Image Stegnalysis through Deep Learning and LSB

Nitin Ranjan[1] and Sureshkumar N[2]

[1]18BCE0272, School of Computer Science and Engineering, Vellore Institute of Technology
[2]Associate Professor, SCOPE, VIT, Vellore

***Abstract*—Image Steganalysis is the detection of the message within the cover-image. It is often one of the most difficult processes in image processing because the information encoded in the image can often be fully retrieved only when the algorithm used to encode and embed it is known. This would mean that there is a need for a model that can understand the encryption or stenographic models. Most of these models rely upon deep learning alone. The author proposes the use of a RESNET-50 deep learning model with additional filters. The model proposed achieves an average accuracy of 75 percent on the datasets tested.**

***Abstract*—Keywords: Steganalysis, RESNET-50,, encryption, decryption,deep learning, epoch, hyperparameters.**

I. Introduction

Steganalysis is the study of detecting hidden messages using steganography. This is analogous to crypto-analysis and cryptography. Steganalysis can prevent the covert communication by analyzing the statistical distribution of the carrier, detecting and even extracting the secret messages embedded in the digital image. However, it is very difficult to accurately model the original carrier image using statistical methods, which will undoubtedly hinder the detection of secret information in the steganographic image. So, most practical applications rely on machine learning models. In 2015 Imagenet competition, Microsoft came up with a deep learning model of 8 layers as a solution for stegnalysis. Ever since then, deep learning has been the preferred method for carrying out stegnalysis.

II. Related Work

In [1]The authors propose a mechanism to use unpredictable distortion matrices for data hiding. The cover images are then grouped into subsequences, each coded with a static payload. Finally, a spatial-rich-model stegnalyser is used with an ensemble classifier and the process is carried out over a very large sample set to minimize loss. In [2] however, image foresnics are used to separate processed and investigated images before stegnalysis exploiting the fragility of image manipulation detection to reduce the false alarm rate by fragile detection of gamma transformation and LSB matching. In [3], the authors propose the removal of MSB planes as statistics suggest that embedding is usually not carried out in these planes. Further, since the image actually shrinks after data embedding, the authors propose the use of diagonal elements of the co-occurrence matrix to reduce the effect of asymmetry caused in the same due to data embedding. In [4]image net model is used to carry out stagnalysis transfer learning constructed using a double pipeline architecture – for cross-entropy and the optimization through a 10-weight decay. The hyper-parameters for each of the sample was calculated and the model was executed on the ALSAKA-II dataset. In [5], a universal stenalysis engine is proposed - a system that holds all the known steganlysis models and then based on the cover image analysis, a suitable algorithm,a suitable classification model for feature extraction and thus a new feature set classifier is modelled and trained. The authors in [6] propose preprocessing to be carried out in CNN based image processing because under reduction, the frequency for certain values becomes too high. This involves the convolution of reduced stegno images only through a trained model instead of convoluting both reduced stegno images and reduced cover images. This helps in selective increase in pixel value frequencies. In [7],proposal is for the utilization of deep neural networks to train a model that can distinguish between stegno images and normal images, generating two symmetrical subnets similar to [4] in three phases. This creates an adaptive stegnalysis model that extracts a hyper parameter through training and classification. In [8], through the use of Deep learning with ensemble learning, the authora proposes that a contournet transformation be used based on a statistical analysis of the pixels. The ROC model is utilized and accuracy is computed using a training model. The authors finally conclude that while on grayscale images, such a model can yield high accuracy, it is not exactly the case with colourful images. [9] presents a novel framework with three parts. The first has four parts – shallow feature extraction, un-looped feature extraction, deep feature reduction and classification. The second and third involve residual learning models. The framework reduces memory dependency and processor usage. The results yield a new baseline architecture that can be used to detect a new hyper-parameter. However detection for a new hyper-parameter increases the computational cost of the module. The authors conclude that stegnalysis models do not very accurately determine encrypted images and rather use heuristics to detect noise in images. In [10], suggestion is made over several methodologies and tricks to carry out both pre-processing and stegnalysis, including the creation of a model that directly inputs the DCT coefficients, further aided by a multi-layer model, pseudo labelling of features, stacking those features with a tree – all this built upon a CNN based model. [11] proposes a model which involves an ensemble of several filters and tools – a binary relevance multi-label filter followed by a Logistical Regression classifier and finally a margin FML - to ensure that the model uses the right set of images for classification and stegnalysis, reducing false positives. In [12], the authors propose steganalysis based on LSB matching by using Ye network (YeNet). The YeNet produces a 2-D vector output which are the dimensions of the features extracted. A linear model is compared to the dimensions thus obtained and the origin is treated as a critical point. This is used to classify the image as a stego or a cover image. The experiments suggest that in YeNet, the output of stego image changes regularly with the embedding rate, the mapping relationship between embedding rate and feature distribution can be fitted, so as to estimate the embedding rate of a given stego. [13] states that during stegnalysis, the system should use diagonal elements of the matrix just as suggested in [3] to eliminate noise. It also suggests the use of Gaussian elimination equations. Then a probability model is used to distinguish between cover images and stegno images. The authors conclude that while there is marginal noise in the restored image as well, however, in smaller sections of the image, if there is no correlation between adjacent pixels, a probability function is again employed to detect an absorber edge and embedded data. [14] points out that noises can be eliminated by splitting exposure time and then averaging the images under several captures making it useful for images under motion, which are seemingly blur. This is restored by splitting the motion under several captures. Retrieve a normalized correlation coefficient. Finally, an average from all the normalized pixels is obtained. This reduces blurring as well as noise. [15] is similar to [14] where, a novel methodology that is useful for reduction in time complexity and improvement in error rate is proposed. This is achieved through introducing Legrange multipliers over a blurred image modelled like an object in linear motion. This transforms the image into an equation of matrices and finally a restoration algorithm is applied. This restoration algorithm is also based on legrange multiplier. [16] makes a comparison between inverse filtering and Wiener filtering on blurred images. The authors conclude that both the filters work perfectly under conditions where images are blurred and have minimal, marginal or no noise. However, in the presence of noise, a new methodology is needed. In [17], authors propose several steps of filtering – beginning with median filtering, wiener filtering, regularization filtering, finally followed by Lucy-Richardson filtering. This is then measured using several metrics like the PSNR, MSE and SSIM values.

III. Dataset

1) I have used ALASKA-II dataset because it is has a set of unique elements, has yielded very poor rates of false positives. It is also a fresh dataset released in 2020.
2) I have also used the naïve image dataset present to retest the results obtained on the first dataset.

IV. Proposed Work

We have used the transfer learning mechanism proposed in [4] over a ResNet-50 deep learning network as proposed in my base research article [18].
Repeat the same without transfer learning.
The model shall be facilitated with a LSB filter as proposed in [3] and [12].

A. Aim of Experiment

The aim behind different models were different. As discussed, in the model with Transfer Learning, the dataset should quickly adapt to the base stenographic functions used. While, without Transfer Learning, the purpose was to make the algorithm perfect to identify a family of steganography algorithms and not to all steganography functions.

B. Software and Harware Requirements

1) Python 3.9.2
2) TPU Accelarator
3) ALASKA2 Dataset

I used the Kaggle Jupyter Notebook Platform to achieve these.

C. Supposed Challenges

Increasing network depth does not work by simply stacking layers together. Deep networks are hard to train because of the vanishing gradient problem — as the gradient is back-propagated to earlier layers, repeated multiplication may make the gradient extremely small. As a result, as the network goes deeper, its performance gets saturated or even starts degrading rapidly.

V. Experimental Results

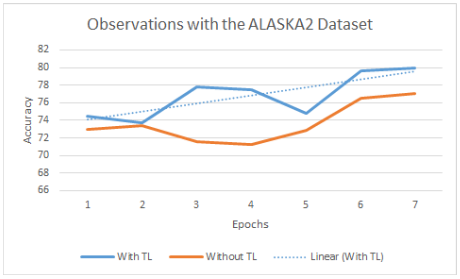


Fig. 1. fig 1: Observations recorded on Alaska2 Data set.

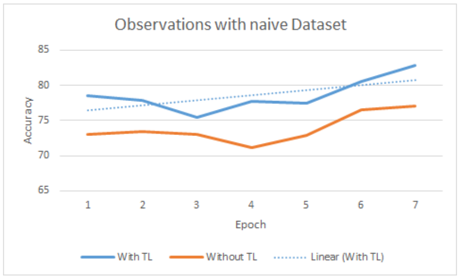


Fig. 2. Observations recorded on Naive Data set. The accuracy is fairly less in the model without Transfer Learning and does not vary a lot when the epochs are varied.

VI. Observation

The following results can be arrived upon based on the experimental observations recorded with both the data-sets:

1) 1As already demonstrated in Experimental Observations, the accuracy and loss in both Transfer Learning based model was 77percent accuracy compared to 58 percent loss.
2) The transfer learning based model that uses two epochs has a loss of 63 percent as compared to the 58 percent loss in the same model but with a single epoch of training. This might be because Deep Learning models approach a saturation when the same model undergoes various levels of training. This was one of the reasons why I opted for RESNET-50 over RESNET-150 or higher. The Transfer Learning paradigm only increases the effective number of layers in the Deep Learning Model and is slower because a more comprehensive memory of the layers has to be maintained by the algorithm.
3) Without Transfer Learning, the accuracy was around 74 percent for RESNET – 50 model with ALASKA2 dataset and around 75 percent for Naïve Dataset.
4) In the Naive dataset that I used once the model was successfully trained with ALASKA2 dataset, the accuracy achieved was about 78 percent compared to a loss statistics of around 58 percent during model training. This is when the number of epochs is 1.
5) The Naive Dataset training was faster and slightly better results were obtained as compared to ALASKA2 dataset, probably because the Naive Dataset is DCT encoded and LSB is a simple and yet effective way to decode the same.
6) The conclusions are based on the two datasets that I used to train and test the model. This might however be specific to the datasets I picked up because the accuracy largely depends upon the number of steganography algorithms used in the data studied under the model as well. This is because the Transfer learning model will remember basic algorithms when trained while the non-transfer learning model will learn some specific families of algorithms better.

VII. Conclusion

So, I hereby conclude that this algorithm which includes a simple LSB Filter put before a RESNET-50 model trained with Transfer Learning Paradigm has an average accuracy score of 75 percent with the average training losses amounting to 60 percent. And the transfer learning model clearly outweighs the model without transfer learning as in most cases, the user will not run over a set number of epochs in any deep learning model.

VIII. Suggestions and further improvements

Using the Xception paradigm(Extreme Inception) model that was introduced yet again for the ImageNet 2015 dataset might help to further improve the RESNET+LSB model accuracy. The Xception model outperformed the then best model (before the ImageNet 2015 competition where Microsoft proposed the RESNET model) by eliminating the need to learn 3D mapping and replacing it with learning a 2D+1D map. The model might also be improved further with accuracy of about 85 percent achieved if the CNN model being used in the RESNET-50 is replaced with a DenseNet Model. Densenet is a Recurrent CNN model with added memory. However, I did not suggest it for two reasons – it will make the model very complicated.

1) The Densenet model itself behaves as a small deep learning paradigm. In this case, several Densenets might have the same effect on the model as increasing the number of deep learning layers or transfer learning layer has – increased saturation in training and increase loss.
2) Additionally, the RESNET model is experimentally known to be the strongest Stegnalysis mode. However, the researcher is open to the idea of combining these two models for the sake of experimental verification of point (2).

References

[1] Jyoti Neginal , and Dr Ruskar Fatima, ”An adaptive stenographic technique, ascertaining upper bounds of embedding” *IEEE,2019*

[2] Ping Wang, and others, ”Steganalysis aided by fragile detection of image manipulations” *IEEE,2019*

[3] M. Abolghasemi, and others, ”Steganalysis of LSB Matching Based on Co-Occurrence Matrix and Removing Most Significant Bit Planes”*IEEE, 2008*

[4] Yassie Yousfie, and others , ”ImageNet Pre-trained CNNs for JPEG Steganalysis” *IEEE, 2020*

[5] Yanping Tan, and others, ” New Design Method about the Universal Steganalysis Classifier in Digital Image” *IEEE, 2020*

[6] Hiroya Kato, and others, ”A Preprocessing Methodology by Using Additional Steganography on CNN-based Steganalysis” *IEEE, 2020*

[7] Wieke You, and others, ” Siamese CNN for Image Steganalysis” *IEEE, 2021*

[8] Nour Mohamed, and others, ”A Review of Color Image Stegnalysis in the Tranform Domain” *IEEE, 2020*

[9] Wonhyuk Ahn, and others, ”Local-Source Enhanced Residual Network for Steganalysis of Digital Images” *IEEE, 2020*

[10] Kaizaburo Chubachi, ”An Ensemble Model using CNNs on Different Domains for ALASKA2 Image Steganalysis” *IEEE, 2020*

[11] Tayebe Abazar, and others, ”A binary relevance adaptive model-selection for ensemble stegnalysis” *17th International ISC Conference on Information Security and Cryptology (ISCISC) 2020*

[12] Yu Sun, and Taiyun Li, ”A Method for Quantitative Steganalysis Based on Deep Learning” *IEEE, 2019*

[13] Varlen Grabski, ”Digital Image Restoration Based on Pixel Simultaneous Detection Probabilities” *IEEE, 2009*

[14] Zile Wei, ”Digital Image Restoration by Exposure-Splitting and Registration” *IEEE 2004*

[15] Igor Stojanovic, ”Application of Non-Iterative Method in Digital Image Restoration” *IEEE*

[16] Mohammed Mahmudur Rahman Khan, and others, ”Digital Image Restoration in Matlab: A Case Study on Inverse and Wiener Filtering” *IEEE 2018*

[17] Reecturaj Mishra, and others, ”Digital Image Restoration using Image Filtering Techniques” *IEEE 2019*

[18] Kiaming Hu, and others, ”Deep Residual Learning for Image Recognition” *arXiv:1512.03385v1 [cs.CV] 10 Dec 2015*

CSE4019: Image Processing

Studying and proposing solution for Steganalysis through a deep learning model and LSB

PROJECT REPORT

Nitin Pramod Ranjan

18BCE0272

Under the guidance of

Prof Sureshkumar N

Acknowledgment

The author expresses his gratitude to the course faculty for CSE3009, Prof Sureshkumar N for permitting and guiding this the work behind this report. The author also expresses gratitude towards VIT, Vellore for including this course in the curriculum and allowing me a chance to understand the subject better.

Nitin Ranjan
18BCE0272

The presentation for review 2:
https://drive.google.com/file/d/1s17XQHXy4uQSxADavNVSUnAiauWQiOZ4/view?usp=sharing
The presentation for review 3:
https://drive.google.com/file/d/1ucYSeuvkgzpH3Q3hp10ZWwJdmIfbuCWc/view?usp=sharing

Index

Abstract:

Image Steganalysis is the detection of the message within the cover-image. It is often one of the most difficult processes in image processing because the information encoded in the image can often be fully retrieved only when the algorithm used to encode and embed it is known. This would mean that there is a need for a model that can understand the encryption or stenographic models. Most of these models rely upon deep learning alone. The author proposes the use of a RESNET-50 deep learning model with additional filters. The model proposed achieves an average accuracy of 75% on the datasets tested.
Keywords: *Steganalysis, RESNET-50,, encryption, decryption,deep learning, epoch, hyperparameters.*

Introduction

Steganalysis is the study of detecting hidden messages using steganography. This is analogous to crypto-analysis and cryptography.
Steganalysis can prevent the covert communication by analyzing the statistical distribution of the carrier, detecting and even extracting the secret messages embedded in the digital image. However, it is very difficult to accurately model the original carrier image using statistical methods, which will undoubtedly hinder the detection of secret information in the steganographic image. So, most practical applications rely on machine learning models. In 2015 Imagenet competition, Microsoft came up with a deep learning model of 8 layers as a solution for stegnalysis. Ever since then, deep learning has been the preferred method for carrying out stegnalysis.

Related Work:

[1] The authors propose a mechanism to use unpredictable distortion matrices for data hiding. The cover images are then grouped into subsequences, each coded with a static payload. Finally, a spatial-rich-model stegnalyser is used with an ensemble classifier and the process is carried out over a very large sample set to minimize loss.
[2] The authors have proposed the usage of image foresnics to separate processed and investigated images before stegnalysis. It tries to exploit the fragility of image manipulation detection to reduce the false alarm rate. This is achieved through fragile detection of gamma transformation and LSB matching.
[3] The authors propose the removal of MSB planes as statistics suggest that embedding is usually not carried out in these planes. Further, since the image actually shrinks after data embedding, the authors propose the use of diagonal elements of the co-occurrence matrix to reduce the effect of asymmetry caused in the same due to data embedding.
[4] An image net model to carry out stagnalysis transfer learning was constructed using a double pipeline architecture – the first to use cross-entropy and the second to optimize it through a 10-weight decay. The hyper-parameters for each of the sample was calculated and the model was executed on the ALSAKA-II dataset. The authors

conclude that their experiment yielded good results, however, could yield better if the hyper-parameters were more clearly evaluated.

[5] The authors propose a universal stenalysis engine. Basically a system that holds all the known steganlysis models and then based on the cover image analysis, a suitable algorithm is chosen. Further, a suitable classification model is chosen for feature extraction. Thus, a new feature set classifier is modelled and trained. Finally the results are yielded.

[6] The authors propose a preprocessing to be carried out while carrying out CNN based image processing. This is mostly because under reduction, the frequency for certain values becomes too high. This involves the convolution of reduced stegno images only through a trained model instead of convoluting both reduced stegno images and reduced cover images. This helps in selective increase in pixel value frequencies.

[7] The authors propose the utilization of deep neural networks to train a model that can distinguish between stegno images and normal images. They then generated two symmetrical subnets similar to [4] in three phases. This creates an adaptive stegnalysis model that extracts a hyper parameter through training and classification and then yield a stegnalytical result.

[8] The authors propose the use of Deep learning with ensemble learning. The author proposes that a contournet transformation be used based on a statistical analysis of the pixels. The ROC model is utilized and accuracy is computed using a training model. The authors finally conclude that while on grayscale images, such a model can yield high accuracy, it is not exactly the case with colourful images.

[9] A novel framework is proposed. The framework has three parts. The first has four parts – shallow feature extraction, un-looped feature extraction, deep feature reduction and classification. The second and third involve residual learning models. The framework reduces memory dependency and processor usage. The results yield a new baseline architecture that can be used to detect a new hyper-parameter. However detection for a new hyper-parameter increases the computational cost of the module. The authors conclude that stegnalysis models do not very accurately determine encrypted images and rather use heuristics to detect noise in images.

[10] The authors suggest several methodologies and tricks to carry out both pre-processing and stegnalysis. It includes a the creation of a model that directly inputs the DCT coefficients, further aided by a multi-layer model, pseudo labelling of features, stacking those features with a tree – all this built upon a CNN based model.

[11] The authors propose a model which involves an ensemble of several filters and tools – a binary relevance multi-label filter followed by a Logistical Regression classifier and finally a margin FML. This is to ensure that the model uses the right set of images for classification and stegnalysis and reduce false positives.

[12] The authors propose steganalysis based on LSB matching by using Ye network (YeNet). The YeNet produces a 2-D vector output which are the dimensions of the features extracted. A linear model is compared to the dimensions thus obtained and the origin is treated as a critical point. This is used to classify the image as a stego or a cover image. The experiments suggest that in YeNet, the output of stego image changes regularly with the embedding rate, the mapping relationship between embedding rate and feature distribution can be fitted, so as to estimate the embedding rate of a given stego.

[13] The authors state that during stegnalysis, the system should use diagonal elements of the matrix just as suggested in [3]. This can be used to eliminate noise. The author has suggested the use of Gaussian elimination equations. Then a probability model is used to distinguish between cover images and stegno images. The authors conclude that while there is marginal noise in the restored image as well, however, in smaller sections of the image, if there is no correlation between adjacent pixels, a probability function is again employed to detect an absorber edge and embedded data.

[14] The author points out that noises can be eliminated by splitting exposure time and then averaging the images under several captures. This is useful for images under motion, which are seemingly blur. This is restored by splitting the motion under several captures. Retrieve a normalized correlation coefficient. Finally, an average from all the normalized pixels is obtained. This reduces blurring as well as noise.

[15] The paper is similar to [14]. The author proposes a novel methodology that is useful for reduction in time complexity and improvement in error rate. This is achieved through introducing Legrange multipliers over a blurred image modelled like an object in linear motion. This transforms the image into an equation of matrices and finally a restoration algorithm is applied. This restoration algorithm is also based on legrange multiplier.

[16] The author makes a comparison between inverse filtering and Wiener filtering on blurred images. The authors conclude that both the filters work perfectly under conditions where images are blurred and have minimal, marginal or no noise. However, in the presence of noise, a new methodology is needed.

[17] The author proposes several steps of filtering – beginning with median filtering, wiener filtering, regularization filtering, finally followed by Lucy-Richardson filtering. This is then measured using several metrics like the PSNR, MSE and SSIM values.

Dataset

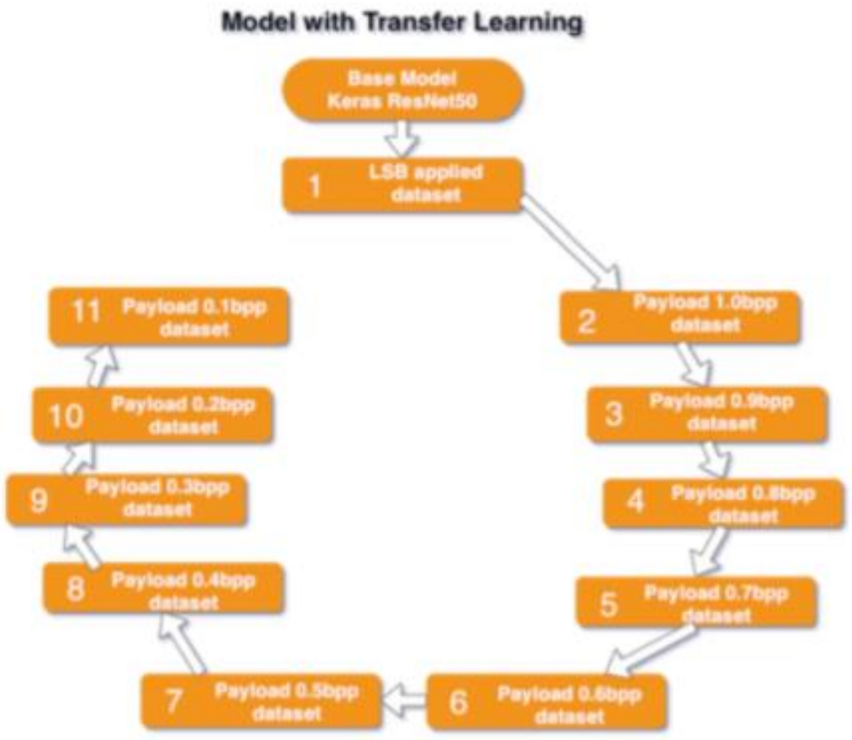
1. I have used ALASKA-II dataset because it is has a set of unique elements, has yielded very poor rates of false positives. It is also a fresh dataset released in 2020.
2. I have also used the naïve image dataset present to retest the results obtained on the first dataset.

The link to both the datasets have been provided in Appendix [A].

Proposed Work

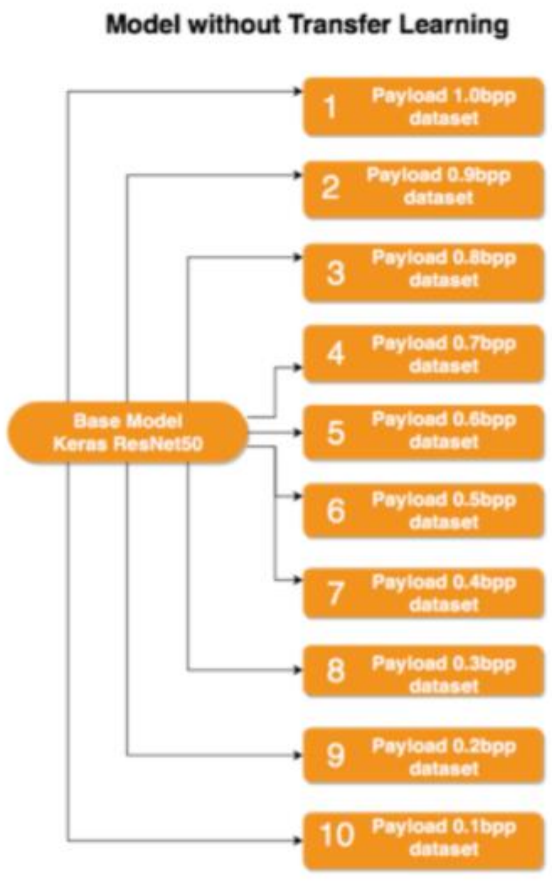
- We have used the transfer learning mechanism proposed in [4] over a ResNet-50 deep learning network as proposed in my base research article [18].
- Repeat the same without transfer learning.
- The model shall be facilitated with a LSB filter as proposed in [3] and [12].

With Transfer Learning



- On the first test branch, transfer learning method has been applied to model on different payload rates.
- First step was that the model has been trained on the dataset that applied with Least Significant Bit (LSB) steganography algorithm, the very base and easy to detect one.
- The steganography algorithms hide information using the LSB technique but with more difficulty to detect. Hence, the model should learn how the LSB works.
- The dataset should quickly adapt to the base stenographic functions used
- After LSB training step, the same model was transferred to the steganography dataset for training.

Without Transfer Learning



- A normal pre-trained model (without skip connection) would have to be trained separately on separate payload data.
- The model should achieve more successful predictions and less error rates on the dataset hardest to detect which is 0.1bpp payload rated.
- On every step from LSB and 1.0bpp dataset to 0.1bpp dataset, model would learn better how does the steganography algorithm operate and hide messages in the cover images.
- The model should start gaining the ability to distinguish even the smallest bit changes out of chain learning steps.
- In the end, the model would be able to detect all the stego images applied in every different embedding payload values using the same steganography algorithm.

Terms

CNN

Convolutional networks (CNNs) are the new and most used element of computer vision solutions. They have been standard benchmarking algorithms for a decade now. In most steganalysis models, CNN is used to reduce computational costs and a lower requirement for number of parameters in large models.

ResNet

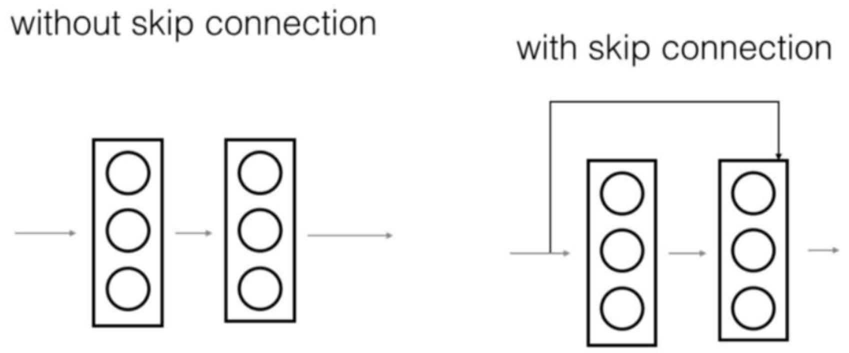
ResNet is short for **Residual Networks. It** is a classic neural network that is used as a backbone for many computer vision tasks. The most important observation about ResNet is that it allows an easy training of neural networks.
AlexNet, the winner of ImageNet 2012, was the model that apparently kick started the focus on deep learning, and it had only 8 convolutional layers, the VGG network had 19 and Inception or GoogleNet had 22 layers and ResNet 152 had 152 layers. ResNet-50 is basically a smaller version of ResNet 152 and frequently used as a starting point for transfer learning.

It is based on the original paper [18].

Transfer Learning

Transfer learning is a machine learning method where a model developed for a task is reused as the starting point for a model on a second task. In this project, the author proposes two models for using ResNet – 50. One uses transfer learning, the other does not.

Skip Connection



The figure on the left is stacking convolution layers together one after the other. On the right we still stack convolution layers as before but we now also add the original input to the output of the convolution block. This is called skip connection. In these architectures, skip connections are used to pass information from the downsampling layers to the upsampling layers.

Aim of the Experiment

The aim behind different models were different. As discussed, in the model with Transfer Learning, the dataset should quickly adapt to the base stenographic functions used.
While, without Transfer Learning, the purpose was to make the algorithm perfect to identify a family of steganography algorithms and not to all steganography functions.

Supposed Challenges

Increasing network depth does not work by simply stacking layers together. Deep networks are hard to train because of the vanishing gradient problem — as the gradient is back-propagated to earlier layers, repeated multiplication may make the gradient extremely small. As a result, as the network goes deeper, its performance gets saturated or even starts degrading rapidly.

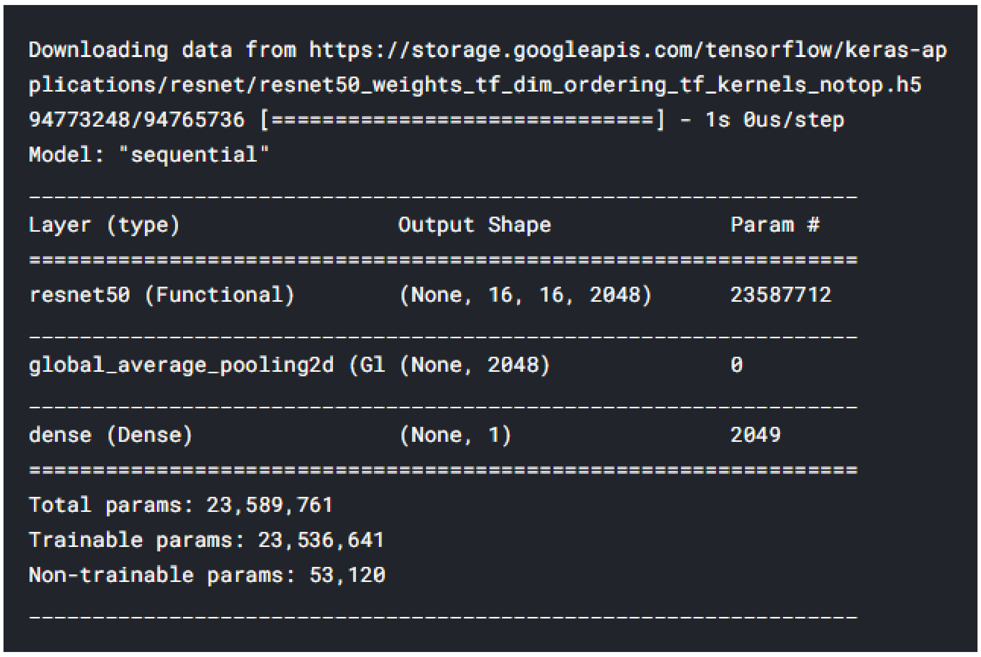
Software/Hardware Requirements -

1. Python 3.9.2
2. TPU Accelerator
3. Alaska 2 Database

Platform Used: Kaggle Jupyter Notebook

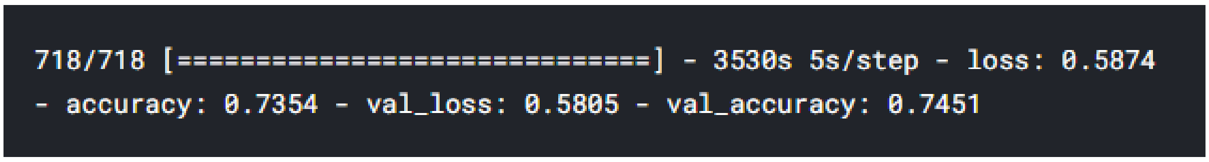
Experimental Observations on ALASKA2 Dataset With Transfer Learning

Single Epoch



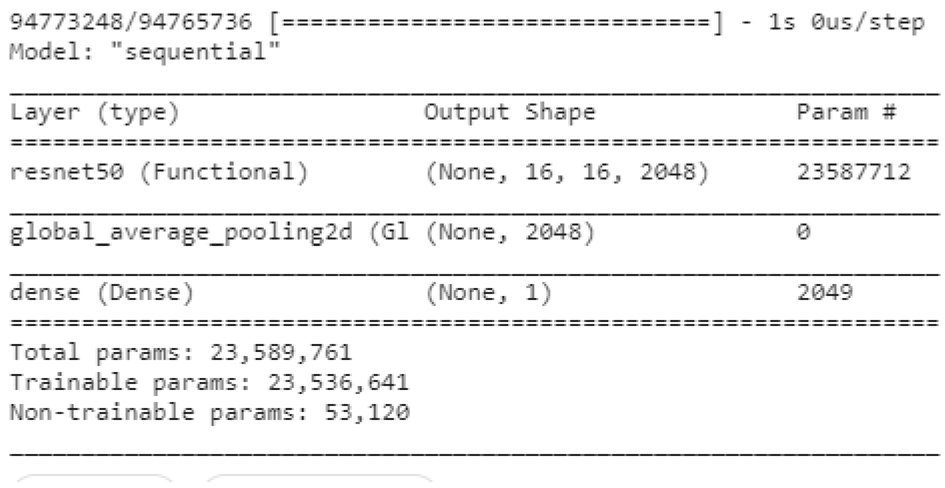
```
Downloading data from https://storage.googleapis.com/tensorflow/keras-ap
plications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
94773248/94765736 [==============================] - 1s 0us/step
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
resnet50 (Functional)        (None, 16, 16, 2048)      23587712
_________________________________________________________________
global_average_pooling2d (Gl (None, 2048)              0
_________________________________________________________________
dense (Dense)                (None, 1)                 2049
=================================================================
Total params: 23,589,761
Trainable params: 23,536,641
Non-trainable params: 53,120
_________________________________________________________________
```

Model Summary Yielded after training:



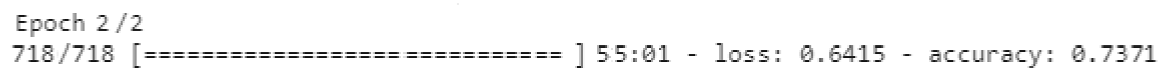
```
718/718 [==============================] - 3530s 5s/step - loss: 0.5874
- accuracy: 0.7354 - val_loss: 0.5805 - val_accuracy: 0.7451
```

Two Epochs



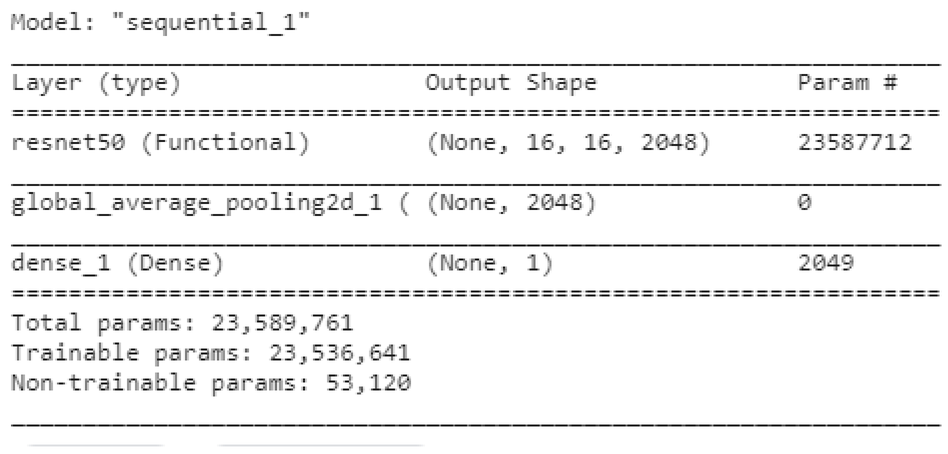
```
94773248/94765736 [==============================] - 1s 0us/step
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
resnet50 (Functional)        (None, 16, 16, 2048)      23587712
_________________________________________________________________
global_average_pooling2d (Gl (None, 2048)              0
_________________________________________________________________
dense (Dense)                (None, 1)                 2049
=================================================================
Total params: 23,589,761
Trainable params: 23,536,641
Non-trainable params: 53,120
_________________________________________________________________
```

Model Summary Yielded after Training:



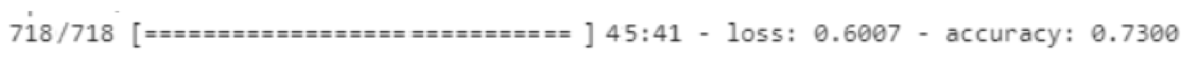
```
Epoch 2/2
718/718 [============================] 55:01 - loss: 0.6415 - accuracy: 0.7371
```

Without Transfer Learning



```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
resnet50 (Functional)        (None, 16, 16, 2048)      23587712
_________________________________________________________________
global_average_pooling2d_1 ( (None, 2048)              0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 2049
=================================================================
Total params: 23,589,761
Trainable params: 23,536,641
Non-trainable params: 53,120
_________________________________________________________________
```

Model Summary Yielded:



```
718/718 [============================] 45:41 - loss: 0.6007 - accuracy: 0.7300
```

Observations on naïve Dataset

The Naive Dataset was again tested under the same code.
It has images that have information encoded in them using DCT (Discrete Cosine Transformations).
This was done to ascertain the observations recorded with ALASKA2 Dataset.
The experimental results were fairly similar with an average accuracy of 78% with a single epoch.

RESULTS AND CONCLUSION

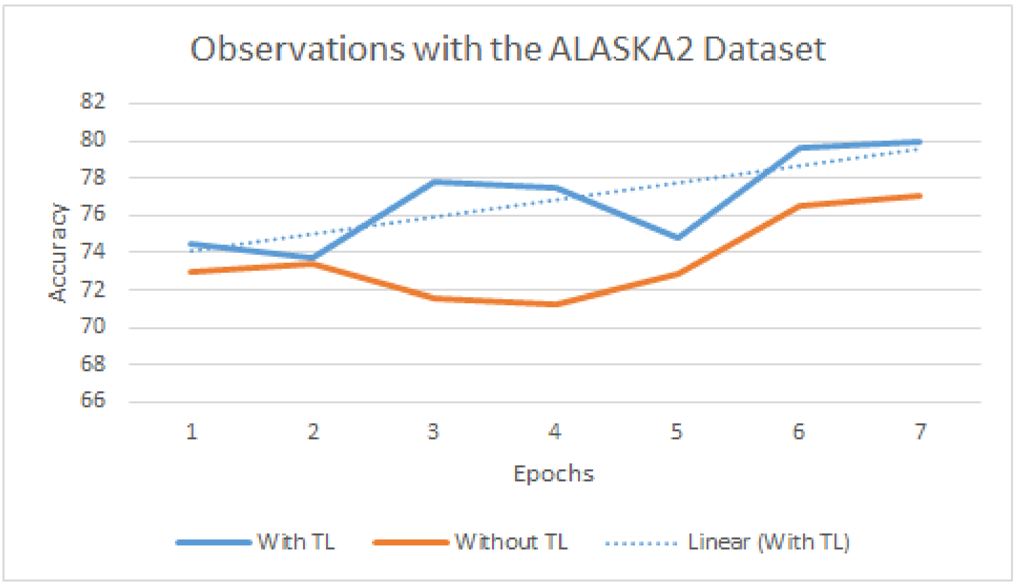


Figure Res(1) (a): Observations recorded on Alaska2 Dataset.

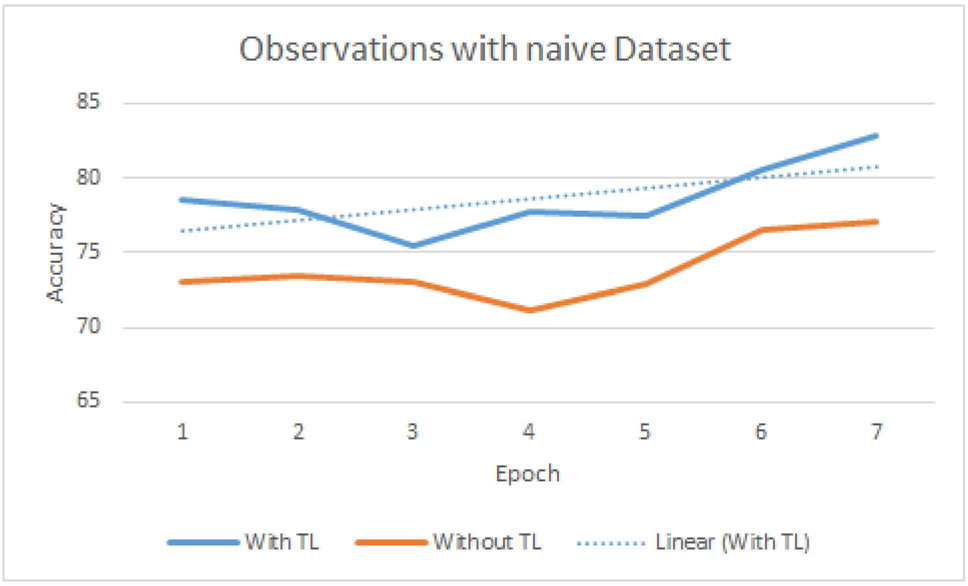


Figure Res(1)(b): Observations recorded on the naïve dataset. The accuracy is fairly less in the model without Transfer Learning and does not vary a lot when the epochs are varied.

The following results can be arrived upon based on the experimental observations recorded with both the datasets:

1. As already demonstrated in Experimental Observations, the accuracy and loss in both Transfer Learning based model was ~77% accuracy compared to ~58% loss for ALASKA2 dataset.
2. The transfer learning based model that uses two epochs has a loss of ~63% as compared to the 58% loss in the same model but with a single epoch of training. As already discussed, this is because Deep Learning models approach a saturation when the same model undergoes various levels of training. This was one of the reasons why I opted for RESNET-50 over RESNET-150 or higher. The Transfer Learning paradigm only increases the effective number of layers in the Deep Learning Model and is slower because a more comprehensive memory of the layers has to be maintained by the algorithm.
3. Without Transfer Learning, the accuracy was around 74% for RESNET – 50 model with ALASKA2 dataset and around 74% for Naïve Dataset.
4. In the Naive dataset that I used once the model was successfully trained with ALASKA2 dataset, the accuracy achieved was about 78% compared to a loss statistics of around 58% during model training. This is when the number of epochs is 1.
5. The Naive Dataset training was faster and slightly better results were obtained as compared to ALASKA2 dataset, probably because the Naive Dataset is DCT encoded and LSB is a simple and yet effective way to decode the same.

6. The conclusions are based on the two datasets that I used to train and test the model. This might however be specific to the datasets I picked up because the accuracy largely depends upon the number of steganography algorithms used in the data studied under the model as well. This is because the Transfer learning model will remember basic algorithms when trained while the non-transfer learning model will learn some specific families of algorithms better.

So, I hereby conclude that this algorithm which includes a simple LSB Filter put before a RESNET-50 model trained with Transfer Learning Paradigm has an average accuracy score of ~77% with the average training losses amounting to ~60%. And the transfer learning model clearly outweighs the model without transfer learning as in most cases, the user will not run over a set number of epochs in any deep learning model.

IMPROVEMENTS AND FURTHER SUGGESTIONS

Using the Xception paradigm(Extreme Inception) model that was introduced yet again for the ImageNet 2015 dataset might help to further improve the RESNET+LSB model accuracy. The Xception model outperformed the then best model (before the ImageNet 2015 competition where Microsoft proposed the RESNET model) by eliminating the need to learn 3D mapping and replacing it with learning a 2D+1D map.

The model might also be improved further with accuracy of about 85% achieved if the CNN model being used in the RESNET-50 is replaced with a DenseNet Model. Densenet is a Recurrent CNN model with added memory. However, I did not suggest it for two reasons –

1. It will make the model very complicated.
2. The Densenet model itself behaves as a small deep learning paradigm. In this case, several Densenets might have the same effect on the model as increasing the number of deep learning layers or transfer learning layer has – increased saturation in training and increase loss.
3. Additionally, the RESNET model is experimentally known to be the strongest Stegnalysis mode. However, the researcher is open to the idea of combining these two models for the sake of experimental verification of point (2).

APPENDIX – A

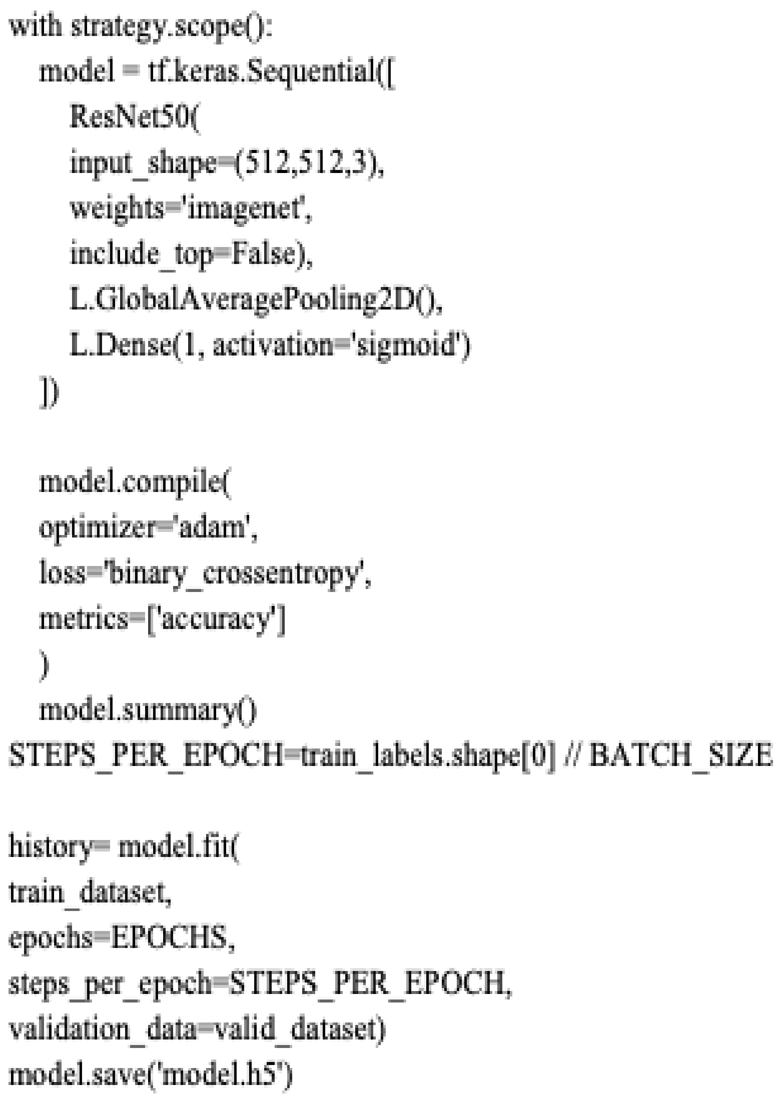
ALASKA 2 DATA-SET CAN BE ACCESSED AT :
https://www.kaggle.com/c/alaska2-image-steganalysis
THE NAIVE APPROACH DATASET CAN BE ACCESSED AT:
https://www.kaggle.com/tanulsingh077/steganalysis-complete-understanding-and-model/data

APPENDIX - B

Code for model with Transfer Learning

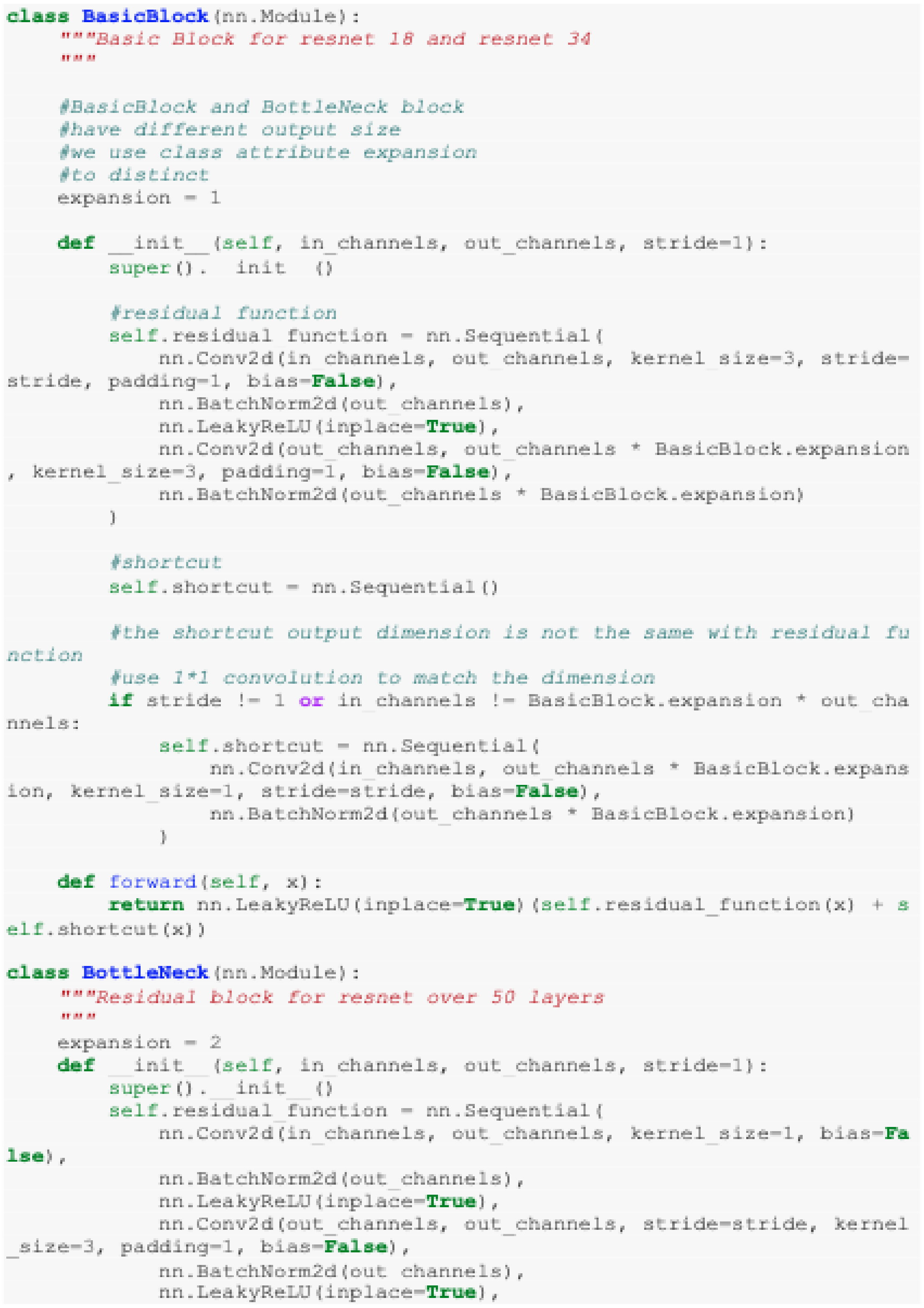
```
import math, re, os

import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from kaggle_datasets import KaggleDatasets
import tensorflow as tf
import tensorflow.keras.layers as L
from sklearn import metrics
from sklearn.model_selection import train_test_split

try:
   # TPU detection. No parameters necessary if TPU_NAME environment variable is
   # set: this is always the case on Kaggle.
   tpu = tf.distribute.cluster_resolver.TPUClusterResolver()
   print('Running on TPU ', tpu.master())
except ValueError:
   tpu = None

if tpu:
   tf.config.experimental_connect_to_cluster(tpu)
   tf.tpu.experimental.initialize_tpu_system(tpu)
   strategy = tf.distribute.experimental.TPUStrategy(tpu)
else:
   # Default distribution strategy in Tensorflow. Works on CPU and single GPU.
   strategy = tf.distribute.get_strategy()

print("REPLICAS: ", strategy.num_replicas_in_sync)
# For tf.dataset
AUTO = tf.data.experimental.AUTOTUNE

# Data access
GCS_DS_PATH = KaggleDatasets().get_gcs_path()

# Configuration
EPOCHS = 2
```

```
BATCH_SIZE = 16 * strategy.num_replicas_in_sync
def append_path(pre):
   return np.vectorize(lambda file: os.path.join(GCS_DS_PATH,pre,file))
sub = pd.read_csv('/kaggle/input/alaska2-image-steganalysis/sample_submission.csv')
train_filenames=np.array(os.listdir('/kaggle/input/alaska2-image-
steganalysis/Cover/'))
np.random.seed(0)
positives = train_filenames.copy()
negatives = train_filenames.copy()

np.random.shuffle(positives)
np.random.shuffle(negatives)

jmipod=append_path('JMiPOD')(positives[:10000])
juniward=append_path('JUNIWARD')(positives[10000:])
uerd=append_path('UERD')(positives[20000:30000])

pos_paths=np.concatenate([jmipod, juniward,uerd])
test_paths=append_path('Test')(sub.Id.values)
neg_paths =append_path('Cover')(negatives[:30000])
train_paths=np.concatenate([pos_paths,neg_paths])
train_labels= np.array([1] * len(pos_paths)+[0] * len(neg_paths))
train_paths, valid_paths, train_labels,
valid_labels=train_test_split(train_paths,train_labels,test_size=0.2,random_state=42)
def decode_image(filename, label=None, image_size=(512,512)):
   bits=tf.io.read_file(filename)
   image=tf.image.decode_jpeg(bits,channels=3)
   image=tf.cast(image,tf.float32)#image to tf.float32 data type
   image=tf.image.resize(image,image_size)

   if label is None:
      return image
   else:
      return image,label

def data_augment(image,label=None):
   image=tf.image.random_flip_left_right(image)
   image=tf.image.random_flip_up_down(image)

   if label is None:
        return image
    else:
        return image,label
train_dataset = (tf.data.Dataset
                .from_tensor_slices((train_paths,train_labels))
                .map(decode_image, num_parallel_calls=AUTO)
                .cache()
                .repeat()
                .shuffle(1024)
                .batch(BATCH_SIZE)
                .prefetch(AUTO)
                )
valid_dataset= (tf.data.Dataset
               .from_tensor_slices((valid_paths,valid_labels))
               .map(decode_image, num_parallel_calls=AUTO)
               .batch(BATCH_SIZE)
               .prefetch(AUTO)
              )
test_dataset= (tf.data.Dataset
              .from_tensor_slices(test_paths)
              .map(decode_image, num_parallel_calls=AUTO)
              .batch(BATCH_SIZE)
             )
def build_lrfn(lr_start=0.0001,lr_max=0.000075,
             lr_min=0.000001, lr_rampup_epochs=20,
             lr_sustain_epochs=0,lr_exp_decay=0.8):
    lr_max = lr_max * strategy.num_replicas_in_sync

    def lrfn(epoch):
        if epoch < lr_rampup_epochs:
            lr = (lr_max- lr_start)/lr_rampup * epoch + lr_start
        elif epoch < lr_rampup_epochs + lr_sustain_epochs:
            lr = lr_max
        else:
            lr = (lr_max - lr_min) * lr_exp_decay ** (epoch - lr_rampup_epochs -
lr_sustain_epochs) + lr_min
            return lr

        return lrfn
from tensorflow.keras.applications.resnet50 import ResNet50
```

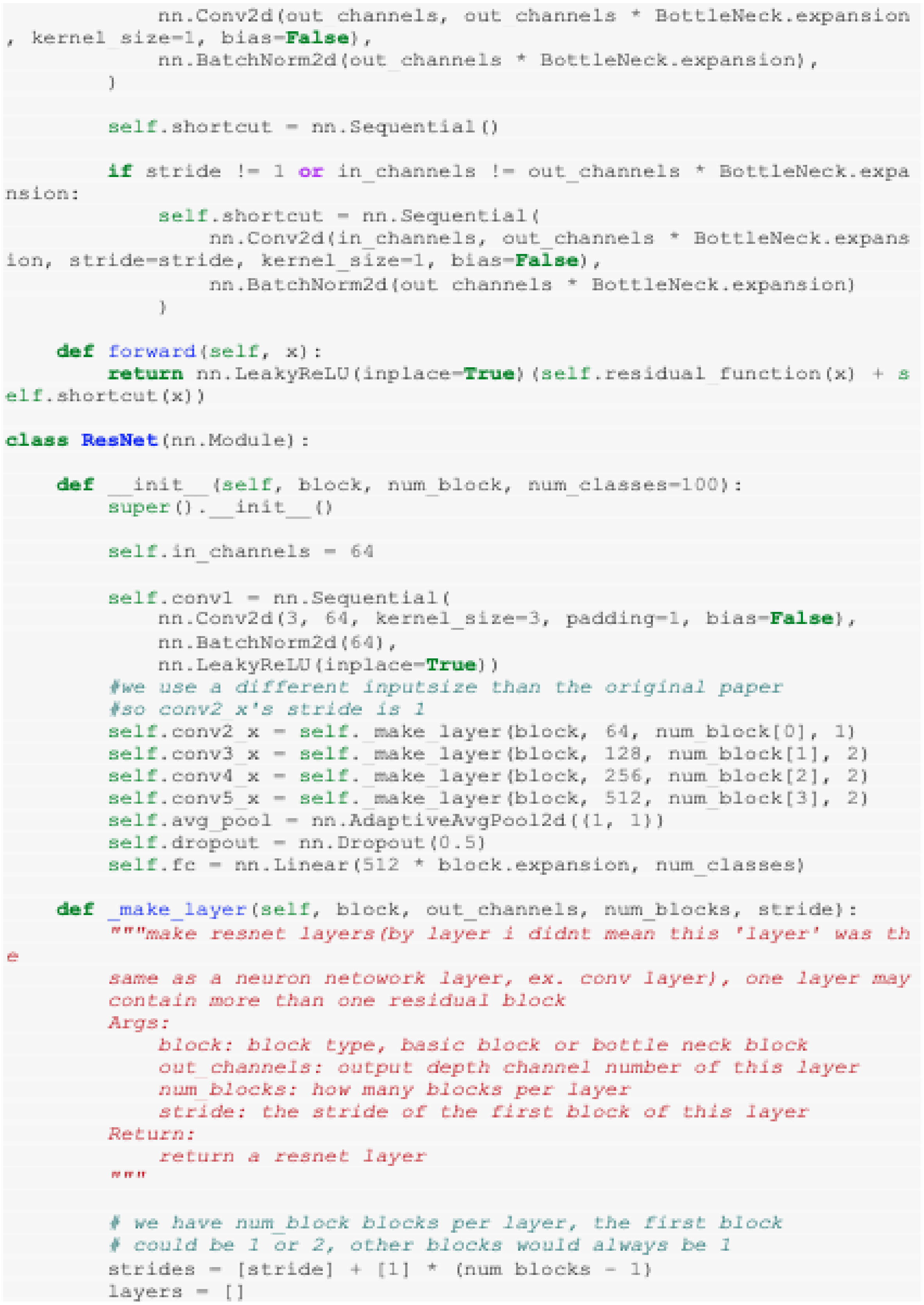


```
with strategy.scope():
    model = tf.keras.Sequential([
        ResNet50(
        input_shape=(512,512,3),
        weights='imagenet',
        include_top=False),
        L.GlobalAveragePooling2D(),
        L.Dense(1, activation='sigmoid')
    ])

    model.compile(
    optimizer='adam',
    loss='binary_crossentropy',
    metrics=['accuracy']
    )
    model.summary()
STEPS_PER_EPOCH=train_labels.shape[0] // BATCH_SIZE

history= model.fit(
train_dataset,
epochs=EPOCHS,
steps_per_epoch=STEPS_PER_EPOCH,
validation_data=valid_dataset)
model.save('model.h5')
```

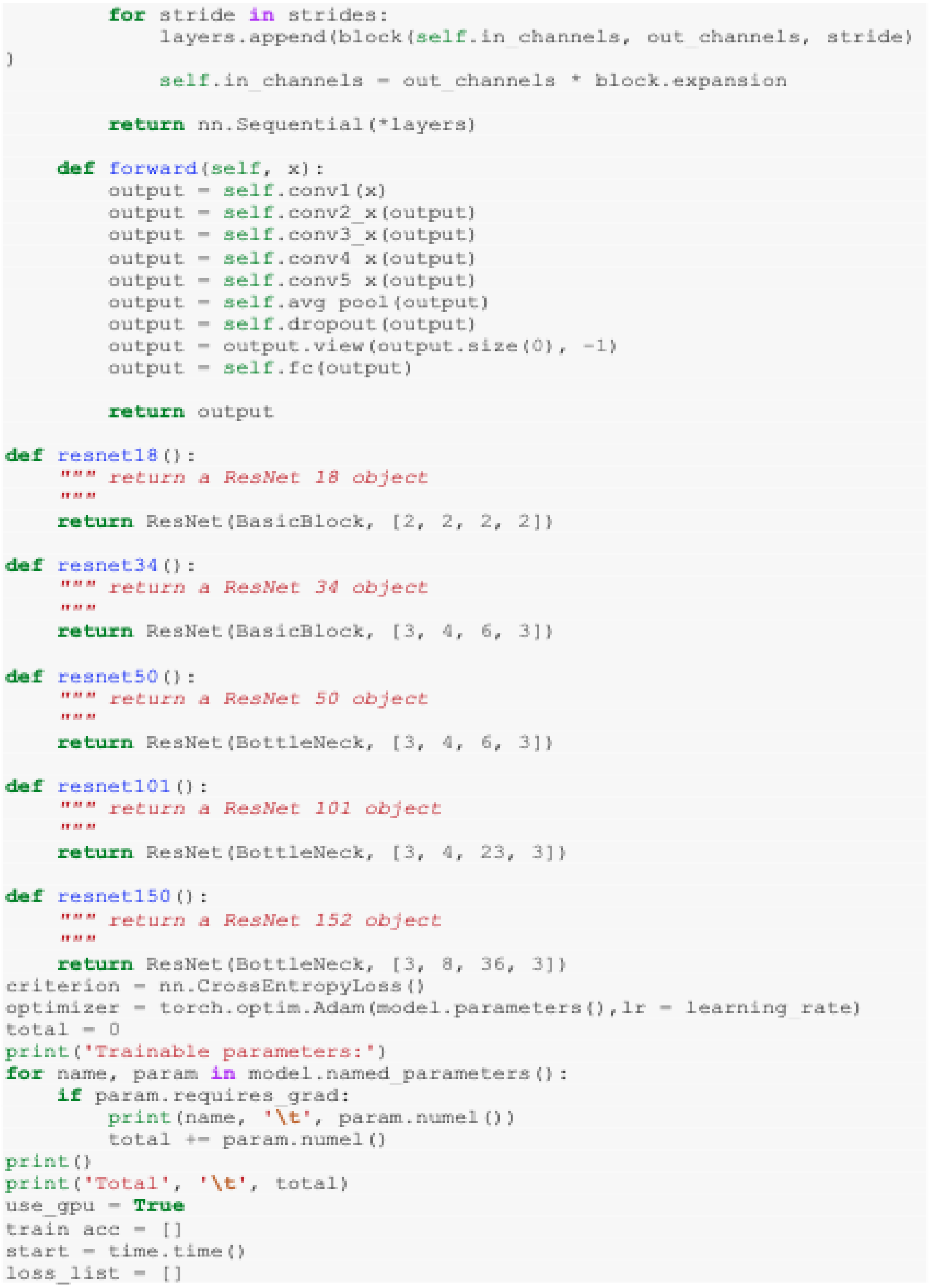
```
import pandas as pd
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
import torchvision.datasets as datasets
import matplotlib.pyplot as plt
import numpy as np
import time
from sklearn.metrics import *
# Normalize training set together with augmentation
transform_train = transforms.Compose([
    transforms.RandomCrop(32, padding=4),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize(mean=[x / 255.0 for x in[0.507, 0.487, 0.441]]
,
                                     std=[x / 255.0 for x in [0.267, 0.
256, 0.276]])
])

# Normalize test set same as training set without augmentation
transform_test = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize(mean=[x / 255.0 for x in[0.507, 0.487, 0.441]]
,
                                     std=[x / 255.0 for x in [0.267, 0.
256, 0.276]])
])
learning_rate=0.001
num_epochs = 1
momentum=0.9
weight_decay=1e-5
batch_size_train=256
batch_size_test=256
batch_size = 256
pd.read_csv("…/kaggle/datset/naïve/submission.csv")
trainset = df(root = "./data",
                                         train=True,
                                         download=True,
                                         transform=transform_train)
trainloader = torch.utils.data.DataLoader(
    trainset, batch_size=batch_size_train, shuffle=True, num_workers=4)

testset = torchvision.datasets.CIFAR100(root = "./data",
                                        train=False,
                                        download=True,
                                        transform=transform_test)
testloader = torch.utils.data.DataLoader(
    testset, batch_size=batch_size_test, shuffle=False, num_workers=4)
```



```
class BasicBlock(nn.Module):
    """Basic Block for resnet 18 and resnet 34
    """

    #BasicBlock and BottleNeck block
    #have different output size
    #we use class attribute expansion
    #to distinct
    expansion = 1

    def __init__(self, in_channels, out_channels, stride=1):
        super().__init__()

        #residual function
        self.residual_function = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, kernel_size=3, stride=
stride, padding=1, bias=False),
            nn.BatchNorm2d(out_channels),
            nn.LeakyReLU(inplace=True),
            nn.Conv2d(out_channels, out_channels * BasicBlock.expansion
, kernel_size=3, padding=1, bias=False),
            nn.BatchNorm2d(out_channels * BasicBlock.expansion)
        )

        #shortcut
        self.shortcut = nn.Sequential()

        #the shortcut output dimension is not the same with residual fu
nction
        #use 1*1 convolution to match the dimension
        if stride != 1 or in_channels != BasicBlock.expansion * out_cha
nnels:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_channels, out_channels * BasicBlock.expans
ion, kernel_size=1, stride=stride, bias=False),
                nn.BatchNorm2d(out_channels * BasicBlock.expansion)
            )

    def forward(self, x):
        return nn.LeakyReLU(inplace=True)(self.residual_function(x) + s
elf.shortcut(x))

class BottleNeck(nn.Module):
    """Residual block for resnet over 50 layers
    """
    expansion = 2
    def __init__(self, in_channels, out_channels, stride=1):
        super().__init__()
        self.residual_function = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, kernel_size=1, bias=Fa
lse),
            nn.BatchNorm2d(out_channels),
            nn.LeakyReLU(inplace=True),
            nn.Conv2d(out_channels, out_channels, stride=stride, kernel
_size=3, padding=1, bias=False),
            nn.BatchNorm2d(out_channels),
            nn.LeakyReLU(inplace=True),
```

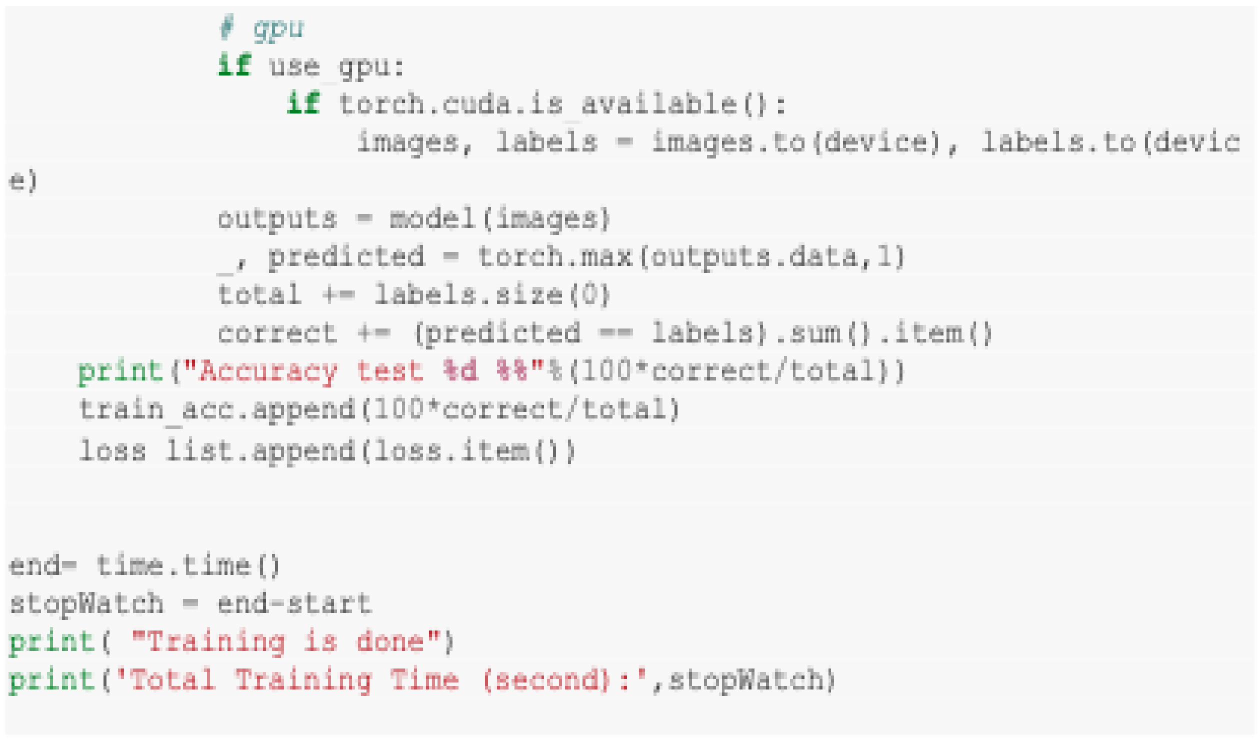


```
                nn.Conv2d(out_channels, out_channels * BottleNeck.expansion
, kernel_size=1, bias=False),
                nn.BatchNorm2d(out_channels * BottleNeck.expansion),
            )

        self.shortcut = nn.Sequential()

        if stride != 1 or in_channels != out_channels * BottleNeck.expa
nsion:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_channels, out_channels * BottleNeck.expans
ion, stride=stride, kernel_size=1, bias=False),
                nn.BatchNorm2d(out_channels * BottleNeck.expansion)
            )

    def forward(self, x):
        return nn.LeakyReLU(inplace=True)(self.residual_function(x) + s
elf.shortcut(x))

class ResNet(nn.Module):

    def __init__(self, block, num_block, num_classes=100):
        super().__init__()

        self.in_channels = 64

        self.conv1 = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=3, padding=1, bias=False),
            nn.BatchNorm2d(64),
            nn.LeakyReLU(inplace=True))
        #we use a different inputsize than the original paper
        #so conv2_x's stride is 1
        self.conv2_x = self._make_layer(block, 64, num_block[0], 1)
        self.conv3_x = self._make_layer(block, 128, num_block[1], 2)
        self.conv4_x = self._make_layer(block, 256, num_block[2], 2)
        self.conv5_x = self._make_layer(block, 512, num_block[3], 2)
        self.avg_pool = nn.AdaptiveAvgPool2d((1, 1))
        self.dropout = nn.Dropout(0.5)
        self.fc = nn.Linear(512 * block.expansion, num_classes)

    def _make_layer(self, block, out_channels, num_blocks, stride):
        """make resnet layers(by layer i didnt mean this 'layer' was th
e

        same as a neuron netowork layer, ex. conv layer), one layer may
        contain more than one residual block
        Args:
            block: block type, basic block or bottle neck block
            out_channels: output depth channel number of this layer
            num_blocks: how many blocks per layer
            stride: the stride of the first block of this layer
        Return:
            return a resnet layer
        """

        # we have num_block blocks per layer, the first block
        # could be 1 or 2, other blocks would always be 1
        strides = [stride] + [1] * (num_blocks - 1)
        layers = []
```



```
        for stride in strides:
            layers.append(block(self.in_channels, out_channels, stride)
)
            self.in_channels = out_channels * block.expansion

        return nn.Sequential(*layers)

    def forward(self, x):
        output = self.conv1(x)
        output = self.conv2_x(output)
        output = self.conv3_x(output)
        output = self.conv4_x(output)
        output = self.conv5_x(output)
        output = self.avg_pool(output)
        output = self.dropout(output)
        output = output.view(output.size(0), -1)
        output = self.fc(output)

        return output

def resnet18():
    """ return a ResNet 18 object
    """
    return ResNet(BasicBlock, [2, 2, 2, 2])

def resnet34():
    """ return a ResNet 34 object
    """
    return ResNet(BasicBlock, [3, 4, 6, 3])

def resnet50():
    """ return a ResNet 50 object
    """
    return ResNet(BottleNeck, [3, 4, 6, 3])

def resnet101():
    """ return a ResNet 101 object
    """
    return ResNet(BottleNeck, [3, 4, 23, 3])

def resnet150():
    """ return a ResNet 152 object
    """
    return ResNet(BottleNeck, [3, 8, 36, 3])
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(),lr = learning_rate)
total = 0
print('Trainable parameters:')
for name, param in model.named_parameters():
    if param.requires_grad:
        print(name, '\t', param.numel())
        total += param.numel()
print()
print('Total', '\t', total)
use_gpu = True
train_acc = []
start = time.time()
loss_list = []
```

```
running_loss = 0
total_step = len(trainloader)
for epoch in range(num_epochs):
    for i, (images, labels) in enumerate(trainloader):
        # gpu
        if use_gpu:
            if torch.cuda.is_available():
                images, labels = images.to(device), labels.to(device)

        outputs = model(images)

        loss = criterion(outputs, labels)



        # backward and optimization
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
        if i % 25 == 24: # print every 500 mini batches
            print("[%d, %5d] loss: %.3f" % (epoch + 1, i+1,running_loss
/25))
            running_loss = 0.0

        if i % 7 == 0:
            print("epoch: {} {}/{}".format(epoch,i,total_step))


    # train
    correct = 0
    total = 0
    with torch.no_grad():
        for data in trainloader:
            images, labels = data


            # gpu
            if use_gpu:
                if torch.cuda.is_available():
                    images, labels = images.to(device), labels.to(devic
e)


            outputs = model(images)
            _, predicted = torch.max(outputs.data,1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
    print("Accuracy train %d %%"%(100*correct/total))
    train_acc.append(100*correct/total)

    # test

    correct = 0
    total = 0
    with torch.no_grad():
        for data in testloader:
            images, labels = data
```



```
            # gpu
            if use_gpu:
                if torch.cuda.is_available():
                    images, labels = images.to(device), labels.to(devic
e)
            outputs = model(images)
            _, predicted = torch.max(outputs.data,1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
    print("Accuracy test %d %%"%(100*correct/total))
    train_acc.append(100*correct/total)
    loss_list.append(loss.item())


end= time.time()
stopWatch = end-start
print( "Training is done")
print('Total Training Time (second):',stopWatch)
```

Bibliography and References:

[1] “An adaptive stenographic technique, ascertaining upper bounds of embedding” by Jyoti Neginal and Dr Ruskar Fatima, Second International Conference on Smart Systems and Inventive Technology (ICSSIT 2019) IEEE Xplore Part Number: CFP19P17-ART; ISBN:978-1-7281-2119-2
[2] “Steganalysis aided by fragile detection of image manipulations” by Ping Wang et al., 2019, Multimedia Tools and Applications (2019) 78:23309–23328.
[3] “Steganalysis of LSB Matching Based on Co-Occurrence Matrix and Removing Most Significant Bit Planes” by M. Abolghasemi et al. , International Conference on Intelligent Information Hiding and Multimedia Signal Processing, 978-0-7695-3278- 3/08 , 2008 IEEE
[4] “ImageNet Pre-trained CNNs for JPEG Steganalysis” by Yassie Yousfi et al., WIFS‘2020, December, 6-11, 2020, New York, USA. 978-1- 7281-9930-6/20/, 2020 IEEE.
[5] “A New Design Method about the Universal Steganalysis Classifier in Digital Image” by Yanping Tan et al., 2020 2nd International Conference on Machine Learning, Big Data and Business Intelligence (MLBDBI), IEEE, 2020
[6] “A Preprocessing Methodology by Using Additional Steganography on CNN-based Steganalysis” by Hiroya Kato et al., GLOBECOM 2020 - 2020 IEEE Global Communications Conference | 978-1-7281-8298-8/20, 2020 IEEE
[7] “A Siamese CNN for Image Steganalysis” by Weike You et al. , IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY, VOL. 16, 2021, IEEE
[8] “A Review of Color Image Steganalysis in the Transform Domain” by Nour Mohamed et al., 2020 14th International Conference on Innovations in Information Technology (IIT), IEEE
[9] “Local-Source Enhanced Residual Network for Steganalysis of Digital Images” by Wonhyuk Ahn et al. , Digital Object Identifier 10.1109/ACCESS.2020.3011752, IEEE Access, 2020
[10] “An Ensemble Model using CNNs on Different Domains for ALASKA2 Image Steganalysis” by Kaizaburo Chubachi, 2020 IEEE International Workshop on Information Forensics and Security (WIFS), WIFS‘2020, December, 6-11, 2020, IEEE, 2020
[11] “A Binary Relevance Adaptive Model-Selection for Ensemble Steganalysis” by Tayebe Abazar et al., 17th International ISC Conference on Information Security and

Cryptology (ISCISC'2020), Iran University of Science and Technology, September 9- 10, 2020
[12] “A Method for Quantitative Steganalysis Based on Deep Learning” by Yu Sun and Tianyun Li, 2019 IEEE 2nd International Conference on Information Systems and Computer Aided Education (ICISCAE), IEEE
[13] “Digital Image Restoration Based on Pixel Simultaneous Detection Probabilities” by Varlen Grabski, IEEE TRANSACTIONS ON NUCLEAR SCIENCE, VOL. 56, NO. 3, JUNE 2009, IEEE
[14] “Digital Image Restoration by Exposure-Splitting and Registration” by Zile Wei et al, Proceedings of the 17th International Conference on Pattern Recognition (ICPR’04), IEEE
[15] “Application of Non-Iterative Method in Digital Image Restoration” by Igor Stojanovic et al. , IEEE
[16] “Digital Image Restoration in Matlab: A Case Study on Inverse and Wiener Filtering” by Mohammed Mahmudur Rahman Khan et al., International Conference on Innovation in Engineering and Technology (ICIET) 27-28 December, 2018, IEEE
[17] “Digital Image Restoration using Image Filtering Techniques” by Reecturaj Mishra et al. , 2019 International Conference on Automation, Computational and Technology Management (ICACTM), IEEE 2019
[18] “Deep Residual Learning for Image Recognition” by Kiaming Hu et al. , arXiv:1512.03385v1 [cs.CV] 10 Dec 2015