Universidad de Chile
Facultad de Ciencias Físicas y Matemáticas
Departamento de Ingeniería Eléctrica

Preinforme

Tema 2 - Estimación de mapas de oclusión en redes neuronales

Integrantes: André Teixeira
Raydel Ortigueira
Profesor: Pablo Estévez V.
Auxiliar: Ignacio Reyes J.
Ayudantes: Esteban Reyes D.
Germán García Jara
Mauricio Jofre
Miguel Videla A.
Nicolás Tapia Rivas

Fecha de realización: 7 de octubre de 2019
Fecha de entrega: 7 de octubre de 2019
Santiago, Chile

Índice de Contenidos

Índice de Figuras

Índice de Tablas

1. Introdución

Una de las técnicas que ha surgido para visualizar el comportamiento interno de modelos de clasificación es el mapa de oclusión.

1.1. Mapas de Oclusión

El mapa de oclusión de una imagen arbitraria es un mapa que indica cuales regiones de la imagen son más importantes para el modelo cuando efectúa la clasificación (Fig.1). Este mapa se genera al calcular la certeza que produce el clasificador para la clase de la imagen analizada, cuando a la entrada de este se colocan varias versiones de la imagen, donde cada una de ellas tiene tapada una región distinta, la cual se va desplazando a través de la imagen (Fig. 2).

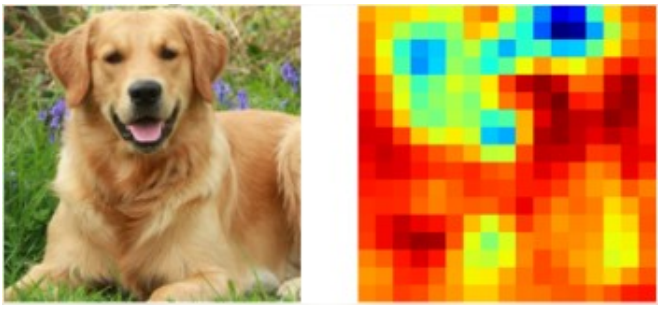

Figura 1: Ejemplo de mapa de oclusión

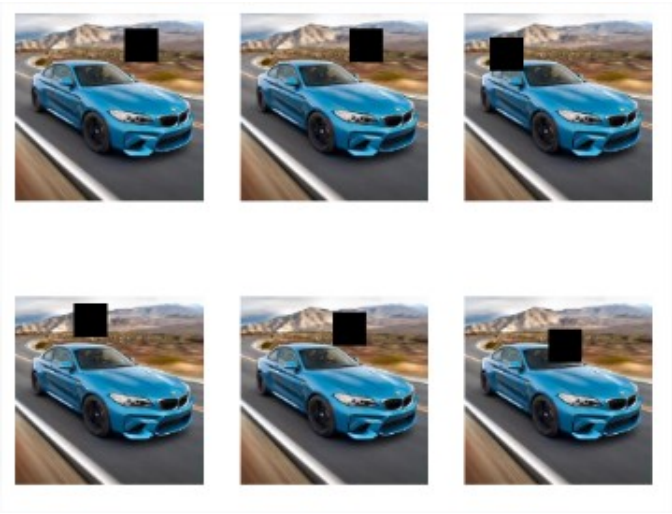

Figura 2: Desplazamiento de los parches que ocultan información de la imagen

La omisión de información de la imagen para el clasificador causará un impacto en el valor de la certeza al clasificar la imagen, por lo que si son omitidas las regiones de la imagen que más información útil aportan al clasificador el valor de la certeza del modelo será pobre. El resultado de estas predicciones iterativas, causadas por el continuo desplazamiento del parche que oculta

información, será un mapa que mostrará las regiones de la imagen donde las certezas más bajas corresponden a las regiones más importantes que necesita el modelo para clasificar correctamente [1].

1.2. Problema

Cada pixel del mapa de oclusión es obtenido cuando el modelo efectúa una inferencia por cada posición del parche que oculta información de la imagen. Por lo tanto, si la imagen cuenta con NxN pixeles, entonces el mapa requiere de NxN inferencias del modelo, por lo que el problema de calcular un mapa de oclusión es que consume muchísimo tiempo.

1.3. Solución propuesta

En este proyecto se buscará estimar estos mapas de oclusión con un autoencoder, el que si aprende a estimar correctamente reduciría el tiempo de cálculo del mapa de NxN a 1. (Fig. 3).

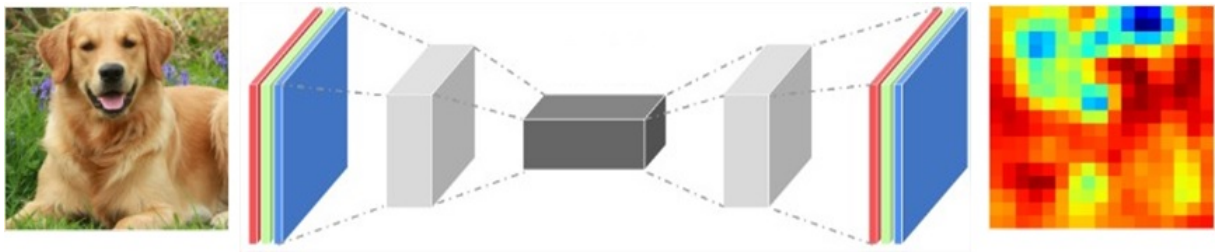

Figura 3: Autoencoder convolucional genérico para solucionar el problema.

1.4. Autoencoder

Un autoencoder es una red neuronal artificial no supervisada que aprende a comprimir y codificar eficientemente los datos, luego aprende a reconstruir los datos de la representación codificada reducida a una representación lo más cercana posible a la entrada original. El entrenamiento luego implica el uso de la propagación hacia atrás para minimizar la pérdida de reconstrucción de la red [2].

1.5. Objetivos del proyecto

Objetivo general:

Reducir el tiempo de cálculo de los mapas de oclusión utilizando un autoencoder convolucional para estimarlos.

Objetivos específicos:

- Entrenar una red convolucional simple para clasificar supernovas y no-supernovas a partir de imágenes astronómicas [4].
- Calcular los mapas de oclusión de las imágenes utilizando el clasificador entrenado.
- Entrenar un autoencoder convolucional con los mapas de oclusión calculados.

2. Base datos

2.1. Datos a usar

La base de datos disponible [3] para este proyecto cuenta con 100.000 ejemplos de imágenes astronómicas clasificadas en 2 categorías: Supernovas y No Supernovas. Cada ejemplo cuenta con 4 tipos de imágenes sumando un total de 400.000 imágenes en la base de datos.

Fueron extraídas de la base de datos sus dimensiones y fueron cuantificados algunos aspectos de interés donde la información más notable es que la base de datos se encuentra balanceada, cuenta con igual cantidad de ejemplos positivos y negativos (Tabla 1). Luego fueron extraídas algunas estadísticas de interés donde se evidencia que los datos están normalizados entre 0 y 1. (Tabla 2)

En este proyecto, la base de datos se dividió en 75 % para entrenamiento y 25 % para pruebas. De los cuales 75 % de entrenamiento, 25 % fueron utilizados para la validación.

Nombre Archivo: HiTS2013_100k_samples(4_channels)_images_labels.pkl

Tabla 1: Dimensiones de los datos

Dimensiones de los datos	
Dimensiones de la base de datos	100000x21x21x4
Dimensiones de las imágenes	21x21x1
Número de ejemplos	100000
Cantidad de tipos de imágenes	4
Cantidad de ejemplos positivos	50000
Cantidad de ejemplos negativos	50000

Tabla 2: Estadísticas de los datos

Estadísticas de los datos	
Valor máximo	1
Valor mínimo	0
Valor promedio	0.43698978
Valor de varianza	0.04454142
Valor de desviación estándar	0.21104838

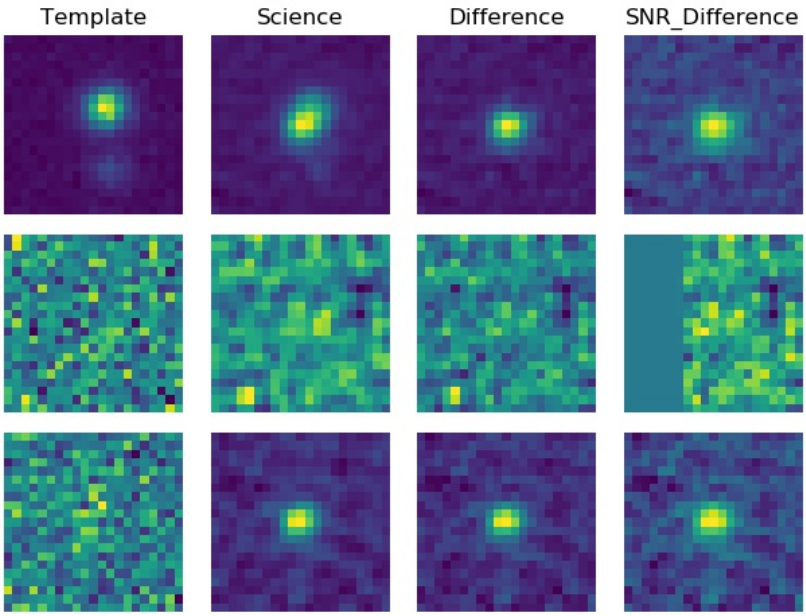

Figura 4: Muestra de 3 ejemplos de las imágenes que contiene la base de datos

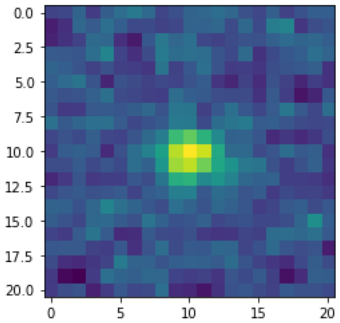

Figura 5: Ejemplo Positivo de una Imagen de Supernova

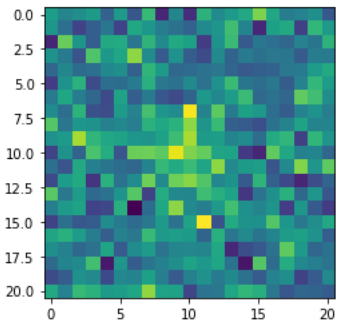

Figura 6: Ejemplo Negativo de una Imagen de Supernova

3. Programación

3.1. Software Usado y Librerías Usadas

Para el proyecto propuesto se utilizó Jupyter Notebook con lenguaje base Python 3 para permitir una ejecución secuencial, en bloque y por partes, y así facilitar la interpretación y prueba del código, debugging y la presentación de resultados.

Las librerías utilizadas fueron:

- Numpy: por su utilidad en el reformateo de imágenes y listas, además de la normalización de valores de píxeles.
- Keras con Backend TensorFlow: para la creación, evaluación, prueba y procesamiento de imágenes de modelos de redes neuronales.
- Pickle: para cargar la base de datos y guardar resultados.
- Matplotlib para mostrar imágenes y plotear los resultados de las métricas y presentar la matriz de confusión.

4. Clasificador

4.1. Arquitectura

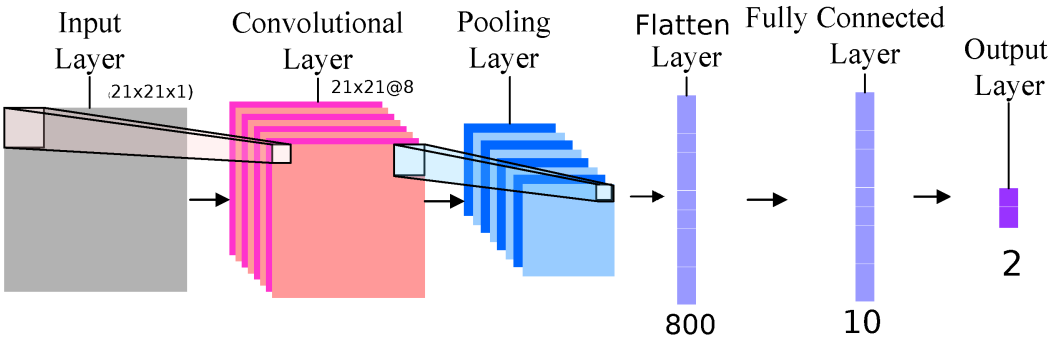

Figura 7: Arquitectura de la Red Neuronal utilizada

Se probaron varios modelos, con varias capas convolucionales y / o fully connected y el que presentó mejores resultados en las pruebas de accuracy de validación fue la arquitectura con:

1 capa convolucional con 8 mapas de características más Maxpooling de 2, donde se hace flatten para 2 fullyconnected compuestas por 10 neuronas en la capa oculta y 2 neuronas en la capa de salida.

En la capa convolucional se hace Zero Padding para que los feature maps mantengan la dimension.

Entre la capa oculta y la capa de salida se estableció un valor de Dropout igual a 0.5 para mejorar los resultados en términos de sobreajuste.

```
#########################################################
##                    Con Zero Padding                 ##
#########################################################
##    21x21x1 -> 21x21x8 -> 10x10x8 -> FC=10 -> FC=2   ##
#########################################################

def CNN_model(num_filters = 8 , filter_size = 3, pool_size = 2, batch= 10):
    #num_filters = 8  #profundidad
    #filter_size = 3  #3x3 filter (conv3)
    #pool_size = 2    # Max Pooling de 2
    #batch= 10        #batch size


    model = Sequential([
      Conv2D(num_filters, filter_size, padding='same', input_shape=input_shape),
      MaxPooling2D(pool_size=pool_size),
    ])

    model.add(Flatten())
    model.add(Dense(10))
    model.add(Activation('relu'))
    model.add(Dropout(0.5))
    model.add(Dense(2))
    model.add(Activation('softmax'))

    sgd = optimizers.SGD(lr=0.01)   #Stochastic gradient descent optimizer
    loss_cce = losses.sparse_categorical_crossentropy   #sparse categorical crossentropy

    model.compile(optimizer=sgd,
                  loss=loss_cce,
                  metrics=['accuracy'])
    return model
```

```
model=CNN_model()
```

Figura 8: Código Modelo de Clasificador

4.2. Salidas deseadas, Función objetivo, Principio de optimización

La salida deseada del clasificador es un vector bidimensional con la probabilidad de que no sea supernova y supernova en cada posición, respectivamente.

La funcion objetivo que queremos minimizar es la función Loss sparse categorical crossentropy. De todas las funciones, esta es la que mejor se ajusta a nuestro clasificador y base de datos, porque queremos clasificar las imágenes entre 2 clases con etiquetas binarias.

El optimizador utilizado fue el stochastic gradient descent con un valor de learning rate igual a 0.01, porque se quería actualizar los pesos en minibatches y en comparación con el optimizador de Adam, mostraba mejores resultados de prueba en términos de Loss y Accuracy.

4.3. Lista de Parámetros del Clasificador

Tabla 3: Parámetros Clasificador

Parámetro	Valor Por Defecto	**Descripción**
tipo	3	Tipo de Clase de Imagen
num_filters	8	Número de feature maps, profundidad de la capa
filter_size	3	Tamaño del filtro cuando se hace la convolución
pool_size	2	Cuanto se quiere hacer de Pooling
batch	10	Tamaño de los minibatches
lr	0.01	Learning rate
x_train		Ejemplos para entrenamiento
x_test		Ejemplos para prueba
r_x_train		Ejemplos para entrenamiento preprocessados
y_train		Labels para entrenamiento
y_test		Labels para prueba

4.4. Criterios de detención

Como criterios de detención fueron utilizados dos:

- Número de Epochs: es el número de veces que el programa se repite, donde en cada época tenemos 56250 iteraciones que corresponden al número de muestras de entrenamiento.
- EarlyStopping: este criterio finaliza el entrenamiento para evitar el sobreajuste y que los resultados empeoren, donde el criterio de parada se estableció cuando la curva de validación de la loss function alcanzara su valor mínimo.

EarlyStopping para evitar el sobreajuste y evitar que los resultados empeoren.

EarlyStopping(monitor='val_loss', mode='min', verbose=1)

Aquí finaliza el entrenamiento si encuentra un mínimo en los valores de loss en la validación.

5. Recursos Computacionales

5.1. Número Parametros Y Espacio En Memória

Calculo de pesos/parametros y espácio en la memoria (1 peso= 4 bytes)

Image: [21x21x1] weights: 0
CONV3-8: [21x21x8] weights: ((3*3+1)* 1)*8 = 80 pesos , memoria=320 bytes
maxpool2: [10x10x8] weights: 0
Flatten: [1x1x800] weights: 0
FC-10: [1x1x10] weights: (800+1)*(10+1) = 8010 pesos , memoria=32040 bytes
FC-10: [1x1x2] weights: (10+1)*(2+1) = 22 pesos , memoria=88 bytes

Número total de parámetros o pesos = 8112, memoria= 32448 bytes.

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 21, 21, 8)	80
max_pooling2d_1 (MaxPooling2	(None, 10, 10, 8)	0
flatten_1 (Flatten)	(None, 800)	0
dense_1 (Dense)	(None, 10)	8010
activation_1 (Activation)	(None, 10)	0
dropout_1 (Dropout)	(None, 10)	0
dense_2 (Dense)	(None, 2)	22
activation_2 (Activation)	(None, 2)	0

Total params: 8,112
Trainable params: 8,112
Non-trainable params: 0

Figura 9: Model Summary

5.2. Tiempo de Iteraciones y Epochs

Los resultados de los tiempos de inferencia fueron un promedio de 24 segundos por epoch con 421 μ.segundos por iteración.

6. Occlusion Map Generator

Para generar los mapas de oclusión requeridos se creó la clase llamada 'OcclusionGenerator'.

Esta clase es usada para crear imágenes ocluidas, es decir, la función 'flow' crea una imagen con un parche negro con valores de píxel igual a 0.Al volver a llamar a la función, se mueve el parche negro por toda la imagen (Fig.10).

Con esta clase también es posible generar minibatches de imágenes ocluidas normalizadas utilizando la función 'gen_minibatch'

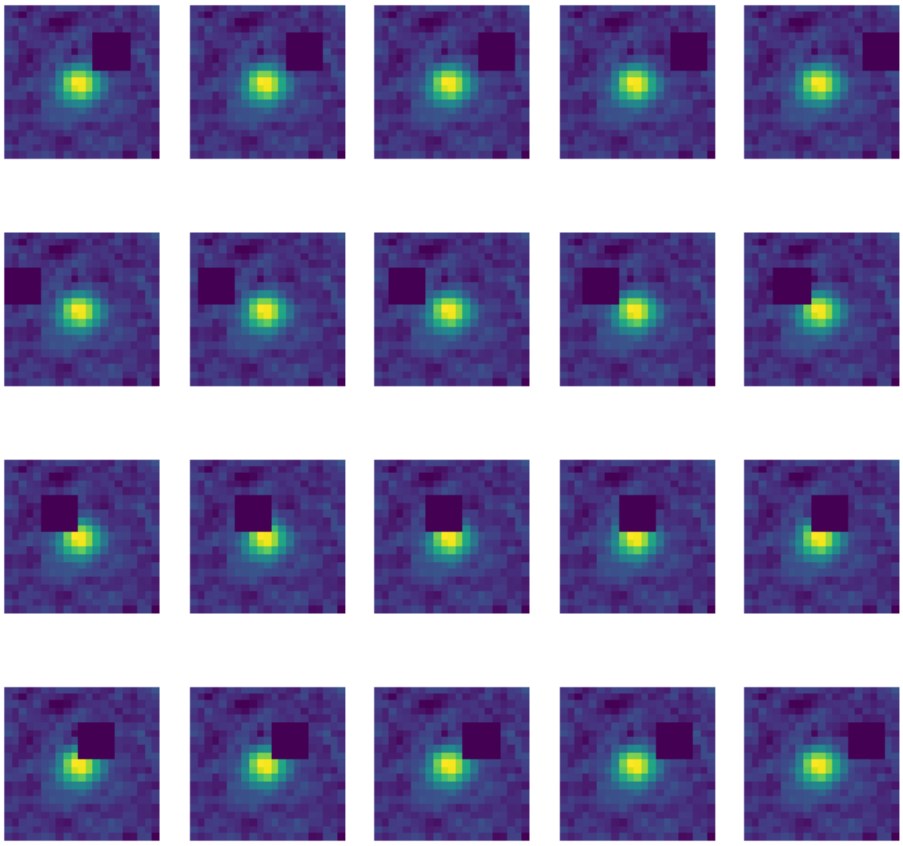

Figura 10: Ejemplo de una proba de la clase del generador de oclusión

Para generar la lista de heatmaps para una determinada imagen, creamos la función gen_heatmap, que se encarga de cargar la imagen, clasificarla con el clasificador entrenado que fue creado y guardado previamente, llamado 'super_nova_class.model'.

Luego, la función gen_heatmap genera minibatches y devuelve una lista de mapas de calor. Este proceso se repite para todos los minibatches.

Para obtener el mapa de oclusión finalmente, fusionamos la lista de heatmaps en una sola imagen, sumándolos y normalizándolos para que los píxeles tengan valores entre 0 y 1.

7. Carta Gantt

Actividades	Horas	Inicio	Final	07-oct	14-oct	21-oct	28-oct	04-nov	11-nov	18-nov
Tarea 1	3	07/10/2019	07/10/2019							
Tarea 2	3	08/10/2019	08/10/2019							
Tarea 3	3	09/10/2019	09/10/2019							
MP2 - (Viernes 11/10)	0,25	11/10/2019	11/10/2019							
Tarea 4	3	21/10/2019	21/10/2019							
MP3 - (Viernes 25/10)	0,25	25/10/2019	25/10/2019							
MP4 - (Jueves y Viernes) (4-8/11)	0,25	04/11/2019	04/11/2019							
Tarea 5	3	11/11/2019	11/11/2019							
Tarea 6	3	15/11/2019	15/11/2019							
Informe Final (18-22/11)	0,25	18/11/2019	18/11/2019							

Figura 11: Carta GANTT

No. Tarea	Tareas
1	Entrenar una red convolucional simple para clasificar las imágenes astronómicas en supernovas y no-supernovas.
2	Calcular mapas de oclusión de imágenes utilizando el clasificador entrenado.
3	Crear una base de datos con mapas de oclusión para entrenar al autoencoder.
4	Entrenar un autoencoder convolucional con los mapas de oclusión calculados.
5	Elaborar el posters de presentación final.
6	Elaborar el informe final que incluye correcciones.

Figura 12: Descripción de las tareas

8. Resultados esperados

Si un autoencoder es entrenado para estimar mapas de oclusión y aprende a estimar correctamente, entonces podrá ser utilizado para calcular estos mapas y reducir el tiempo de procesamiento.

8.1. Medidas de desempeño para evaluar los modelos

- Exactitud
- Función de pérdida
- Matriz de confusión
- Tiempo de inferencia

8.2. Forma en que se presentarán los resultados

- Gráficos de Exactitud y Función de pérdida
- Tabla de Matriz de confusión

9. Resultados Preliminares

Como resultados preliminares fue entrenado nuestro clasificador, donde se obtuvieron valores de accuracy mayores del 90 % con un valor de 94.6 % y un valor de loss function de 0.14, ambos en el proceso de validación (Fig.14-15). Además, se muestran la matriz de confusión lograda (Fig.16) y los mapas de oclusión obtenidos para una imagen clasificada como supernova y otra clasificada como no supernova(Fig.17-18).

9.1. Metricas

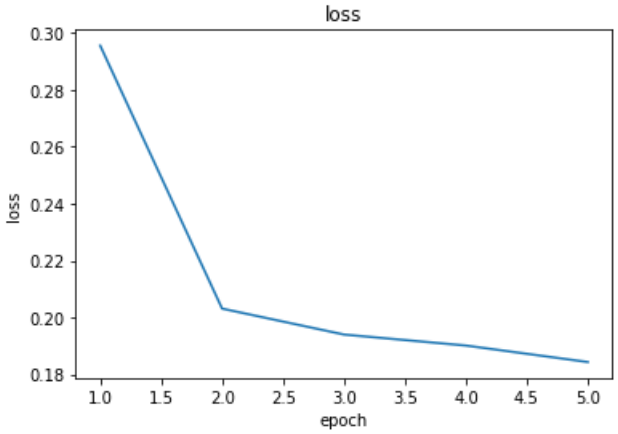

Figura 13: Gráfico de Loss por Epoch

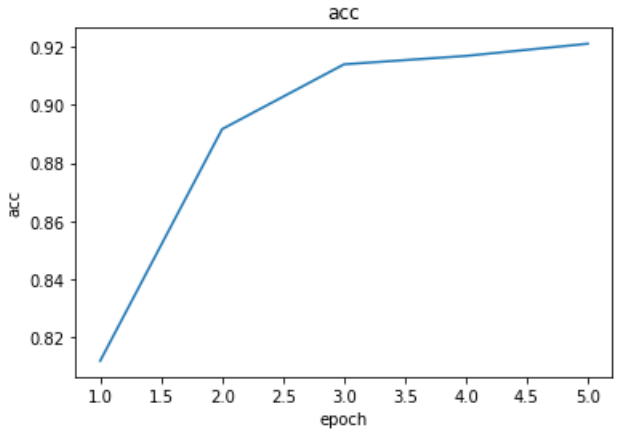

Figura 14: Gráfico de Accuracy por Epoch

```
Testing Accuracy y Loss...

25000/25000 [==============================] - 2s 64us/step

Loss: 14.458135744988917 %

Accuracy: 94.612 %
```

Figura 15: Resultados de Accuracy y Loss en la prueba

9.2. Matriz confusión

```
Confusion Matrix, without normalization
[[12386   105]
 [ 1242 11267]]
```

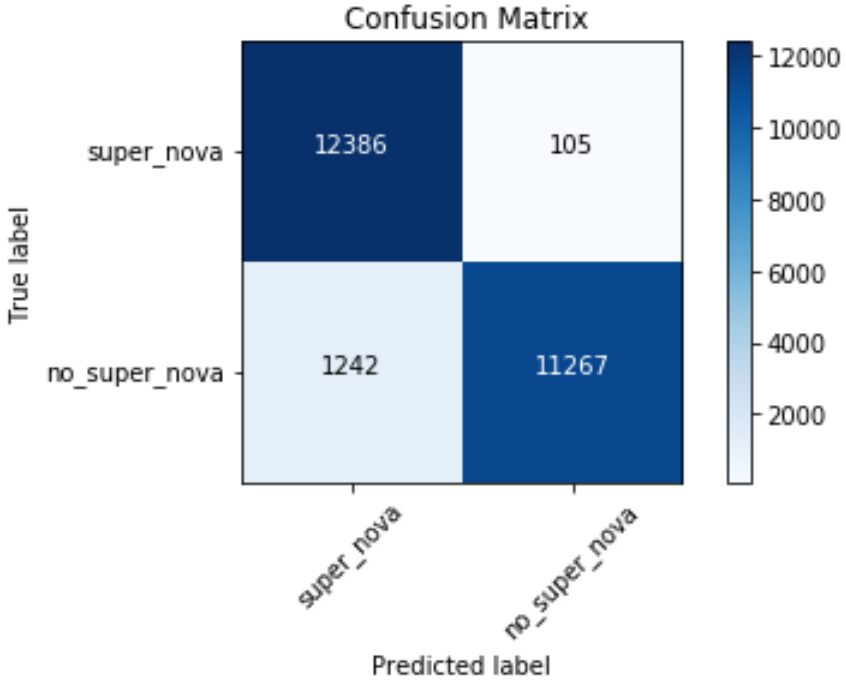

Figura 16: Matriz de confusión no normalizada

9.3. Mapas de Oclusión

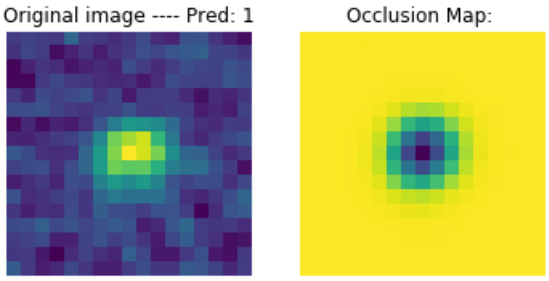

Figura 17: Ejemplo Mapa de Oclusión para una muestra Positiva

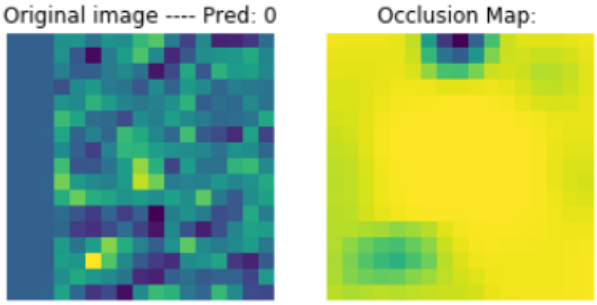

Figura 18: Ejemplo Mapa de Oclusión para una muestra Negativa

Referencias

[1] *Occlusion experiments for image segmentation,*
https://github.com/akshaychawla/Occlusion-experiments-for-image-segmentation

[2] W.Badr, *Autoencoder: What is it? And what is it used for? (Parte 1),*
https://towardsdatascience.com/auto-encoder-what-is-it-and-what-is-it-used-for-part-1-3e5c6f017726

[3] E.Dirk, *Bases de datos de imágenes astronómicas: HiTS2013,*
https://drive.google.com/drive/folders/1bjm8VnxkxrZIYpSLaOCLziyHuHngu3U_

[4] G. Cabrera-Vives, I. Reyes, F. Förster, P. A. Estévez, and J.-C. Maureira, "Deep-HiTS: Rotation Invariant Convolutional Neural Network for Transient Detection," Astrophys. J., vol. 836, no. 1, p. 97, 2017.