Real and Synthetic Image Detection with CNN's

Anonymous Author(s)
Affiliation
Address
email

Abstract

0.1 Motivating Problem:

With the advancements of AI, all forms of digital communication are now compromised. Words, images, videos, even one's own likeness and voice can be imitated by AI. As these forms of mimicry improve, and they will, society will be plunged into the age of mass misinformation. Our trust in all systems is established through communication and the erosion of this trust will bring about conflicts and collapse. Due to AI's ease of use and accessibility, the option to propagate **propaganda**, commit **fraud** and cause **social division** can be given to anyone. The paper aims to tackle the growing and dangerous segment of **AI generated images**. In the future, this may be extended to the frames within videos as well.

0.2 Objective and Methodology:

The paper contrasts and compares the application of various **convolutional neural nets** (CNN's) to classify images as either AI or Non-AI. Using 3 CNN's, two of which are **pretrained**, we train their performance on a training set of 100,000 images and evaluate on a test set of 20000 images. These images were taken from the **CIFAKE dataset** and were all compressed and had the same aspect ratio. This was to reduce the complexity of model training due to our limited computational power. We then choose the best performing CNN and tune its hyperparameters in hopes of optimising its results.

0.3 Results and Conclusion:

Without tuning the validation accuracies over 3 epochs for our 3 models, **LeNet, AlexNet and ResNet50**, are 0.91, 0.975, 0.982 respectively. We found that tuning the residual networks hyperparameters, learning rate and epochs, didn't result in an improvement and so we were left with its original pretrained model as our best performer. We expected the residual networks, which are deep neural networks with **residual blocks** and **skip connections** to outperform shallower networks. But we did not expect all 3 to perform well overall. This may have been due to the pixelated images having less detail and so less intricate patterns which were easier for all three models to learn.

Convolutional Neural Networks

Dense neural networks, e.g feed forward neural networks, require their input to be a vector. An image will be flattened into a great vector with all the pixels and connected to all the neurons. But this process does not account for the spatial positioning of pixels relative to one another. This lack of spatial understanding fails to capture essential features of an image like edges, textures or whole objects. This obstacle is overcome by Convolutional Neural Networks with their use of convolutional

Submitted to 37th Conference on Neural Information Processing Systems (NeurIPS 2023). Do not distribute.

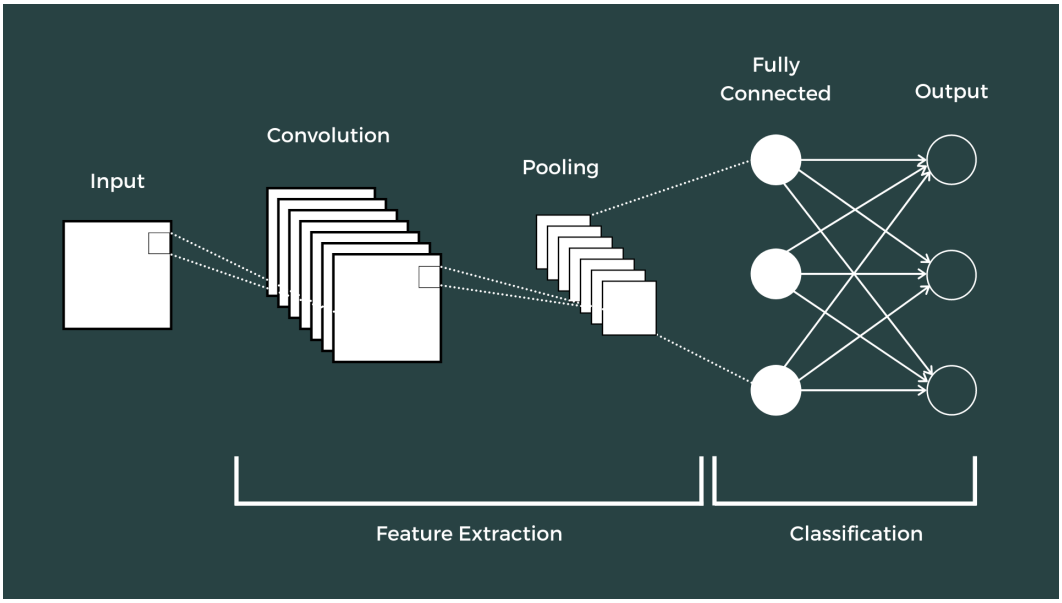

Figure 1: CNN Example

layers. Conv layers use a kernel which is a filter that passes over regions of an image and extracts features through convolutions operations. This is **translational variance** because the features are detected regardless of their position in the image. These are then passed to each layer of dense neurons to learn about. This allows each layer of neurons to be responsible for processing a particular region of an image at a time and all neurons in the same layer can be made to learn and apply the same function to each region they are assigned to using **weight sharing**.

The learning is enhanced by the **locality principle** where pixels near each other are inherently correlated. So in a particular layer of neurons, 2 neurons may be assigned different square regions of pixels. But their weight to, let's say the top right pixel of their respective region is the same. This applies to all the other pixel positions as well. Each neuron in the same layer also has the same bias. Keep in mind the bias and weights are generated randomly across layers but are uniform within their layer. Since neurons start with the same weights for their assigned pixels, when we perform updates to the models they will still be performing the same function to their input even if that function is not the same as prior to the updates.

These CNN's have new hyperparameters from other neural networks. Some of the common ones are as follows. The kernel size of the conv layer determines the region of pixels that influences the output of each neuron, this is the **receptive field** of a neuron. **Kernels** are often square so that the convolution operation is performed uniformly across it. The distance this filter is slid across the image is known as the **stride**. By default this value is 1 so it can capture as many details as possible. It could go up to the size of the kernel but that may result in losing details that could be at the borders of 2 regions. Sometimes the kernel size and stride can cause the kernel to mismatch with the dimension of the input. In this case we use some amount of **padding** by adding a layer of pixels around the border of the input. These pixel values in the new layer around the border are usually zero or the same as the pixel values at the border. Kernel sizes are often odd because when the stride is 1 and with some padding because that causes the output feature map to have the same dimensions as the input. This may be desirable if we want to preserve all the spatial information for later layers.

A problem arises when the **feature channels** created from one conv layer are passed to another conv layer. More feature channels meaning more parameters that need to be trained, which is computationally intensive. An input image has height, width and 3 colour channels and so is captured by a 3d tensor. A fourth dimension exists, that is the number of images in the batch. We will have more parameters (**weights and biases**) for each new **feature map** and this is multiplied by the number of images in the batch. Due to weight sharing, despite having a lot of neurons the actual distinct parameters will be in the tens of thousands which is far fewer than other dense neural networks which

could have millions. But the parameters of CNN's can still explode if you have consecutive conv layers.

Using **pooling layers** the essence of feature maps can still be retained by taking either the max value or the average value of a particular region of conv layers output. The number of neurons is reduced by a factor of the pooling layers size. This downsamples the feature maps, reducing their size so that when they are passed through another conv layer, the output feature maps do not result in an explosion of distinct parameters that need training and updating. Conv and Pooling layers are often repeated across a CNN with **ReLu activation** functions sprinkled in to introduce non-linearity to the model. This is done by turning negative values from the performed convolution to zero's. The effect of this is an improvement in the models ability to generalise and learn diverse patterns.

0.4 LeNet

One of the first CNNs developed by LeCun et al. (1998). Originally designed to recognise handwriting, it was the beginning of a chain reaction of interest in how deep learning can help us with real-life scenarios. LeNet is referred to as LeNet-5 as it has 5 layers in its architecture. 2 conv layers, 1 max pooling layer and 3 fully connected layers. The image size will need be adjusted from 224*224 to 32*32 so that the model can process it.

0.5 AlexNet

Going up chronologically and in complexity, we have AlexNet. It is regarded as a groundbreaking deep-learning model that was first presented by Krizhevsky et al. (2012). It has demonstrated superior performance in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) due to its deeper architecture. It outperformed other models and was cutting the top-5 error rate from the previous year's winner by almost half, from 25.8 percent to 15.3 percent. Its input size is 224*224 which matches our original input size.

0.6 ResNet50

Introduced by He et al. (2015), we have the residual networks. An innovative type of CNN with many variations to its layers depth, creating many variations to the model. Out of all the variations we choose ResNet-50 (50 layers) for its balance of depth with training time. It was also selected because its input iage requirement was 224*224 which matched our input images. Its key innovation lies in its residual blocks and skip connections, which unlike other CNN's allow it to pass information from earlier layers to later layers without significant alterations from the earlier layers. We will see how this alleviates the vanishing gradient problem.

Methodology

0.7 Data

Using a pretrained Alexnet and Resnet allowed us to take advantage of transfer learning. These models were trained on much larger data sets for the **general purpose of image classification**. Having already been trained means their neurons would have learned general functions that may still be useful for our use case of detecting AI images. We replaced the last classification layer of these 2 models to train to better suit our **binary classification** needs. Adding this new layer means it will have new weights which connect to the layer before it. These new weights do require training and so the models will be trained on our data set. This dataset by Bird (2023) has 100,000 images for a training set and 20,000 images for a test set. These are split 50/50 for real and AI-created. Relative to popular alternatives this dataset's size falls between moderate to large. Our hope is that much of the functionality that was trained into the pre-trained models will transfer over to our classification task with a smaller data set. We did not resort to **freezing** the weights of earlier layers because we wanted to make use of the fully functionality of the pretrained model. That being said, it was an option for us to freeze earlier layers because training on our smaller dataset is unlikely to make any meaningful updates to the weights of the earlier layers in the models. Adding **extra layers** also didn't seem necessary because these models are already quite deep and have sufficient conv layers to extract essential details in our smaller, compressed data set.

We acquired our data from **Kaggle** which has a free API that allows us to achieve this. The data was augmented in the original dataset as well as by our team. The images were originally compressed which adds noise and distortion to it. While the images are still identifiable by people a CNN may learn from it differently than if it was not compressed. The **compressions** and size reduction reduced the complexity of images as well as our memory requirements. This may result in faster convergence and improved model accuracy but the loss of information could lead to generalisation issues.

0.8 Pre-processing the data

Our preprocessing steps included the following. The images were resized to 224x224 to standardise the inputs across all dimensions. For AlexNet, which wants an input of 227x227, we do not mind this minor change as PyTorch has an adaptive feature to account for small changes. Due to their shared weights and local receptive fields, the convolution layers in the foundation of architectures of AlexNet are naturally adaptable and can handle small changes in input size. For LeNet, the difference is much larger since we want 32x23. However, we'll just **resize** it in one of its few layers to adjust for this change. Next we used **horizontal flipping** where the chance for the image to flip horizontally was 0.5. This will improve data diversity, where even the slightest feature may be important for the conv layer to pick up to aid in classification later on. Then we adjusted the **brightness and contrast** with a 20 percent chance of images in the training set. This should allow the model to generalise better to dynamic changes in the lighting conditions seen in real-life scenarios. Changes to the data like these should prevent overfitting because it artificially raises the diversity of our data.

Then **normalisation** was performed, the mean and standard deviation used to normalise the pixel values are usually derived from the **ImageNet dataset** (mean: [0.485, 0.456, 0.406], std: [0.229, 0.224, 0.225]). This phase, which matches the distribution of the new input data with that of the training data used in the original model, is crucial when training models using weights pre-initialized from models learned on ImageNet. **Tensor Conversion and Channel Correction** is done using the Albumentations method ToTensorV2(). It changes the channels from HxWxC to CxHxW and allows for images to become tensors so that Pytorch models can understand them.

0.9 Hyper parameters

After selecting our best performing baseline model we then attempt to improve performance by adjusting its hyper parameters. We will increase and decrease the **learning rate** of the best model to see how it affects validation loss and accuracy. We expect an increase in learning rate to speed up the convergence to a solution. This means the model learns to approximate the underlying data faster which should be reflected in better accuracy or loss. On the other hand this could result in the model overshooting the optimal solution. In which case perhaps lowering the learning rate could result in stable convergence which could reach an optimal solution more accurately. This would in turn also suggest better validation loss and accuracy.

Even if no improvement is found due to the pretrained models already having found optimal weights, we will make a note of whether any changes to the results were within our expectations. The experiments will be conducted over more epochs to monitor the loss function and validation accuracy's behaviour. To **evaluate** our models we will use loss, accuracy, ROC curve and the confusion matrix.

LeNet

0.10 Model Architecture

LeNet's simple architecture makes it easy to implement but also faster to train. It uses a few layers making is **shallower** compared to the other models we will use.It's especially suited for small images because fewer pixels means fewer parameters for the model to train as well as update during **backpropogation**. This makes it possible to run it many times on different samples of data.

We expect it to perform worse in terms of validation accuracy because we acknowledge that the compressed images may not have a significant amount of details that would need many conv layers to begin with. It's also likely that its validation loss will be quite high by the last epoch because of the lack of regularisation methods such as drop out thats used in the later model.

Layer	Configuration	Details
Conv1	Input channels: 3	RGB images
	Output channels: 6	
	Kernel size: 5×5	
	Activation: ReLU	
MaxPool	Kernel size: 2×2	
	Stride: 2	
Conv2	Input channels: 6	
	Output channels: 16	
	Kernel size: 5×5	
	Activation: ReLU	
FC1	Input features: $16 \times 53 \times 53$	
	Output features: 120	
	Activation: ReLU	
FC2	Input features: 120	
	Output features: 84	
	Activation: ReLU	
Output	Input features: 84	
	Output features: 1	(for binary classification)
	Activation: None	(raw output scores)

0.11 Results and Interpretation

LeNet has remarkably high validation accuracy of 0.91 on our test set, which we suspected may happen due to the data being compressed and losing complexity. As can be seen in table 1, the validation loss was highest of the 3 models (0.23) by a large margin. This tells us the model suffers from overfitting which may be due to its lack of regularisation methods e.g dropout. The model may have memorised noise and distortions in the training data, as those were introduced when the data was compressed and resized. This could be the cause of the overfitting, that tells us it cannot generalise well to unseen data.

Epoch	Train Loss / Accuracy	Validation Loss / Accuracy
Epoch 1/3	0.4723 / 0.7639	0.3674 / 0.8353
Epoch 2/3	0.3198 / 0.8629	0.2530 / 0.8964
Epoch 3/3	0.2582 / 0.8939	0.2272 / 0.9102

Table 1: Training and Validation Metrics for LeNet

AlexNet

0.12 Model Architecture

The image goes through a series of conv, ReLU, max pooling layers, an adaptive average pooling layer and finally the classifier layers. The conv layers aim to extract features and local patterns. This may include things like edges and image textures. These are normalised, where the ReLu function is applied to them to introduce non-linearity. Then the input is passed to pooling layers to reduce their spatial dimensions to reduce the processing burden for later layers. This is repeated until we reach the fully connected layers represented by the linear function. Just before being inputted into the classifier functions the adaptive average pooling layer summarises the information from all the

Table 2: AlexNet's Convolutional Layers

Layer	Configuration
Conv1	Input channels: 3, Output channels: 64, Kernel size: 11×11, Stride: 4×4, Padding: 2×2 ReLU, MaxPool2d(3, 2)
Conv2	Input channels: 64, Output channels: 192, Kernel size: 5×5, Stride: 1×1, Padding: 2×2 ReLU, MaxPool2d(3, 2)
Conv3	Input channels: 192, Output channels: 384, Kernel size: 3×3, Stride: 1×1, Padding: 1×1 ReLU
Conv4	Input channels: 384, Output channels: 256, Kernel size: 3×3, Stride: 1×1, Padding: 1×1 ReLU
Conv5	Input channels: 256, Output channels: 256, Kernel size: 3×3, Stride: 1×1, Padding: 1×1 ReLU, MaxPool2d(3, 2)

Table 3: AlexNet's Adaptive Average Pooling Layer

Layer	Configuration
AdaptiveAvgPool	Output size: 6×6

feature maps, condensing them into a vector that can be inputted into the fully connected layers. These layers associate the images to specific classes for classification and they are followed up by ReLu and dropout layers. **Dropout** is a regularisation method that ignores the output of random neurons during the forward and backward passes through the network. Doing so forces the model to not become reliant on particular neurons and perform by learning general patterns that should mitigate overfitting. The final FC layer uses either a **sigmoid function** for binary classification or a softmax function to assign probability scores to each class.

0.13 Results and Interpretation

As expected, having more conv layers and pooling layers resulted in lower loss and better accuracy than LeNet, that was 0.067 and 0.975 respectively. These are shown on table 5.The model has a high validation accuracy and low loss.

Based of the **drastic reduction** we now see in validation loss, from LeNet's 0.26 to AlexNet's 0.056 we can be more confident of the models ability to generalise to new images and is good at distinguishing between Non-AI and AI images. This is due to having more conv layers that enable feature detection and extraction despite the images having some noise and distortion added from being compressed.

Table 4: AlexNet's Classifier Layers

Layer	Configuration
Dropout1	Probability: 0.5
FC1	Input features: 9216, Output features: 4096 ReLU, Dropout(0.5)
FC2	Input features: 4096, Output features: 4096 ReLU
Output	Input features: 4096, Output features: 1

Epoch	Train Loss / Accuracy	Validation Loss / Accuracy
Epoch 1/3	0.0678 / 0.9748	0.0837 / 0.9704
Epoch 2/3	0.0616 / 0.9770	0.0844 / 0.9695
Epoch 3/3	0.0576 / 0.9786	0.0670 / 0.9755

Table 5: Training and Validation Metrics for AlexNet

Residual Networks

0.14 Problem:

Prior to residual networks, deep neural networks were time intensive to train for any practical applications. Their depth meant that each additional layer introduces another random **weight matrix** that is multiplied with the activation function. With each additional layer to the end, the original input gets scrambled repeatedly until it is indistinguishable from **random noise**. When updates are performed for optimisation, the computed loss at the end of the model is backpropagation through the network. But because of the scrambling, that loss is not informative to the later layers because their inputs were mostly random noise. But it's not just the later layers, the earlier layers are rendered useless as well. This is because when the **gradients** of the **loss function** are back propagated they are multiplied with each random weight matrix as well. The gradients represent the rate of change of the loss function with respect to the network's parameters (weights and biases); they're needed to convey the parameter changes needed to adjust the loss function by a specific amount. By the time these gradients reach earlier layers, their values will continue to shrink to zero from the multiplications done through **chain rule**. Due to this, no meaningful updates can be made to the weights, this is the **vanishing gradient problem**.

So because the beginning layers receive unhelpful gradients and the later layers receive unhelpful random noise, the model's loss takes many epochs to reduce during training and the gradient descent algorithm becomes inefficient and unhelpful. This results in insignificant decreases in the loss per epoch and is worsened by the training time being impractically long per epoch due to neural networks hunger for large amounts of data.

0.15 Resnet's Solution:

Residual networks allow for the input to later layers and gradients to earlier layers, to be less jumbled up. It does this through **skip connections and residual blocks**. A description of the data's movement for 1 block is as follows. The input goes through two ways simultaneously, the first is through the block and the second is it skips the block by going through an **identity function** that returns itself at the end of the block. So at the end of the block there is the modified input and the original input. These two inputs are in the form of tensors which are combined by either adding or concatenating. This new input now passes through the rest of the network which is composed of these **bottlenecks** in the same way.

That means that the output from any block can be passed to much later layers in the network, so any block can now work with the original input unchanged as well as a version of the data that has been modified by previous layers. These same paths for the input are used by the gradients during backpropagation to reach the earlier layers faster and without much shrinkage. Now all the layers can receive meaningful updates. So by using the shorter paths the model loss can steadily decrease in the first few epochs and as time is spent training, the gradients passed through the residual blocks will still contribute to meaningful updates. All this allows the model's loss to steadily decrease per epoch so it can outcompete a shallow network.

0.16 ResNet50's Architecture

The initial conv layer uses a 7x7 kernel and has 64 filters. Every conv layer is followed by batch normalization, and at the end of every bottleneck is also a ReLU activation function.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 2: ResNet50 in the middle

After the initial conv layer, we get a max pooling layer for downsampling the feature maps to prevent hyperparameters from growing and slowing down training.

Typical residual blocks group 2 convolutional layers together with a round of batch normalization after each convolutional layer, finished by a ReLU activation function. ResNet50's residual blocks are a modified form called 'bottlenecks' that have 3 conv layers each. Then there are 4 groups of bottlenecks. For each convolutional layer in all the bottlenecks, the kernel is size 1x1, 3x3, and 1x1 respectively. The first 1x1 conv layer reduces complexity so it's less expensive for the 3x3 conv layer to extract features from it. Then the last 1x1 conv layer projects the depth back up to the original amount of spatial dimensions.

After all the bottlenecks, we have the average pooling layer to account for the information in all the feature maps and condense it to a fixed length vector that's inputted into the fully connected layer.

Here the softmax activation function is applied to the input, converting the logits into probabilities that are assigned to all the classes. These values are between zero and one and sum to one for all the classes.

0.17 Results and Interpretation

Due to the residual blocks, ResNet50 **outperformed** the shallower models as expected. Both loss and accuracy for the baseline model improved across each epoch to reach impressive values of 0.056 and 0.98 on table 6. If we could train for longer, we expected to see it continue to improve at a slower and slower rate. Compared to a deep model without residual blocks, the loss would be relatively high over the first several epochs without notable decreases. We compared our base ResNet50 model to two cases where we increased the epochs to 5 for each and in one case increased the learning rate and in the other case decreased the learning rate. Increasing the learning rate showed that the more **aggressive updates** to the model's weights would result in faster convergence in fewer epochs; decreasing would have the opposite effect.

Using the **confusion matrix** we call calculate the models precision, recall and F measure. Our real images are represented by 1 and synthetic images by zero. The confusion matrix quadrants (1,1), (0,1), (1, 0) and (0,0) are labeled A,B,C and D respectively. Note these coordinates are (x,y). **Precision** is the proportion of true 'real image' predictions out of all the real images. **Recall** is the true 'real image' predictions out of all the real images in the test data. Precision tells us of all our predictions of a particular class, what proportion of them are actually reliable, such as thing is relevant in contexts like reviewing photographic evidence in court. Optimising for recall is useful because it tells us if our model is better at predicting real or synthetic images. I.e how sensitive the model is to a particular class of predictions. **F-measure** gives us a value that summarises and balances the information of precision and recall.

Model	Epoch	Train Loss / Accuracy	Validation Loss / Accuracy
Baseline ResNet50	Epoch 1/3	0.1282 / 0.9494	0.0723 / 0.9729
	Epoch 2/3	0.0631 / 0.9763	0.0598 / 0.9785
	Epoch 3/3	0.0437 / 0.9836	0.0558 / 0.9791
ResNet50 (Lower LR)	Epoch 1/5	0.1862 / 0.9254	0.0955 / 0.9642
	Epoch 2/5	0.0955 / 0.9633	0.0754 / 0.9729
	Epoch 3/5	0.0712 / 0.9735	0.0655 / 0.9758
	Epoch 4/5	0.0576 / 0.9785	0.0588 / 0.9795
	Epoch 5/5	0.0448 / 0.9837	0.0564 / 0.9797
ResNet50 (Higher LR)	Epoch 1/5	0.6340 / 0.6690	0.3990 / 0.8177
	Epoch 2/5	0.3147 / 0.8681	0.3023 / 0.8777
	Epoch 3/5	0.2227 / 0.9118	0.3053 / 0.8710
	Epoch 4/5	0.1896 / 0.9257	0.2185 / 0.9111
	Epoch 5/5	0.1650 / 0.9358	0.1409 / 0.9468

Table 6: Comparison of Training and Validation Metrics for Different ResNet50 Configurations

Our ResNet50 model presents 0.9798 precision, 0.9795 recall and 0.9796 f measure. These were calculated from the confusion matrix on figure 3. These are all really high values are are likely due to the simplicity in details from the compressed images. If these models were to be applied to more AI images those would need to be compressed to match the size requirements for these models. Doing so could still prove to be a highly useful tool in AI image detection.

Conclusion

Our ResNet 50 model has the highest validation accuracy (although not my much). Adjusting the learning rate did not improve its validation accuracy or loss but perhaps it could by trying out a larger range of values. Our ResNet50 model presents 0.9798 precision, 0.9795 recall and 0.9796 f measure. These are all really high values are are likely due to the simplicity in details from the compressed images. If these models were to be applied to more AI images those would need to be compressed to match the size requirements for these models. Doing so could still prove to be a highly useful tool in AI image detection.

References

[1] Bird, J. J. (2023, April 26). CI-Fake: Real and AI-generated synthetic images [Data set]. Kaggle. https://www.kaggle.com/datasets/birdy654/cifake-real-and-ai-generated-synthetic-images

[2] He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep residual learning for image recognition. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 770–778. https://doi.org/10.1109/CVPR.2016.90

[3] LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324. https://doi.org/10.1109/5.726791

[4] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. In Advances in Neural Information Processing Systems (pp. 5998–6008).

[5] Krizhevsky, A., & Hinton, G. (2009). Learning multiple layers of features from tiny images.

[6] Real images are from Krizhevsky & Hinton (2009), fake images are from Bird & Lotfi (2024).

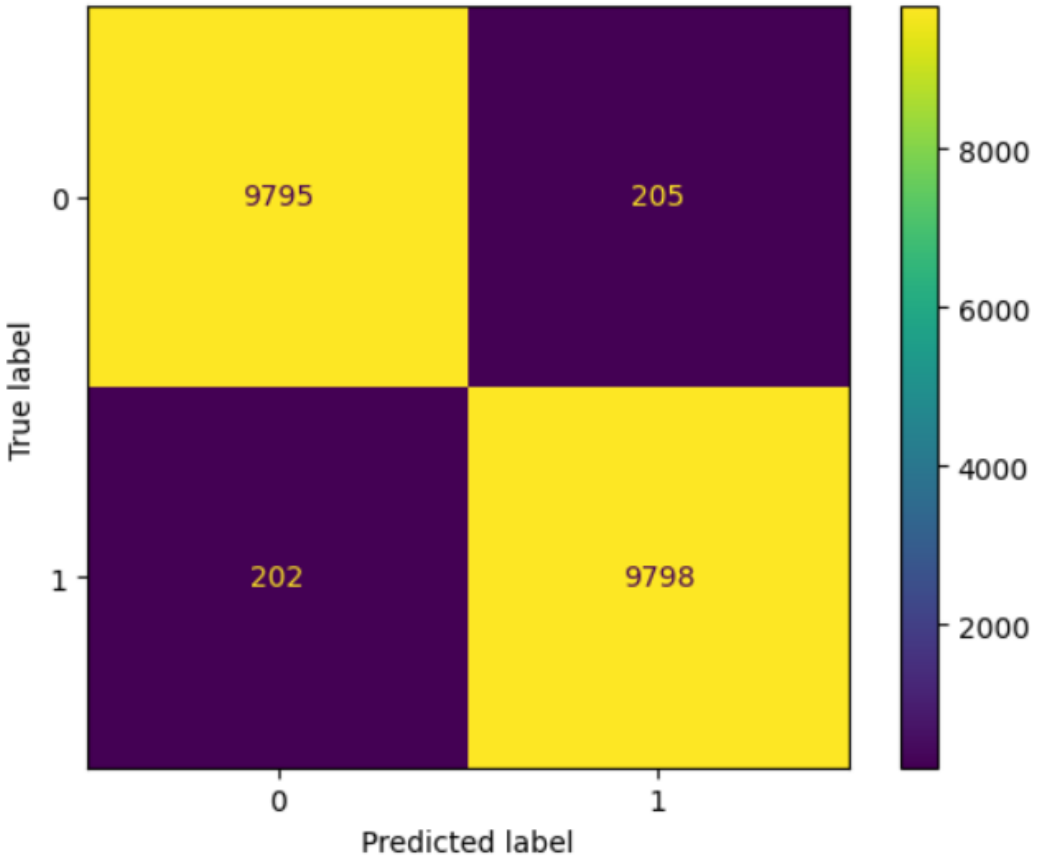

Figure 3: Baseline ResNet50 Confusion Matrix

1 Statement about individual contributions

Each of our teammates believes we split the work fairly and equally. We also assigned tasks that stuck to our strengths. Everyone's contribution was 33.33 percent each.

23007 - Completed Research Paper.
28364 - Completed Presentation, assisted with research paper.
27647 - Completed Code and plots.