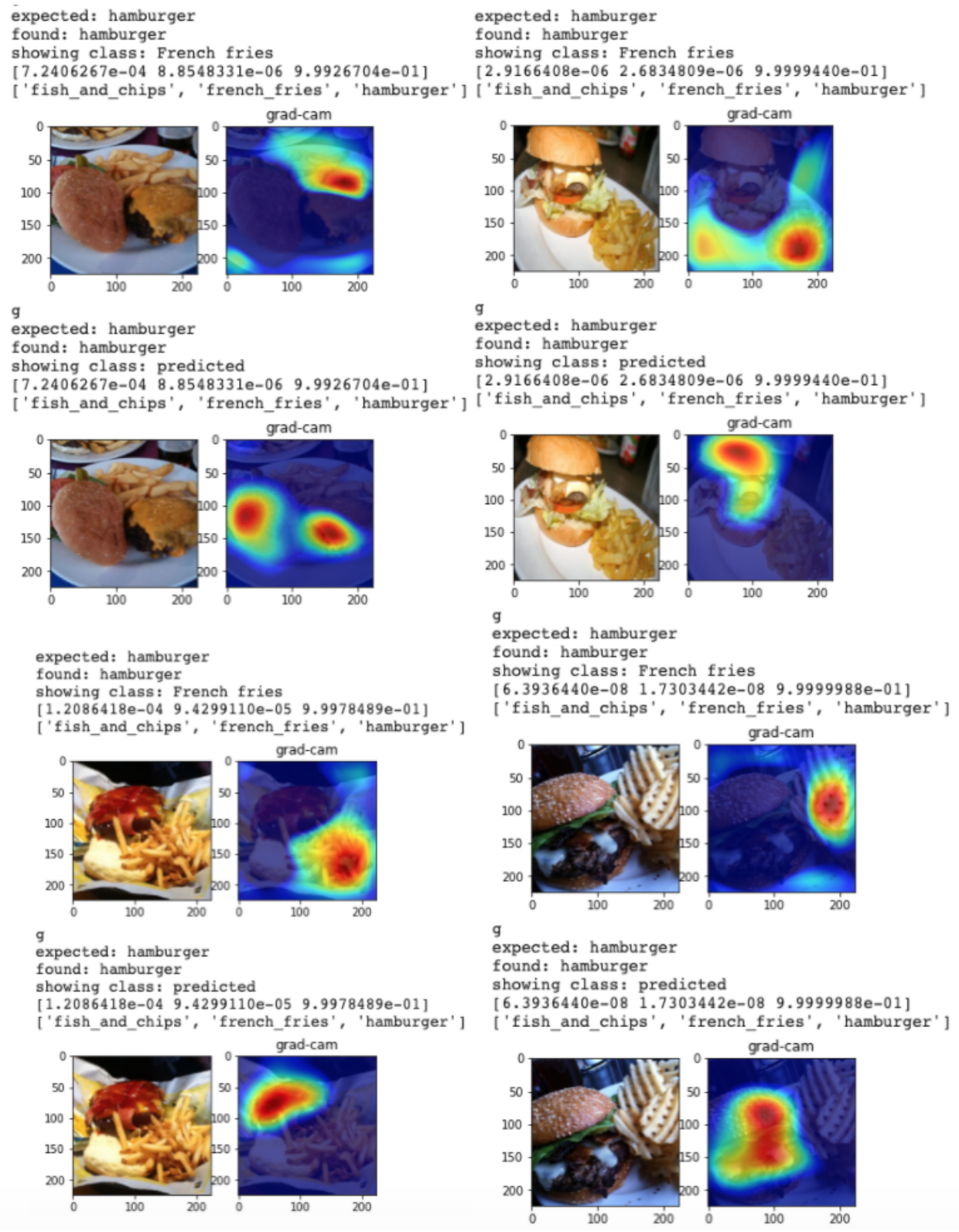


Imatge 37: Imatges de mostra d'explicabilitat sobre Food101.

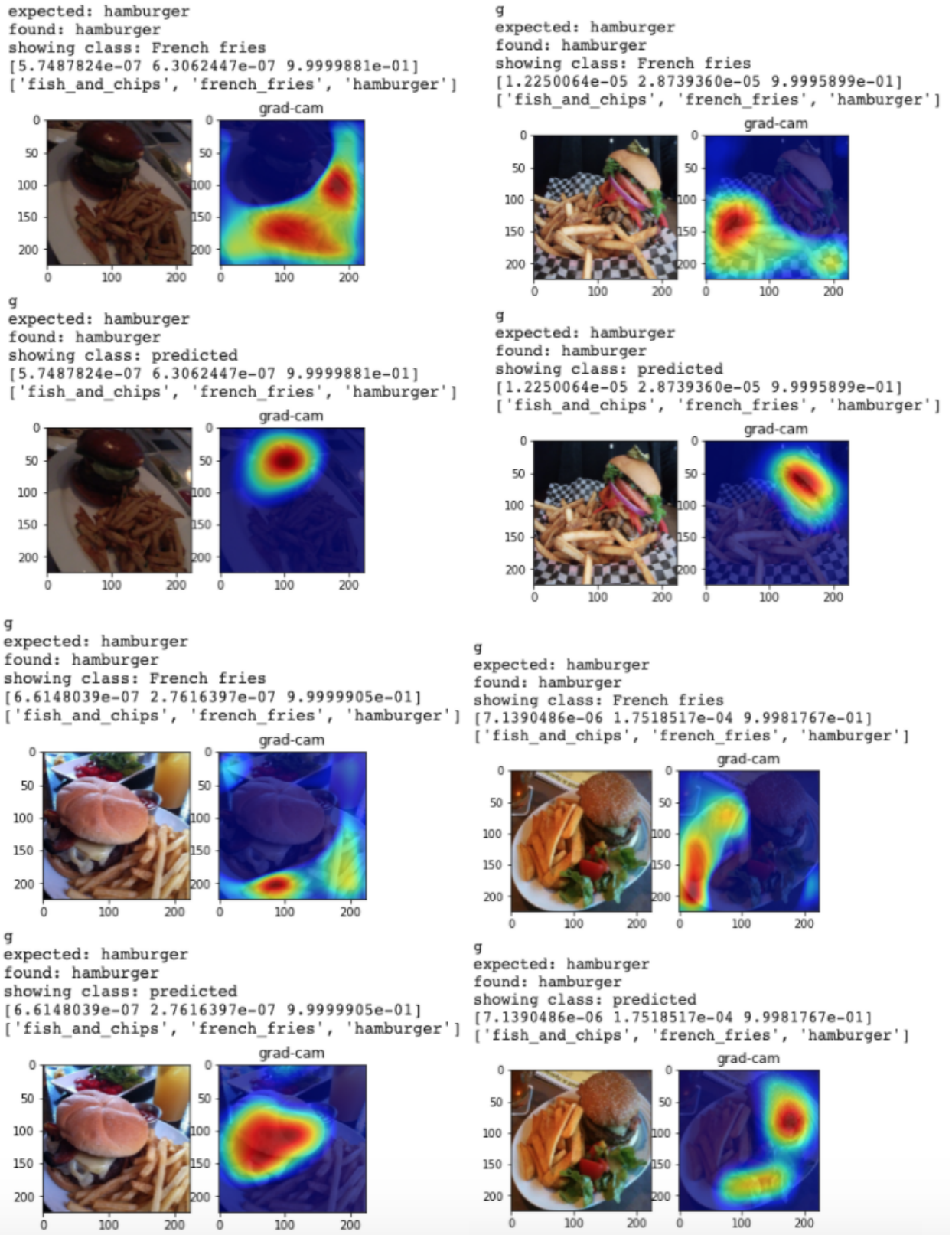
B.2 Food201



Imatge 38: Imatges de Food201 amb explicabilitat sobre patates (dalt) i hamburgueses (baix). Amb puntuacions per classe.



Imatge 39: Imatges de Food201 amb explicabilitat sobre patates (dalt) i hamburgueses (baix). Amb puntuacions per classe.



Imatge 40: Imatges de Food201 amb explicabilitat sobre patates (dalt) i hamburgueses (baix). Amb puntuacions per classe.

Bibliografia

[1] Prantik Howlader Vineeth N Balasubramanian Aditya Chattopadhyay, Anirban Sarkar, *Grad-cam++: Improved visual explanations for deep convolutional networks*, https://arxiv.org/abs/1710.11063, 2017.

[2] Vivek Rathod Anoop Korattikara Alex Gorban Nathan Silberman Sergio Guadarrama George Papandreou Jonathan Huang Kevin Murphy Austin Myers, Nick Johnston, *Im2calories: towards an automated mobile vision food diary*, https://static.googleusercontent.com/media/research.google.com/es//pubs/archive/44321.pdf, 2015.

[3] Agata Lapedriza Aude Oliva i Antonio Torralba Bolei Zhou, Aditya Khosla, *Learning deep features for discriminative localization*, https://arxiv.org/abs/1512.04150, 2015.

[4] Lukas Bossard, Matthieu Guillaumin, and Luc Van Gool, *Food-101 – mining discriminative components with random forests*, European Conference on Computer Vision, 2014.

[5] Jason Brownlee, *A gentle introduction to dropout for regularizing deep neural networks*, https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/, 2018.

[6] ———, *A gentle introduction to the rectified linear unit (relu)*, https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/, 2019.

[7] Andrew Dabydeen, *Transfer learning using resnet50 and cifar-10*, https://medium.com/@andrew.dabydeen/transfer-learning-using-resnet50-and-cifar-10-6242ed4b4245, 2019.

[8] James Dellinger, *Weight initialization in neural networks: A journey from the basics to kaiming*, https://towardsdatascience.com/weight-initialization-in-neural-networks-a-journey-from-the-basics-to-kaiming-954fb9b47c79, 2019.

[9] Xavier Glorot, *Understanding the difficulty of training deep feedforward neural networks*, http://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf.

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville, *Deep learning*, MIT Press, 2016, http://www.deeplearningbook.org.

[11] Mengnan Du Fan Yang Zijian Zhang Sirui Ding Piotr Mardziel Xia Hu Haofan Wang, Zifan Wang, *Score-cam: Score-weighted visual explanations for convolutional neural networks*, https://arxiv.org/abs/1910.01279, 2019.

[12] Onel Harrisson, *Machine learning basics with the k-nearest neighbors algorithm*, https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761, 2018.

[13] Sergey Ioffe i Christian Szegedy, *Batch normalization: Accelerating deep network training b y reducing internal covariate shift*, https://arxiv.org/abs/1502.03167, 2015.

[14] Terence Parr i Jeremy Howard, *The matrix calculus you need for deep learning*, https://explained.ai/matrix-calculus/.

[15] George V Jose, *Useful plots to diagnose your neural network*, https://towardsdatascience.com/useful-plots-to-diagnose-your-neural-network-521907fa2f45, 2019.

[16] Thomas Brox Martin Riedmiller Jost Tobias Springenberg, Alexey Dosovitskiy, *Striving for simplicity: The all convolutional net*, https://arxiv.org/abs/1412.6806, 2014.

[17] Shaoqing Ren i Jian Sun. Kaiming He, Xiangyu Zhang, *Delving deep into rectifiers: Surpassing human-level performance on imagenet classification*, https://arxiv.org/abs/1502.01852, 2015.

[18] Frederik Kratzert, *Understanding the backward pass through batch normalization layer*, https://kratzert.github.io/2016/02/12/understanding-the-gradient-flow-through-the-batch-normalization-layer.html, 2016.

[19] Suki Lau, *Learning rate schedules and adaptive learning rate methods for deep learning*, https://towardsdatascience.com/learning-rate-schedules-and-adaptive-learning-rate-methods-for-deep-learning-2c8f433990d1, 2017.

[20] Assaad MOAWAD, *Neural networks and back-propagation explained in a simple way*, https://medium.com/datathings/neural-networks-and-backpropagation-explained-in-a-simple-way-f540a3611f5e, 2018.

[21] Magnus Erik Hvass Pedersen, https://github.com/Hvass-Labs/TensorFlow-Tutorials/blob/master/10_Fine-Tuning.ipynb.

[22] Abhishek Das Ramakrishna Vedantam Devi Parikh Dhruv Batra Ramprasaath R. Selvaraju, Michael Cogswell, *Grad-cam: Visual explanations from deep networks via gradient-based localization*, https://arxiv.org/abs/1610.02391, 2016.

[23] Adrian Rosebrock, *Understanding regularization for image classification and machine learning*, https://www.pyimagesearch.com/2016/09/19/understanding-regularization-for-image-classification-and-machine-learning/, 2016.

[24] Sumit Saha, *A comprehensive guide to convolutional neural networks — the eli5 way*, https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53, 2018.

[25] ALAKH SETHI, *Supervised learning vs. unsupervised learning – a quick guide for beginners*, https://www.analyticsvidhya.com/blog/2020/04/supervised-learning-unsupervised-learning/.

[26] Tavish Srivastava, *Introduction to k-nearest neighbors: A powerful machine learning algorithm (with implementation in python*, https://www.analyticsvidhya.com/blog/2018/03/introduction-k-neighbours-algorithm-clustering/, 2018.

[27] tabayashi0117, https://github.com/tabayashi0117/Score-CAM.

[28] Stanford University, *Cs231n: Convolutional neural networks for visual recognition*, http://cs231n.stanford.edu/syllabus.html, 2020.

[29] yosuke toda, https://github.com/totti0223/gradcamplusplus.