Universitat Oberta de Catalunya (UOC)

Màster Universitari en Ciència de Dades (*Data Science*)

TREBALL FINAL DE MÀSTER

Classificació del Melanoma mitjançant visualització artificial

Autor: Pol Casellas La Rosa

Tutor: Jordi de la Torre Gallart

Vic, 3 de gener de 2021

Abstract

Es denomina càncer al conjunt de les malalties en les quals s'observa un procés descontrolat en la divisió cel·lular en qualsevol part del cos. Avui en dia, el càncer és una de les malalties amb major índex de mortalitat.

Més concretament, el càncer de pell és la tipologia de càncer més habitual entre la població humana. El melanoma n'és la forma més mortal, causant un 75% de les morts per càncer epitelial. Tot i l'elevat índex de mortalitat, de la mateixa manera que en altres càncers, una ràpida detecció pot provocar una major efectivitat del tractament.

Actualment, l'avaluació del diagnòstic del càncer de pell per part dels dermatòlegs ha millorat notablement gràcies a la dermatoscòpia. No obstant, es podria millorar significativament la precisió dels diagnòstics mitjançant algoritmes de classificació d'imatges.

Les eines d'anàlisis d'imatges que automatitzen el diagnòstic de melanoma tenen un gran potencial per a millorar la precisió dels diagnòstics dels dermatòlegs i contribuir en una ràpida detecció del melanoma augmentant, d'aquesta manera, la probabilitat de cura de la melanoma a milions de persones.

Al llarg d'aquest projecte s'utilitzen conjunts d'imatges de pacients per tal de determinar l'existència o no de melanoma mitjançant algoritmes de classificació d'imatges.

Agraïments

L'elaboració d'aquest treball ha estat possible gràcies a l'ajuda i el suport de diverses persones; a totes elles gràcies. En particular:

M'agradaria donar les gràcies a Jordi de la Torre, el tutor del treball, qui m'ha acompanyat en aquest bonic viatge al món de la visualització artificial i m'ha ajudat sempre que ho he necessitat.

Tot el que tinc els hi dec als meus pares i a la meva germana petita Laia, sense vosaltres res hagués estat possible.

Vull agrair a l'Ariadna per estar sempre al meu costat i donar-me el seu suport, fins i tot quan les coses no sortien tal i com esperava.

Per altre banda, agrair a un bon amic com és l'Oriol Barbany per a aportar-me una visió externa crítica i orientadora.

També m'agradaria recordar tots i cada un dels professors que al llarg del màster m'han transmès els seus valuosos coneixements, així com la seva passió per a la ciència de dades.

Finalment, donar les gràcies als amics de tota la vida per confiar incondicionalment en mi com a científic de dades.

Índex

Índex de figures

Índex de taules

Capítol 1

Introducció

El melanoma és la forma de càncer de pell més mortal, concretament causa el 75% de les morts per càncer epitelial. Igual que en la resta de càncers, una ràpida detecció pot provocar una major efectivitat en el tractament. Al llarg de l'estudi es pretén desenvolupar una eina automatitzada capaç d'identificar el melanoma en fotografies de taques a la pell.

Motivació i objectius

Aquest projecte de ciència de dades aplicat a les ciències de la salut i més concretament a la dermatologia pretén col·laborar mitjançant algoritmes d'intel·ligència artificial en el diagnòstic del melanoma, contribuint d'aquesta manera en la millora de la salut de milions de persones.

Motivació

De totes les branques de la ciència de dades, una de les quals he gaudit més ha estat el *Deep Learning* o aprenentatge profund. Aquest fet, m'encoratge a enfrontar-me a un problema tan complex i alhora tan apassionant com és la detecció del melanoma en imatges de lesions cutànies. Per altra banda, tot hi no haver participat mai en cap projecte de ciències de la salut, resulta evident que l'idea de contribuir en la millora de la salut de milions de persones és altament motivadora.

Objectius

L'objectiu d'aquest projecte és desenvolupar una eina automatitzada d'anàlisis d'imatges que sigui capaç d'identificar el melanoma en fotografies de taques a la pell. El melanoma és una malaltia mortal, però, com la resta de càncers, una ràpida i precisa detecció pot fer que el

tractament sigui més eficaç i d'aquesta manera augmentar la probabilitat que el pacient superi la malaltia.

Objectiu principal

Desenvolupament mitjançant algoritmes d'aprenentatge profund d'una eina automàtica capaç d'identificar el melanoma en fotografies de taques a la pell.

Objectius parcials

Per tal de poder desenvolupar l'algoritme d'intel·ligència artificial, abans és necessari resoldre problemes parcials:

- Netejar i transformar de manera òptima el conjunt de dades.
- Trobar els models i hiperparàmetres que optimitzin els resultats.

Metodologia

Aquest projecte es centra en el desenvolupament, mitjançant el llenguatge de programació *Python*, d'una eina automàtica d'anàlisis d'imatges capaç d'identificar el melanoma en fotografies de taques a la pell.

Per fer-ho s'utilitzaran mètodes de programació àgil, adaptant el codi a qualsevol mena de canvi que pugui sorgir durant el projecte per tal d'augmentar la probabilitat d'èxit. En el transcurs del projecte es desglossarà l'objectiu principal en petits objectius o fites a complir en curts terminis de temps.

1.1 Abast del projecte

L'abast d'aquest projecte es centra en la competició *Kaggle* de l'any 2020 per a la classificació del melanoma. Aquesta competició organitzada per a la *Society for Imaging Informatics in Medicine (SIIM)* i la *International Skin Imaging Collaboration (ISIC)* té per objectiu el desenvolupament d'una eina automàtica d'anàlisis d'imatges que sigui capaç d'identificar el melanoma en fotografies de lesions epitelials.

Per tal d'aconseguir aquest objectiu es faciliten prop de 33.000 diagnòstics de pacients formats per imatges i metadades (Identificador del pacient, gènere, edat, lloc anatòmic de la lesió)

etiquetades en dues categories: Sa i Melanoma.

El conjunt de test el qual l'eina automàtica pretén predir la seva categoria està format per a 10.982 diagnòstics. L'objectiu principal d'aquest treball es centra a obtenir una categorització el màxim fiable possible dels diagnòstics anteriors.

Capítol 2

Estat de l'art

El càncer[5] és una malaltia que s'origina quan les cèl·lules d'una part del cos creixen de forma extraordinària i descontrolada; aquestes cèl·lules són conegudes com a cèl·lules cancerígenes. L'alta mortalitat d'aquesta malaltia és conseqüència de que aquestes cèl·lules utilitzen els nutrients de la resta de cèl·lules debilitant-les fins al punt de poder provocar la seva mort. D'aquesta manera el sistema immunitari de l'ésser humà afectat és debilitat, essent incapaç de protegir el cos.

El que es coneix com a càncer o tumor maligne s'inicia quan les cèl·lules cancerígenes fan metàstasis, és a dir, són capaces de reproduir-se en altres parts del cos. El càncer pot afectar a diferents parts del cos; els pulmons, el pit, el fetge, la pròstata, la pell, etc.

Aquest treball es centra amb el càncer de pell i, més concretament, amb el melanoma. Actualment el càncer de pell es classifica en dos grups; no melanoma i melanoma. Aquest segon és la tipologia més mortal de càncer epitelial i el que s'expandeix de forma més ràpida en l'organisme afectat. És per aquest motiu que el següent projecte es focalitza en la detecció del melanoma a partir de taques en la pell.

Actualment, en la detecció del melanoma, una eina molt utilitzada pels dermatòlegs és la regla del ABCDE:

- **A** (*Asymmetry*): Grau d'asimetria de les taques.
- **B** (*Border*): Taques amb límits irregulars.
- **C** (*Colour*): Varietat de colors en les taques.
- **D** (*Diameter*): Diàmetre de les taques.

- **E** (*Evolution*): Canvis en les taques.

En aquest aspecte, *Dang N. H. Thanh*[5] utilitza tècniques d'aprenentatge profund de preprocessament, segmentació i detecció d'imatges per tal d'establir, mitjançant la regla del ABCDE, una puntuació numèrica a cadascuna de les imatges estudiades. A partir d'aquesta puntuació l'autor classifica les imatges en funció de la presència o absència del melanoma.

Tot i que el mètode de classificació d'imatges sigui de caire més tradicional i no utilitzi xarxes neuronals per a la classificació d'imatges, el tractament previ de les imatges resulta d'interès pel desenvolupament d'aquest treball.

A continuació, s'analitzen els mètodes utilitzats i passos seguits per a diferents autors per tal de maximitzar la precisió dels models de classificació d'imatges basat en l'aprenentatge profund.

2.1 Augment del conjunt de dades

L'augment del conjunt de dades es basa en aplicar transformacions a les imatges del conjunt per tal de crear variacions als objectes sense afectar la semàntica a alt nivell d'aquesta. Seguidament, es llisten alguns dels beneficis d'aquesta pràctica:

- Permet ampliar el conjunt de dades d'entrenament i afegir diversitat a les dades.
- Produeix un afecte de regularització, ajudant al model a generalitzar de forma més eficaç.
- Augmenta la precisió del model.
- Dificulta el sobrentrenament.
- Col·labora en la tolerància d'errors.
- Millora la robustesa del model.
- Incrementa la velocitat d'aprenentatge del model

En aquest aspecte, *Masdevallia*[6], *Bo*[3] i *Deotte*[4] apliquen tècniques molt similars per tal d'augmentar les dades del conjunt:

- Rotacions aleatòries de les imatges.
- Modificacions d'inclinació.

- Girs horitzontals.
- Girs verticals.
- Aplicacions de zoom.
- Alteracions del color i la llum.
- Saturació d'imatges.

2.2 Preprocessat d'imatges

Treballar amb imatges de la pell humana comporta certes dificultats[1] ja que habitualment les dades o imatges procedeixen de diferents fonts, on les fotografies s'han obtingut amb condicions desiguals. Aquest fet pot generar a un seguit de problemàtiques:

- **Variacions de mida i forma**: La baixa homogeneïtat de mida i forma entre les diferents lesions de pell provoquen que la identificació de la lesió sigui un problema complex.
- **Soroll i presència d'artefactes**: S'entén per soroll aquells objectes introduïts en el moment de la captura de la imatge. En imatges de lesions epitelials és comú trobar-hi sorolls en forma de pèls, vasos sanguinis o butllofes.
- **Límits borrosos i irregulars**: Algunes imatges de lesions de la pell es caracteritzen per a tenir límits borrosos o irregulars, dificultant així la localització dels límits de la lesió.
- **Baix contrast**: En alguns casos existeix poc contrast de colors entre la lesió i la pell sana, complicant d'aquesta manera la correcta identificació de la lesió.
- **Il·luminació**: El color de la imatge de la lesió, els rajos de llum o els reflexos poden afectar a la il·luminació de les imatges demoscòpiques.

Per tal de poder utilitzar les imatges en models d'aprenentatge profund és necessari netejar, normalitzar i estandarditzar les imatges del conjunt d'entrada.

2.2.1 Presència de pèl

Com s'ha explicat anteriorment, les imatges mèdiques de lesions de pell acostumen a estar cobertes per artefactes capil·lars; en algunes imatges el pèl està sobreposat a l'àrea de la lesió, mentre que en altres casos no. La presència de pèl pot afectar la precisió de les eines automatitzades de diagnòstic.

En aquest aspecte existeixen diferents mètodes de tractament del pèl per tal de millorar la precisió de la classificació del model:

- **Detecció i eliminació del pèl**: aquest mètode es basa en identificar el pèl de les imatges i aplicar interpolacions per tal d'eliminar-lo. En aquest sentit, *Dang N. H. Thanh*[5] utilitza una curvatura adaptativa per detectar els pèls i aplicar-hi mètodes de pintat. Els mètodes de pintat de la regió afectada pel pèl són una tasca gens trivial.

- **Augment de pèl**: d'entrada l'augment de pèl pot semblar un mètode per a dificultar l'aprenentatge del model, ja que sembla que el soroll afegit pot afectar el rendiment del model en les èpoques d'entrenament. No obstant això, l'augment del pèl col·labora en evitar el sobrentrenament ajudant al model a generalitzar millor i, d'aquesta manera, contribuir en el ràpid aprenentatge de l'algoritme.

  En aquest sentit, *Masdevallia*[6] utilitza una metodologia per a la simulació pèl; a partir de les dades d'entrada s'identifiquen i es segmenten màscares capil·lars reals que, posteriorment, són inserides de forma aleatòria a cadascuna de les imatges del conjunt d'entrenament.

2.2.2 Normalització del color

En la segmentació de les lesions epitelials, els models de colors juguen un paper clau ja que el to de les lesions i el to de pell acostumen a tenir colors molt similars. En aquest àmbit, la normalització del color té per objectiu discriminar el to de pell de la imatge per tal de facilitar als sistemes d'intel·ligència artificial la segmentació de les lesions de pell.

En el model de color RGB, el to de pell i les lesions epitelials solen tenir una major afectació en el canal vermell que en el verd i el blau. Per aquest motiu *Dang N. H. Thanh*[5] normalitza la banda vermella del model de color RGB:

Sigui $u = (u_R, u_G, u_B)$ una imatge en color de les lesions cutànies, on u_R, u_G, u_B són la intensitat de la imatge de la banda vermella, la banda verda i la banda blava, respectivament es defineix la banda vermella normalitzada com:

$$\widehat{u_R} = \frac{u_R}{\sqrt{u_R^2 + u_G^2 + u_B^2}}$$

Aquesta nova banda resulta adequada per a la segmentació d'imatges epitelials, ja que elimina l'efecte de la intensitat de la imatge i no depèn de l'ombra ni d'altres efectes.

2.3 Tècniques de classificació d'imatges per a lesions de pell

Existeixen diverses tècniques per a la classificació d'imatges de lesions cutànies que poden ser útils en el procés de detecció del melanoma. *Adekanmi Adegun*[1] examina les tècniques tradicionals de classificació d'imatges:

- **Classificadors basats en instàncies**: aquests classificadors comparen les instàncies de patrons nous en vers les instàncies amb patrons reconeguts sense realitzar generalitzacions. Un exemple seria l'algoritme *K-Neighbor*.
- **Arbres de decisió**: en aquests models en forma d'arbre, cada atribut actua com un node per tal de crear regles de decisió per a la classificació d'imatges.
- **SVM (Support Vector Machines)**: model d'aprenentatge supervisat que realitza classificació i anàlisi d'imatges, identificant patrons a partir d'un model que utilitza regles per a la classificació binària mitjançant la màxima separació d'un hiperplà.
- **Mètodes tradicionals basats en la intel·ligència**: es basen en la intel·ligència computacional per tal d'aplicar inferències. Alguns exemples són la xarxa neuronal artificial, el sistema d'inferència basat en difusió i els algorismes genètics.
- **Xarxes neuronals profundes**: utilitzen models computacionals de múltiples capes de processament amb arquitectura profunda per tal d'aprendre i poder representar les dades amb múltiples nivells d'abstracció.

En aquest sentit, el projecte es centra en les xarxes neuronals profundes, més concretament analitzarem els mètodes aplicats per *Masdevallia*[6], *Bo*[3] i *Deotte*[4] per tal de classificar en melanoma.

En la resolució del problema de la classificació del melanoma, els tres autors citats anteriorment, han utilitzat diferents configuracions dels algoritmes *EfficientNet*[7]. Aquesta família d'algoritmes basats en l'escala composta permeten aconseguir un rendiment altament eficaç sense comprometre l'eficiència dels recursos ni del model. Generalment, i també en el cas concret de la classificació del melanoma, els algoritmes *EfficientNet* obtenen un gran rendiment amb els models *Imagenet* o *Noisy-Student* preentrenats [6], [4].

2.4 Acceleració de models d'aprenentatge profund

En la competició *Kaggle* que tenia per objectiu la classificació del melanoma, els participants *Masdevallia*[6], *Bo*[3] i *Deotte*[4], utilitzen els acceleradors **TPU** proporcionats per *Kaggle*.

Els TPU (*Tensor Processing Units*) són acceleradors de hardware especialitzats en tasques de *Deep Learning*, els quals redueixen dràsticament el temps d'aprenentatge del model i també en milloren notablement la seva predicció.

2.5 Calibratge dels hiperparàmetres

Una de les eines més utilitzades per tal de trobar els hiperparàmetres que optimitzin el model és el **K-Fold cross validation**[2], un mètode estadístic d'avaluació i comparació d'algoritmes d'aprenentatge que utilitza la divisió del conjunt en diferents segments.

Aquesta tècnica consisteix en crear k particions aleatòries de la mateixa mida del conjunt de dades original. D'aquests k conjunts s'utilitzen $k-1$ per a l'entrenament del model i un per a l'avaluació del model (conjunt de test). Aquest procés es repeteix k vegades.

Aquesta tècnica utilitzada en [6], [3] i [4] contribueix en la selecció de valors òptims dels hiperparàmetres, a més de proporcionar una idea general del rendiment de l'algoritme i evitar el sobreajustament del model.

2.6 Millora de prediccions mitjançant TTA

En el diagnòstic mèdic a partir d'imatges és molt comú l'ús del **TTA** (*test-time augmention*), una tècnica que es basa en augmentar cadascuna de les imatges del conjunt test n vegades de forma aleatòria i seguint el patró d'augment de dades aplicat en les imatges d'entrenament. Aquest mètode s'ha utilitzat en [6], [3] i [4] per tal d'obtenir una predicció més robusta. A l'hora d'aplicar el TTA cal tenir en compte l'augment important del cost computacional que pot significar.

2.7 Acoblament de models

En les seves respectives publicacions *Masdevallia*[6], *Bo*[3] i *Deotte*[4] destaquen la importància de l'acoblament de models (*Ensemble models*), una tècnica que permet combinar les prediccions

de múltiples models reduint la variància de les prediccions i augmentant-ne la precisió.

En aquest aspecte, els autors citats anteriorment, destaquen la importància de l'ús de les metadades del conjunt per tal d'obtenir un millor rendiment en la tasca de classificació del melanoma.

Capítol 3

Disseny i Desenvolupament

3.1 Parametrització

Com a bones pràctiques de programació àgil, en primer lloc es declaren un seguit de variables per tal de parametritzar el codi amb l'objectiu que aquest sigui fàcilment adaptable a canvis.

Més concretament, es declaren les següents variables:

- *DEVICE*: Configuració del Dispositiu a utilitzar (*TPU* o *GPU*).
- *SEED*: Nombre de *Seed* a utilitzar en el *K-Fold*.
- *FOLDS*: Nombre de *Folds* del *K-Fold*.
- *IMG_SIZES*: Mida de les imatges a importar.
- *RESIZE*: Mida òptima de les imatges per a cadascun dels models *EfficientNet*.
- *INC2019*: Booleà que indica l'ús de les dades de la competició *Kaggle* del 2019 .
- *INC2018*: Booleà que indica l'ús de les dades de la competició *Kaggle* del 2018 i 2017.
- *BATCH_SIZES*: Mida del *Batch*.
- *EPOCHS*: Nombre d'èpoques d'entrenament.
- *CROP_SIZE*: Mida del retall aleatori per a cada mida d'imatge original.
- *AUGMENT*: Indicador d'augment de dades.
- *HAIR_AUGM*: Indicador d'augment de pèl.

- *RED_NORM*: Indicador normalització canal vermell.
- *MODEL_WEIGHTS*: Pesos predeterminats per als models, s'utilitzen tant *imagenet* com *noisy-student*.
- *EFN*: Model *EfficientNet* a utilitzar
- *OPTIMIZER*: Funció optimitzadora a utilitzar.
- *EXECUTION*: Identificador de l'execució.

Addicionalment a les variables anteriors es defineix una configuració pels models. Aquesta configuració consta de diferents variables relacionades amb l'entrenament o predicció dels models com algunes de les variables definides anteriorment o d'altres com podrien ser la taxa d'aprenentatge o alguns índexs per a la transformació d'imatges.

3.2 Càrrega dades

El pròxim pas en el desenvolupament de l'eina automàtica per a la predicció de melanoma és la càrrega dels conjunts de dades a utilitzar en el transcurs de la investigació. Més concretament, amb l'objectiu d'augmentar el conjunt de dades inicial per tal d'evitar el sobrentrenament, es carreguen dos conjunts de variables:

- **Melanoma**: Conjunt de dades de la competició de *Kaggle* per a la classificació del melanoma de l'any 2020.
- **Isic2019**: Conjunts de dades de la competició de *Kaggle* per a la classificació del melanoma dels anys 2019, 2018 i 2017.

Ambdós conjunts de dades han estat proporcionats per la comunitat *Kaggle* com a fitxers d'entrada per a l'entrenament dels models en les consegüents competicions que tenien com a objectiu el diagnòstic automàtic del melanoma.

Aquests conjunts estan formats per fitxers *CSV* amb informació sobre el pacient diagnosticat (metadades) en format *DICOM* i les seves respectives imatges de lesions epitelials en format *JPEG* i *TFRecord*.

Addicionalment, es carreguen les dades del conjunt de test, és a dir, la informació d'aquells pacients amb taques a la pell dels quals s'haurà de determinar de forma automàtica si es tracta o no de melanoma, en la competició de l'any 2020. La figura 3.1 conté un mostreig de les imatges del conjunt d'entrenament.

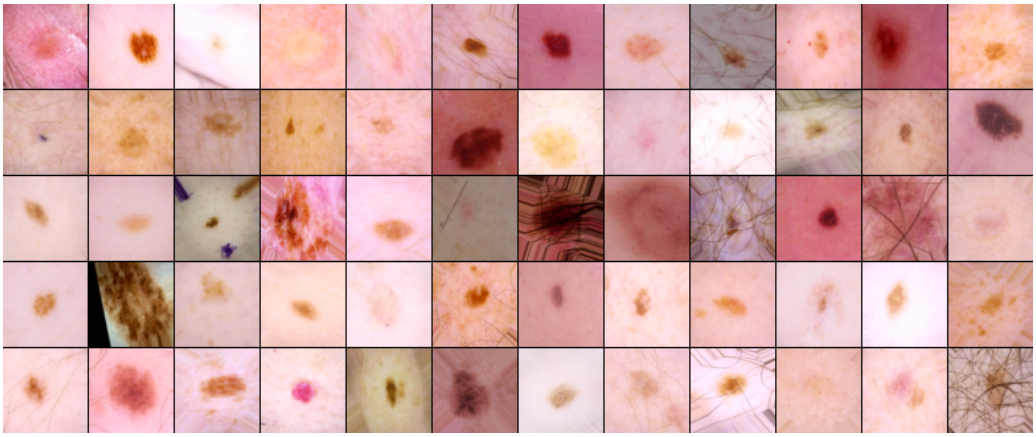


Figura 3.1: Exemple conjunt d'entrenament

3.3 Augment del conjunt de dades

Amb l'objectiu d'ampliar el conjunt de dades d'entrenament i afegir diversitat a les dades s'evita el sobrentrenament i augmenta la precisió del model. Per aquest motiu, s'apliquen les transformacions d'imatges llistades a continuació al conjunt d'entrenament. Per fer-ho, s'adapten les funcions *get_mat*, *transform* i *prepare_image* [4].

- Rotacions aleatòries de les imatges.
- Retalls d'imatges aleatoris.
- Aplicacions aleatòries de zoom.
- Desplaçaments d'imatges aleatoris.
- Modificacions d'inclinació.
- Girs horitzontals.
- Girs verticals.
- Alteracions de la tonalitat de la imatge.
- Alteracions del contrast de la imatge.
- Alteracions de la brillantor de la imatge.

En la figura 3.2 es visualitzen les diferents transformacions aleatòries aplicades per a una de les imatges del conjunt d'entrenament.

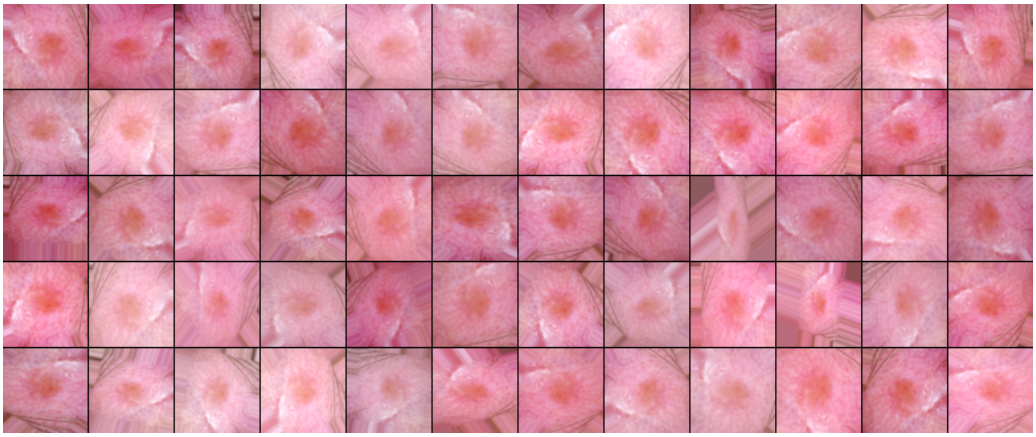


Figura 3.2: Exemple augments conjunt de dades

3.4 Preprocessat i transformació d'imatges

3.4.1 Augment de pèl

Com s'explica en la secció l'Estat de l'art, les imatges mèdiques de lesions de pell acostumen a estar cobertes per artefactes capil·lars que afecten negativament en la precisió de les eines automàtiques de diagnòstic.

En aquest aspecte, amb l'objectiu de millorar la precisió del model i col·laborar en evitar el sobrentrenament mitjançant la funció *hair_aug_tf* [6] s'insereixen, de forma aleatòria, teixits capil·lars en les imatges de diagnòstic. La figura 3.3 conté un mostreig del conjunt d'entrenament amb augment de pèl aleatori.

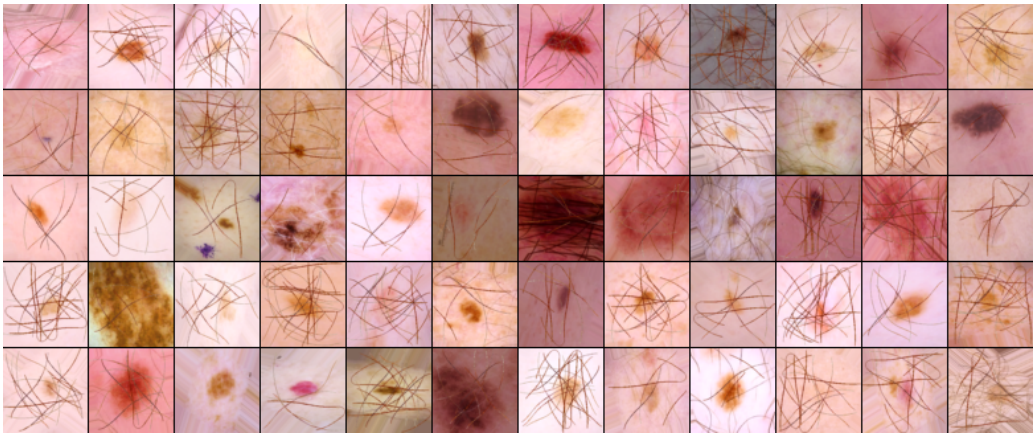


Figura 3.3: Exemple augment de pèl

3.4.2 Normalització del color

En l'apartat anterior es detalla que en el model de color RGB, el to de pell i les lesions epitelials solen tenir una major afectació en el canal vermell. En aquest aspecte, per tal discriminar el to de pell de la imatge en l'execució del model, es normalitza la banda vermella del model de color RGB:

Sigui $u = (u_R, u_G, u_B)$ una imatge en color de les lesions cutànies, on u_R, u_G, u_B són la intensitat de la imatge de la banda vermella, la banda verda i la banda blava respectivament, es defineix la banda vermella normalitzada com:

$$\widehat{u_R} = \frac{u_R}{\sqrt{u_R^2 + u_G^2 + u_B^2}}$$

La normalització del color vermell es duu a terme mitjançant la funció de *Python norm_color* definida seguidament:

```
def norm_color(image, norm = True):
    # Comprovacio sobre si aplicar normalitzacio
    if norm:
        img = image
        # Normalitzacio del canal RGB vermell
        red_norm = img[:,:,0]/tf.sqrt(tf.math.reduce_sum(img**2, axis = -1))

        # Creacio d'una nova imatge utilitzant el canal vermell normalitzat
            juntament amb els canals verd i blau originals
        norm_img = tf.stack([red_norm,img[:,:,1],img[:,:,2]], axis = -1)

        # Retorn imatge normalitzada
        return norm_img
    else:
        return image
```

En la figura 3.4 es mostra com es normalitza el color vermell en un mostreig del conjunt d'entrenament.

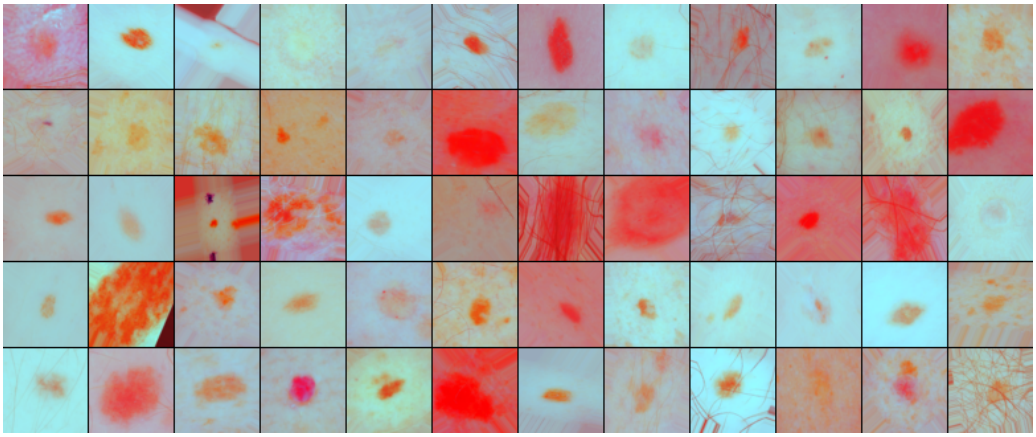


Figura 3.4: Exemple normalització color vermell

3.5 Definició del model

Per tal de resoldre el problema de la detecció del melanoma mitjançant la visualització artificial, es defineix un model basat en xarxes neuronals profundes. Més concretament, en el model definit s'utilitza (mitjançant la parametrització inicial) qualsevol dels algoritmes de la família *EfficientNet*. La xarxa neuronal definida està formada per tres capes:

- Capa base formada per una xarxa neuronal profunda de l'algoritme ***EfficientNet*** a utilitzar.
- Capa ***GlobalAveragePooling2D***.
- Capa **densa** amb funció d'activació *sigmoid*.

Per a la construcció del model s'utilitza la funció *build_model* definida a continuació:

```
def build_model(dim, ef):

    # Definicio de la mida d'entrada
    inp = tf.keras.layers.Input(shape=(dim,dim,3))

    # Importacio del model EfficientNet (EF) i els pesos definits inicialment
    base = ef(input_shape=(dim,dim,3),weights=MODEL_WEIGHTS,include_top=False)
    x = base(inp)

    # S'afegeix al model una capa GlobalAveragePooling2D i una capa densa amb
        funcio d'activacio sigmoid.
    x = tf.keras.layers.GlobalAveragePooling2D()(x)
```

```
    x = tf.keras.layers.Dense(1,activation='sigmoid')(x)

    # Definicio del model
    model = tf.keras.Model(inputs=inp,outputs=x)

    # Definicio funci de perdua
    losses = tf.keras.losses.BinaryCrossentropy(label_smoothing =
        CFG['label_smooth_fac'])

    # Compilacio del model
    model.compile(optimizer=CFG['optimizer'],loss=losses,metrics=['AUC'])
    return model
```

3.5.1 Taxa d'aprenentatge

Amb l'objectiu d'accelerar l'aprenentatge del model s'utilitza una adaptació de la funció *get_lr_callback* [4],[6] en la qual es defineix la taxa d'aprenentatge en funció de l'època d'entrenament; essent menys exigent en les primeres èpoques d'entrenament del model i augmentant l'exigència d'aquesta en les capes més avançades, reduïnt així el temps d'entrenament.

3.6 Calibratge dels hiperparàmetres

Seguidament, amb l'objectiu de calibrar els diferents hiperparàmetres per tal d'optimitzar la precisió del model definitiu, es desenvolupa una validació creuada *K-Fold*.

Amb la finalitat de poder prendre les decisions adequades sobre quins hiperparàmetres optimitzen la precisió del model, es visualitzen gràficament (figura 3.5) les mètriques AUC i *Loss* de cadascún dels *Folds*.

Addicionalment, es calcula la mitjana aritmètica de la mètrica AUC màxima obtinguda en cadascú dels K *Folds*:

$$\overline{AUC} = \frac{\sum_{i=1}^{K} \max AUC_i}{K}$$

on AUC_i és la métrica AUC del *Fold* i i K el nombre de *Folds*.

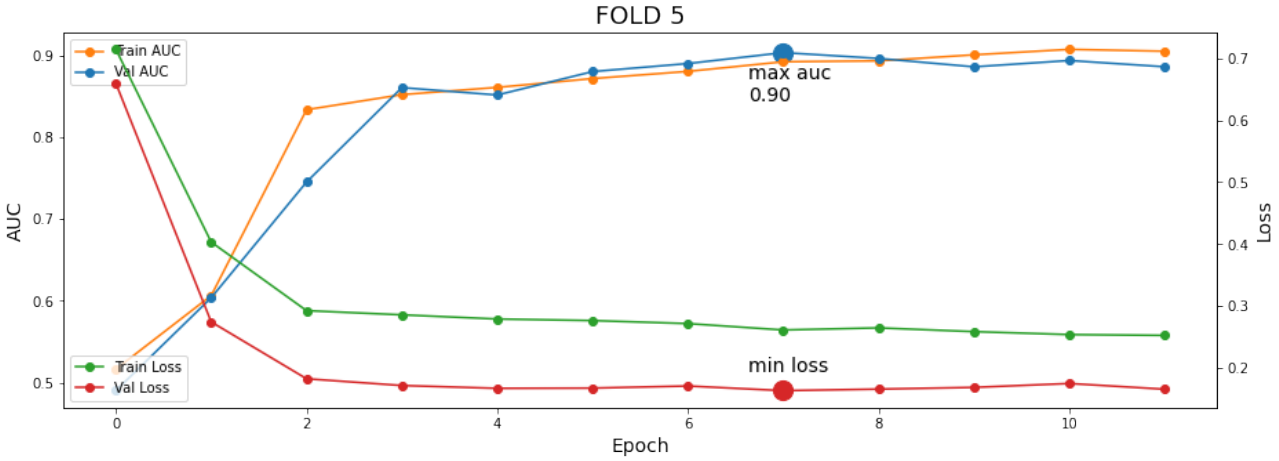


Figura 3.5: Exemple gràfic K-Fold

3.7 Entrenament model

Un cop fixats els hiperparàmetres òptims pel rendiment del model s'entrena el model creat anteriorment mitjançant el codi *Python* que es mostra a continuació:

```
# Definir i crear el model Efficientnet a partir de la funcio build_model
constructor = getattr(efn, EFN)
with strategy.scope():
    model = build_model(dim=RESIZE[EFN],ef=constructor)

# Mitjancant la funcio get_dataset, obtenir els conjunts d'entrenament i validacio
ds_train = get_dataset(files_train, CFG, shuffle=True, repeat=True)
ds_test = get_dataset(files_test, CFG, labeled=False).unbatch().take(12*5)

# Entrenament del model
print('Entrenant el model ',EFN,'...')
history = model.fit(ds_train, epochs=EPOCHS, callbacks = [get_lr_callback(CFG)],
    steps_per_epoch=count_data_items(files_train)/BATCH_SIZES//REPLICAS, verbose=1)
print('\n Finalitzat entrenament del model ', EFN,' \n')
```

3.8 Predicció conjunt de test

Seguint les pautes marcades en l'estat de l'art s'augmenta el conjunt test de forma aleatòria i seguint el patró d'augment de dades aplicat en el conjunt d'entrenament, és a dir, s'apliquen

l'ús de *TTA*.

Un cop aplicades les transformacions al conjunt de test, a partir del model entrenant, s'obté la predicció de la classe a la qual pertany cadascuna de les imatges del conjunt de test augmentat.

A continuació es transforma el vector amb les prediccions per tal d'obtenir una única predicció per a cada imatge del conjunt test calculada amb la predicció mitjana de totes les seves transformacions.

En el codi *Python* que es mostra a continuació, es dur a terme la predicció del conjunt de test i aquesta s'exporta en un fitxer *CSV*:

```
# Calcul dels passos TTA a aplicar en la prediccio del conjunt
cnt_test = count_data_items(files_test)
steps = cnt_test / (CFG['batch_size'] * REPLICAS) * CFG['tta_steps']

# Conjunt de Test Augmentat per tal de fer la prediccio mitjanant TTA
ds_test_aug = get_dataset(files_test, CFG, repeat=True, labeled=False,
    return_image_names=False)

# Prediccio de la classe a la qual pertany cadascuna de les imatges del conjunt de
    test augmentat a partir del model EfficientNet creat anteriorment
pred = model.predict(ds_test_aug,verbose=1, steps=steps)


# Transformacio del vector amb les prediccions per tal d'obtenir una unica
    prediccio per a cada imatge del conjunt test calculada amb la prediccio
# mitjana de totes les transformacions de la mateixa imatge
preds = np.stack(pred)
preds = preds[:,:cnt_test* CFG['tta_steps']]
preds = preds[:df_test.shape[0]*CFG['tta_steps']]
preds = np.stack(np.split(preds, CFG['tta_steps']),axis=1)
preds = np.mean(preds, axis=1)
pred = preds.reshape(-1)

# Obtencio del nom de les imatges del conjunt de proba.
ds_test = get_dataset(files_test, CFG, repeat=False, labeled=False,
```

```
    return_image_names=True)
image_names = np.array([img_name.numpy().decode("utf-8") for img, img_name in
    iter(ds_test.unbatch())])

# Creacio de conjunt de dades amb els resultats obtinguts
resultats = pd.DataFrame(dict(image_name = image_names,target = pred))
resultats = resultats.sort_values('image_name')

# Creacio de fitxer amb format CSV amb els resultats obtinguts
resultats.to_csv(f'{EXECUTION}.csv', index=False)

resultats.head()
```

3.9 Acoblament de models

Finalment i, per tal de reduir la variància entre les prediccions dels diferents models i augmentar, d'aquesta manera, la precisió de la predicció final, es duu a terme un acoblament de models (*Ensambling*) combinant a partir de la mitjana aritmètica aquells models que han mostrat una millor precisió en el conjunt de prova. Per fer-ho s'utilitza el següent codi *Python*:

```
# Llegir les prediccions de cadascun dels models seleccionats
BASEPATH = "../input/melanomapredictions"
pred_5 = pd.read_csv(os.path.join(BASEPATH, '5.csv'))
pred_9 = pd.read_csv(os.path.join(BASEPATH, '9.csv'))
pred_15 = pd.read_csv(os.path.join(BASEPATH, '15.csv'))
pred_17 = pd.read_csv(os.path.join(BASEPATH, '17.csv'))

# Combinacio dels models mitjancant el calcul de la prediccio mitjana (ensambling)
pred = (pred_5['target']+pred_9['target']+pred_15['target']+pred_17['target'])/4

# Creacio del conjunt de dades amb els resultats obtinguts
resultats = pd.DataFrame(dict(image_name = pred_5['image_name'],target = pred))
resultats = resultats.sort_values('image_name')

# Creacio de fitxer amb format CSV amb els resultats obtinguts
resultats.to_csv(f'pred_ensamble.csv', index=False)
```

Capítol 4

Experiments i Resultats

En el capítol anterior es mostra com s'ha desenvolupat l'eina d'aprenentatge automàtic que té per objectiu classificar imatges de diagnòstics de pacients en funció de si s'expressa o no el melanoma. En el present apartat s'expliciten un seguit de proves que s'han dut a terme per tal d'optimitzar el rendiment del model a entrenar i, d'aquesta manera, obtenir una eina automàtica capaç de detectar amb la màxima precisió la presencia o no del melanoma.

Per tal de reduir el temps d'execució destinat a calibrar els diferents hiperparàmetres del model, en primer lloc s'utilitza el model més senzill de la família *Efficientnet*, és a dir, el model *Efficientnet-B0*. Un cop definits els hiperparàmetres que optimitzin la precisió del model s'analitza la precisió obtinguda per a cadascun dels models *Efficientnet*.

En primer lloc, i com a punt d'inici, es defineix la configuració inicial que es mostra a continuació:

- **Mida del Batch**: 16
- **Èpoques d'entrenament**: 12
- **Optimitzador**: *adam*
- **Incloure Dades 2020**: Verdader
- **Incloure Dades 2019**: Fals
- **Incloure Dades 2018**: Fals
- **Augment de dades**: Fals
- **Pesos predeterminats**: *imagenet*

- **Augment de pèl**: Fals
- **Normalització del Color**: Fals
- **Model Efficienet**: *Efficienet-B0*

Mitjançant la configuració definida anteriorment, els resultats obtinguts no són positius ja que a conseqüència de la falta de dades per a la fase d'entrenament, el model presenta sobrentrenament.

La mètrica *AUC* del gràfic de la figura 4.1 evidencia aquest fet perquè el rendiment en el conjunt d'entrenament és molt superior que al del conjunt de prova, on decau a partir de la quarta època d'entrenament i la precisió del model és propera a 0.5. Això significa que la predicció del conjunt de prova feta a partir del model té poca més precisió que una predicció aleatòria.

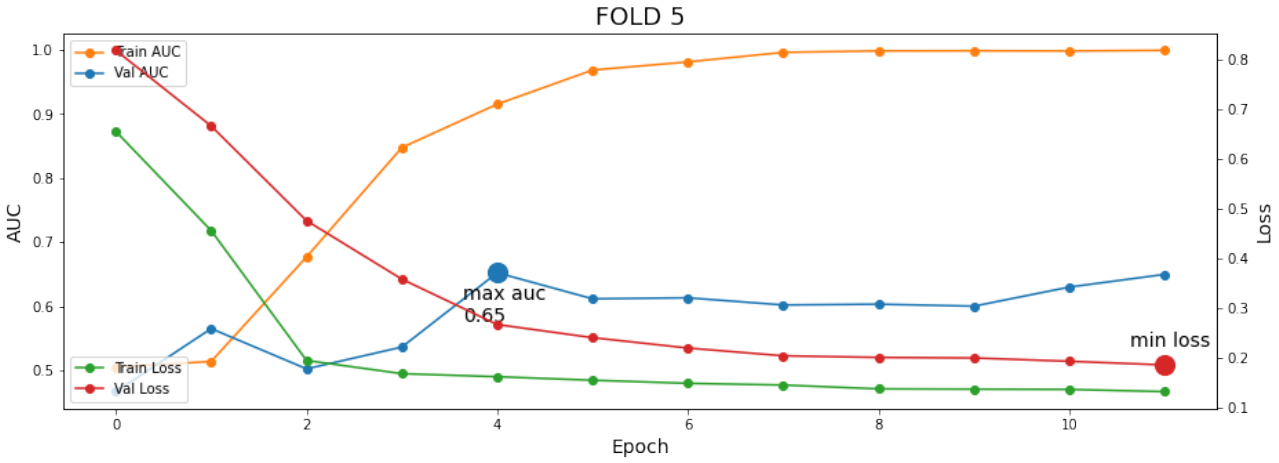


Figura 4.1: Gràfic resultats per a un dels *Folds* de l'execució inicial

Seguidament i, per tal d'evitar, el sobrentrenament es duen a terme tres noves execucions utilitzant conjunts de dades més extensos:

2. S'inclouen les dades de l'any 2019
3. S'inclouen les dades dels anys 2019, 2018 i 2017
4. S'inclouen les dades dels anys 2019, 2018 i 2017 i s'augmenta el conjunt de dades

Els resultats obtinguts en la segona i tercera execució, tot i millorar de forma poc significativa el resultats de la primera, continuen mostrant evidències que a causa de la falta de dades el model s'especialitza en el conjunt d'entrenament i no és capaç de predir correctament el conjunt

de validació.

No obstant això, en la tercera execució i gràcies a l'augment de dades, s'aprecien millores significatives de la precisió del model en el conjunt de validació. En la figura 4.2 es mostren gràficament l'evolució de les mètriques *AUC* i *Loss* al llarg de les 12 èpoques d'entrenament. En ella es pot observar com el rendiment del model no millora significativament a partir de la vuitena època d'entrenament, fet que apunta que és suficient utilitzar 12 èpoques en l'entrenament del model.

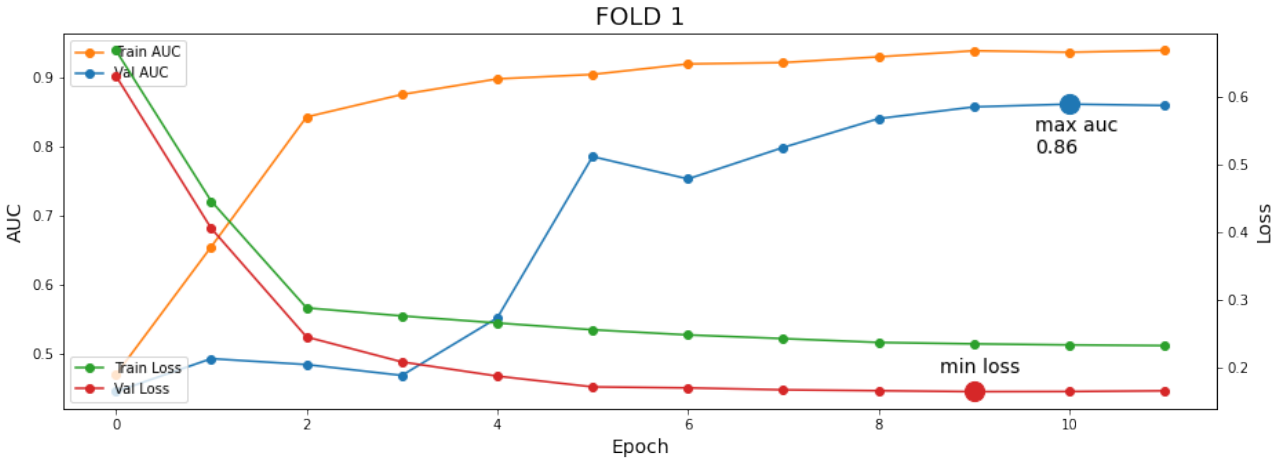


Figura 4.2: Gràfic resultats per a un dels *Folds* de la execució 4

Seguidament, i tenint en compte la millora de la precisió del model utilitzant l'augment de dades, es duen a terme una bateria d'experiments per tal d'esbrinar quina configuració dels hiperparàmetres següents resulta òptima per a la precisió del model:

- **Pesos predeterminats**: Es realitzen proves amb els pesos prentrenats *imagenet* i *noisy-student*.
- **Model Efficienet**: S'executa el model pels algoritmes *Efficienet B0 - B6*
- **Augment de pèl**: S'experimenta tenint en compte o no la inclusió de teixit capil·lar.
- **Normalització Color Vermell**: S'efectuen proves normalitzant o no el color vermell.

En la taula 4.1 es mostra un resum dels resultats obtinguts per a les proves descrites anteriorment. Més concretament, i per tal de poder comparar els diferents models, es mostra la mètrica $\overline{AUC}$ descrita en el capítol anterior.

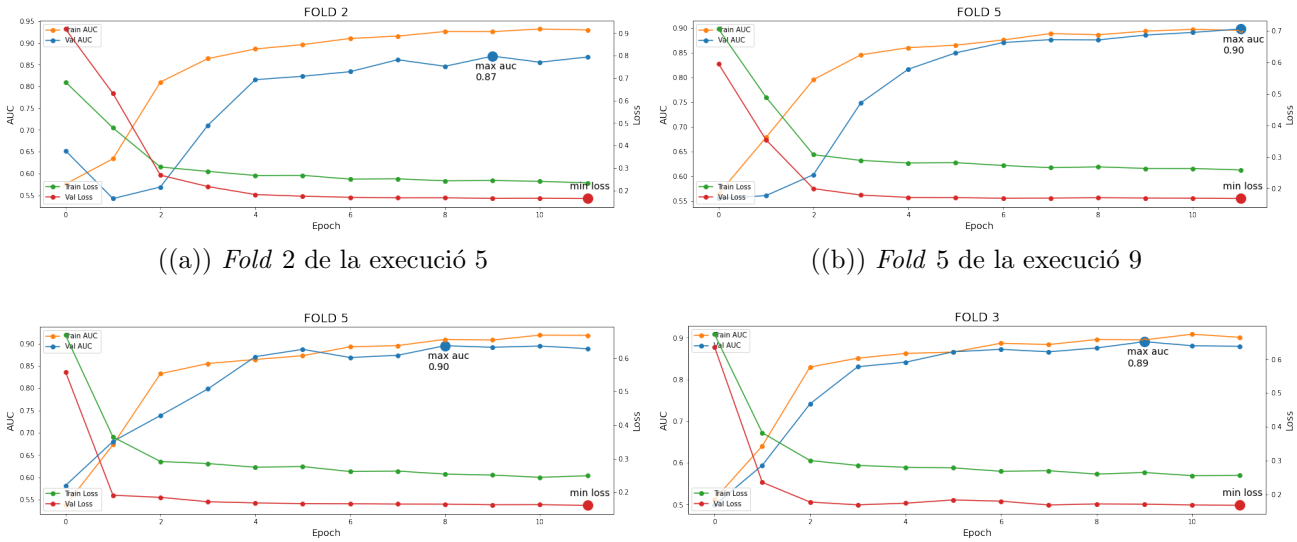
ID	Dades	Pesos	Efficienet	Augment	Pèl	Color	$\overline{AUC}$
1	2020	imagenet	B0	No	No	No	0,624
2	20 i 19	imagenet	B0	No	No	No	0,72
3	20, 19 i 18	imagenet	B0	No	No	No	0,742
4	20, 19 i 18	imagenet	B0	No	No	No	0,842
5	20, 19 i 18	noisy-student	B0	No	No	No	0,868
6	20, 19 i 18	noisy-student	B0	Si	No	No	0,86
7	20, 19 i 18	imagenet	B0	Si	No	No	0,86
8	20, 19 i 18	imagenet	B0	Si	Si	Si	0,822
9	20, 19 i 18	noisy-student	B0	Si	Si	Si	0,862
10	20, 19 i 18	noisy-student	B0	Si	No	Si	0,838
11	20, 19 i 18	noisy-student	B1	Si	Si	Si	0,848
12	20, 19 i 18	noisy-student	B2	Si	Si	Si	0,856
13	20, 19 i 18	noisy-student	B3	Si	Si	Si	0,83
14	20, 19 i 18	noisy-student	B4	Si	Si	Si	0,854
15	20, 19 i 18	noisy-student	B5	Si	Si	Si	0,886
16	20, 19 i 18	noisy-student	B6	Si	Si	Si	0,858
17	20, 19 i 18	imagenet	B5	Si	Si	Si	0,878
18	20, 19 i 18	imagenet	B6	Si	Si	Si	0,872

Taula 4.1: Resum de les execucions

Com es pot comprovar en la taula 4.1, els experiments que millor precisió han mostrat en els conjunts de validació han estat les execucions amb identificador 5, 9, 15 i 17. En la figura 4.3, es mostra el comportament de les mètriques AUC i *Loss* per a un dels *folds* de cadascuna d'aquestes execucions.

En els quatre casos que es mostren en la figura s'aprecia com la precisió del model (tant en el conjunt d'entrenament com en el conjunt de prova) augmenta significativament en les quatre primeres èpoques d'entrenament i llavors s'estabilitza amb una precisió propera al 0.9. Un fet similar succeeix amb la mètrica *Loss* que a partir de la segona època d'entrenament s'estabilitza amb valors propers al 0.2 per ambdós conjunts.

En l'estabilització de les mètriques avaluades en cadascun dels models executats es pot afirmar que el nombre d'èpoques d'entrenament seleccionades (12) són suficients per a entrenar el model. No obstant això, i per tal de confirmar la hipòtesi anterior, en la figura 4.4 es mostra el resultat d'un dels *Folds* de l'execució 9 amb 20 èpoques d'entrenament.



((a)) *Fold* 2 de la execució 5

((b)) *Fold* 5 de la execució 9

((c)) *Fold* 5 de la execució 15

((d)) *Fold* 3 de la execució 17

Figura 4.3: Gràfics amb els resultats per a un dels *Folds* de la execucions 5,9,15 i 17

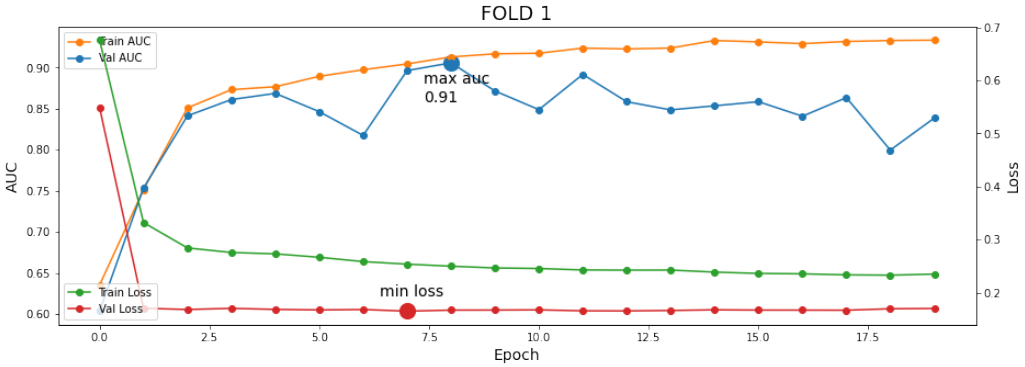


Figura 4.4: Gràfic resultats per a un dels *Folds* de l'execució 5 amb 20 èpoques d'entrenament

Per altra banda, també s'ha executat alguns dels models anteriors augmentant la mida del *batch* a 32. Amb aquest augment no s'aprecia cap millora significativa en la precisió dels models. No obstant això, el que sí que es veu afectat és el temps d'execució dels models ja que aquest augmenta significativament. Per aquests motius, es manté la mida del *batch* a 16 tal com s'ha fixat inicialment.

Un cop fixats el hiperparàmetres que optimitzen el rendiment del model, aquest s'entrena utilitzant totes les dades del conjunt d'entrenament per a predir de forma automàtica a la classe a la qual pertany els diagnòstics del conjunt de test.

4.1 Resultats

Un cop predites les classes a la qual pertanyen cadascun dels diagnòstics del conjunt de prova, aquestes s'exporten en un fitxer *CSV* per tal de ser enviades a *Kaggle* i d'aquesta forma obtenir la precisió a aquest conjunt.

En la taula 4.2 es mostren els resultats obtinguts per a les prediccions dels quatre models seleccionats en l'apartat anterior:

ID	Private Score	Public Score
5	0.8901	0.9257
9	0.8741	0.8910
15	0.8963	0.9237
17	0.9026	0.9203

Taula 4.2: Resultats de les prediccions

On la mètrica *Private Score* definida per *Kaggle* avalua el model amb aproximadament el 66% de les dades del conjunt de prova i la mètrica *Pulic Score* avalua el model amb aproximadament el 34% de les dades del conjunt de prova.

Com es pot observar en la taula 4.2 l'execució 17 amb una *Private Score* superior al 0.9 i una *Pulic Score* superior al 0.92 és l'execució que millor precisió mostra en el conjunt de prova.

Finalment, i per tal de millorar els resultats anteriors, tal com s'ha descrit en el capítol anterior es duu a terme un *Ensambling* de models per a la mitjana amb l'objectiu de reduir la variància entre les prediccions i augmentar la predicció final. En la taula 4.3 es mostren els resultats d'utilitzar la tècnica d'acoblament descrita anteriorment per diferents combinacions dels models utilitzats anteriorment:

ID Models	Private Score	Public Score
5, 9, 15, 17	0.9021	0.9255
5, 15, 17	0.9062	0.9307
5, 15	0.9024	0.9302
5, 17	0.9031	0.9269
15, 17	0.9074	0.9293

Taula 4.3: Resultats *Ensambling* de Models

Com es pot observar en la taula 4.3, a partir de l'acoblament de diferents models, la precisió final millora de forma notable. Més concretament, es pot observar que la predicció amb millor precisió s'ha obtingut a partir de la mitjana dels models 5, 15 i 17; obtenint *Private Score* superior al 0.9 i una *Pulic Score* superior al 0.93.

Capítol 5

Conclusions

En l'apartat anterior s'ha utilitzat el mètode estadístic d'avaluació *K-Fold Cross Validation* per tal de seleccionar els valors òptims dels hiperpàrametres així com obtenir una primera idea del rendiment general dels diferents algoritmes.

Mitjançant l'ús del *K-Fold* s'ha dividit $k = 5$ vegades el conjunt d'entrenament en dos subconjunts; entrenament i validació. Sobre aquestes particions s'ha entrenat i avaluat diferents models per tal de seleccionar els hiperpàrametres que optimitzin l'encert del model.

En la taula 4.1 s'observa la precisió mitjana de cadascun dels models entrenats en el conjunt de validació. A partir d'aquestes proves s'ha seleccionat els hiperpàrametres d'aquelles execucions que mostren una millor predicció en el conjunt de validació. En aquest aspecte es destaca l'eficàcia i beneficis d'alguns dels experiments realitzats anteriorment:

Augment de les dades

En els resultats obtinguts en les primeres execucions del capítol anterior s'aprecia la importància de la mida del conjunt d'entrenament pel correcte rendiment del model. Per tal d'augmentar el conjunt de dades, s'ha utilitzat dues tècniques:

- **Utilitzar altres conjunts de dades similars**: per a l'entrenament del model s'utilitzen les dades de les competicions de *Kaggle* dels anys 2019, 2018 i 2017.
- **Augment de dades**: a partir de tècniques d'augment de dades s'apliquen lleugeres transformacions a les imatges sense canviar la seva semàntica a alt nivell.

Mitjançant les dues tècniques descrites anteriorment, el conjunt de les dades per a l'entrenament està format per a un major nombre de dades. D'aquesta manera s'obtenen diferents beneficis per a l'entrenament del model:

- Augment de la diversitat en el conjunt de dades.
- Millor generalització del model.
- Millor precisió del model.
- Evita el sobrentrenament.
- Col·labora en la tolerància d'errors.
- Augment de la robustesa del model.
- Major velocitat d'aprenentatge.

Augment de pèl

Com s'ha explicat en l'estat de l'art, una de les complicacions al treballar amb imatges de pell humana està en evitar el soroll creat per a la presència d'artefactes introduïts en el moment de la captura de la imatge. En el cas d'estudi es pot observar en la figura 3.1 que algunes de les imatges de lesions epitelials contenen artefactes capil·lars que generen soroll i poden dificultar la generalització del model.

Per tal de solucionar la problemàtica anterior s'ha utilitzat una metodologia per a la simulació realista de pèl, introduint a les fotografies del conjunt diferents imatges capil·lars reals presegmentades prèviament.

D'aquesta manera s'ha permès al model potenciar els beneficis de l'augment de dades; millorar la precisió de la classificació, augmentar la capacitat per a la generalització, evitar el sobrentrenament o augmentar la velocitat d'aprenentatge.

Normalització del color

Tal i com s'ha vist en el segon capítol, per tal de discriminar el to de pell de les imatges, s'ha dut a terme una normalització del canal vermell en el model de color RGB.

Aquesta pràctica ha permès al model obtenir una millor generalització augmentant, d'aquesta manera, l'encert dels resultats obtinguts.

Acoblament de models

Finalment, es destaca la millora significativa dels resultats obtinguts en l'apartat anterior després de combinar les prediccions de múltiples models. A partir d'aquesta tècnica s'ha obtingut els següents beneficis:

- Reducció de la variància de les precisions dels diferents models.
- Millora de la precisió de les prediccions obtingudes.

Mitjançant l'ús de les tècniques descrites anteriorment i ajustant els hiperpàrametres de forma òptima, tal com s'indica en l'apartat anterior, l'eina automàtica de processament d'imatges pel diagnòstic del melanoma en lesions epitelials té una taxa d'encert superior al 90%.

Aquesta eina pot contribuir en millorar de forma significativa la detecció d'aquesta malaltia per part dels dermatòlegs, augmentant d'aquesta manera l'efectivitat del tractament a aplicar i, com a conseqüència, incrementar la probabilitat de supervivència dels pacients diagnosticats.

Capítol 6

Línies de treball futures

L'eina d'intel·ligència artificial creada al llarg d'aquest projecte obre la porta a diferents línies d'investigació o de treball futures; a continuació se'n llisten algunes:

- Implementació de l'eina en un entorn clínic real per tal de facilitar als dermatòlegs el diagnòstic del melanoma i d'aquesta forma afavorir en la salut de milions de persones.
- Millorar el rendiment dels models utilitzant tècniques de preprocessat i tractament d'imatges avantguardistes que apareguin de forma posterior.
- Augmentar les dades del conjunt d'entrenament ja sigui a partir d'altres conjunts de dades existents o bé mitjançant dades de nous diagnòstics, per tal de millorar la precisió dels models.
- Extrapolar l'experiència i coneixements adquirits en aquest projecte a altres eines de visió artificial similars per a la millora de diagnòstics clínics com podrien ser:
  - Anàlisis de radiografies.
  - Anàlisis d'ecografies.
  - Anàlisis de tomografies.
- Utilitzar els models entrenats per a resoldre altres problemes relacionats amb la classificació d'imatges mitjançant visió artificial com podrien ser els que es plantegen a continuació:
  - Controls de qualitat en el sector industrial.
  - Detecció i classificació d'objectes.
  - Eines d'arbitratge automàtic en determinats esports.
  - Detecció de paràsits o malalties en plantes.

Bibliografia

[1] Adekanmi Adegun and Serestina Viriri. Deep learning techniques for skin lesion analysis and melanoma cancer detection: a survey of state-of-the-art. *Artificial Intelligence Review*, June 2020.

[2] Bosch Rué Anna, Casas Roma Jordi, and Toni Lozano Bagén. *Deep Learning principios y fundamentos*. Editorial UOC, July 2019.

[3] Bo. 1st place solution. *Kaggle: siim-isic-melanoma-classification*, August 2020.

[4] Chris Deotte. Triple stratified kfold with tfrecords. *Kaggle: siim-isic-melanoma-classification*, July 2020.

[5] Dang N. H. Thanh V. B. Surya Prasath Le Minh Hieu and Nguyen Ngoc Hien. Melanoma skin cancer detection method based on adaptive principal curvature, colour normalisation and feature extraction with the abcd rule. *Journal of Digital Imaging*, December 2019.

[6] Masdevallia. 3rd place solution overview. *Kaggle: siim-isic-melanoma-classification*, August 2020.

[7] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *International Conference on Machine Learning*, May 2019.

Annex A

Disseny i Desenvolupament (codi Python)

A.1 Importació de llibreries

Importació de les llibreries *Python* a utilitzar durant el desenvolupament de la solució:

```
# Importacio de llibreries

import numpy as np
import pandas as pd
import os, random, re, math, time, gc
random.seed(a=42)
from glob import glob
import tensorflow as tf
import tensorflow.keras.backend as K
import efficientnet.tfkeras as efn
from sklearn.model_selection import KFold
import matplotlib.pyplot as plt
from kaggle_datasets import KaggleDatasets
import PIL
```

A.2 Parametrització

Declaració de variables:

```
# Dispositiu (TPU o GPU)
DEVICE = "TPU"
# DEVICE = "GPU"

# SEED per K-FOLD.
SEED = 42

# Nombre de Folds
FOLDS = 5

# Mida de les imatges a utilitzar
# Mides disponibles: 256, 384, 512, 768
IMG_SIZES = 512

# Mida optima de les imatges per a cada model EfficientNET
RESIZE = {'EfficientNetB0': 224, 'EfficientNetB1' : 240, 'EfficientNetB2': 260,
    'EfficientNetB3': 300, 'EfficientNetB4': 380, 'EfficientNetB5': 456,
    'EfficientNetB6': 528}

# Incloure dades de competicions anteriors (2019/2018)?
INC2019 = True
INC2018 = True

# Mida del Batch i Epoques d'entrenament
BATCH_SIZES = 32
EPOCHS = 12

# Mida del retall aleatori per a cada mida d'imatge original (256, 384, 512, 768):
CROP_SIZE = {256: 250, 384: 370, 512: 500, 768: 750}

# Indicador d'augment de dades en el conjunt d'entrenament
AUGMENT = True
```

```
# Indicador d'augment de pel
HAIR_AUGM = True

# Indicador normalitzacio canal vermell
RED_NORM = True


# Pesos predeterminats per als models, utilitzem tant imagenet com noisy-student
MODEL_WEIGHTS = 'imagenet'

# Model EfficientNet a utilitzar
EFN = 'EfficientNetB5'

# Funcio optimitzador a utilitzar
OPTIMIZER = 'adam'

# Identificador de l'execucio del model

EXECUTION = 'Ex17'
```

```
# Definir configuracio
CFG = dict(

    # Mida del Batch
    batch_size = BATCH_SIZES,

    # Dimensions de les imatges
    read_size = IMG_SIZES,
    crop_size = crop_size[IMG_SIZES],
    net_size = RESIZE[EFN],

    # Taxa d'aprenentatge
    LR_START = 0.000003,
    LR_MAX = 0.000020,
    LR_MIN = 0.000001,
    LR_RAMPUP_EPOCHS = 5,
    LR_SUSTAIN_EPOCHS = 0,
    LR_EXP_DECAY = 0.8,
```

```
    # Epoques d'entrenament:
    epochs = EPOCHS,

    # Variables relacionades amb l'augment de dades
    rot = 180.0,
    shr = 1.5,
    hzoom = 6.0,
    wzoom = 6.0,
    hshift = 6.0,
    wshift = 6.0,

    # Indicador d'augment de pel
    hair_augm = hair_augm,

    # Indicador normalitzacio canal vermell
    red_norm = red_norm,

    # Optimitzador
    optimizer = OPTIMIZER,
    label_smooth_fac = 0.05,

    # Pasos TTA
    tta_steps = 25
)
```

A.3 Configuració TPU

A continuació, per tal d'accelerar l'aprenentatge en la fase d'entrenament i optimitzar l'eficàcia dels models, s'estableix connexió amb l'accelerador **TPU** proporcionat per *Kaggle*.

```
if DEVICE == "TPU":
    print("connecting to TPU...")
    try:
        tpu = tf.distribute.cluster_resolver.TPUClusterResolver()
        print('Running on TPU ', tpu.master())
    except ValueError:
```

```
        print("Could not connect to TPU")
        tpu = None

    if tpu:
        try:
            print("initializing TPU ...")
            tf.config.experimental_connect_to_cluster(tpu)
            tf.tpu.experimental.initialize_tpu_system(tpu)
            strategy = tf.distribute.experimental.TPUStrategy(tpu)
            print("TPU initialized")
        except _:
            print("failed to initialize TPU")
    else:
        DEVICE = "GPU"

if DEVICE != "TPU":
    print("Using default strategy for CPU and single GPU")
    strategy = tf.distribute.get_strategy()

if DEVICE == "GPU":
    print("Num GPUs Available: ",
        len(tf.config.experimental.list_physical_devices('GPU')))


AUTO     = tf.data.experimental.AUTOTUNE
REPLICAS = strategy.num_replicas_in_sync
print(f'REPLICAS: {REPLICAS}')
```

A.4 Càrrega dades

Càrrega dels conjunts de dades:

```
# Importacio del directori de Kaggle per a l'entrada de dades

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
```

```
        print(os.path.join(dirname, filename))

# Carrega de dades del conjunt d'entrenament
GCS_PATH = KaggleDatasets().get_gcs_path('melanoma-%ix%i'%(IMG_SIZES,IMG_SIZES))
GCS_PATH2 = KaggleDatasets().get_gcs_path('isic2019-%ix%i'%(IMG_SIZES,IMG_SIZES))
files_train = np.sort(np.array(tf.io.gfile.glob(GCS_PATH + '/train*.tfrec')))

# Carrega de dades del conjunt de test (noms exercici 2020)
BASEPATH = "../input/siim-isic-melanoma-classification"
df_test = pd.read_csv(os.path.join(BASEPATH, 'test.csv'))
files_test = np.sort(np.array(tf.io.gfile.glob(GCS_PATH + '/test*.tfrec')))
```

A.5 Augment del conjunt de dades

S'adapten les funcions *get_mat* i *transform* [4] per tal de realitzar les transformacions a les imatges de lesions epitelials:

```
#Funcio que retorna una matriu 3x3 amb els index de transformacio
def get_mat(rotation, shear, height_zoom, width_zoom, height_shift, width_shift):

    # Conversio de graus a radians
    rotation = math.pi * rotation / 180.
    shear    = math.pi * shear  / 180.


    def get_3x3_mat(lst):
        return tf.reshape(tf.concat([lst],axis=0), [3,3])

    # Definicio matriu de rotacio
    c1   = tf.math.cos(rotation)
    s1   = tf.math.sin(rotation)
    one  = tf.constant([1],dtype='float32')
    zero = tf.constant([0],dtype='float32')

    rotation_matrix = get_3x3_mat([c1, s1, zero,
                                   -s1, c1,  zero,
                                   zero, zero, one])
```

```
    # Definicio matriu tall
    c2 = tf.math.cos(shear)
    s2 = tf.math.sin(shear)

    shear_matrix = get_3x3_mat([one, s2, zero,
                                zero, c2,  zero,
                                zero, zero, one])

    # Definicio matriu zoom
    zoom_matrix = get_3x3_mat([one/height_zoom, zero,     zero,
                               zero,           one/width_zoom, zero,
                               zero,           zero,          one])

    # Definicio matriu de desplacament
    shift_matrix = get_3x3_mat([one, zero, height_shift,
                                zero, one, width_shift,
                                zero, zero, one])

    # Retorn objecte K.dot amb la informacio de la matriu de transformacio 3x3
    return K.dot(K.dot(rotation_matrix, shear_matrix),
                 K.dot(zoom_matrix,   shift_matrix))


# Aquesta funcio te com a parametre d'entrada una imatge de la mida [dim,dim,3]
# Sortida: Imatge girada, tallada, ampliada i desplaada a latzar

def transform(image, cfg):
    DIM = cfg["read_size"]
    XDIM = DIM%2 #fix for size 331

    # Assignacio aleatoria dels valors de rotacio, tall, ampliacio i desplacament
    rot = cfg['rot'] * tf.random.normal([1], dtype='float32')
    shr = cfg['shr'] * tf.random.normal([1], dtype='float32')
    h_zoom = 1.0 + tf.random.normal([1], dtype='float32') / cfg['hzoom']
    w_zoom = 1.0 + tf.random.normal([1], dtype='float32') / cfg['wzoom']
    h_shift = cfg['hshift'] * tf.random.normal([1], dtype='float32')
    w_shift = cfg['wshift'] * tf.random.normal([1], dtype='float32')

    # Obtencio de la matriu de transformacio mitjancant la funcio get_mat
```

```
    m = get_mat(rot,shr,h_zoom,w_zoom,h_shift,w_shift)

    # Llistat de destinacio dels pixels de la imatge
    x   = tf.repeat(tf.range(DIM//2, -DIM//2,-1), DIM)
    y   = tf.tile(tf.range(-DIM//2, DIM//2), [DIM])
    z   = tf.ones([DIM*DIM], dtype='int32')
    idx = tf.stack( [x,y,z] )

    # Rotacio dels pixels de destinacio pels pixels originals
    idx2 = K.dot(m, tf.cast(idx, dtype='float32'))
    idx2 = K.cast(idx2, dtype='int32')
    idx2 = K.clip(idx2, -DIM//2+XDIM+1, DIM//2)

    # Obtencio dels valors dels pixels originals
    idx3 = tf.stack([DIM//2-idx2[0,], DIM//2-1+idx2[1,]])
    d    = tf.gather_nd(image, tf.transpose(idx3))

    # Retorn de la imatge girada, tallada, ampliada i desplacada a l'atzar
    return tf.reshape(d,[DIM, DIM,3])
```

A.6 transformació de dades

A.6.1 Augment de pèl

A partir de la funció *hair_aug_tf* [6] que utilitza una simulació de pèl, es realitza un augment de pèl aleatori en les imatges de ferides cutànies.

```
# HAIR AUGMENTATION

# Carrega del conjunt d'imatges formades per pels
GCS_PATH_hair_images = KaggleDatasets().get_gcs_path('melanoma-hairs')
hair_images = tf.io.gfile.glob(GCS_PATH_hair_images + '/*.png')
hair_images_tf=tf.convert_to_tensor(hair_images)

# Definicio del nombre maxim de pels a inserir a cada imatge
n_max= 20
```

```
# Com que les imatges dels pels tenen una mida 256x256, es transformen i s'escalen
    les imatges a mida 256x256
if IMG_SIZES != 256:
    scale=tf.cast(CFG['crop_size']/256, dtype=tf.int32)


def hair_aug_tf(input_img, augment=True):

    if augment:

        # Copia de la imatge original per tal de no modificar-la
        img = tf.identity(input_img)

        # Desnormalitzar: retornar la imatge de 0-1 a 0-255
        img = tf.multiply(img, 255)

        # Assignacio aleatoria del nombre de pels a augmentar (fins a n_max)
        n_hairs = tf.random.uniform(shape=[],
            maxval=tf.constant(n_max)+1,dtype=tf.int32)

        im_height = tf.shape(img)[0]
        im_width = tf.shape(img)[1]

        # En el cas que no s'insereixin pels, es retorna la imatge normalitzada a
            [0,1]
        if n_hairs == 0:
            img = tf.multiply(img, 1/255)
            return img

        # Seguidament en cada iteracio fins n_hairs (definit previament) s'insereix
            un pel en la imatge.
        for _ in tf.range(n_hairs):

            # Obtencio aleatoria d'una imatge de pel
            i = tf.random.uniform(shape=[],
                maxval=tf.shape(hair_images_tf)[0],dtype=tf.int32)
            fname = hair_images_tf[i]
            bits = tf.io.read_file(fname)
            hair = tf.image.decode_jpeg(bits)
```

```
# En cas que sigui necessari, escalar la imatge del pel a mida 256x256
if IMG_SIZES != 256:
    new_width = scale*tf.shape(hair)[1]
    new_height = scale*tf.shape(hair)[0]
    hair = tf.image.resize(hair, [new_height, new_width])

# Girs aleatoris de la imatge del pel
hair = tf.image.random_flip_left_right(hair)
hair = tf.image.random_flip_up_down(hair)

# Rotacions aleatories de la imatge del pel
n_rot = tf.random.uniform(shape=[], maxval=4,dtype=tf.int32)
hair = tf.image.rot90(hair, k=n_rot)

# Alcada i amplada de la imatge del cabell
h_height = tf.shape(hair)[0]
h_width = tf.shape(hair)[1]

# Definicio de les coordenades de la part superior esquerra de la regio
    dinteres (roi) on es realitzara laugment
roi_h0 = tf.random.uniform(shape=[], maxval=im_height - h_height + 1,
    dtype=tf.int32)
roi_w0 = tf.random.uniform(shape=[], maxval=im_width - h_width + 1,
    dtype=tf.int32)

# Definicio regio d'interes
roi = img[roi_h0:(roi_h0 + h_height), roi_w0:(roi_w0 + h_width)]

# Conversio a escala de grisos la imatge del pel
hair2gray = tf.image.rgb_to_grayscale(hair[:, :, :3])

# Definicio llindar de tolerancia
mask = hair2gray>10

img_bg = tf.multiply(roi, tf.cast(tf.image.grayscale_to_rgb(~mask),
    dtype=tf.float32))
hair_fg = tf.multiply(tf.cast(hair[:, :, :3], dtype=tf.int32),
    tf.cast(tf.image.grayscale_to_rgb(mask), dtype=tf.int32))
```

```
            dst = tf.add(img_bg, tf.cast(hair_fg, dtype=tf.float32))

            paddings = tf.stack([[roi_h0, im_height-(roi_h0 + h_height)], [roi_w0,
                im_width-(roi_w0 + h_width)],[0, 0]])
            # Pad dst amb zeros perque tingui la mateixa forma que la imatge.
            dst_padded=tf.pad(dst, paddings, "CONSTANT")

            # Creacio d'un boolea mask amb zeros als pixels del segment d'augment i
                uns a qualsevol altre lloc
            mask_img=tf.pad(tf.ones_like(dst), paddings, "CONSTANT")
            mask_img=~tf.cast(mask_img, dtype=tf.bool)

            # Foradar la imatge original a la ubicacio del segment d'augment
            img_hole=tf.multiply(img, tf.cast(mask_img, dtype=tf.float32))

            # Insercio del segment augmentat al forat creat
            img = tf.add(img_hole, dst_padded)

        # Es retorna la imatge normalitzada a [0,1]
        img = tf.multiply(img, 1/255)

        return img
    else:
        return input_img
```

A.6.2 Normalització del color

Es normalitza la banda vermella del model de color RGB per tal de discriminar el to de pell de la imatge de les lesions de pell, per fer-ho s'utilitza la funció *norm_color*:

```
def norm_color(image, norm = True):
    # Comprovacio sobre si aplicar normalitzacio
    if norm:
        img = image
        # Normalitzacio del canal RGB vermell
        red_norm = img[:,:,0]/tf.sqrt(tf.math.reduce_sum(img**2, axis = -1))
```

```
        # Creacio d'una nova imatge utilitzant el canal vermell normalitzat
            juntament amb els canals verd i blau originals
        norm_img = tf.stack([red_norm,img[:,:,1],img[:,:,2]], axis = -1)

        # Retorn imatge normalitzada
        return norm_img
    else:
        return image
```

A.7 Lectura conjunts de dades

Seguidament s'adapten les funcions *read_labeled_tfrecord*, *read_unlabeled_tfrecord*, *prepare_image* i *get_dataset* [4]

```
# Funcio per a llegir les metadades emmagatzemades en els fitxers tfrecord
    etiquetats
def read_labeled_tfrecord(example):
    tfrec_format = {
        'image'                     : tf.io.FixedLenFeature([], tf.string),
        'image_name'                : tf.io.FixedLenFeature([], tf.string),
        'patient_id'                : tf.io.FixedLenFeature([], tf.int64),
        'sex'                       : tf.io.FixedLenFeature([], tf.int64),
        'age_approx'                : tf.io.FixedLenFeature([], tf.int64),
        'anatom_site_general_challenge': tf.io.FixedLenFeature([], tf.int64),
        'diagnosis'                 : tf.io.FixedLenFeature([], tf.int64),
        'target'                    : tf.io.FixedLenFeature([], tf.int64)
    }
    example = tf.io.parse_single_example(example, tfrec_format)
    return example['image'], example['target']


# Funcio per a llegir les metadades emmagatzemades en els fitxers tfrecord no
    etiquetats
def read_unlabeled_tfrecord(example, return_image_name):
    tfrec_format = {
        'image'                     : tf.io.FixedLenFeature([], tf.string),
```

```
        'image_name'                : tf.io.FixedLenFeature([], tf.string),
    }
    example = tf.io.parse_single_example(example, tfrec_format)
    return example['image'], example['image_name'] if return_image_name else 0

# Funcio de tractament de la imatge
def prepare_image(img, cfg=None, augment=True):

    # Copia de la imatge original per tal de no modificar-la
    img = tf.image.decode_jpeg(img, channels=3)

    # Escalar la imatge
    img = tf.image.resize(img, [cfg['read_size'], cfg['read_size']])

    # Es normalitza la imatge a [0,1]
    img = tf.cast(img, tf.float32) / 255.0

    if augment:

        # Transformacio de la imatge mitjancant la funcio transform
        img = transform(img, cfg)

        # Aplicacio de voltes aleatories
        img = tf.image.random_flip_left_right(img)

        # Alteracio aleatoria de la tonalitat de la imatge
        img = tf.image.random_hue(img, 0.01)

        # Alteracio aleatoria de la saturacio de la imatge
        img = tf.image.random_saturation(img, 0.7, 1.3)

        # Alteracio aleatoria del contrast de la imatge
        img = tf.image.random_contrast(img, 0.8, 1.2)

        # Alteracio aleatoria de la brillantor de la imatge
        img = tf.image.random_brightness(img, 0.1)

        # Augment aleatori de pel mitjanant la funci hair_aug_tf
        img = hair_aug_tf(img, augment=cfg['hair_augm'])
```

```
        # Normalitzacio del canal de Color Vermell mitjancant la funcio norm_color
        img = norm_color(img, norm=cfg['red_norm'])

    else:
        img = tf.image.central_crop(img, cfg['crop_size'] / cfg['read_size'])

    img = tf.image.resize(img, [cfg['net_size'], cfg['net_size']])

    # Retorn de la imatge transformada

    img = tf.reshape(img, [cfg['net_size'],cfg['net_size'], 3])

    return img

# Funcio que comptabilitza el nombre d'imatges del conjunt
def count_data_items(filenames):
    n = [int(re.compile(r"-([0-9]*)\.").search(filename).group(1))
         for filename in filenames]
    return np.sum(n)

# Funcio per obtenir el conjunt de treball (entrenament o test) a partir dels
    fitxers d'entrada

def get_dataset(files, augment = False, shuffle = False, repeat = False,
                labeled=True, return_image_names=True, batch_size=16, dim=256):

    ds = tf.data.TFRecordDataset(files, num_parallel_reads=AUTO)
    ds = ds.cache()

    # Insercio de repeticions de dades en el conjunt de fitxers ds
    if repeat:
        ds = ds.repeat()

    # Barreja aleatoria del conjunt de fitxers
    if shuffle:
        ds = ds.shuffle(1024*8)
        opt = tf.data.Options()
        opt.experimental_deterministic = False
```

```
        ds = ds.with_options(opt)

    # Es llegeixen les metadades emmagatzemades en el conjunt de fitxers ds
        mitjancant les funcions read_labeled_tfrecord i read_unlabeled_tfrecord
    if labeled:
        ds = ds.map(read_labeled_tfrecord, num_parallel_calls=AUTO)
    else:
        ds = ds.map(lambda example: read_unlabeled_tfrecord(example,
            return_image_names), num_parallel_calls=AUTO)

    # Tractament de la imatge mitjancant la funcio prepare_image
    ds = ds.map(lambda img, imgname_or_label: (prepare_image(img, augment=AUGMENT,
        cfg=CFG),imgname_or_label), num_parallel_calls=AUTO)

    # Ajustament de la mida del Batch
    ds = ds.batch(batch_size * REPLICAS)
    ds = ds.prefetch(AUTO)

    # Retorn conjunt
    return ds
```

A.8 Mostrar conjunts dades

A continuació es defineix la funció *show_dataset* per a mostrar exemples de les imatges del conjunt de dades i posteriorment es mostra el conjunt:

```
# Funcio per a mostrar les imatges d'un conjunt de dades
def show_dataset(thumb_size, cols, rows, ds):
    mosaic = PIL.Image.new(mode='RGB', size=(thumb_size*cols + (cols-1),
        thumb_size*rows + (rows-1)))

    for idx, data in enumerate(iter(ds)):
        img, target_or_imgid = data
        ix  = idx % cols
        iy  = idx // cols
        img = np.clip(img.numpy() * 255, 0, 255).astype(np.uint8)
        img = PIL.Image.fromarray(img)
```

```
        img = img.resize((thumb_size, thumb_size), resample=PIL.Image.BILINEAR)
        mosaic.paste(img, (ix*thumb_size + ix,
                           iy*thumb_size + iy))

    display(mosaic)

# Mostrar el conjunt d'entrenament mitjancant la funcio show_dataset
ds = get_dataset(files_train, CFG, labeled=False).unbatch().take(12*5) # augment =
    False
show_dataset(64, 12, 5, ds)
```

A.9 Definició del model

Per a la construcció del model s'utilitza la funció *build_model* definida a continuació:

```
def build_model(dim, ef):

    # Definicio de la mida d'entrada
    inp = tf.keras.layers.Input(shape=(dim,dim,3))

    # Importacio del model EfficientNet (EF) i els pesos definits inicialment
    base = ef(input_shape=(dim,dim,3),weights=MODEL_WEIGHTS,include_top=False)
    x = base(inp)

    # S'afegeix al model una capa GlobalAveragePooling2D i una capa densa amb
        funcio d'activacio sigmoid.
    x = tf.keras.layers.GlobalAveragePooling2D()(x)
    x = tf.keras.layers.Dense(1,activation='sigmoid')(x)

    # Definicio del model
    model = tf.keras.Model(inputs=inp,outputs=x)

    # Definicio funcio de perdua
    losses = tf.keras.losses.BinaryCrossentropy(label_smoothing =
        CFG['label_smooth_fac'])

    # Compilacio del model
```

```
    model.compile(optimizer=CFG['optimizer'],loss=losses,metrics=['AUC'])
    return model
```

A.9.1 Taxa d'aprenentatge

Amb l'objectiu d'accelerar l'aprenentatge del model, s'utilitza una adaptació de la funció *get_lr_callback* de [4] i [6] en la qual es defineix la taxa d'aprenentatge en funció de l'època d'entrenament

```
def get_lr_callback(cfg):

    # Obtencio d'informacio de la configuracio CFG definida
    lr_start = cfg['LR_START']
    lr_max = cfg['LR_MAX'] * strategy.num_replicas_in_sync
    lr_min = cfg['LR_MIN']
    lr_ramp_ep = cfg['LR_RAMPUP_EPOCHS']
    lr_sus_ep = cfg['LR_SUSTAIN_EPOCHS']
    lr_decay = cfg['LR_EXP_DECAY']

    def lrfn(epoch):

        # Definicio de la taxa d'aprenentatge per a les epoques d'entrenament
            inicials
        if epoch < lr_ramp_ep:
            lr = (lr_max - lr_start) / lr_ramp_ep * epoch + lr_start

        # Definicio de la taxa d'aprenentatge per a les epoques d'entrenament
            intermedies
        elif epoch < lr_ramp_ep + lr_sus_ep:
            lr = lr_max

        # Definicio de la taxa d'aprenentatge per a les epoques d'entrenament
            intermedies
        else:
            lr = (lr_max - lr_min) * lr_decay**(epoch - lr_ramp_ep - lr_sus_ep) +
                lr_min
```

```
        # Retorn taxa d'aprenentatge
        return lr

    lr_callback = tf.keras.callbacks.LearningRateScheduler(lrfn, verbose=False)
    return lr_callback
```

A.10 Calibratge dels hiperparàmetres

S'utilitza la validació creuada *K-Fold* per tal de calibrar els hiperparàmetres del model, posteriorment es visualitzen gràficament els resultats del model.

```
# Crear objecte KFold amb el numero de particions definides inicialment
kfold = KFold(n_splits=FOLDS, shuffle = True, random_state = SEED)
max_auc = []

for fold, (train_index, test_index) in enumerate(kfold.split(np.arange(FOLDS))):

    # Netejar memria de la TPU
    if DEVICE=='TPU':

        if tpu: tf.tpu.experimental.initialize_tpu_system(tpu)

    # Obtenir els fitxers del conjunts de validacio
    files_train_fold = tf.io.gfile.glob([GCS_PATH + '/train%.2i*.tfrec'%x for x in
        train_index])

    # En cas que apliqui afeguir les dades del 2019
    if INC2019:
        files_train_fold += tf.io.gfile.glob([GCS_PATH2 + '/train%.2i*.tfrec'%x for
            x in train_index*2+1])
        print("S'utilitzen les dades del 2019")

    # En cas que apliqui afeguir les dades del 2018 i 2017
    if INC2018:
        files_train_fold += tf.io.gfile.glob([GCS_PATH2 + '/train%.2i*.tfrec'%x for
            x in train_index*2])
        print("#### S'utilitzen les dades del 2018 i 2017")
```

```
# Barrejar conjunt d'entrenament
np.random.shuffle(files_train); print('#'*25)

# Obtenir fitxers dels conjunt validacio
files_valid_fold = tf.io.gfile.glob([GCS_PATH + '/train%.2i*.tfrec'%x for x in
    test_index])

# Mitjancant la funcio get_dataset, obtenir els conjunts d'entrenament i
    validacio
ds_train = get_dataset(files_train_fold, augment=AUGMENT, shuffle=True,
    repeat=True,dim=RESIZE[EFN], batch_size = BATCH_SIZES)

ds_valid = get_dataset(files_valid_fold,augment=False,shuffle=False,
    repeat=False,dim=RESIZE[EFN])

# Netejar la sessio
K.clear_session()

print('#'*25); print('FOLD',fold+1); print('#'*25)

# Definir i crear el model Efficientnet a partir de la funcio build_model
constructor = getattr(efn, EFN)
with strategy.scope():
    model = build_model(dim=RESIZE[EFN],ef=constructor)

# Desar el millor model de cada FOLD
sv = tf.keras.callbacks.ModelCheckpoint(
    'fold-%i.h5'%fold, monitor='val_loss', verbose=0, save_best_only=True,
    save_weights_only=True, mode='min', save_freq='epoch')

# Entrenament del model
print('Entrenant el model ',EFN,'...')
history = model.fit(ds_train, epochs=EPOCHS, callbacks =
    [sv,get_lr_callback(CFG)],
                    steps_per_epoch=count_data_items(files_train)/BATCH_SIZES
                    //REPLICAS,
                    validation_data=ds_valid,verbose=0)
```

```
# Netejar la memoria
del model; z = gc.collect()


# Mostrar graficament l'evolucio al llarg de les epoques d'entrenament de les
    metriques AUC i Loss pels conjunts d'entrenament i validacio
plt.figure(figsize=(15,5))

# Metrica AUC
plt.plot(np.arange(EPOCHS),history.history['auc'],'-o',label='Train
    AUC',color='#ff7f0e')
plt.plot(np.arange(EPOCHS),history.history['val_auc'],'-o',label='Val
    AUC',color='#1f77b4')

# Representar el valor maxim de la metrica AUC al conjunt de validacio
x = np.argmax( history.history['val_auc'] ); y = np.max(
    history.history['val_auc'] )
xdist = plt.xlim()[1] - plt.xlim()[0]; ydist = plt.ylim()[1] - plt.ylim()[0]
plt.scatter(x,y,s=200,color='#1f77b4'); plt.text(x-0.03*xdist,y-0.13*ydist,'max
    auc\n%.2f'%y,size=14)

# Definir eixos i llegenda
plt.ylabel('AUC',size=14); plt.xlabel('Epoch',size=14)
plt.legend(loc=2)

# Metrica loss
plt2 = plt.gca().twinx()
plt2.plot(np.arange(EPOCHS),history.history['loss'],'-o',label='Train
    Loss',color='#2ca02c')
plt2.plot(np.arange(EPOCHS),history.history['val_loss'],'-o',label='Val
    Loss',color='#d62728')

# Representar el valor minim de la metrica AUC al conjunt de validacio
x = np.argmin( history.history['val_loss'] ); y = np.min(
    history.history['val_loss'] )
ydist = plt.ylim()[1] - plt.ylim()[0]
plt.scatter(x,y,s=200,color='#d62728'); plt.text(x-0.03*xdist,y+0.05*ydist,'min
    loss',size=14)
```

```
    # Definir eixos, titol i llegenda
    plt.ylabel('Loss',size=14)
    plt.title('FOLD %i'%(fold+1),size=18)
    plt.legend(loc=3)

    # Mostrar grafic
    plt.show()

    # Desar val_auc maxima
    max_auc.append(np.max( history.history['val_auc'] ))

# Calcular i mostrar per pantalla la val_auc mitjana de tots els folds
print('val_AUC mitjana: ', np.mean(max_auc))
```

A.11 Entrenament model

A continuació s'entrena el model definit anteriorment

```
# Definir i crear el model Efficientnet a partir de la funcio build_model
constructor = getattr(efn, EFN)
with strategy.scope():
    model = build_model(dim=RESIZE[EFN],ef=constructor)

# Mitjancant la funcio get_dataset, obtenir els conjunts d'entrenament i validacio
ds_train = get_dataset(files_train, CFG, shuffle=True, repeat=True)
ds_test = get_dataset(files_test, CFG, labeled=False).unbatch().take(12*5)

# Entrenament del model
print('Entrenant el model ',EFN,'...')
history = model.fit(ds_train, epochs=EPOCHS, callbacks =
    [get_lr_callback(CFG)],steps_per_epoch=count_data_items(files_train)/BATCH_SIZES//REPLIC
    verbose=1)
print('\n Finalitzat entrenament del model ', EFN,' \n')
```

A.12 Predicció conjunt de test

Seguidament es realitza la predicció de la classe a la qual pertany cadascuna de les imatges del conjunt de test i s'exporta el resultat en un fitxer CSV:

```
# Calcul dels passos TTA a aplicar en la prediccio del conjunt
cnt_test = count_data_items(files_test)
steps = cnt_test / (CFG['batch_size'] * REPLICAS) * CFG['tta_steps']

# Conjunt de Test Augmentat per tal de fer la prediccio mitjanant TTA
ds_test_aug = get_dataset(files_test, CFG, repeat=True, labeled=False,
    return_image_names=False)

# Prediccio de la classe a la qual pertany cadascuna de les imatges del conjunt de
    test augmentat a partir del model EfficientNet creat anteriorment
pred = model.predict(ds_test_aug,verbose=1, steps=steps)


# Transformacio del vector amb les prediccions per tal d'obtenir una unica
    prediccio per a cada imatge del conjunt test calculada amb la prediccio
# mitjana de totes les transformacions de la mateixa imatge
preds = np.stack(pred)
preds = preds[:,:cnt_test* CFG['tta_steps']]
preds = preds[:df_test.shape[0]*CFG['tta_steps']]
preds = np.stack(np.split(preds, CFG['tta_steps']),axis=1)
preds = np.mean(preds, axis=1)
pred = preds.reshape(-1)

# Obtencio del nom de les imatges del conjunt de proba.
ds_test = get_dataset(files_test, CFG, repeat=False, labeled=False,
    return_image_names=True)
image_names = np.array([img_name.numpy().decode("utf-8") for img, img_name in
    iter(ds_test.unbatch())])

# Creacio de conjunt de dades amb els resultats obtinguts
resultats = pd.DataFrame(dict(image_name = image_names,target = pred))
resultats = resultats.sort_values('image_name')
```

```
# Creacio de fitxer amb format CSV amb els resultats obtinguts
resultats.to_csv(f'{EXECUTION}.csv', index=False)

resultats.head()
```

A.13 Acoblament de models

Finalment, se seleccionen els models amb major precisió i es realitza un acoblament de models a partir de la mitjana:

```
# Llegir les prediccions de cadascun dels models seleccionats
BASEPATH = "../input/melanomapredictions"
pred_5 = pd.read_csv(os.path.join(BASEPATH, '5.csv'))
pred_9 = pd.read_csv(os.path.join(BASEPATH, '9.csv'))
pred_15 = pd.read_csv(os.path.join(BASEPATH, '15.csv'))
pred_17 = pd.read_csv(os.path.join(BASEPATH, '17.csv'))

# Combinacio dels models mitjancant el calcul de la prediccio mitjana (ensambling)
pred = (pred_5['target']+pred_9['target']+pred_15['target']+pred_17['target'])/4

# Creacio del conjunt de dades amb els resultats obtinguts
resultats = pd.DataFrame(dict(image_name = pred_5['image_name'],target = pred))
resultats = resultats.sort_values('image_name')

# Creacio de fitxer amb format CSV amb els resultats obtinguts
resultats.to_csv(f'pred_ensamble.csv', index=False)
```