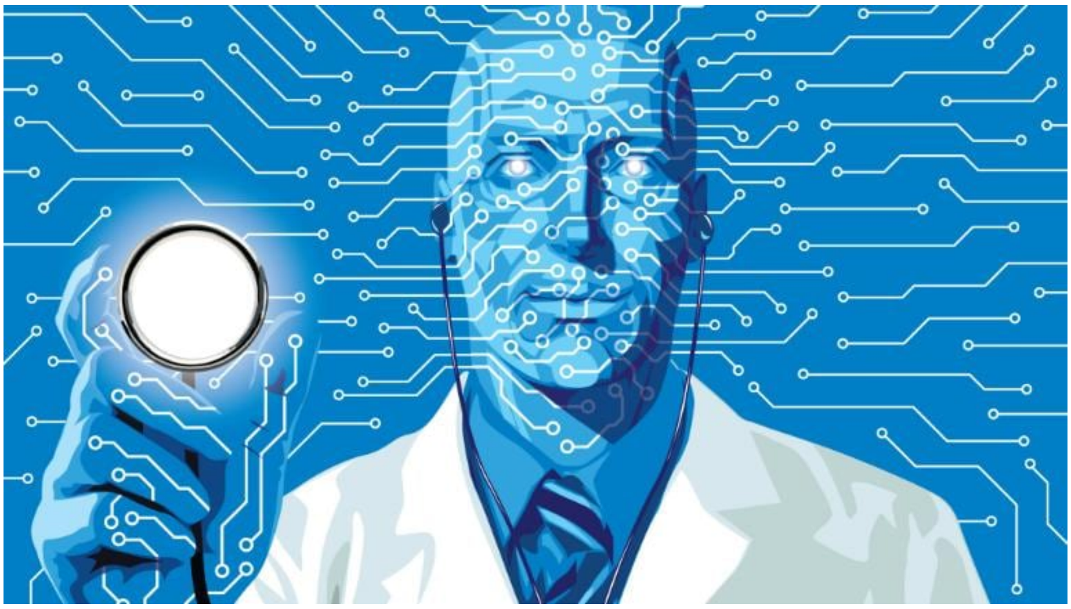
CHEST X-RAY (COVID-19 & PNEUMONIA) DETECTION REPORT



Prepared for: COMP534 - Applied AI 2021

Prepared by:

Aditya Naik
Student ID: 201574102
sganaik@liverpool.ac.uk

Reza Fazli
Student ID: 201538160
sgrfazli@liverpool.ac.uk

Yuzhe Qiu
Student ID: 201535477
Y.Qiu30@liverpool.ac.uk

Introduction

COVID-19 is a viral respiratory infection that affects the human lungs and has been declared to be a global pandemic. Due to the spreadability of the COVID-19 virus, early detection is crucial, as this will protect both patients and others around them. Early diagnosis of patients infected by the COVID-19 virus and special care and treatment are the best ways to combat the pandemic. This becomes a bit difficult as one of the symptoms of COVID-19 is pneumonia [1], which is basically the inflammation of lung tissue which could be induced by viruses and bacteria other than the coronavirus as well. People of all ages can die from viral pneumonia other than COVID-19. In order to diagnose any patient that shows pneumonia symptoms, chest X-rays (CXR), as well as computer tomography scans (CT) are the best options, but sometimes even expert radiologists find determining the cause of pneumonia difficult due to this similarity.

To assist these radiologists in providing a diagnosis quickly and accurately, we try to harness the power of deep learning. Computer vision, a subset of deep learning, has proven to be a promising method for detecting medical conditions through the analysis of medical imagery. In our project, we will be making use of the chest X-ray-covid19-pneumonia dataset which is open source and available on the Kaggle platform along with pre-trained models to develop a multi-class classification model to help distinguish the X-ray images of COVID-19, other pneumonia and healthy patients.

Data cleaning

Our dataset consists of images of X-ray scans for COVID19, pneumonia and normal patients. The data is loaded in the form of directories leading to the images along with the class label and stored in a dataframe. The images have three channels and are of varying sizes. The sizes are restricted later on while implementing the pre-trained models as they have in-built preprocessing to bring the data to the appropriate size. The training dataset seems to be imbalanced with pneumonia accounting for 3418 images while COVID-19 and normal images make up for 460 and 1266 images respectively. This could cause problems as the model will be forced to give more importance to pneumonia and would also provide misleading accuracy values. We alleviate this issue by randomly selecting 460 images from the normal and pneumonia categories resulting in a balanced training dataset with each category having 460 images [3]. Although this greatly reduces our training dataset size, we overcome the bias induced due to the smaller number of training images by implementing a technique called Data augmentation.

Data Augmentation [4]

During the training process, we make use of the Image preprocessing and augmentation API of Keras known as ImageDataGenerator. The following operations were performed by the API on our data:

Rescaling: Performs min-max scaling for all pixels of each channel
Rotation: The images are rotated within a range of -10 to +10 degrees
Lateral shift (width shift): Images are shifted laterally by 10% in either direction
Vertical shift (height shift): Images are shifted vertically by 10% in the top-down direction
Horizontal flip: Mirrored images of the original X-ray scans are taken across the vertical axis

The rescaling operation is to bring all the inputs to the same scale so as to enable the model to train faster and avoid local optima and is part of the data preprocessing and not augmentation. On the other hand, the rotation, vertical and lateral shifting and horizontal flipping are data augmentation operations. The data obtained after using these operations are treated as new data even though it was obtained by slightly modifying the original training images. This helps us reduce the bias induced in our training data due to the sub-sampling discussed in the previous section. In addition to this data augmentation also acts as a regulariser as it forces the model to be more tolerant to variations in size, position and orientation of the image and makes it focus on the important features. [5]

Training, Validation and Testing data

There are a total of 1380 training images and 1288 test images. The training data is first shuffled by using the panda's **sample** method with the **frac** parameter set to 1 which shuffles all the rows in the data frame without replacement at random. The test images are kept aside and remain untouched until the final model is trained and validated. The training data on the other hand is further divided into the training data and validation data in the 80:20 ratio. This is done by specifying the **validation_split** parameter as 0.2 in the ImageDataGenerator API object and subsequently specifying the **subset** parameter in the train_data_generator and val_data_generator objects as **'training'** and **'validation'** respectively. We also set the **batch size for training as 32** in the train_data_generator object as larger batch sizes result in instabilities while smaller batch sizes do not exploit the power of parallel computing using GPUs. [6]

Libraries used

The Pandas library was used to implement data frames to store the directories for each image along with its class labels for each image in the training, validation and testing data. The NumPy library was used to make calculations and manipulate the data arrays. Matplotlib for plotting the training and validation plot as well as displaying the images from the dataset. Sklearn was used to display the confusion matrix and calculate the precision, recall and f1 score for training validation and test. The pre-trained models we imported and modified using TensorFlow and Keras.

Pre-trained models and Transfer learning [7]

Transfer learning is a popular and very powerful approach that in short can be summed up as the process of learning from a pre-trained model that was trained to perform a similar task but on a different dataset [5]. This method involves cutting off the last few layers of a pre-trained model and retraining them to accomplish a new task. It is obvious, that deep neural networks consume large amounts of training time and computing resources, and are prone to overfitting. So instead of developing a CNN model from scratch, we use powerful pre-trained models, the **ResNet50** and the **GoogleNet**. These are deep and complex models trained by experts with access to large amounts of computing resources. These models were trained on the ImageNet dataset with 1000 classes of images. In our work, we first load the two pre-trained models with the ImageNet weights but replace the top layer with a softmax dense layer having only 3 outputs as we have only three classes. We then directly test the models on our dataset as it is without training and compare the performance and advantages of each. Next, we select the appropriate model out of the two and customise the top layers using the hyper-parameter tuning and train the final model in two stages, the details of which we will see in the next few sections.

ResNet50 [8]

ResNets make use of a technique called **residual learning.** This utilises skip connections or shortcut connections to enable the network to model the difference between the target function and the input which is the basis for residual learning. The presence of many skip connections allows the training to progress quickly even if many intermediate layers are stuck or learning slowly as the input signal is available throughout the network [5]. These distinct advantages enabled the model to be deeper with fewer parameters and provide better accuracy in the ILSVRC challenge in 2015.
The ResNet50 consists of 50 residual units. Each unit has two convolutional layers with Batch normalisation and Relu activation function using 3x3 filters and the same padding with a stride of 1.

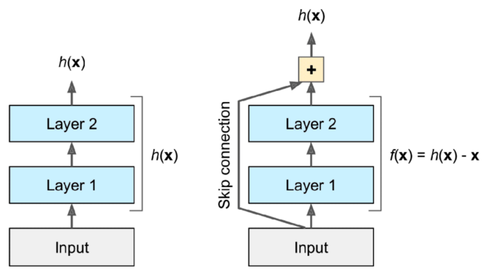
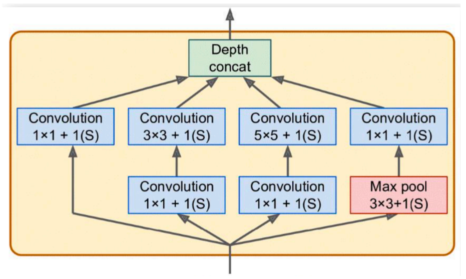


Figure 14-15. Residual learning

Image taken from Aurelion Geron, E 2020," Hands-on machine learning with Scikit-learn, Keras and TensorFlow", O'Reilly Media, Inc

GoogLe Net (inceptionV3) [9]

This network is made up of a number of **inception modules** in which the input signal is first copied and fed to four different layers. Each pair of convolutional layers ([1x1,3x3] and [1x1,5x5]) acts like a single powerful convolutional layer, capable of capturing more complex patterns. **The 1x1 convolutional layers help capture cross channel patterns** and serve as a **bottleneck layer** by producing fewer feature maps compared to the inputs which cut computational costs and the total number of parameters resulting in faster and more accurate performance. It also makes use of **local response normalisation** which inhibits neurons in neighbouring feature maps resulting in the specialisation of different features. As a result of these advantages, this model was the winner of the ILSVRC challenge in 2014.



Inception module Image taken from Aurelion Geron, E 2020," Hands-on machine learning with Scikit-learn, Keras and TensorFlow", O'Reilly Media, Inc

As we can see each of these models has its own distinct advantages. We now load both these pre-trained models with the ImageNet weights without the top softmax layer and replace it with a softmax layer with 3 outputs and test the models directly without training and see how they perform.

We see from the table below that both models perform quite badly. The reason for this is that although the lower layers are doing their job well and detecting low-level patterns the actual decision making happens at the top layers and since we have added an untrained softmax layer on the top it does not give proper results. So to make the model take advantage of the trained layers and produce good metrics we will have to add a few dense layers to the top of the pre-trained model and train it on the dataset. We select the GoogLe net (inceptionV3) to do this out of the two because it provides a higher validation accuracy of 0.779 on the ImageNet dataset [10] compared to 0.749 for ResNet50 [11]. The inceptionV3 also has approximately 22 million parameters (model 2 summary in the notebook) while the ResNet50 has about 23.5 million parameters (model 1 summary in the notebook) resulting in a lesser number of computations in Google net.

Model	Train accuracy	Validation Accuracy	Train Precision	Validation Precision	Train Recall	Validation Recall	Train F-score	Validation F-score
ResNet50	0.3324	0.3374	0.1108	0.1123	0.3333	0.3333	0.1663	0.1680
GoogLe Net	0.3216	0.3297	0.1108	0.2168	0.3333	0.2866	0.1663	0.1970

Hyper-parameter tuning

The pre-trained model is loaded with the ImageNet weights and the layers are locked to prevent training. The hyper-parameter tuning is performed to add the dense layers on the top. We try to find out the best option for the number of hidden layers, neurons per layer, activation function, optimiser and initial learning rate. We do not change the loss function and keep it fixed at **sparse_categorical_crossentropy** as the labels have sparse values and are mentioned in the class_model parameter in the flow_from_dataframe method of the ImageDataGenerator object, using any other loss functions results in errors. We implement a function called BuildNetwork that builds the dense network on top of the pre-trained model. To build this neural network we will be using batch normalisation [5] which normalises the inputs before sending them into the next layer. This tends to speed up training by avoiding the vanishing gradients problem [5]. We will not be using this method for layers with the 'selu' activation function as these layers are self-normalising. In addition to this, we also use the He_normal initialisation strategy for ReLU activation functions and its variants [5] and the Glorot initialisation for sigmoid and tanh functions [5] and LeCun initialisation for the selu function [5] instead of random, as random initialisation tends to bring in instability at the early stages of training resulting in longer training times and causing the network to get stuck in local optima [5]. The model is then trained for each value of the hyper-parameter in the list for 5 epochs and the hyper-parameters which produce the highest validation sparse categorical accuracy is selected. The validation is performed on the validation set and does not make use of k-fold cross-validation.

Parameters selected:
Activation functions: ['sigmoid', 'tanh', 'relu', 'elu', 'selu']
Number of hidden layers : [0, 1, 2, 3, 4]
Neurons per layer: [32, 64, 128, 256]
Optimisers : ['Adam','RMSprop', 'Adagrad', 'Adadelta', 'Adamax', 'Nadam']

Initial learning rate : [0.1, 0.01, 0.001, 0.0001]
We will not be using a constant learning rate as this might cause the model to train very slowly if it's small or cause it to oscillate and never converge to the global minimum. To avoid this we will be using a learning rate scheduler [12] while training the final model. This helps take advantage of the high initial learning rate at the beginning and progressively drops its value as it progresses resulting in a fine-tuned search for the minimum.

Epochs: For tuning the epochs we will be using the Early stopping callback while training the final model based on the above-tuned hyper-parameters. We will choose a sufficiently high number of epochs and start training the model with the early stopping callback with patience = 5 indicating that if the validation sparse categorical accuracy doesn't improve for 5 successive epochs the training will stop [5].

Hidden Layers (Figure 2)

The higher the number of hidden layers in the network enables the model to detect more patterns and hence learn better. But at the same time, a very deep network may cause overfitting and also increase the computational time and resources. 3 is selected because it has the highest validation accuracy with no signs of overfitting as the difference between the scores is not high.

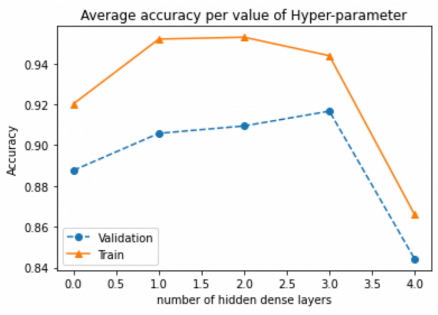


Figure 2

Hidden Units (Figure 3)

Similar to the number of layers, the number of nodes in a layer is also important. Too many could overfit the data by detecting unnecessary patterns among the variables itself while too few tend to result in a high bias due to under-fitting. In this case, we choose 32 as it has the highest training and validation score compared to the others.

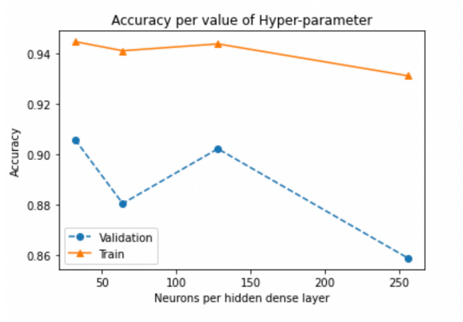


Figure 3

Activation Function (Figure 4)

Selected parameter: ‘selu’

The major advantage of using SELU is that it provides **self-normalization** (that is output from SELU activation will preserve the mean of 0 and standard deviation of 1) and this solves the vanishing or exploding gradients problem[13]. Selu is selected because it results in the best validation score due to its self normalising behaviour.

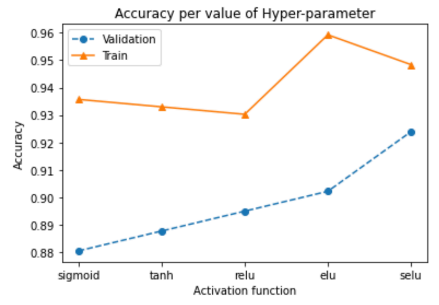


Figure 4

Optimisers (Figure 5)

Nadam is the best optimiser for this task as is indicated by its high training and validation score. This basically combines the advantages of Adam with that of using the Nesterov momentum.

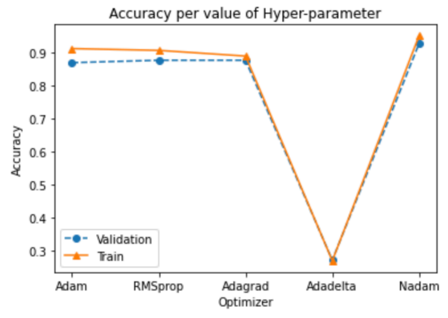


Figure 5

Initial Learning Rate (Figure 6)

We will be using a learning rate scheduler while training the final model [5]. This helps take advantage of the high initial learning rate at the beginning and progressively drops its value as training goes on resulting in a fine-tuned search for the minimum. The best initial value seems to be 0.001 which is the second smallest value among the chosen ones with the highest training and validation scores. The reason for this is that since it is a pre-trained model not much learning is required and so a smaller learning rate is better.

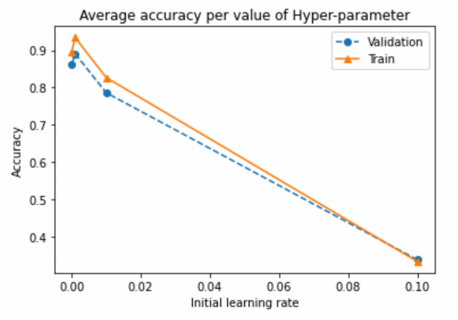


Figure 6

Final Model

For the final model we choose the following parameters:

Hidden layers = 3,

Hiden Units = 32

Activation function = ‘selu’,

Optimiser = ‘Nadam’

Loss function = ‘sparse_categorical_crossentropy',

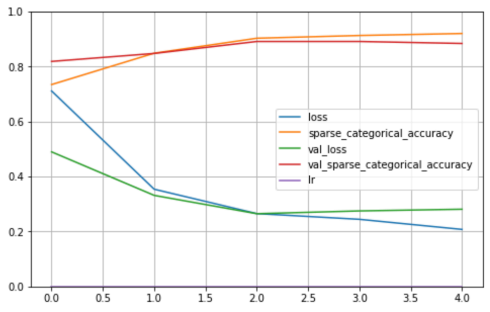
Initial Learning rate = 0.001,

batch size = 32

The final model was trained in two stages:

Stage 1

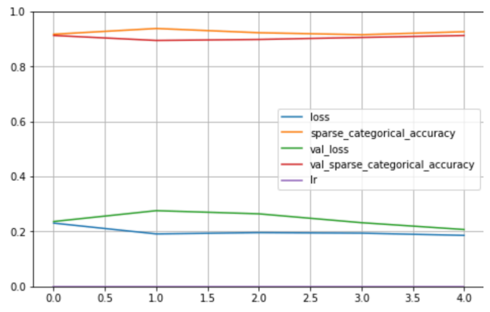
The InceptionV3 model was loaded along with the ImageNet weights and without the top layer. All the layers of the pre-trained model were locked in order to prevent any changes to the weights. A global average pooling layer was applied on top of this model followed by **3 layers of 32 dense hidden units** with a **‘selu’ activation function**. Finally, the last softmax layer was added with three outputs as we have three classes. The model is then compiled with optimiser as **Nadam** and loss function as **sparse categorical cross-entropy**. It was then trained with a **0.001 learning rate** with **batch size 32** for **5 epochs**. We follow this method of training in the first stage because all the lower layers have been trained to detect low-level features but the topmost dense layers have been newly added and the weights are newly initialised. By allowing the model to train with the low layers locked the dense network will learn to detect patterns in the low-level features which have been already learnt when the pre-trained model was trained on the ImageNet dataset. It also has the added advantage of greatly reducing training time and resources as we don’t have to update the weights of the lower layers which would otherwise take hours to train. The training metrics are shown in the plot. There is a significant drop in the training and validation loss and a rise in the training and validation accuracy and no signs of overfitting as the difference between the validation and trining accuracy is very small at the end of training. The **training and validation accuracy** at the end of this stage was **0.9031** and **0.8913** respectively while the t**raining and validation loss** were clocked at **0.2444** and **0.249** respectively.



Metrics vs epochs for stage 1 (Figure 7)

Stage 2

For stage 2 all the layers of the model are unlocked and allowed to train. Although the weights in the lower layers are already at the optimal values they need to be trained for a few epochs on our current dataset for fine-tuning. This stage will take much more time per epoch but the training will require only a few epochs to reach the optimal state. From the plot, we see that the training ends in **5 epochs** as the early stopping callback kicks in and ends the training as it detects no further improvement in the validation accuracy. From curves, we see that, although not much, there is a slight improvement in the accuracy and reduction in the training and validation loss from where the model had left off in stage 1. The **training and validation accuracy** at the end of this stage was **0.9429** and **0.9058** respectively while the **training and validation loss** were clocked at **0.1660** and **0.2588** respectively.



Metrics vs. Epochs for stage 2 (Figure 8)

Model Evaluation

This fully trained model is then evaluated on the never before seen test data and the summary of all the metrics are displayed in the following table:

Dataset	Sparse categorical accuracy	Precision	Recall	F-score
Training data	0.9429	0.9438	0.9410	0.9412
Validation Data	0.9058	0.9312	0.9313	0.9307
Test Data	0.9270	0.9083	0.9155	0.9116

We see that our model performs very well on the test data as well with all metrics above 90% values. Comparing these results to the previous outcomes obtained by evaluating the pre-trained models on the dataset directly without customising it, we see that there is a significant amount of improvement in terms of classification accuracy which rose steeply from about **33%** to just over **90%** for the new model. The precision and recall and f-score for our final model are also greater than **90%**. **But among all these metrics we focus our attention on the recall**. Our task in this project is to assist Radiologists in detecting COVID19 by analysing the Chest X-rays of patients in an attempt to detect the disease early so that it is possible to advise the patient to isolate it so as to prevent the spread of the virus to other people. The reason for this is that even if we detect a false positive it's okay as even if the patient doesn't not have COVID19, we could just advise the patient to isolate at home greatly reducing the risk of spreading the virus which would otherwise be high if the model detects false negatives (in case of a model with low recall) as the sick patient would be allowed to go free without isolation. **Our final model which has a recall of 91.55% on the test data proves that it is the best model for the task.**

Conclusion

Our project demonstrated the power of Computer vision in the field of Radiology and how it could assist medical practitioners in making faster and more accurate diagnoses. We also see the effectiveness of the concept of knowledge transfer in the use of transfer learning. By taking a model trained on a completely different dataset for a similar problem we saw that training such a model with a few tweaks to its top layers and hardly any training epochs can produce a model with high classification accuracy. We also see its ability to greatly reduce training time and computational resource consumption as we do not need to train such a large model from scratch. Only the new custom top layers require most of the training while the rest of the layers require very little to no training thereby greatly reducing the need for expensive equipment like state of the art GPUs. Although our model had a great performance in terms of accuracy, it could have been better. The reason for it stagnating at the current value is most likely due to the bias induced because of sub-sampling the dataset in an attempt to balance it. This process has most likely dropped some of the important data which could have improved the features learnt. Data augmentation assists in reducing this problem but it does not eliminate it. Image resolution also plays an important role in feature detection, but since our models are configured to take in only images of size 224x224 a lot of the information about the spatial features is lost which could be another reason for its limit. The best option to improve performance would be to get more training data consisting of high-resolution images and modify the pre-trained models to take in these images of high resolution. In addition to this, the use of a deeper and newer pre-trained model (ex. SE net) could be explored to see if improves the metrics.

Challenges of the project

The main challenge of the project was an un-balanced dataset as there were not enough positive covid images. This left us with no choice but to sub-sample the dataset of the other classes at random and ignore potentially important images. The next major task was training the models with the lack of sufficient computational power since we were using normal computers and processors, it was time-consuming due to the high level of computation and resources required.

Task allocation

Aditya Naik: Data cleaning, Data Augmentation, Hidden layers and activation function tuning, training and validation plots, Final model implementation and testing and analysis, Report writing

Reza Fazli: ResNet50 loading and testing, Hidden units and optimiser tuning and analysis, Report writing

Yuzhe Qiu: GoogleNet loading and testing, Initial learning rate and epochs tuning and analysis, Report writing

References

[1] https://www.webmd.com/lung/covid-and-pneumonia#1

[2]https://medium.com/analytics-vidhya/handling-imbalanced-dataset-in-image-classification-dc6f1e13aeee

[3] ResearchGate. (n.d.). *(PDF) Handling imbalanced datasets: A review*. [online] Available at: https://www.researchgate.net/publication/228084509_Handling_imbalanced_datasets_A_review.

[4] Krizhevsky, A., Sutskever, I. and Hinton, G.E. (2012). *ImageNet Classification with Deep Convolutional Neural Networks*. [online] Neural Information Processing Systems. Available at: https://homl.info/80 [Accessed 10 May 2022].

[5] Aurelion Geron, E 2020," Hands-on machine learning with Scikit-learn, Keras and TensorFlow", O'Reilly Media, Inc

[6] Masters, D. and Luschi, C. (2018). Revisiting Small Batch Training for Deep Neural Networks. *arXiv:1804.07612 [cs, stat]*. [online] Available at: https://homl.info/smallbatch [Accessed 10 May 2022].

[7] Zhuang, F., Qi, Z., Duan, K., Xi, D., Zhu, Y., Zhu, H., Xiong, H. and He, Q. (2019). A Comprehensive Survey on Transfer Learning. *arXiv:1911.02685 [cs, stat]*. [online] Available at: https://arxiv.org/abs/1911.02685.

[8] He, K., Zhang, X., Ren, S. and Sun, J. (2015). Deep Residual Learning for Image Recognition. *arXiv:1512.03385 [cs]*. [online] Available at: https://homl.info/82 [Accessed 10 May 2022].

[9] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V. and Rabinovich, A. (2015). *Going Deeper With Convolutions*. [online] www.cv-foundation.org. Available at: https://homl.info/81 [Accessed 10 May 2022].

[10] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J. and Wojna, Z. (2015). *Rethinking the Inception Architecture for Computer Vision*. [online] arXiv.org. Available at: https://arxiv.org/abs/1512.00567.

[11] He, K., Zhang, X., Ren, S. and Sun, J. (2015). *Deep Residual Learning for Image Recognition*. [online] arXiv.org. Available at: https://arxiv.org/abs/1512.03385.

[12]Senior, A., Heigold, G., Marc', A., Ranzato, K. and Yang (n.d.). AN EMPIRICAL STUDY OF LEARNING RATES IN DEEP NEURAL NETWORKS FOR SPEECH RECOGNITION. [online] Available at: https://homl.info/63.

[13]https://medium.com/@kshitijkhurana3010/activation-functions-in-neural-networks-ed88c56b611b