Deep Learning – Assignment 2 (Convolutional Networks)

Name: Shane Quinn Date: 29/04/2021

Student Number: R00144107

Part A: Convolutional Neural Networks (CNN)

Part (i)

The 6 CNN architectures implemented in this section consist of one or several convolutional and pooling layers, followed by one densely connected layer and a SoftMax layer. The different architectures investigated in this section can be seen below in Table 1

	Convolutional Layers (C)	Pooling Layers (P)	Final Validation Accuracy	Execution Time (s)	Architecture
Baseline	1	1	.56	52	CP
Basic 1	2	2	.68	49	CPCP
Basic 2	5	5	.84	57	CPCPCPCPCP
Basic 3	6	6	.84	58	CPCPCPCPCPCP
Basic 4	6	3	.81	838	CCPCCPCCP
Basic 5	5	4	.80	126	CPCPCCPCP

Table 1 – Basic CNN Architectures

In the graphs below we can see how the various models described in the above table performed

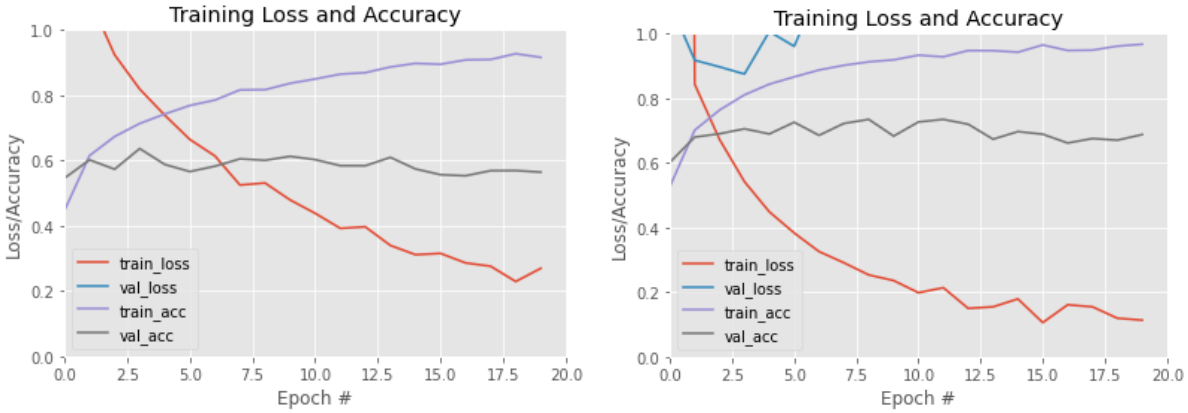

Figure 1 – Baseline & Basic Architecture 1

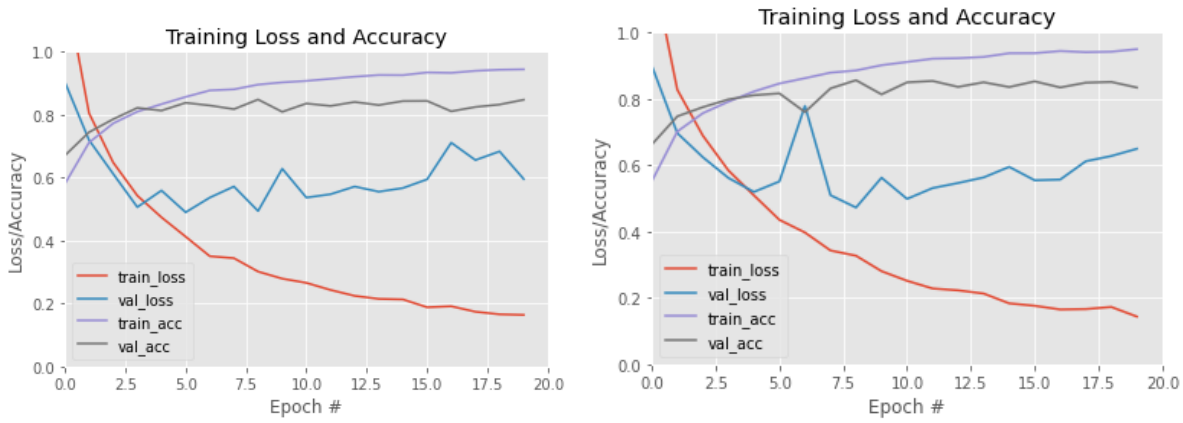

Figure 2 – Basic Architecture 2 & 3

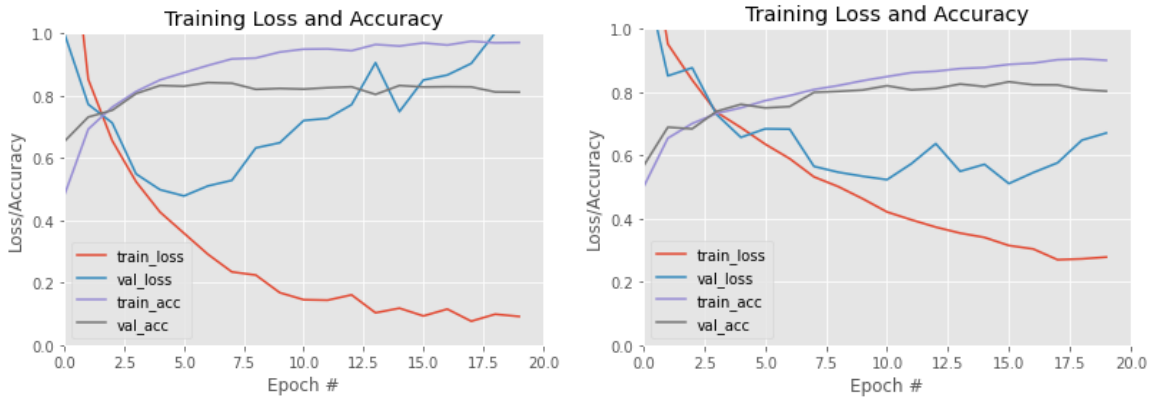

Figure 3 – Basic Architecture 4 & 5

We can see that as we increase convolutional and pooling layers the validation accuracy increases. The basic architecture sees a maximum validation score of around .58, whereas introducing 1 extra convolution and pooling layer increases the accuracy to approximately .68. Basic architectures 2 & 4 show how as we add convolutional and pooling layers, we can see accuracies as high as .84. One drawback of the addition of extra layers is the increase in execution time, which we can see from Table 1 above, while in these basic models there is not a large difference.

In Basic Architecture 4 (seen in Figure 3) we can see very aggressive overfitting. This neural network architecture consists of 6 convolutional layers and 3 pooling layers, in the order of 2 convolution layers followed by a pooling layer repeated 3 times. The number of filters in convolutional layers decreases as we progress from 254 to 32. We can see the training accuracy in this model is extremely high however the model begins to overfit around epoch 5. The execution time for this model was also extremely high. This is due to having 2 convolutional layers with 254 filters directly after each other. Leading to a far higher number of tuneable parameters than the other models which helps explain the aggressive over-fitting we're seeing. The addition of a pooling layer between these 2 layers would reduce the number of tuneable parameters thus mitigating against over-fitting, we could also explore regularisation techniques or data augmentation, to address over-fitting.

Data Augmentation

For this section Basic Architecture 3 and 5 will be used to investigate the potential benefits of using data augmentation techniques on these model performances. Data augmentation is the process of artificially generating new data, so the model has more data to aid learning and can help improve

the model performance. It also has the added benefit of reducing the model's chance to overfit as it may never see the same image twice.

The introduction of data augmentation techniques increases the runtime of the model drastically. In this implementation we used the following data augmentation techniques

- Cropping
- Zoom
- Vertical/Horizontal mirroring
- Rotation

Below graphs show a comparison of how our two models perform when data augmentation is implemented vs when it is not

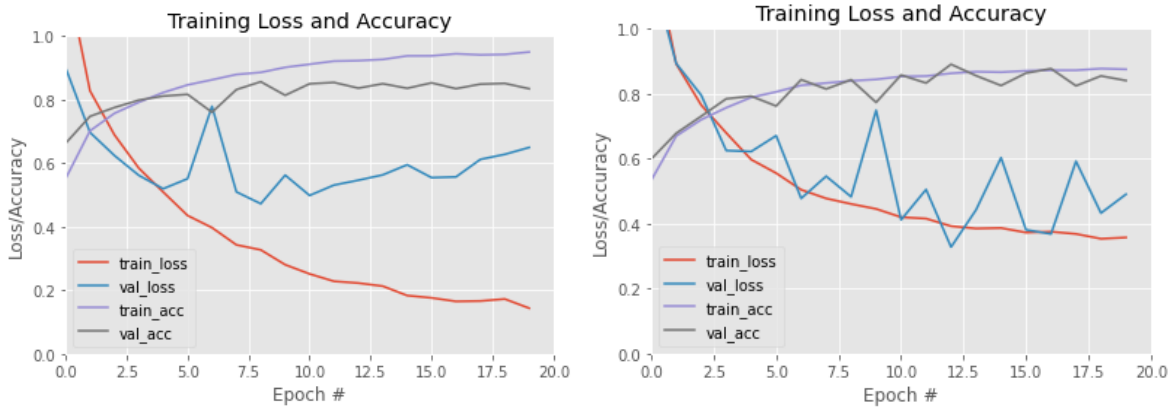

Figure 4 – Basic Architecture 3 (Left standard, right with data augmentation implemented)

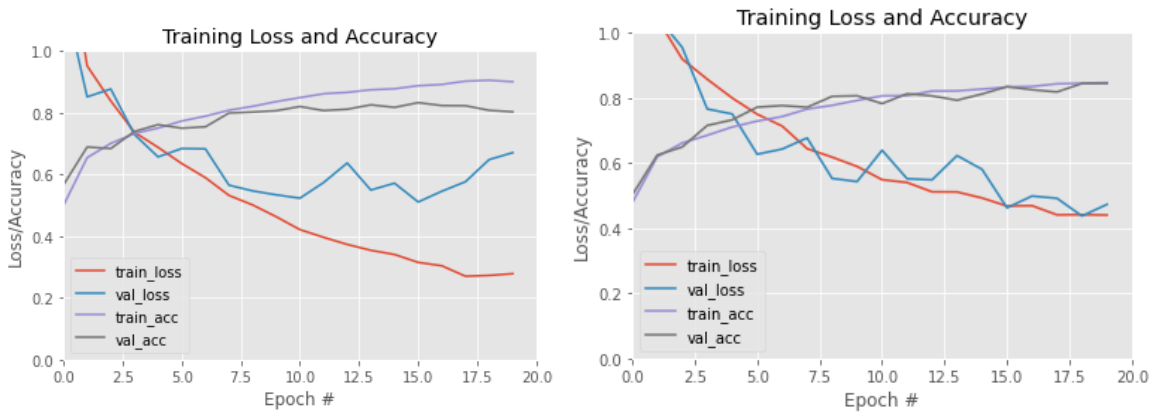

Figure 5 – Basic Architecture 5 (Left standard, right with data augmentation implemented)

As we can see there is a clear decrease in over-fitting in both cases, as well as evidence of reduced over-fitting we can see that the validation accuracy in both cases increases slightly with the addition of data augmentation, this is particularly visible in architecture 5. The table below shows the impact of data augmentation on execution time. For the same number of epochs, the execution time is drastically increased.

	Standard	**With Data Augmentation**
Basic 3	58s	438s
Basic 5	126s	486s

Table 2 – Model Execution time standard vs with data augmentation

There is no evidence of over-fitting within the 20 epochs for this architecture, running this model for 50 epochs and 100 epochs we can see that it the model achieves almost .90 validation accuracy without overfitting. This demonstrates the power of data augmentation techniques and how in some cases it can mitigate against the risk of over-fitting.

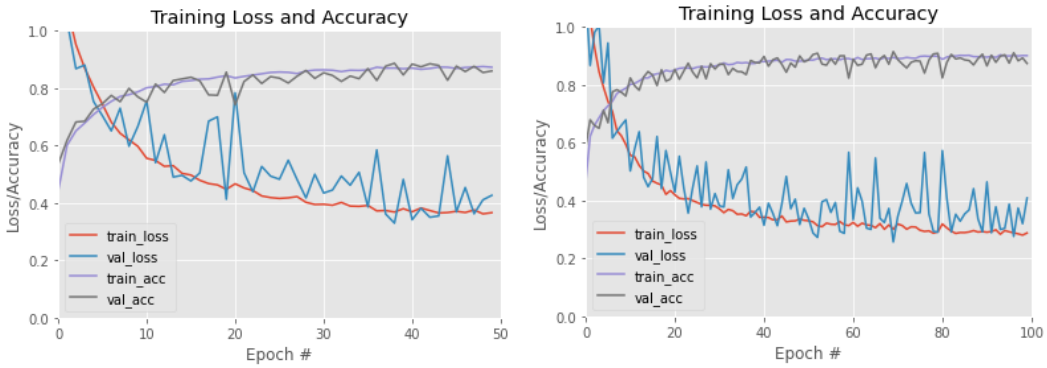

Figure 6 – Basic architecture 5, 50 epochs (L) & 100 epochs (R)

Next I carried out some experiments to investigate the effects of the certain types of data augmentation techniques used by removing them individually, and training our model with just the remaining 3 techniques.

Without mirror

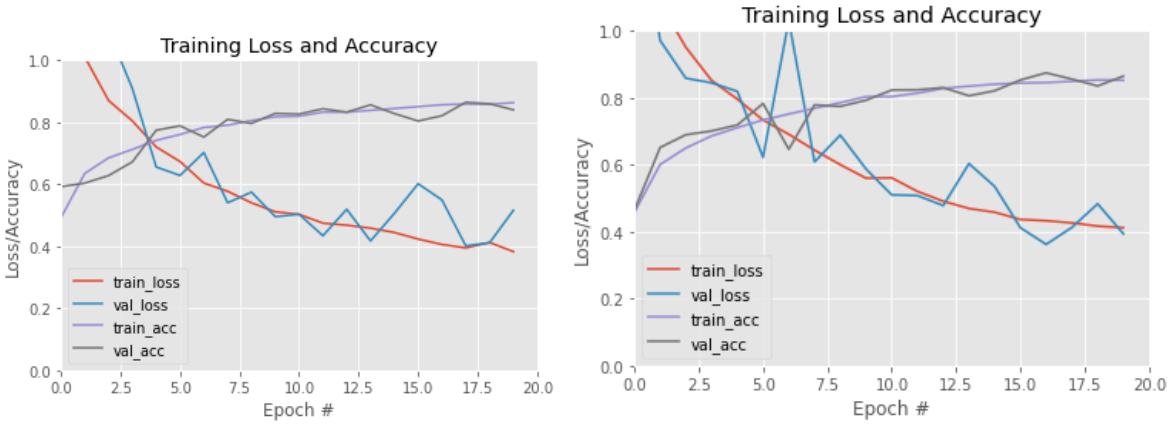

Figure 7 – Without mirror (Left) – Without cropping (Right)

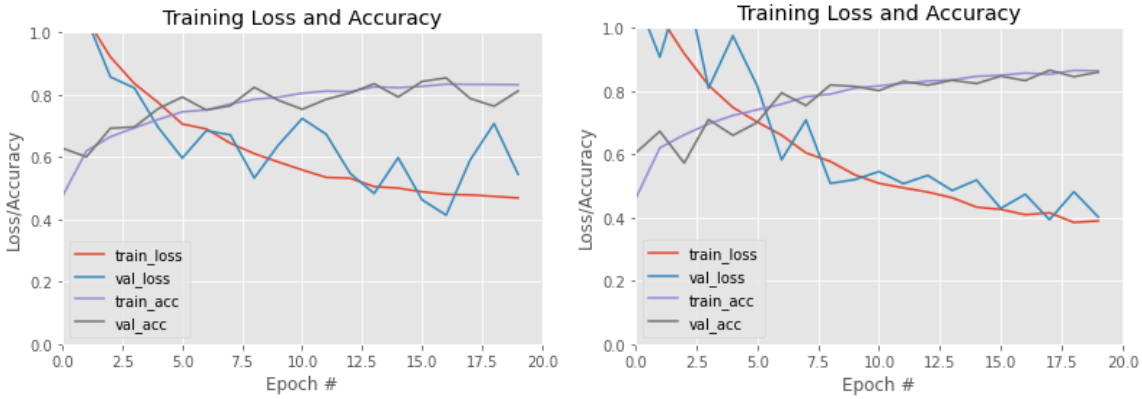

Figure 8 – Without rotation (Left) – Without Zoom (Right)

As we can see the removal of any one data augmentation technique does not have a great deal of effect on the validation accuracy of the model.

A new data augmentation technique described in 'Deep Adversarial Data Augmentation for Extremely Low Data Regimes (DADA)' [1] uses GAN's to optimize data augmentation for datasets which have only a small set of labelled data. This technique was used to augment an electroencephalographic (EEG) signal dataset, which is used for the classification of brain signals. Using DADA researches found an increase in performance on average of 1.7% when compared to the state-of-the-art CNN-SAE [2] model.

Another recent data augmentation paper titled, 'Circular Shift: An Effective Data Augmentation Method For Convolutional Neural Network On Image Classification' [3]. This method addresses the impact of the data augmentation techniques used in this implementation described earlier, on small images or dispersed feature of objects. The goal of Circular Shift is to mitigate against losing valuable information that cropping or rotating could cause. This is achieved by splitting the image into 'K' segments, rotating and shifting these segments creating a new image.

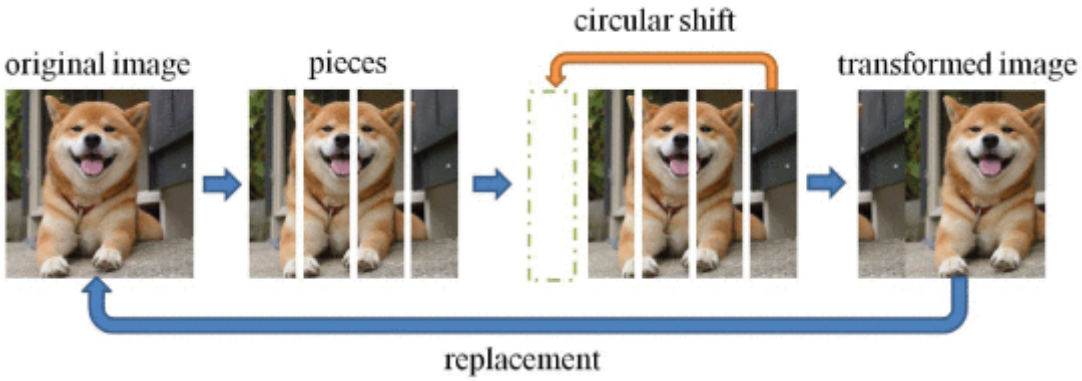

Figure 9 – Circular Shift operation [3]

The above figure shows the operation of circular shift. As we can see none of the original data is lost during the operation making this data augmentation technique beneficial for small images.

One-shot learning

In 'Data Augmentation Using Learned Transformations for One-Shot Medical Image Segmentation' [4] researchers were able to generate MRI scans from just one segmented scan leading to significant improvements over the current state-of-the-art in MRI image segmentation. The issue with using techniques implemented in the accompanied code (available as part of Keras's 'ImageDataGenerator' method) is they struggle to generate genuine variations and are sensitive to hyper-parameter tuning. Using a CNN based on U-Net [5] they implement an auto-encoder seen below,

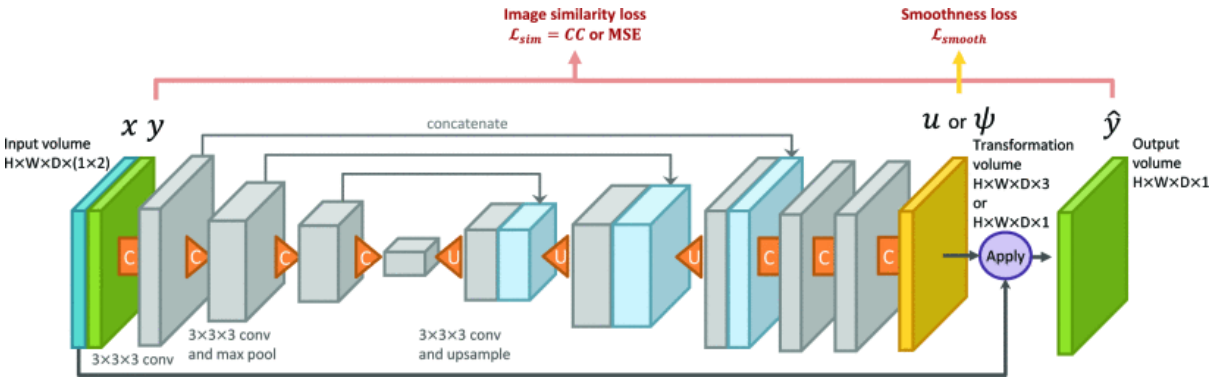

Figure 10 – One-shot learning data augmentation method [4]

This method allows the model to learn non-linear shapes and deformations seen in MRI tumour scans from just one image.

Zero-shot learning

'An Introduction to Zero-Shot Learning: An Essential Review' [6] describes how zero-shot learning could potentially address the issue in deep learning that is finding large labelled datasets. Large labelled datasets can in many cases require specialists to manually label them costing hours of work. This paper describes how Zero-Shot learning uses the semantic relationship of instances to generate new samples.

Again looking at a EEG dataset discussed previously, 'EZSL-GAN: EEG-based Zero-Shot Learning approach using a Generative Adversarial Network' [7] describes a new method for dealing with small amounts of data. EZSL-GAN works by encoding EEG data using a Gated Recurrent Unit (GRU), followed by a GAN and finally a classification network which discriminates between generated and unseen EEG signals.

Part (ii) – Ensembles

An ensemble neural network architecture is the combination of multiple model predictions aggregated together. Neural networks by their stochastic nature are susceptible to a great deal of variance considering weights are initialised randomly. By training multiple models and combining their predictions we can draw a more reliable, accurate prediction better than any single model.

The obvious drawback of implementing ensembles is the computational cost. Most high performing neural network architecture's such as AlexNet and Inception are ensemble based however due to this high computational cost of training multiple models they are rarely used in real-time applications.

Methodology

The simplest form of ensemble is to use the same baseline model with different weights. To achieve this the developer would train their baseline model multiple times, saving the best performing mode each time and at the inference stage they will load these saved models and aggregate their predictions. For models that can take a very long to train this method would take a lot computation power to train an ensemble. Snapshot ensembles is an answer to this, which is where the model is trained once, and multiple snapshots are taken as it is trained, and these snapshots are used to build the ensemble.

To increase variability of our base learners more than one model was used. For the ensemble method implemented in the accompanied code I picked the two best performing architectures from

Part A (i) (Basic Architecture 3 and Basic Architecture 5) and created 2 models from each architecture. Using checkpointing to save the best performing model in each.

Another method for increasing variability is to train each baseline model on different subsets of the training data, this was not undertaken in this implementation however we are using data augmentation techniques in our baseline models so no model see's the same image multiple times. To combine the results of all models I aggregated the output of each learners SoftMax layer.

Variability & Performance

The approach taken to investigate the impact of implementing an ensemble on variance was to build the same ensemble several times and record the performance of each of the base learners as well as the ensemble, below is the resulting accuracies, means and standard deviations observed.

	Accuracy 1	Accuracy 2	Accuracy 3	Accuracy 4	Accuracy 5	Mean	Std Dev
m1 (Basic 3)	0.876	0.88	0.904	0.876	0.906	0.8884	0.011662
m2 (Basic 3)	0.895	0.903	0.902	0.904	0.849	0.8906	0.003536
m3 (Basic 5)	0.848	0.86	0.852	0.863	0.847	0.854	0.006016
m4 (Basic 5)	0.897	0.883	0.871	0.89	0.892	0.8866	0.009601
Ensemble	0.922	0.92	0.921	0.925	0.918	0.9212	0.001871

Table 3 – Ensemble and base model accuracy

From the table above we can see that the mean performance of our ensemble model is >3% better than the highest mean performing baseline model. This table also shows the benefits of using an ensemble to mitigate against the variance seen in our baseline models. The standard deviation of our ensemble model is 0.001871 while our baseline models' standard deviations range from 0.0035 – 0.0116, indicating that the ensemble model is almost twice as resistant to variance as the best performing baseline model in terms of variance.

Research (Ensemble advancements)

A more intricate design is to weigh each of the baseline models differently. This could help us to achieve a better performing overall model. These weights could be trained to an optimal value.

'A novel diversity-guided ensemble of neural network based on attractive and repulsive particle swarm optimization' [8] describes an effective method of optimizing base learner weights which considers both accuracy and variance of the base learners. This is achieved using attractive and repulsive particle swarm optimization.

Introducing a weighted base learner ensemble could help us achieve a more accurate overall ensemble.

Part B: Transfer Learning

Feature Extractor

Transfer learning is a common technique used when designing neural network architectures, it leverages pre-existing designs. These models can be used as feature extractors which in turn feed into another application specific ML model.

In the implemented example we looked at an array of variations detailed in the table below:

	F1 Score	
	VGG16	**VGG19**
KNeighborsClassifier	0.7625	0.770208

DecisionTreeClassifier	0.671875	0.65375
RandomForestClassifier	0.8591	0.857083
SVC	0.905625	0.898333
SGDClassifier	0.8775	0.871458
LogisticRegression	0.888958	0.880208
GaussianNB	0.62458	0.652917
AdaBoostClassifier	0.510416	0.571667

Table 4 – Transfer Learning combinations

As we can see there is little difference between VGG16 and VGG19. We can see that of the ML classifiers tested the support vector classifier scores the highest, with Logistic regression second highest. Taking this a step further I experimented with hyper-parameter tuning of the SVC using Grid Search. The optimal config with an F1 score of **0.91** is:

- VGG16
- SVC
    - C = 10
    - Kernel = rbf

This experimentation took a long time to carry out, in hindsight a better approach would have been to take a sub sample of the dataset, for hyper parameter tuning.

Fine Tuning

Fine tuning is another branch of transfer learning. This technique also leverages an existing design (in our case VGG), this is connected to a densely connected network. The first time we train the network we freeze the weights in the VGG and just tune our densely connected network. When we have trained this portion of the network, we go back and unfreeze a portion of the pre-trained VGG net. And train them weights

In our implemented example we have VGG16 connected to 254 relu activated neurons followed by our 9 class SoftMax layer. After this was tuned, we unfroze the weights of blocks 3, 4 and 5 of the VGG achieving an accuracy of **0.9635**. The graphs below depict model performance with the weights of VGG frozen (left) and certain layers unfrozen (right)

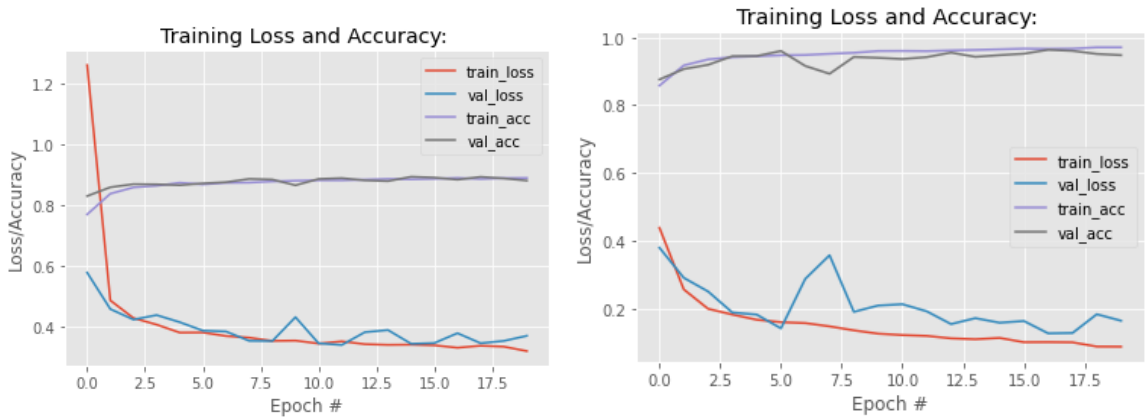

Figure 11 – Fine tuning raining loss and accuracy, Phase 1 (Left) – Phase 2 (Right)

When fine tuning a common practice is to reduce the learning rate in our optimizer, which we done in this example. The learning rate for tuning the densely connected network was 0.001 (default adam learning rate) and this was reduced to 0.0001 for fine tuning. This is done because the VGG

network is already trained on the ImageNet dataset so the weights should not need to be altered that much.

Part C: Adversarial Machine Learning

Adversarial machine learning (ML) is the branch of research which investigates purposely misleading ML algorithms to mis-classify inputs. For example, the image below (to the human observer) is very clearly a bird, or to the trained human observer, a blue jay. The top image has not been altered in any way and is correctly classified as a 'jay' to a very high accuracy. The ML model used in this example is VGG, which is a combination of multiple CNNs developed for high accuracy object detection and won the ImageNet competition in 2014. The bottom image has been altered using adversarial machine learning techniques and is now being classified as a 'mask' to a very high accuracy. Yet, to the human eye these images are identical. This is the basis of adversarial machine learning.

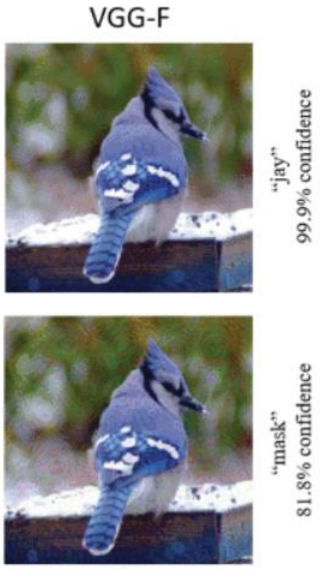

Figure 12 – VGG-F vs Adversarial Machine Learning [9]

To describe adversarial samples, we must first recall that computers interpret numbers and not images as the human eyed does. Deep neural networks (DNNs) take in a large amount of labelled training data, for machine vision problems such as MNIST or ImageNet, images are represented as pixel array of 0-255 greyscale values or 3-dimensional array of RGB pixel values, respectively. A DNN learns and draws relationships between these values and the correct class output so when it is presented with a new (unseen) example it can predict to a high accuracy the correct class.

In 'Explaining and Harnessing Adversarial Examples' [10], published in 2015, Ian Goodfellow et al. describe (contrary to theories at the time) that the effect of adversarial samples on machine learning models was caused by the linear nature of neural networks.

In 'DeepFool: A Simple and Accurate Method to Fool Deep Neural Networks' [11], researchers developed an algorithm to efficiently generate adversarial samples. They describe the minimum perturbation required for a given image to be misclassified. The perturbation seen below in the right two images, is the pixel change summed with the original image to give the adversarial samples on the left.

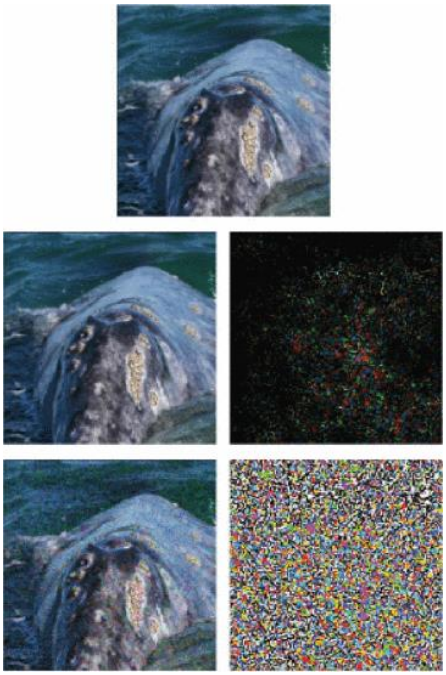

Figure 13 – Deepfool vs fast gradient sign

The Figure above shows the difference in perturbation required to fool a DNN to misclassify this image of a 'whale' as a 'turtle'.

In 'The Limitations of Deep Learning in Adversarial Settings' [12] researchers were able to cause a deep neural network to mis-classify specific input images with a 97% success rate, while just altering 4.02% of the input. All these samples were still successfully classified by human observers. If we consider the equation below to define the minimum perturbation required to achieve the desired class (Y1)

$$argmin_{\delta X} \parallel \delta X \parallel \ s.t.\ F(X + \delta X) = Y1$$

Where δX is the multi-dimensional vectors summed with our original image (X).

Input: $\mathbf{X}$, $\mathbf{Y}^*$, $\mathbf{F}$, Υ, θ

$\mathbf{X}^* \leftarrow \mathbf{X}$
$\delta_{\mathbf{X}} \leftarrow \vec{0}$
$\Gamma = \{1 \ldots |\mathbf{X}|\}$
while $\mathbf{F}(\mathbf{X}^*) \neq \mathbf{Y}^*$ and $||\delta_{\mathbf{X}}|| < \Upsilon$ **do**
    Compute forward derivative $J_{\mathbf{F}}(\mathbf{X}^*)$
    $S(\mathbf{X}, \mathbf{Y}^*) = \texttt{saliency\_map}\,(J_{\mathbf{F}}(\mathbf{X}^*), \Gamma, \mathbf{Y}^*)$
    $i_{max} \leftarrow \arg\max_i S(\mathbf{X}, \mathbf{Y}^*)[i]$
    Modify $\mathbf{X}^*_{i_{max}}$ by θ
    $\delta_{\mathbf{X}} \leftarrow \mathbf{X}^* - \mathbf{X}$
end while
return $\mathbf{X}^*$

Figure 14 – Creating adversarial sample [12]

An important point is that adversarial samples have nothing to do with a model training phase, they are introduced to a model during validation or in production. This leads us to one method to mitigate against the risks of adversarial samples which is to retrain models frequently. The generation of adversarial samples is an active area of research.

References

[1] X. a. W. Z. a. L. D. a. L. Q. a. L. Q. Zhang, “Deep Adversarial Data Augmentation for Extremely Low Data Regimes,” *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 31, no. 10.1109/TCSVT.2020.2967419, pp. 15-28, 2021.

[2] Y. R. T. a. U. Halici, “A novel deep learning approach for classification of EEG motor imagery signals,” *Journal of Neural Engineering,* vol. 14, 2016.

[3] K. a. C. Z. a. W. J. Zhang, “Circular Shift: An Effective Data Augmentation Method For Convolutional Neural Network On Image Classification,” *2020 IEEE International Conference on Image Processing (ICIP),* no. 10.1109/ICIP40778.2020.9191303, pp. 1676-1680, 2020.

[4] A. a. B. G. a. D. F. a. G. J. V. a. D. A. V. Zhao, “Data Augmentation Using Learned Transformations for One-Shot Medical Image Segmentation,” *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR),* no. 10.1109/CVPR.2019.00874, pp. 8535-8545, 2019.

[5] P. F. a. T. B. Olaf Ronneberger, “U-Net: Convolutional Networks for Biomedical”.

[6] O. A. a. S. G. M. Soysal, “2020 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA),” *2020 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA),* no. 10.1109/HORA49412.2020.9152859, pp. 1-4, 2020.

[7] S. a. H. K. a. S. G. a. B. H. Hwang, “EZSL-GAN: EEG-based Zero-Shot Learning approach using a Generative Adversarial Network,” *2019 7th International Winter Conference on Brain-Computer Interface (BCI),* no. 10.1109/IWW-BCI.2019.8737322, pp. 1-4, 2019.

[8] F. a. Y. D. a. L. Q.-H. a. H. D.-S. Han, “A novel diversity-guided ensemble of neural network based on attractive and repulsive particle swarm optimization,” *2015 International Joint Conference on Neural Networks (IJCNN),* 2015.

[9] N. a. M. A. Akhtar, “Threat of Adversarial Attacks on Deep Learning in Computer Vision: A Surve,” *IEEE Access,* vol. 6, pp. 14410-14430, 2018.

[10] J. S. &. C. S. Ian J. Goodfellow, “Explaining and Harnessing Adversarial Examples,” *ICLR 2015.*

[11] S.-M. a. F. A. a. F. P. Moosavi-Dezfooli, “DeepFool: A Simple and Accurate Method to Fool Deep Neural Networks,” *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* pp. 2574-2582, 2016.

[12] N. a. M. P. a. J. S. a. F. M. a. C. Z. B. a. S. A. Papernot, “The Limitations of Deep Learning in Adversarial Settings,” *IEEE European Symposium on Security and Privacy (EuroS P),* pp. 372-387, 2016.