Project Report: Classifying Cosmological Data with Machine Learning

Samuel Gabe
School of Physics and Astronomy
University of Nottingham

11 January 2022

Student ID: [redacted] Samuel Gabe

Abstract

This report documents our project to use machine learning to classify cosmological images based on if they have been gravitationally lensed. From 2016 to 2017, there was a challenge to classify 100,000 of these images based on whether or not they had been gravitationally lensed. The highest scoring submission then was a convolutional neural network (CNN) developed by ASTRO EPFL, which categorised images with an AUROC of 0.93. We investigated the progress of machine learning since then by using the EfficientNetB0 CNN[13] and the TensorFlow Python library (tensorflow.org). We categorised cosmological data of the same type with an AUROC of 0.96 ± 0.01. We used a different dataset to calculate this result, containing 2,000 images instead of the 100,000 which were used in the gravitational lens finding challenge, but we believe there are a sufficient number of images in this dataset such that our result would be the same if the model were tested on the larger dataset.

Contents

Student ID: [redacted] Samuel Gabe

1 Introduction

1.1 Introduction to the Gravitational Lensing Challenge

Strong gravitational lenses are "rare cases in which a distant galaxy or quasar is aligned so closely with a foreground galaxy or cluster of galaxies that the gravitational field of the foreground object creates multiple, highly distorted images of the background object." [4] Before the 2010s, the primary method of finding these lenses was with human inspection. As of 2019, less than a thousand lenses had been found[7]; however, due to the technological advancements of the last few years, many more have been discovered since then. In 2021, 1210 new lenses were discovered using machine learning[4], which is considerably more than the total amount of lenses that had been discovered in 2019.

The goal of the gravitational lensing challenge was to categorise a dataset of 100,000 images based on whether or not they are gravitational lenses. A training dataset was also provided, which contained 20,000 images along with labels for each image specifying whether or not it is a gravitational lens. This training dataset can be used to train a type of program called a neural network to classify the images. The neural network can then be used to classify the larger dataset of 100,000 images. Some images from the training set can be seen in Figure 1.

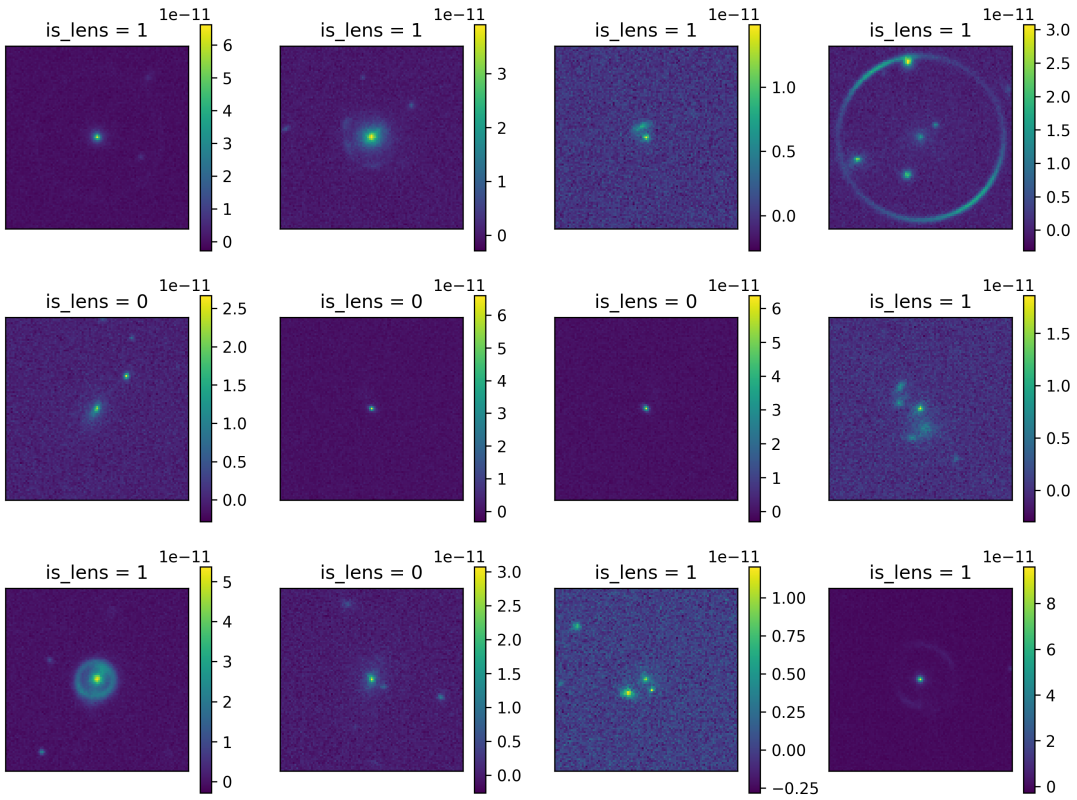

Figure 1: Some images from the training set, along with whether or not each one is a gravitational lens. is lens = 1 indicates the image is a lens. The bars to the right of the images indicate the brightness.

In the gravitational lensing challenge, various methods were used to find lenses. The most successful was the convolutional neural network (CNN). This method accounted for the eight highest scoring submissions and performed better than human inspection.

The paper ranked each submission by finding the receiver operating characteristic (ROC) curve and calculating the area under the curve (AUROC). To understand the ROC curve, we first must understand logistic regression. Figure 2 shows a simple logistic regression model used to classify whether or not images are gravitational lenses, based on some parameter x. We can see that as x increases, the amount of data points with the value False decreases, and the data points with the value True increases. Based on the value of x, we can assign some probability p that the image shows a gravitational lens. We can have a threshold value t, shown in Figure by the dotted orange line, where if $p > t$ we classify the image as a gravitational lens. If $p < t$ we classify it as not a gravitational lens. We can test this model by using images where we know whether or not they are gravitational lenses. For each t, we will have a certain number of false positives and false negatives.

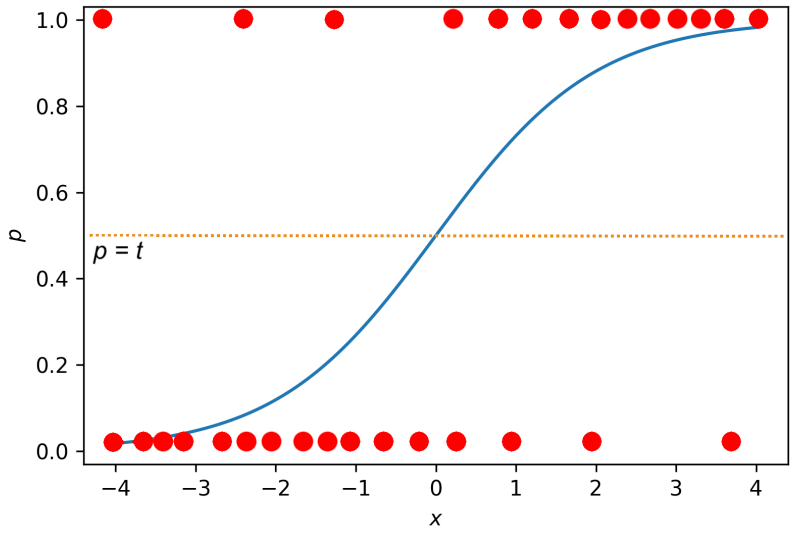

Figure 2: A graph to describe the logistic regression model. This shows the sigmoid function along with some data with some arbitrary parameter x. The data is shown as red points, where the data at $p = 0$ is data where the classification is False (e.g. for our project, the image is not a lens) and the data at $p = 1$ is data where the classification is True (e.g. the image is a lens).

To plot the ROC curve, we plot a graph showing how the false positive rate changes with the false negative rate for various t. A model classifying data randomly would result in a diagonal line from (0, 0) to (1, 1), resulting in an AUROC of 0.5; a model classifying data perfectly would have an AUROC of 1. [7] Examples of ROC curves from classifiers used in the gravitational lens finding challenge are shown in Figure 3.

The ROC curve describes the ability of the model to classify data points into two categories. The graph is created by plotting the true positive rate against the false positive rate, for various values of t. The area under the ROC curve (AUROC) then corresponds to how accurately the model can categorise images of galaxies into the two categories.

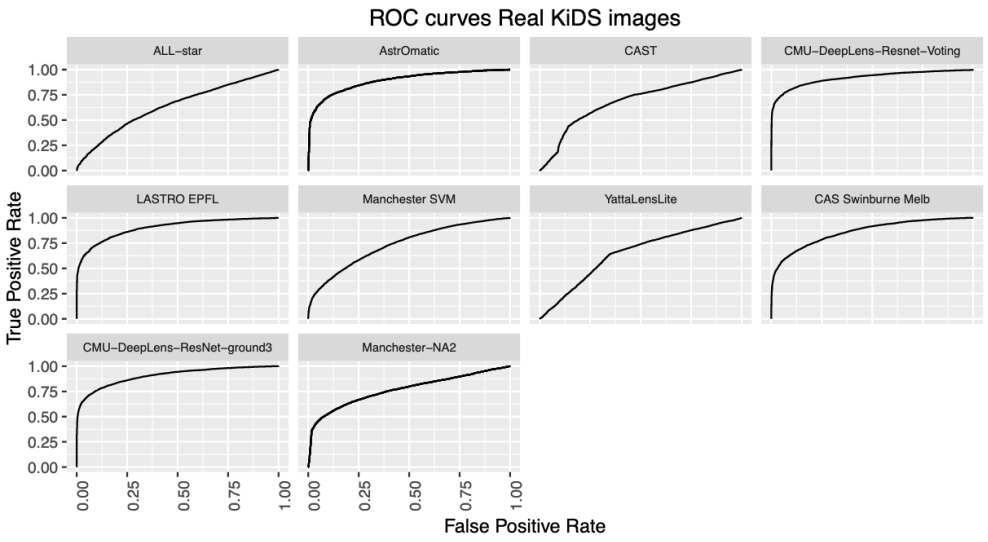

Figure 3: Examples of ROC curves from models in the gravitational lens finding challenge. [7]

1.2 Introduction to Machine Learning

There is a need for computers to place data into different categories based on certain parameters. For example, the MNIST dataset contains images of hand drawn numbers.[2] Every number has been drawn slightly differently. It is practically impossible to manually create an algorithm which can check for certain properties of the image to decide how these numbers should be categorised. The same applies for our dataset with cosmological images. It is unclear to us exactly which features of the image correspond to a gravitationally lensed image and the galaxies are in many different orientations and shapes. Therefore we must use a different solution.

Machine learning is one process which can solve this problem. First, we produce or obtain a dataset, called the training dataset. The training dataset contains the data points and the labels which define how they are classified. We create a machine learning model which takes data as an input. We can then use the dataset to train the model. Using the training dataset, the model will learn how to take a data point and assign it to a category (for this project, there are two categories, one where the image is a gravitational lens and one where it is not). The model is trained over several epochs, or iterations. The model has now been trained, meaning we can use it to classify other similar data. This process, where we use pre-determined labels for our training dataset, is called *supervised learning*. Machine learning has a wide variety of uses across many different fields. It has been used for facial recognition[6], medical diagnoses[12] and analysing the stock market[9].

The highest performing CNN in the gravitational lens finding challenge was developed by LASTRO EPFL, which attained an AUROC score of 0.93. In this project we will attempt to use newer CNN models to better categorise these images. The aim of this project is to determine if there has been an improvement in convolutional neural networks since this challenge was conducted. This report will first discuss machine learning, explaining how an artificial neural network functions. It will then move onto more advanced convolutional neural networks. Then it will explain the problem of overfitting and show ways we can solve this. Then the report will cover how the project was conducted, including the hardware and software used and the

method used to obtain our results. The results will be presented, followed by an analysis and conclusion.

2 Theory

2.1 Network Architecture

Machine learning algorithms have been theorised dating back to at least 1949,[3] but only recently has computational power become sufficient to power these models. Neural networks are computational processing systems which are inspired by how the human brain learns to recognise information. Artificial neural networks (ANNs) take a series of inputs and produce an output. They make use of at least three layers: an input layer, an output layer and one or more hidden layers.[8]

One possible use case of a neural network is to attempt to calculate how well pupils will do on a test. We could calculate this using two metrics: the amount of effort put in by a pupil and their natural talent. These are not quantifiable and therefore cannot be directly used to estimate the pupils' performance on a test. However, we can link these parameters to other parameters which are quantifiable. For instance, we can measure the amount of hours a pupil spends revising for the test and their score on previous tests. The time put into revising can be used to estimate how much effort a pupil has put into a test and their score on previous tests could be used as an indicator of their natural talent, although it will also be affected by the amount of effort they put into those previous tests. Here, we have linked parameters which can't be measured directly to parameters which can be measured directly. The measured parameters can be thought of as the input layer and the parameters which have not been measured can be thought of as a hidden layer. The output layer is the score on the test. A neural network, when given a dataset as described in Section 1.2, can automatically calculate the parameters in the hidden layer.

A training dataset has an input matrix, $\mathbf{X}^{(train)}$ and an output matrix, $\mathbf{Y}^{(train)}$ which are used to train the model. The model will take the input matrix and use the output matrix to create a prediction matrix, $\hat{\mathbf{Y}}(\theta)$, where θ is a parameter of the network. This output matrix evolves over several epochs. The goal of training the neural network is to minimise the loss function,

$$J(\theta) = \frac{1}{N}\sum_{i}^{N} \ell(\hat{\mathbf{y}}^{(\mathbf{i})}(\theta), \mathbf{y}^{(\mathbf{i})}) \tag{1}$$

where $\ell(\hat{\mathbf{y}}^{(\mathbf{i})}(\theta), \mathbf{y}^{(\mathbf{i})})$ is the per-example loss, $\hat{\mathbf{y}}$ is the desired output and $\mathbf{y}$ is the output from the model. The per-example loss depends on the form of the output. Since we are using a binary output (data is placed into two categories), we will use the Binary Cross Entropy (BCE) loss function. This is given by

$$\ell(\hat{\mathbf{y}}^{(\mathbf{i})}(\theta), \mathbf{y}^{(\mathbf{i})}) = -(y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)} \log(1 - \hat{y}^{(i)})). \tag{2}$$

[7]

2.2 Neurons and the Multilayer Perceptron

Neurons links neural networks together. They take an input vector x and produce an output,

$$a = \Phi(z), z = x^T w + b \tag{3}$$

where a is the output, w is the weight of the input and b is a bias of the neuron. The biases can be absorbed into the weights term if we rewrite x and z as vectors,

$$z = \mathbf{x}^T \mathbf{w}, \tag{4}$$

where $\mathbf{x} = (1, x)^T$ and $\mathbf{w} = (b, w)^T$

Each input is considered with respect to its weight. The neuron takes the input vector and weights and uses them to produce an output,

$$a = \Phi(z), \tag{5}$$

where $\Phi(z)$ is the activation function.

Neurons often do not output data with a linear dependency on the input. They usually output a function of the result obtained from combining the inputs and their corresponding weights. One of the most simple activation functions is the sigmoid function,

$$S(x) = \frac{1}{1 + e^{-x}} \tag{6}$$

where x is the input. Other activation functions include linear ($\Phi(z) = z$), hyperbolic tangent ($\Phi(z) = \tanh z$) and ReLU, where

$$\Phi(z) = \begin{cases} z, & \text{if } z > 0 \\ 0, & \text{otherwise.} \end{cases} \tag{7}$$

A multilayer perceptron is a type of neural network with layers of neurons. These neurons link parameters of each network as shown in Figure 4. A neuron takes weighted inputs and produce an output. The multilayer perceptron uses the perceptron activation function, where

$$\Phi(z) = \begin{cases} 1, & \text{if } z > 0 \\ 0, & \text{otherwise.} \end{cases} \tag{8}$$

2.3 Gradient Descent

The training process of the model involves optimising the weights of the neurons. To find the optimal weights which minimise the loss function, we could think about calculating the loss function for a large range of weights. After this, we would know which of these weight leads to the minimum loss function. However, this is highly inefficient and there are in fact much faster ways to reach this minimum. A better method to use is gradient descent. We initialise the model with some initial starting weights. These can be randomly defined, or one can use a set of pre-trained weights. The weights $\mathbf{w}$ are now adjusted some small amount. The new per-example loss functions, $\ell(\hat{\mathbf{y}}^{(\mathbf{i})}(\theta), \mathbf{y}^{(\mathbf{i})})$, are calculated. The gradients $\hat{\mathbf{g}}$ of the loss function in terms of each parameter over the whole dataset are calculated,

$$\hat{\mathbf{g}} = \nabla_{\mathbf{w}} J(\mathbf{w}) \tag{9}$$

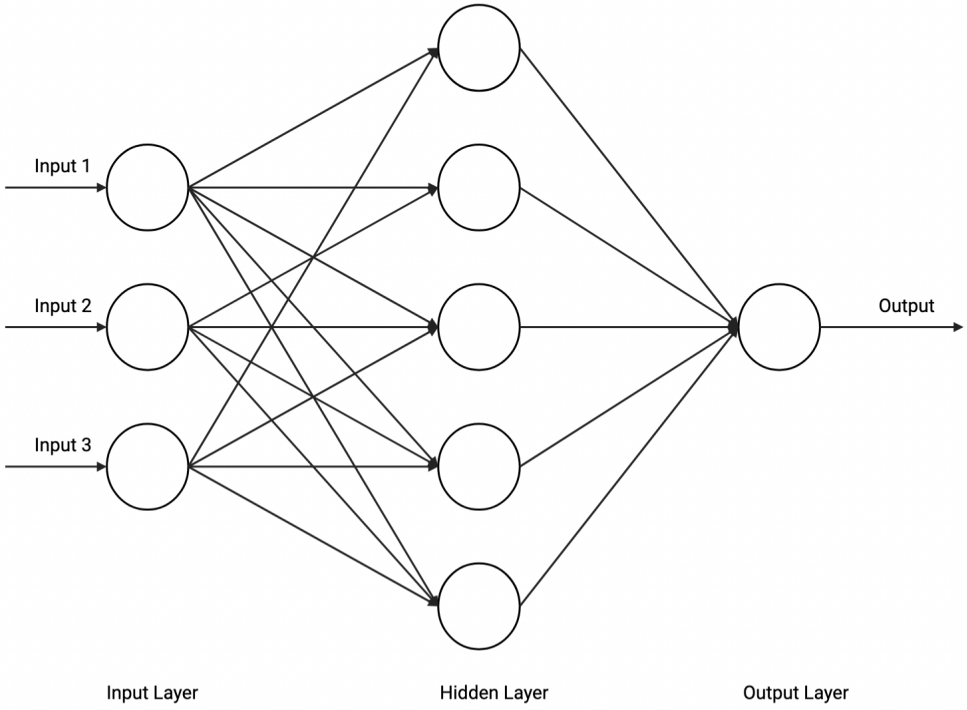

Figure 4: The structure of a multilayer perceptron with four inputs, one output and five hidden parameters.

[10], where N is the total number of images. The gradient of the binary crossentropy loss function is

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = \frac{1}{N} \mathbf{X}^T (\hat{\mathbf{Y}} - \mathbf{Y}). \tag{10}$$

Now, we can adjust the weights in the direction of decreasing gradient,

$$\mathbf{w} \to \mathbf{w} - \alpha \hat{\mathbf{g}}, \tag{11}$$

where α is the learning rate and defines how much the weights should be adjusted each iteration. The learning rate must be carefully defined: if the learning rate is too low, the program will take a long time to reach the minimum of the loss function, but if it is too high it may constantly overshoot the minimum. This can be seen in Figure 5.

However, gradient descent has its flaws. If there is a local minimum, the algorithm may mistakenly think it has found the global minimum when it has only found a local minimum. This is why we need to use an optimisation method. One such method is *stochastic gradient descent*, which is detailed in Section 2.4.

2.4 Stochastic gradient descent

Stochastic gradient descent is an extension of the traditional gradient descent method. Rather than computing the gradient of the loss function for the whole dataset, the algorithm instead

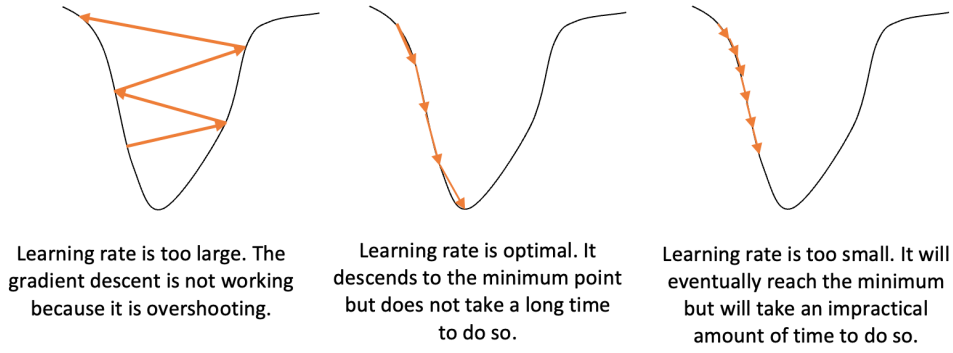

Figure 5: Graphs to show the effect of a learning rate which is too high or too low

computes the gradients for a small random sample of the parameters called a *mini batch*. This vastly improves the run time of the program, because we are calculating far less gradients per epoch. Even though the gradient has only calculated for for a small sample of the parameters, we use it for every parameter.

Stochastic gradient descent has two benefits. As discussed before, it drastically improves the run time of the program. In addition, it means that the gradient will sometimes be estimated higher than it actually is for the parameters which haven't been used in calculating the gradient. This will sometimes cause the algorithm to overshoot and move away from local minima. This is a good thing, because it means that a lower minimum could be found. If we run the program enough times, the global minimum will eventually be found.[10]

2.5 Convolutional Neural Networks

Traditional artificial neural networks are not suited to dealing with image files, which may have tens of thousands of pixels and three colour channels. The number of weights is very large, so the scale of the network will be too large to run using current technology. Additionally, artificial neural networks are prone to overfitting (see Section 2.6). One way of reducing both of these problems is to use a convolutional neural network (CNN). CNNs can recognise patterns in images using filters.

CNNs are comprised of three types of layers:

1. Convolutional layers
2. Pooling layers
3. Fully connected layers.

2.5.1 Convolutional Layers

A convolutional layer has a specified number of filters. A filter is a matrix, usually a square matrix with dimensions 3x3 or 5x5. A section of the image with the same dimensions as the filter is selected. Then, the dot product is calculated between this section of the image and the filter. This dot product is then stored in a new matrix. The filter convolves over every

possible block in the image, storing the dot products in a new matrix. Because we only take one value for each convolution, the width and height of the output matrix will be reduced by $w-1$, where w is the width and height of the convolutional filter, assuming the convolutional filter is square (e.g. for a 3x3 filter, the output matrix will have a width and height 2 pixels less than the width and height of the input matrix). Neurons in convolutional layers do not take inputs from every other neuron in the previous layer.

2.5.2 Pooling Layers

This new matrix is then passed into a pooling layer. There are three types of pooling: max pooling, min pooling and average pooling. The model we will use in this project uses max pooling layers. We again have a filter, which convolves over the image. The stride determines how many pixels the filter moves each convolution - for example, for a stride of 2, the filter will move 2 pixels with each convolution. In each convolution, the maximum value of the pixels in the block is found (hence max pooling). This value is then stored in a new matrix. Again, due to the fact that there are less convolutions than pixels in the image, the dimensions of the new matrix are reduced.

2.5.3 Fully Connected Layers

A CNN will usually use an input layer, followed by, some combination of convolutional layers and max pooling layers, followed by fully connected hidden and output layers.[1] Neurons in convolutional layers do not take inputs from every other neuron in the previous layer, but fully connected layers do. The fully connected input layer has dimensions equal to those of our input data. Then after the convolutional layers, a flattening layer is used to turn the image into a 1D array, so that instead of, say, a 101x101 image matrix, we have a 1x10201 matrix. This is so that we can use a traditional ANN model with our data. Then we have one or more fully connected layers, also known as dense layers. Fully connected layers correspond to the layers found in a typical artificial neural network. The hidden layers have a specified number of neurons, e.g. 64 or 128. Then we have an output layer, which for our project has a dimension of 1, since we only have one output: whether or not our lens is a gravitational lens.

2.6 Overfitting

The methods discussed so far will lead to a model which has been trained to fit to the training data provided. However, the training data, although usually a reasonably large dataset, does not represent every single possibility for data we could provide. Therefore, if we run the model for many epochs, the model could start to categorise the training data with higher and higher accuracy with each epoch, but when we provide a different dataset containing data of the same format, the model will make predictions which get worse and worse with each epoch. The model may become more and more focused on the specific images in the training dataset at the expense of learning the more general features of what makes an image a gravitational lens or not. This phenomenon is called overfitting and there are many ways to combat it.

2.6.1 Data augmentation

The problem of overfitting arises because the training dataset is not general enough for the model to learn how to categorise many types of data. This can be improved by having a

larger training dataset with more data points. As we add more data points to the dataset, the amount of overfitting decreases. However, in this project we only have 20,000 images to work with, so we need to create new images from what we already have. This process is called data augmentation.

One can dramatically increase the number of images in the training dataset by making small changes to the already existing dataset. For example, one could translate, rotate, shear, rescale or flip the images in the dataset to create many new images. However, we have to be careful to only use methods which will not affect how an image could be classified: for example, in the MNIST dataset[2] which contains hand drawn numbers (discussed in Section 1.2), we would have to be careful not to rotate the images 180° because a 6 could turn into a 9. For this project, we do not want any data augmentation which drastically changes the shape of a cosmological object, so we cannot use shearing. This method allows us to dramatically increase the number of images in our training dataset, which reduces overfitting.

Machine learning models perform optimally when data is normalised. This can be done by calculating the mean, μ and standard deviation, σ of all pixels in the dataset. The data can then be normalised as follows:

$$Z = \frac{X - \sigma}{\mu}, \tag{12}$$

where Z is the normalised data and X is the data before normalisation.

2.6.2 Dropout

Another way to reduce overfitting is to use dropout.[11] Sometimes, the weight of one input to a neuron will become much bigger than the others, meaning that the neuron's output is almost entirely dependent on one feature. Dropout fixes this by randomly removing inputs to the neurons, meaning that if we have an input with a disproportionately large weight, there is a random chance of that input being ignored; therefore, neurons can never be dependent on any one feature. This leads to less overfitting.

2.6.3 Validation Data

If we use our training dataset to validate our model's performance, the loss may appear to be lower than it is due to overfitting. To make sure the model is not overfitting, we must use a different set of data. This is called the validation data. When training a neural network, it is common practice to split the given dataset into two separate datasets: the training dataset and the validation dataset. The training dataset is what we use to train the model. Then, we assess our model's performance by using it to categorise the data in the validation dataset. The model has not seen these data before, so we can use this to get an idea of how our model will perform when presented with similar types of data.

Dropout can also improve the performance of neural networks [11], since we have effectively removed a large number of inputs to neurons, which makes for a simpler neural network and therefore an increase in performance.

2.7 Time taken to run the program

Another factor to consider when training a neural network is how long the program takes to run. One method we can use to reduce this is with early stopping. After a certain number of epochs, the model may stop increasing in accuracy. It is impractical and unnecessary to

continue training the model past this point. Early stopping is a mechanism where the program will monitor the AUROC with each epoch. If the validation loss stops decreasing, the program will stop running. However, due to the fluctuating value of the AUROC, the program may stop due to an increase in the AUROC even though the general trend still increase. To avoid the program stopping too early due to fluctuations in the validation loss from epoch to epoch, we can give a patience parameter. This defines how many epochs the program should keep running for after it detects that the validation loss has stopped decreasing.

2.8 Confusion Matrix

The AUROC gives us an indication of how the model performs when categorising data. However, we can not be sure how often it wrongly categorises images which are gravitational lenses versus images which are not. To see this, we can use the confusion matrix M. This is described in Table 1. Here, TN = correctly classified non-lenses, FN = incorrectly classified lenses, FP = incorrectly classified non-lenses, TP = correctly classified lenses.

	Predicted not lens	Predicted lens
Actual not lens	TN	FP
Actual lens	FN	TP

Table 1: The confusion matrix shows how well the data categorises data based on whether it is a gravitational lens or not.

We define the total amount of lenses as P and the total amount of non-lenses as N. The proportion of lenses categorised correctly by the model, or the true positive rate, can be written as

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{P}} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \tag{13}$$

and the proportion of non-lenses categorised correctly by the model, or the true negative rate, can be written as

$$\mathrm{TNR} = \frac{\mathrm{TN}}{\mathrm{N}} = \frac{\mathrm{TN}}{\mathrm{TN} + \mathrm{FP}}. \tag{14}$$

3 Description of how the project was conducted

3.1 Hardware and software used

We used Visual Studio Code to combine the images and run our first models. This allowed us to train very simple models with acceptable speed, but when implementing more complex models, it was clear that our laptops would not have enough computational power to train the models. Therefore, we used Google Colab (http://colab.research.google.com/, a free online software package which allows the user to run Python code through Jupyter notebooks and make use of high performance graphic processing units (GPUs). Our data was stored on Google Drive, which is designed to integrate with Colab. Google Colab provides access to various GPUs, including the Nvidia K80, Nvidia T4 and Nvidia P100. The specific GPU can vary.

3.2 Method

First, we downloaded the space based training set for the 2017 challenge, which can be found at http://metcalf1.difa.unibo.it/blf-portal/gg challenge.html. This contains 20,000 cosmological images with a resolution of 101x101 pixels, along with a table saying whether or not each image is a gravitational lens. The files are in the FITS format, which is commonly used for astronomical images. A single FITS file can hold several images, so we combined the first 18,000 images into one file to use as the training data. Then, the remaining 2,000 images were combined to use as the test data. We handled these FITS files using the AstroPy library for Python. The training dataset has 12574 lenses and 5426 non lenses, while the validation dataset has 1394 lenses and 606 non lenses.

We then create a program to perform our task of creating a model to classify cosmological images based on whether or not they are gravitational lenses. First, we open the training set and the validation set. Then, we convert the image into three channels. This is because the model we will use can only take images with three channels, but the dataset provides images with only one channel. After this, we applied data augmentation to the data in an attempt to reduce overfitting. The data augmentation methods we used were flipping the images both horizontally and vertically, rotating the images different values with a range of 90° and randomly zooming the image with scale factors ranging from 0.6 to 1.4.

After this, we create our model which we will later train. We start by taking the winning model from the gravitational lensing challenge found at [7], since the layers of the model had been detailed. We then compile the model, using the binary crossentropy loss function and the Adam optimiser. The Adam optimiser is an optimiser which builds on stochastic gradient descent (see Section 2.4).[5] After this, trained our model using our training dataset of 18,000 images. After training the model, we used the Scikit-learn library (scikit-learn.org) to calculate the AUROC score. The score over 100 epochs was 0.93, which is the same as what was obtained in the paper, confirming that our code works. We then implemented the EfficientNetB0 model.[13] This model by default takes an input layer with 224x224 pixels and three channels. Using these dimensions means that we would be able to load the ImageNet weights, which have been pre-optimized to make the training process faster; however, resizing our 101x101 images to be this size results in a training dataset of over 23 gigabytes. Google Colab does not have enough RAM for this. EfficientNetB0 allows one to specify not to include the predefined top layers of the model, which are the fully connected layers at the beginning and end of the model. We defined an input layer with 101x101 pixels to match our input data. At the end of the sequence of layers, we added layers as described in the flow chart in Figure 6. We compiled the model, again using the binary cross entropy loss function and ADAM optimizer, and trained the model using our training dataset. We plotted the ROC curve and calculated the AUROC score. We ran this for 100 epochs. The training process was performed 10 times to get an estimate of the uncertainty.

We calculated the confusion matrix by using our model to make predictions for every image in the validation dataset. This gives the probability that the image is a gravitational lens. If the probability is greater than 0.5, we predict that it is a gravitational lens. Otherwise, we predict that it is not a gravitational lens (see Section 2.8. Then, we used TensorFlow's built in confusion matrix function. [1]

[1]The documentation for the TensorFlow confusion matrix function can be found at https://www.tensorflow.org/api docs/python/tf/math/confusion matrix.

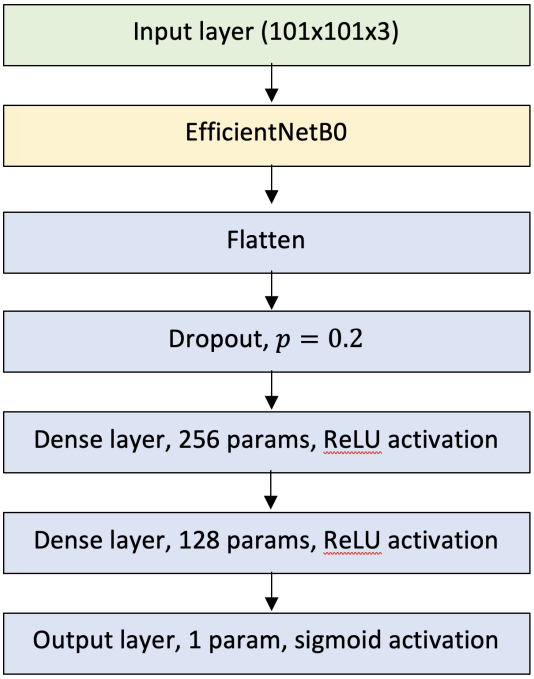

Figure 6: Flow chart describing the architecture of our model.

4 Results

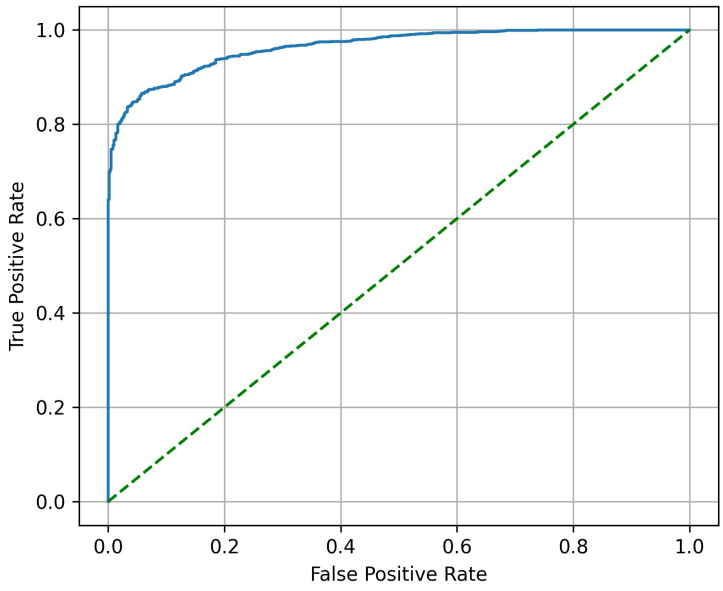

Figure 7: The ROC Curve of the model, as described in Section 1.1. The green dashed line shows what would be obtained by an entirely random model.

The AUROC obtained was 0.96 ± 0.01. The mean and uncertainty were determined by training the model to 100 epochs 10 times and calculating the mean and standard deviation.

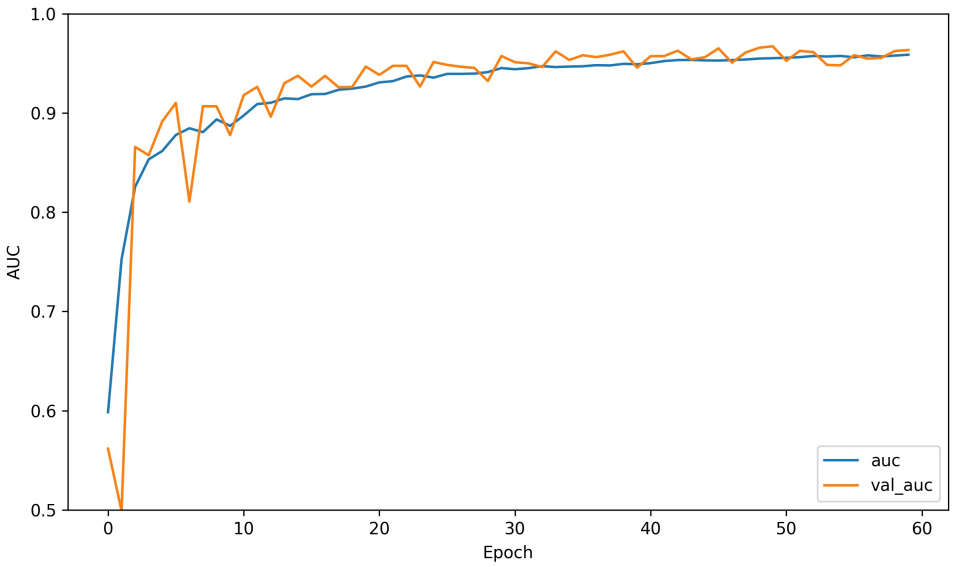

Figure 8: Graph showing how the AUROC score changed with the number of epochs as the model was trained. The line labelled auc shows the training data AUROC, and the line labelled val auc shows the validation data AUROC. When we created this graph, we used early stopping as described in Section 2.7. The program ran for 60 epochs with a patience of 10.

	Predicted not lens	Predicted lens
Actual not lens	436	170
Actual lens	59	1335

Table 2: The confusion matrix obtained using a threshold $t = 0.5$.

The proportion of lenses categorised correctly with $t = 0.5$ was 95.8% and the proportion of non-lenses categorised correctly was 71.9%.

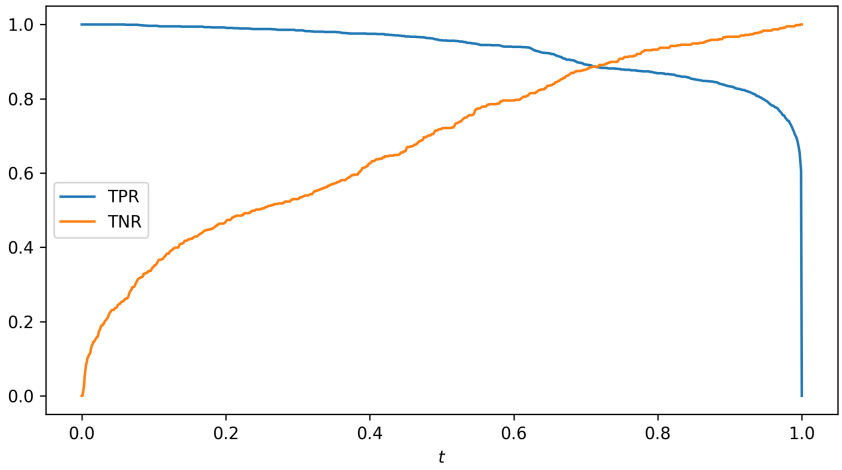

Figure 9: Graph to show how the performance of the model on lenses and non-lenses varies with the threshold t (see Section 2.8).

5 Discussion of Results

The results show that the AUROC score obtained when running the EfficientNetB0 model is higher than that obtained with the ASTRO EPFL model, the highest performing model in the original gravitational lensing challenge. This shows that neural networks have improved since the original challenge took place.

It should be noted that the ASTRO EPFL model used different data to obtain the AUROC score. Whereas we used 2000 validation images from the training set provided, the ASTRO EPFL model used the 100,000 images which the competition provided. If we had used the same dataset, the results may have been slightly different. However, the image sets are the same in nature and our dataset of 2000 is a large sample size, so using the validation data gives a very good estimate of the AUROC which would be obtained if we instead used the larger dataset; however, if we had used the larger dataset, it may have led to a smaller uncertainty in our result.

Other results published since the gravitational lensing challenge took place have also marked a significant improvement. The 2021 paper which found 1210 new lenses used a model which obtained an AUROC score of 0.992 on their validation set.[4] This validation set is different to the one used in this project, so this is not an entirely fair comparison.

Despite the fact that the AUROC attained by the EfficientNetB0 model is higher than that found by the ASTRO EPFL in the 2017 lensing challenge, it still cannot predict with 100% certainty whether or not an image is a gravitational lens. This is due to a variety of factors:

- Some images have lensing features which are too faint to be recognised by the model.
- Some images have been only very slightly affected by gravitational lensing, which could mean the model is not able to recognise that it has been gravitationally lensed.
- Some images are very noisy, so features are not as easily visible.

As expected, the AUROC increases with each epoch. It increases drastically for the first 10 or so epochs, then increases slowly. This is because with more training runs, there is less information for the model to gather about the characteristics of lenses.

We can change the proportion of true positives and true negatives by changing the threshold value t (see Section 1.1). If we increase the threshold value, there will be a lower proportion of false positives, but a higher proportion of false negatives. The opposite is true if we lower the threshold value.

Lenses were categorised with a much higher accuracy than non lenses with $t = 0.5$. If one prefers to find less false positives at the expense of finding less lenses, one can raise the threshold value, as shown in Figure 9.

5.1 Possible future improvements

This experiment could be improved by testing on all 100,000 images from the larger dataset to give a better prediction of how our AUROC compares to those found in the gravitational lens finding challenge. Additionally, a larger training set may result in a higher AUROC than our training set of 18,000 images. The model could also be altered to potentially give a higher AUROC.

6 Conclusions

In recent years, convolutional neural networks have dramatically improved. We can now use them to categorise data more precisely than with human inspection. There has been a noticeable increase in performance since the gravitational lens finding challenge in 2017, where the winning model achieved an AUROC of 0.93. The EfficientNetB0 model produces an AUROC of 0.96 ± 0.01. This is noticeably higher than that achieved in the lens finding challenge in 2017.

Since the original gravitational lens finding challenge in 2017, machine learning technology has seen a noticeable improvement. The EfficientNetB0 model produces an AUROC score of 0.96 ± 0.01, which is higher than the 0.93 score found by the LASTRO EPFL model which won the lens finding challenge. This is the result obtained from training the model on a different dataset, but the datasets are both of the same format; therefore, this AUROC is a reliable indicator of how the model would have performed if using the same dataset as the LASTRO EPFL model.

References

[1] Md Zahangir Alom, Tarek M. Taha, Chris Yakopcic, Stefan Westberg, Paheding Sidike, Mst Shamima Nasrin, Mahmudul Hasan, Brian C. Van Essen, Abdul A. S. Awwal, and Vijayan K. Asari. A state-of-the-art survey on deep learning theory and architectures. *Electronics*, 8(3), 2019. URL: https://www.mdpi.com/2079-9292/8/3/292, doi:10.3390/electronics8030292.

[2] Li Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012. doi:10.1109/MSP.2012.2211477.

[3] D.O. Hebb. *The Organization of Behavior: A Neuropsychological Theory*. Taylor & Francis, 1949. URL: https://books.google.co.uk/books?id=ddB4AgAAQBAJ.

[4] X. Huang, C. Storfer, A. Gu, V. Ravi, A. Pilon, W. Sheu, R. Venguswamy, S. Banka, A. Dey, M. Landriau, and et al. Discovering new strong gravitational lenses in the desi legacy imaging surveys. *The Astrophysical Journal*, 909(1):27, Mar 2021. URL: http://dx.doi.org/10.3847/1538-4357/abd62b, doi:10.3847/1538-4357/abd62b.

[5] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017. arXiv:1412.6980.

[6] S. Lawrence, C.L. Giles, Ah Chung Tsoi, and A.D. Back. Face recognition: a convolutional neural-network approach. *IEEE Transactions on Neural Networks*, 8(1):98–113, 1997. doi:10.1109/72.554195.

[7] R. B. Metcalf, M. Meneghetti, C. Avestruz, F. Bellagamba, C. R. Bom, E. Bertin, R. Cabanac, F. Courbin, A. Davies, E. Decencière, and et al. The strong gravitational lens finding challenge. *Astronomy Astrophysics*, 625:A119, May 2019. URL: http://dx.doi.org/10.1051/0004-6361/201832797, doi:10.1051/0004-6361/201832797.

[8] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks, 2015. arXiv:1511.08458.

[9] Kunal Pahwa and Neha Agarwal. Stock market analysis using supervised machine learning. In *2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)*, pages 197–200, 2019. doi:10.1109/COMITCon.2019.8862225.

[10] Sebastian Ruder. An overview of gradient descent optimization algorithms, 2017. arXiv:1609.04747.

[11] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. URL: http://jmlr.org/papers/v15/srivastava14a.html.

[12] Hidayet Takçı and Saliha Yeşilyurt. Diagnosing autism spectrum disorder using machine learning techniques. In *2021 6th International Conference on Computer Science and Engineering (UBMK)*, pages 276–280, 2021. doi:10.1109/UBMK52708.2021.9558975.

[13] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks, 2020. arXiv:1905.11946.