Exploring Transfer Learning and Fine Tuning

Declan Stockdale

Project outline

This project explores the use of three pretrained models available in the TensorFlow package for transfer learning. The models used are the Inception ResNet v2, Mobilenet v3 (small) and NASNet (mobile). All three models were initially trained on the ImageNet dataset.

Each model will be slightly modified to remove the top layer used for prediction in the ImageNet database and substituted for a new top layer for predicting classes in the Food 101 dataset. The three models will then be trained to assess their performance on the Food 101 dataset with the most accurate performing model undergoing additional fine-tuning for further accuracy improvements.

Introduction

Transfer learning is the process of utilizing a pretrained model which has the ability the detect features are modify the weights and bias of the output to predict new classes of object it may not have seen before. This project initially explores three models: Inception ResNet v2, Mobilenet v3 (small) and NASNet (mobile) for their ability to categories various food items belonging to 101 different classes. Each model was initially trained on the ImageNet database which includes a range of objects from clocks to cars and include some food items. As there is a similarity between the ImageNet and the Food 101 datasets, we should be able to apply transfer learning with minimal modifications are achieve reasonable results.

How does transfer Learning work?

The reason transfer learning works well out of the box is that the early layers in the model have already been trained to detect features such as edges, curves and shapes. They may also be able to determine relative distances between features which will be useful in the Food 101 dataset which would be useful for differentiating between a slice of pizza and a slice of cake which both have similar sector shapes but differ vastly in height.

The models were trained on the ImageNet database and through the training process, were able to identify features that were later fed into the predictive layers for object classification. For our purposes, we are going to keep the earlier feature detection layers and substitute the predictive layers with untrained densely connected layers. We will then input our images into the model and retrain the model with the new predictive layer to predict our new image classes from the new dataset.

Dataset

The Food 101 dataset consists of 1000 images for each of the 101 different food classes from images taken from the food spotting website (http://www.foodspotting.com/). The dataset is available to download from this link [1].

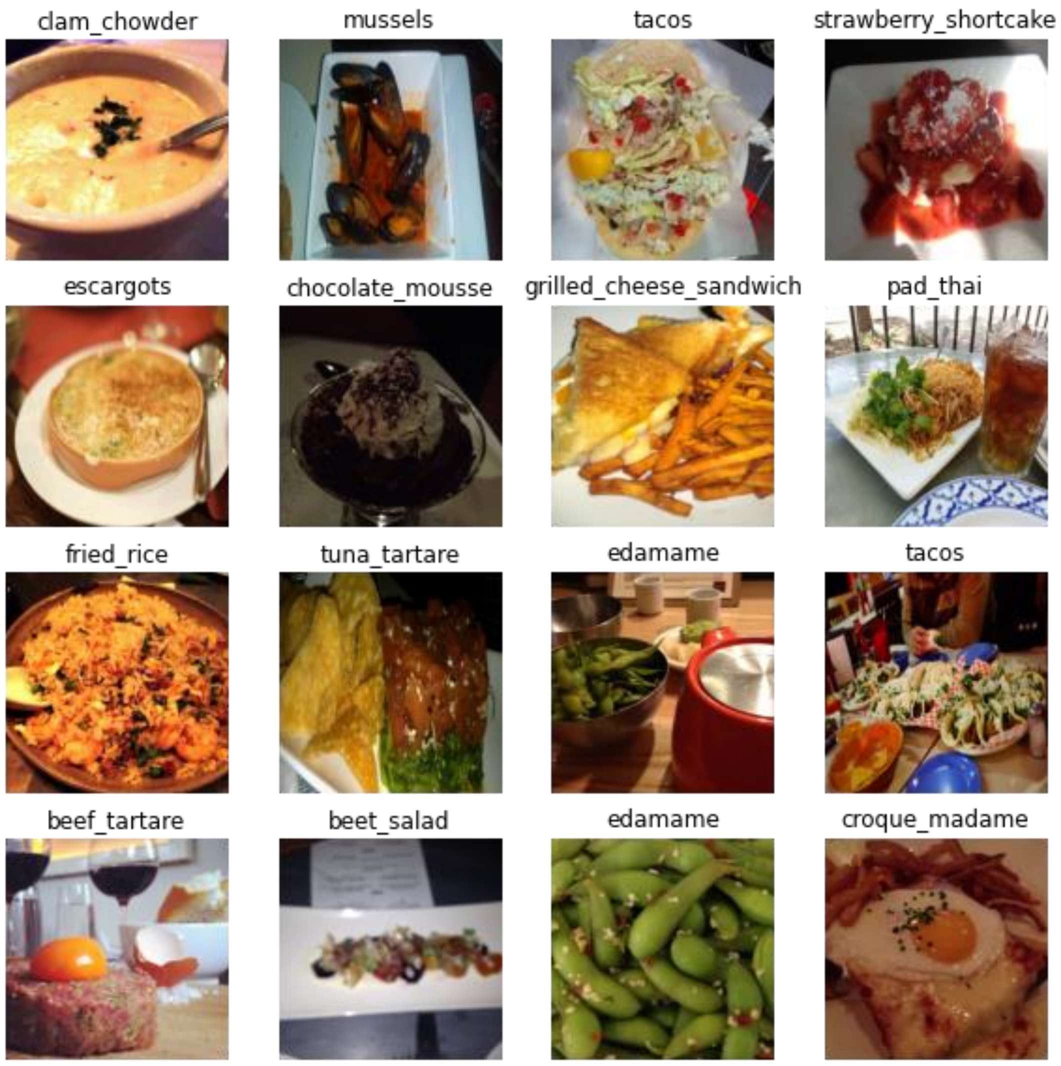


Figure 1. Example of 16 images from the various classes in the Food 101 dataset

From figure 1, it’s quite clear the images are noisy and are not framed particularly well in some cases. This is useful as it represents potential real-world images where the items may not be centered, or the lighting conditions are not optimal. The presentation of the food item can also vary widely within the same class either due to cooking process, plating or various additional sides such as in the various examples of steak in figure 2.

Figure 2. Examples of steak images

History of Deep Learning

Deep learning has experience significant changes with numerous breakthroughs across a wide variety of research areas including but not limited to speech recognition, natural language and many more. The discussed history will focus solely on developments and breakthroughs associated within the computer vision field most notably in image classification.

Deep Learning owes its start to ideas developed in the 1950s and 60s with the creation of the perceptron by Frank Rosenblatt taking inspiration from neurons in the human brain [2]. A perceptron takes various inputs and provides an output and would look familiar to single neurons in modern deep learning networks (figure 3). Due to its simplicity, it was only capable of linear separation seen in figure 4. It was quickly discovered in 1965 by Alexey Ivakhnenko that stacking layers of perceptron's significantly increased the computational power allowing for nonlinear separation and complex modelling [3].

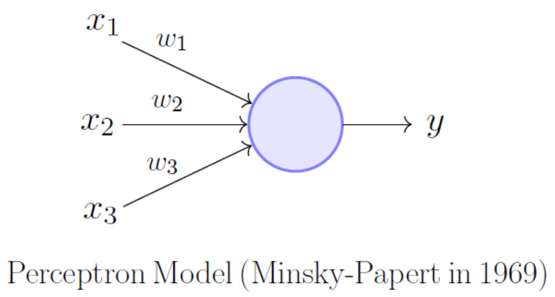


Figure 3. Mathematical model of a perceptron. Image source Neural Networks and Mathematical Models Examples by Ajitesh Kumar

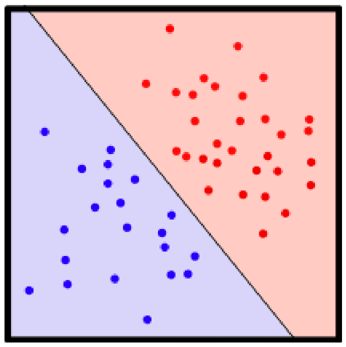


Figure 4. Example of linearly separate data

Backpropagation which is one of the cornerstones of modern deep learning algorithms was first described in 1960 by Henry Kelley utilizing dynamic programming [4]. Two years later, Stuart Dreyfus realized that backpropagation could instead by implemented through the chain rule, increasing the efficiency of the process [5]. Backpropagation was only written in computer code in 1970 by Seppo Linnainmaa and he didn't recognize its potential application to neural networks at the time [6]. It was later in 1974 that Paul Werbos implemented Linnainmaa work to his own research in neural networks [7]. Backpropagation was further pushed into the mainstream research community by continued work by Werbos and later improved upon by Geoffrey Hinton et al [8].

One of the big breakthroughs in computer vision was achieved by Kunihiko Fukushima who developed the neocognitron in 1980 [9]. This was the first algorithm to apply optical character recognition to Japanese characters. It implemented ideas developed by neurophysicists, Hubel and Weisel who developed mathematical descriptions of how cats' eyes respond to light earlier in 1964 (figure 5) [10].

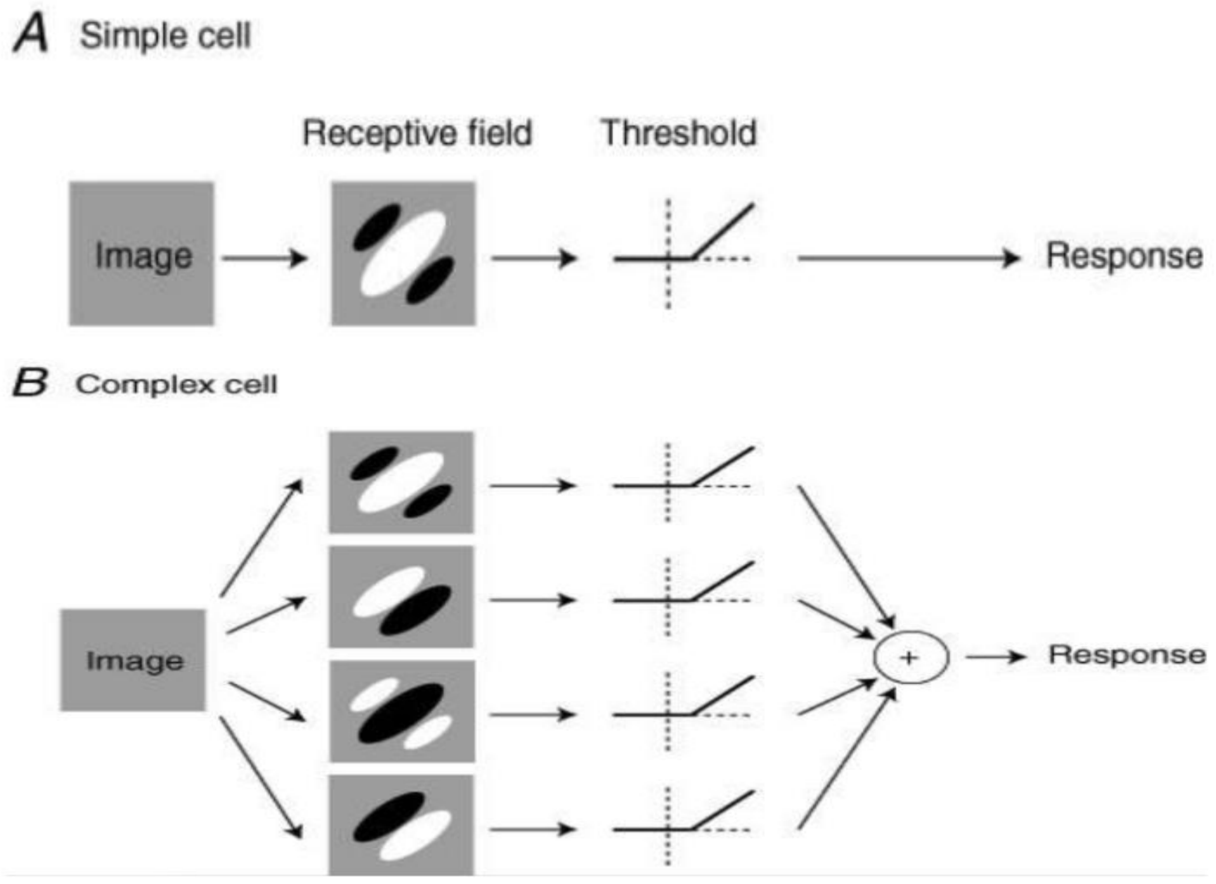


Figure 5. Hubel and Weisel's description of cats eye. Image source What simple and complex cells compute by Matteo Carandini

Another breakthrough in computer vision was achieved by Yann LeCun in 1989 who performed work similar to Fukusima but implemented Deep Convolution Neural Networks using backpropagation to recognize handwritten numbers [11]. All the learning was performed automatically using computers with no manual tuning or initial parameter tuning and was an enormous breakthrough at the time.

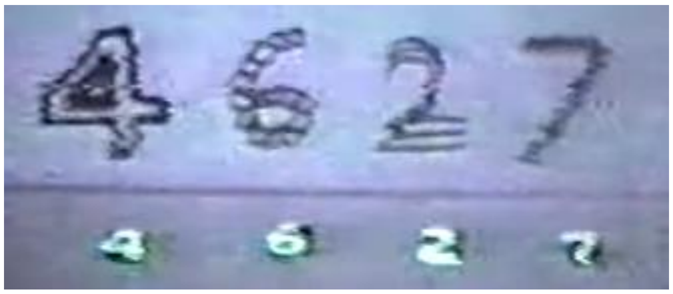


Figure 6. Example of LeCun's handwritten digit recognizer. Screenshot taken from https://www.youtube.com/watch?v=FwFduRA_L6Q

Another important building block of CNNS, max and average pooling was developed in 1990 by Yamaguchi for spoken word recognition [12]. This greatly reduced the dimensionality of the network and reduced training time considerably. It would later be used for computer vision tasks.

The vanishing gradient problem was first identified by Sepp Hochreiter in 1991 where the gradient of the activation function was so small that the network effectively stops training [13]. Numerous efforts to overcome this issue continued throughout the years ahead. Solutions offered included training layers one by one which was tedious and slow. One of the better solutions was proposed by Yoshua Bengio which was to use a ReLu activation function which generally doesn't suffer due to saturating in one direction. Years later residual neural networks were also developed where layers are skipped allowing gradient information to pass around potential bottlenecks [14]. The flip side is the exploding gradient which is where the gradients of the model are so large that they cause the model to become unstable. This can be solved by setting maximum values for the gradient to restrict it to reasonable values.

In 2008 Andrew Ng's research group strongly promoted the use of GPUs for training as they were able to drastically improve training time as they were designed to compute matrices and tensors which were commonplace in computer graphics]. The adoption and investment by GPU companies most notably Nvidia with custom hardware and tailored algorithms have led to an enormous boon in the size of models and the size of training data [15].

A benchmark dataset called ImageNet was first proposed in 2009 [16]. This dataset consisted of approximately 20,000 images in 20 classes and was curated to allow competition between research groups on a level playing field. It has since grown to over 14 million images with a thousand different classes.

One of the most significant contributions to deep learning was the work by Alex Krizhevsky et al with their development of the image recognition model AlexNet in 2012 [17]. It was one of the first GPU trained deep learning models that handily won the 2012 ImageNet competition. It was the first CNN to use ReLu activation function rather than the sigmoid function greatly improving training time. It used 5 convolutional layers and the final 3 layers were densely connected layers. It also applied max pooling which had recently been shown to greatly improve performance on computer vision tasks a few months earlier by Dan Ciresan [18].

Its success generated enormous interest in deep learning and has inspired numerous deep learning models and computational methodologies.

Historical summary

1958 – Perceptron algorithm proposed by RosenBlatt

1960 – Backpropagation through dynamic programming by Kelley

1962 – Backpropagation using chain rule proposed by Dreyfus

1964 – Hubel and Wiesel mathematically describe the visual system of cats visual systems

1965 – Ivakhnenko creates multi layer perceptron

1970 – Linnainmaa writes first computer code for back propagation

1974 – Werbos uses Linnainmaa's code to apply backpropagation to a neural network

1980 – Fukushima develops the neocognitron to identify Japanese characters

1989 – LeCun develops model that can identify handwritten numbers using CNNs and gradient descent

1990 – Max and average pooling first described by Yamaguchi

1991 – First mention of the vanishing gradient problem by Hochreiter

2010 – Nair, Bengio and Hinton are credited with developing the ReLu function to limit vanishing gradient descent

2008 – Ng advocates for increased uptake in GPU for faster training times

2012 - AlexNet wins the ImageNet competition

2014 – First Inception model published in paper "Going deeper with convolutions"

2015 – First residual neural network proposed to counteract vanishing gradient descent in paper "Deep Residual Learning for Image Recognition"

2015 – Inception v2 and v3 published in paper "Rethinking the Inception Architecture for Computer Vision"

2016 – Inception ResNet v1 and v2 first published in paper "Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning

2017- First version of MobileNet published

2017 – NASNet published in paper "Learning Transferable Architectures for Scalable Image Recognition"

2018 – MobileNet v2 published in paper "MobileNetV2: Inverted Residuals and Linear Bottlenecks"

2019 – MobileNet v3 published in paper "Searching for MobileNetV3"

Models used

Inception Resnet v2 and is predecessors

The first version of inception was published in 2014 and its name is a tongue in cheek reference to the movie Inception [19]. It utilizes CNNs and was designed by researchers at Google. It was one of the first architectures to go wider rather than deeper. They achieved this by applying multiple convolutions to the same layer, concatenate the results and finally feeding the result into the next layer. This was done to overcome issues where the background of the targeted object varied greatly so did the size of the target within the image such as in figure 7.

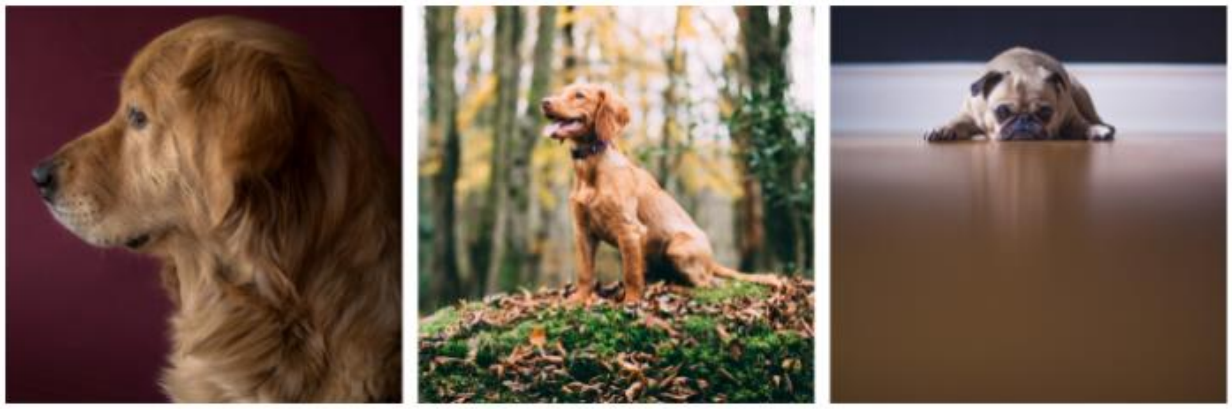


Figure 7. Examples of the same target in different environments.

Source https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202

As the model doesn't initially know where the target is or how large it is in the image, it doesn't know if its better to apply a large convolution for global information or a small convolution to get a smaller part of the image. By applying multiple convolutions, it can get all the information and decide what is important.

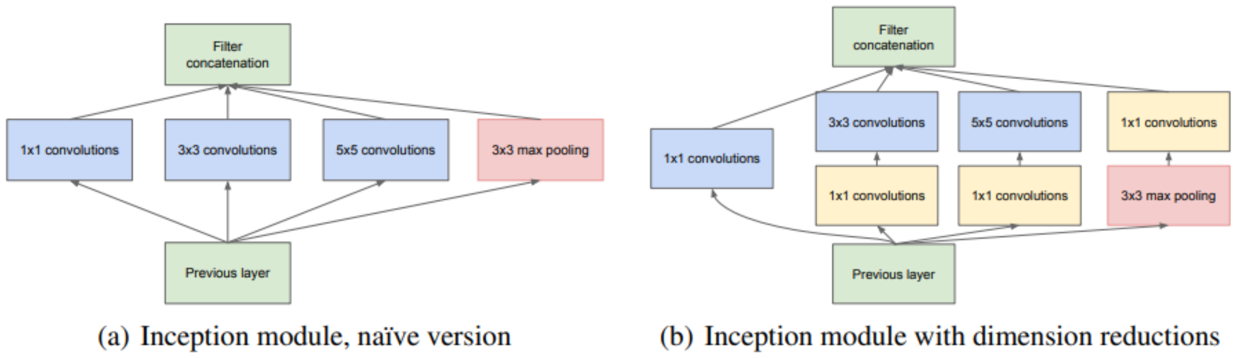


Figure 8. Examples of a) Inception v1 and b) inception v2. The difference is the additional yellow boxes which simplify the computation. Image source https://arxiv.org/pdf/1409.4842v1.pdf

The second and third versions appeared in 2015 following the same idea as in version 1 but reduced the computational costs through slight modifications to the architecture and size of convolutions used, most notably swapping a 5x5 for two 3x3 which is cheaper (3x3 + 3x3 = 18 vs 5x5 = 25 weights) as can be seen in figure 8 above [20]. A 4th version was also developed which streamlined the architecture removing elements which didn't contribute any significance to the final output [21]. It was presented in the same

The success of the Resnet model led to a hybrid between the two architectures resulting the in the foundation for the inception Resnet models used today. The Resnet architecture allows some layers to skip to layers further ahead rather than feed into the next sequential layer as can be seen in figure 9. There are two versions v1 (based off inception v3) and v2 (based off inception v4). Inception ResNet v2 is 164 layers deep and is one of the most successful models in both terms of accuracy and computational cost [22,23]. As of writing this it sits at 364 in the ImageNet ranking with a top 1% of 80.1% [24]. The highest score achieved has a score of 90.94%.

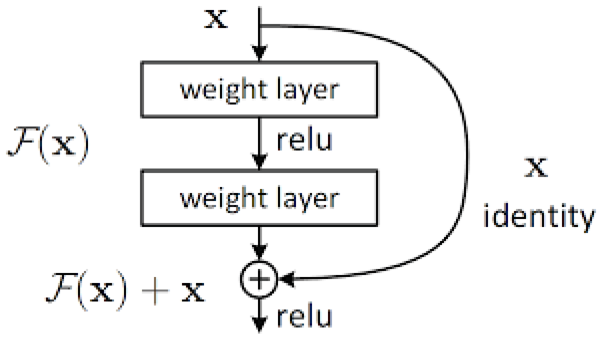


Figure 9. Examples of residual connections which allow outputs from earlier layers to skip steps and feed into a layer further in the network. Image source https://towardsdatascience.com/introduction-to-resnets-c0a830a288a4

MobileNet v3 Small and its predecessors

MobileNet was first published in 2017 by researchers at Google and is that is designed to run on embedded devices such as mobile phones [25]. It focused on light weight deep neural networks based on using a streamlined architecture and as such couldn't use widely used models as they wouldn't run with the available resources. It utilized depth wise separable convolutions followed by pointwise convolutions. In figure 10 below, the input is a tensor with three colour channels. The depth wise convolutions take a channel and performs a convolution reducing the dimensions shown by the red arrows. Afterwards, the newly reduced layers are then restacked in the pointwise convolution step creating the final row in the figure.

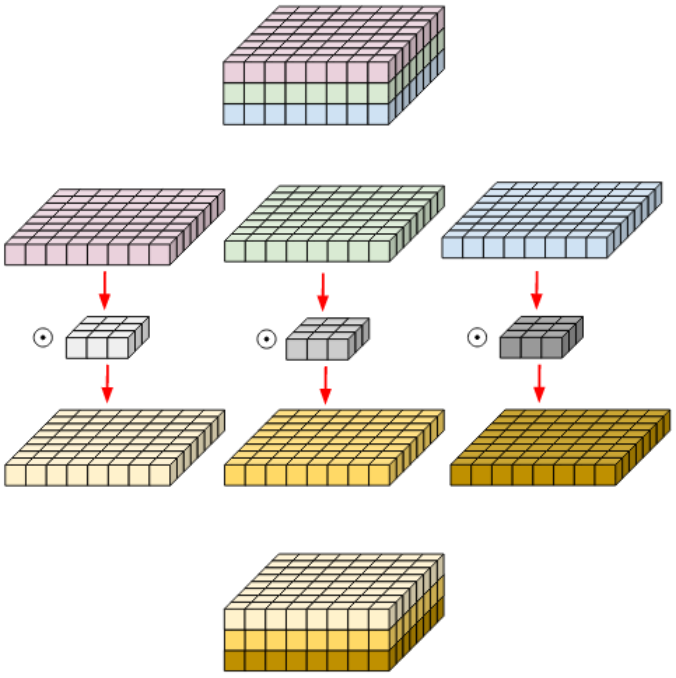


Figure 10. Depth wise and pointwise convolutions used in MobileNet v1. Image source https://medium.com/@zurister/depth-wise-convolution-and-depth-wise-separable-convolution-37346565d4ec

Version 2 came out later updating the architecture to include residual layers inspired by ResNet [26]. It also introduced the concept of an inverted residual structure where a bottleneck is employed which contracts, expands then feeds that into the residual layers. This can be seen in the right-side image in the figure 11. The 1x1 convolution reduces the channel size which is then expanded by the 3x3 convolution and then recompressed.

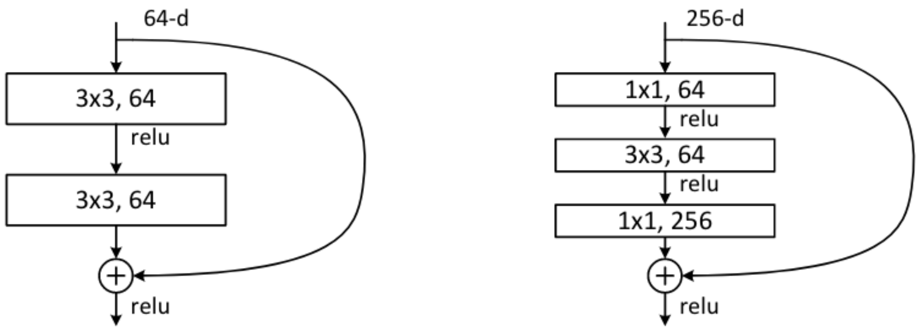


Figure 11. Left image, standard residual connection. Right image, example of inverted residual where image is compressed by 1x1 then expanded by 3x3 convolutions

Version 3 implemented improved algorithms to take advantage of increased computing power and has smart resource allocations to get the most out of the embedded device. It does this in two ways, firstly it utilizes AutoML (Automatic machine learning) to search for the best neural network architecture. The model then adapts the chosen architecture that trims channels that are not contributing (figure 12) [17]. It also implemented the swish and h-swish activation functions which are computationally cheaper than the previously used sigmoid [28].

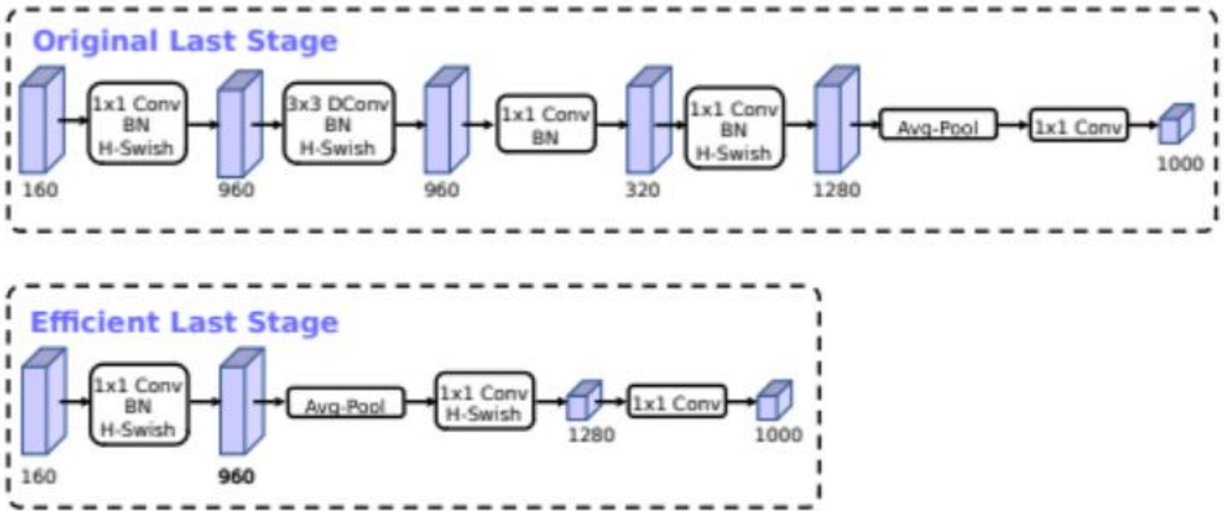


Figure 12. Example of model pruning by AutoML process removing unnecessary elements in initial model (top) to create improved model (bottom)

Another significant difference is the additional squeeze and excite block implanted in the third version. A brief explanation is that this block squeezes the input of each channel into a single value using average pooling which is subsequently fed into a fully connected layer. The excitement comes from the weight of the output of a side channel [29]. This allows the network to attain information across all channels.

There are two versions of MobileNet v3, a Large and a Small version. Only the small version if investigated in this report.

NASNet

NASNet which stands for Neural Architecture Search Network was first published in 2018 once again by researchers at Google [30]. It uses ideas introduced by the Mobilenet v3 in that it uses reinforcement learning to find the optimum CNN architecture. It was one of the first algorithms to use machine learning for architecture engineering of a model and at the time of publication, “the accuracy of the resulting model exceeds all human-designed models – ranging from models designed for mobile applications to computationally-heavy” [31]. The idea was to create a model that can perform well on a smaller dataset then build upon it for a large dataset. The authors chose CiFAR-10 dataset which consists of 60000 images across 10 classes of vehicle. The model was trained on the smaller dataset using reinforcement learning to determine the optimum structure of the normal (orange) and reduction (gray) cells as can be seen in figure 13. This took approximately 2000 working GPU hours. They then took the best cell configurations and stacked them. The resulting model architecture was retrained on the ImageNet dataset.

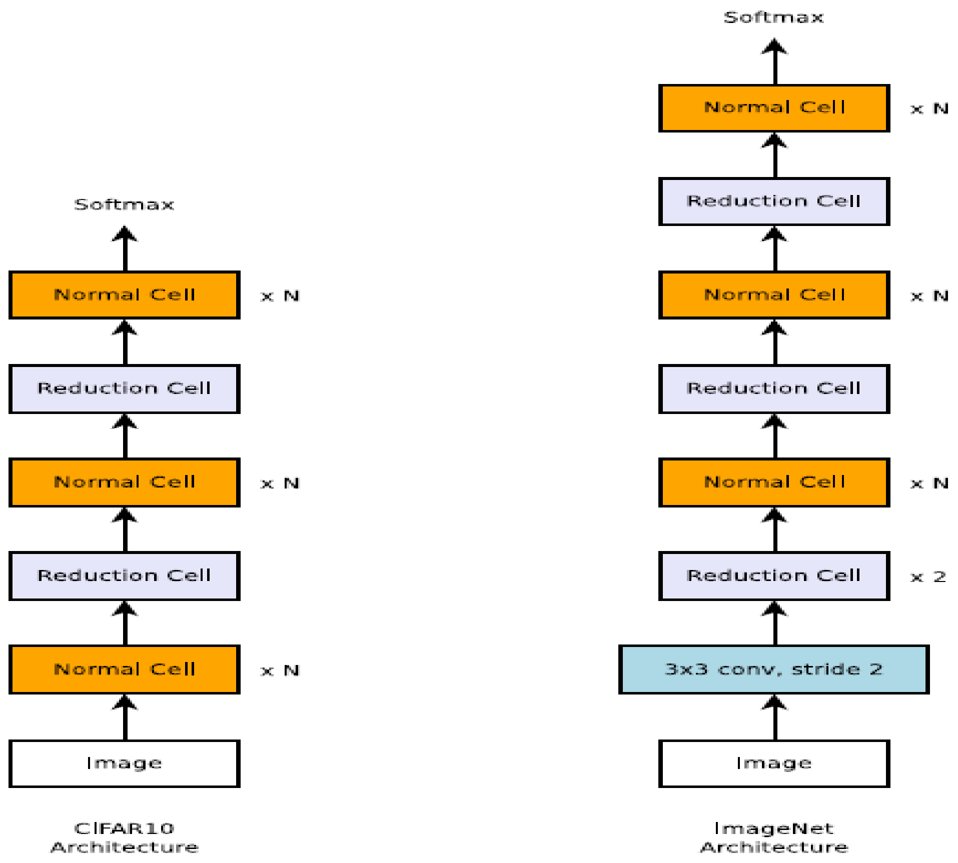


Figure 13. NASNetbasic structure. Composed of stacked normal cells and reduction cells. Left is model used on CiFAR dataset. Right is model used for ImageNet dataset. Image source reference 30

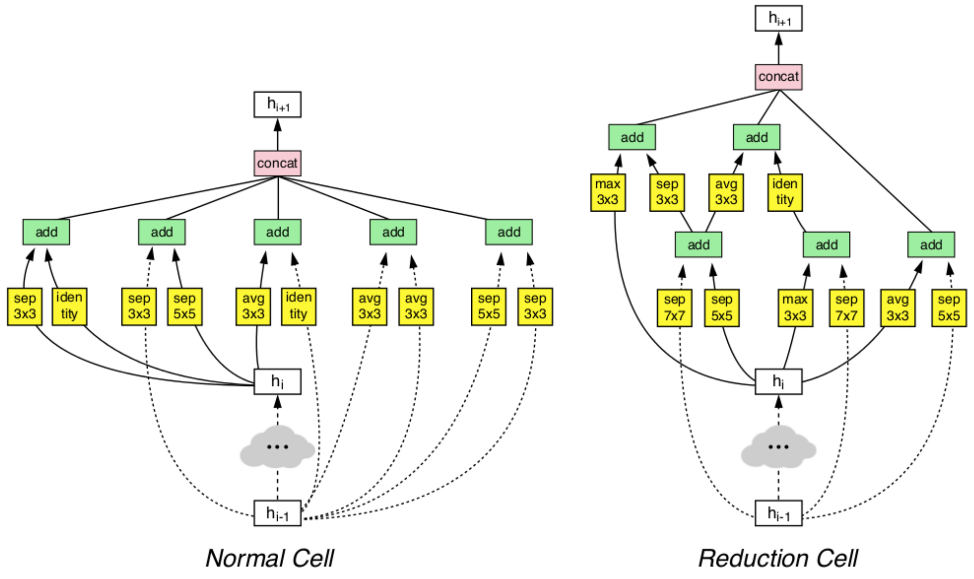
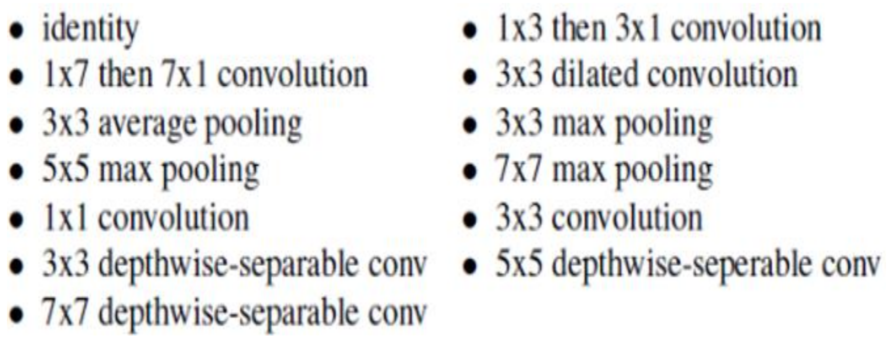


Figure 14. Example of what the normal cell and reduction cell attained through reinforcement learning process. Image source reference 30

The resulting normal cells and reduction cells are attained through reinforcement learning an example can be seen in figure 14. It can observed that residual connections are also present in the network. The possible operations of the normal and reduction cells come from figure 15. The way the operations are chosen is to select two options from the list below (pairs of yellow boxes) along with some way to combine the outputs (green box) followed by concatenating the results. Figure 14 is for B = 5 where there are 5 pairs of yellow boxes. Additional models with different B's exists such as B = 7. It also uses a variant of dropout called Scheduled drop out to drop certain neurons more frequently as the training process progresses.



- identity
- 1x7 then 7x1 convolution
- 3x3 average pooling
- 5x5 max pooling
- 1x1 convolution
- 3x3 depthwise-separable conv
- 7x7 depthwise-separable conv
- 1x3 then 3x1 convolution
- 3x3 dilated convolution
- 3x3 max pooling
- 7x7 max pooling
- 3x3 convolution
- 5x5 depthwise-seperable conv

Figure 15. Potential operations the reinforcement learning algorithm could perform to create the normal and reduction cells.

Latest advancements and potential future of image classification

EfficientNet

EfficientNet was a model proposed by researchers at Google in 2019 that made waves when in achieved state of the art results on ImageNet using significantly less parameters compared to other high accuracy models [32]. The main contribution of the model was to systematically explore how models performed with varying width, depth and input image resolution. They employed AutoML to generate a baseline model on a small dataset. They used a compound scaling method to scale the dimensions (width, depth and resolution) and found the models performed best when the dimension increased in scale together. It also uses the inverted residual network seen in MobileNet v2 (figure 11). An updated version called EfficientNet v2 was released in 2021 and again achieved state of the art performance using an updated parameter searching algorithm [33].

Transformers

Transformers have taken both the natural language processing and the computer vision fields by storm where they were first published in the "attention is all you need" paper in 2017 [34]. A quick look at the current state of the art ImageNet leaderboard shows that most models use transformers in some shape or form [24]. The advantages of transformers over CNN are that while CNNs are fixed with learnt weights, a transformer can learn relationships between pixels in both local space similar to CNN but also across distant pixels. It performs this by breaking the input image into a grid and assigns the pixels in each grid a unique token. Using the tokens, the similarities between tokens can be used to gain a deeper understanding of the interactions between not only neighboring pixels but pixels at significant distances across the image. Transformers also use less memory are quicker to train. As transformers are a relatively new concept, there is a range of research exploring various aspects of the learning process and architecture engineering in the field.

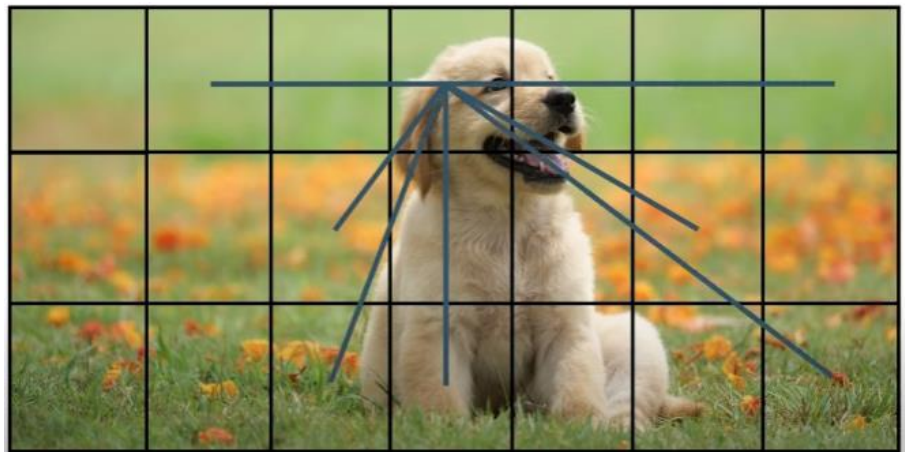


Figure 16. Example of how a transformer would separate an image. The lines are representative of comparing the pixel at the line's intersection with other pixels in other grid squares

Black box investigation and model interpretability

There's been a lot of focus on understanding what happens in the hidden layers of black boxes trying to decode what the network is seeing and how its processing the image. There are quite a few methods to achieve this such as visualizing which neurons have the strongest signal for a given input image. By iterating over images, we can get an understanding of which neurons are associated with each object in an image [35].

In addition, we can generate heatmaps of which region in the input image is associated with the target image [36]. This can be helpful to determine if the model is learning what's intended. An example of this was described in the paper "Can Everyday AI be Ethical? Machine Learning Algorithm Fairness" where the authors tried to create a classifier for huskies' vs wolves [37]. The model only learned that images with grass in them are likely to be huskies as they were domestic animals running around parks or backyards whereas images with white background (snow) was a strong predictor of a picture of a wolf as snow is part of their habitat.

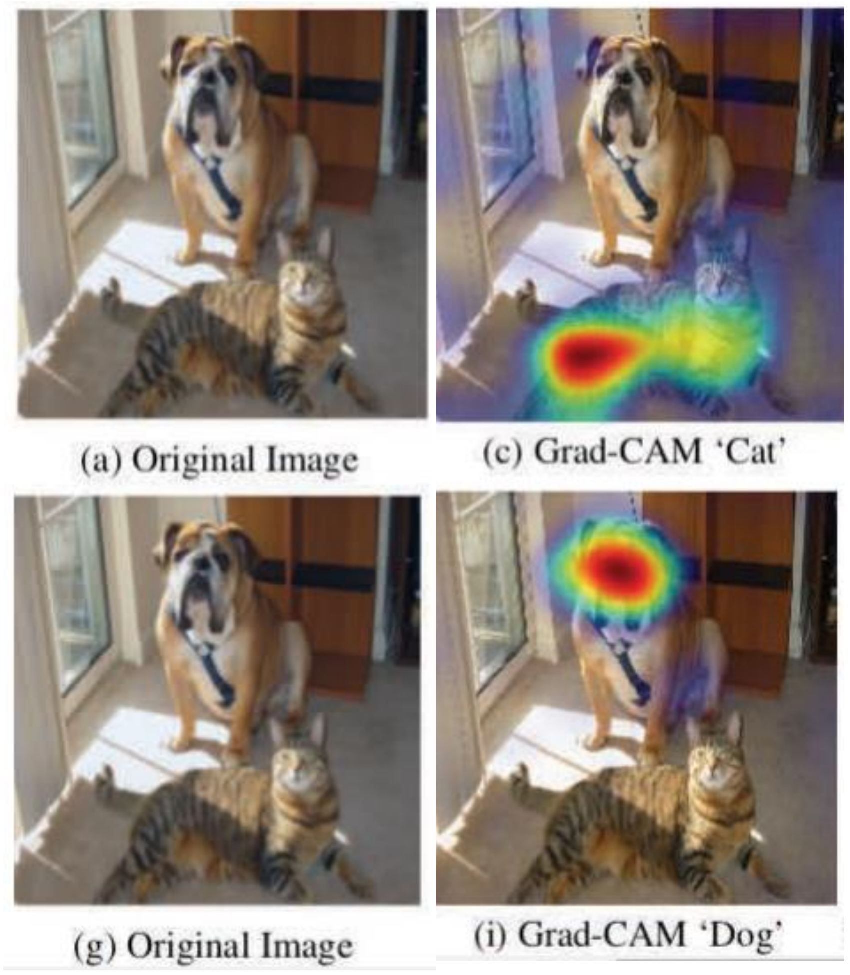


Figure 17. Example of heatmap showing which areas the network responds to for the dog object and the cat object. Image source "Adapted from Analytics of Explainers of Black Box Deep Neural Networks for Computer Vision: A survey

Advances in hardware and potential CPU acceleration

Both Nvidia and more recently AMD have invested enormous resources into their GPU research and development initially tailored for computer games but there has been shift towards creating dedicated hardware and software for the purposes of machine learning. At the moment, Nvidia is the clear market leader in the field due to early adoption and strong online community. AMD is improving their GPUs, competing with Nvidia in gaming performance but the lack of dedicated resources for machine learning has led to a significantly slower uptake [38]. AMD software is open source orientated and may gain more of a market share as both the hardware and software compete with Nvidia products.

GPUs are better out of the box at matrix multiplication compared to CPUs but researchers have developed an algorithm in 2021 that trains 15 times faster on a exotic intel CPU architecture [39]. The CPU isn't that common, and the authors did consult with the Intel engineers to maximize every bit of the CPUs performance. While this algorithm doesn't appear universally applicable to other CPU's it may prove to be useful in the future at reducing training time.

Architecture engineering using Auto ML - AutoML-Zero

The developments of Network Architecture Systems or NAS has already proven itself useful as its seen great results as a key component in MobileNet, NASNet and EfficientNet to name just a few examples. The search space for deep learning architectures is enormous with numerous exotic architectures and likely countless more awaiting discovery. It would be unreasonable to manually explore these search spaces and instead AutoML is employed to find the best outcome. Reinforcement learning is also a hotly studied topic and improvements in that field could lead to improved AutoML discoveries in architecture engineering.

An extreme example of AutoML is AutoML – Zero which starts completely from scratch [40]. It can only use basic matrix, vector and scalar operations to create a model. Unlike other AutoML where it's supplied lists of possible options such as the NASNet, it has to create everything on its own. As of reading about it, it has yet to discover convolutions, but the idea is that it will discover operations that humans haven't discovered and will ultimately expand the predictive capabilities of deep learning models.

Deep Learning on embedded devices

Advances in hardware of mobile devices have led to an increase in deep learning capabilities in both inference and training time. Advances in model architecture and heuristics have led to less resource intensive models which can adequately run-on embedded devices [41]. There are also compression algorithms that compress the model with a trade off in accuracy. Researchers were able to run a famous object identification and tracking algorithm, YOLO (you only look once) is runnable on a modern smart phone through model pruning and compression methods [42]. Improvements in hardware may lead to more advanced models being able to run on modern devices in the near future.

Model set up and comparison

Hardware used for training

Training was performed on a desktop computer running Microsoft Windows 10 equipped with a Nvidia 2080 Super GPU. Relevant libraries were installed such as cudatoolkit 11.0, cudnn 8.2 to enable TensorFlow 2.8.0 to take advantage of the faster computational speed of the GPU. This has numerous advantages compared to utilizing Google Colab in that training can be run for significantly longer without fear of disconnection and the GPU trains faster compared to initial trials on Colab allocated GPUs (1 epoch of inception ResNet took ~585 seconds whereas local GPU took ~320 seconds).

Train Validation Test split

The dataset is structured in a way that each class has its own folder consisting of 100 images. The train and test images are designated in separate text files one for training and another for testing.

The file path and image class labels were put into a pandas dataframe. The training data was shuffled to avoid the model seeing a large number of the same class in one go. Lastly a validation set was split off from the training set. The final image count is 60600, 15149, 25249 for train, validation and test sets.

Model setups

All various models were downloaded using the TensorFlow inbuilt application. When importing the models, the "include_top" value was set to False which removes the last predictive layer of the model. This was substituted with 2 x (256 densely connected layers both with ReLu activation function) and a final 101 densely connected layer using the SoftMax function to predict one of the 101 classes in the Food 101 dataset.

The inception ResNet v3 model input images were set to (299,299,3) whereas the MobileNet and NASNet were set to (224,224,3). These values are defined in the TensorFlow documentation [43-45]. The colour channel was left as 'rgb' as its likely that information is encoded in the colour e.g., the model would need colour to differentiate between chocolate cake and tiramisu and possibly soups.

The batch size was set to 32 and the number of epochs was set at 20. The optimizer used was Adam using the default value of 0.001. As the images went through a generator they were in vector format. The most relevant metric for the multiclass classification problem was categorical cross entropy and chosen metric was accuracy.

Early Stopping was the only call back used as this initial testing stage was to compare how the models performed more or less out of the box. The validation loss was monitored, and the early stopping patience was set to 5. No additional hyperparameter tuning was performed.

Inception Resnet v2 Results

The model ran for 18 epochs before stopping due to early stopping. It can be seen from the figures below that the validation loss plateaus from approximately the 10th epoch onwards. The validation accuracy appears to be increasing and we may be able to improve I if we ran it for additional epochs.

The accuracy on the test set is a disappointing 30.98%. Looking at a confusion matrix the diagonal values are quite good however there are some classes that commonly get mistaken for other classes clam chowder getting mistaken 88 times for lobster bisque or 113 misclassifications of spaghetti bolognaise for spaghetti carbonara.

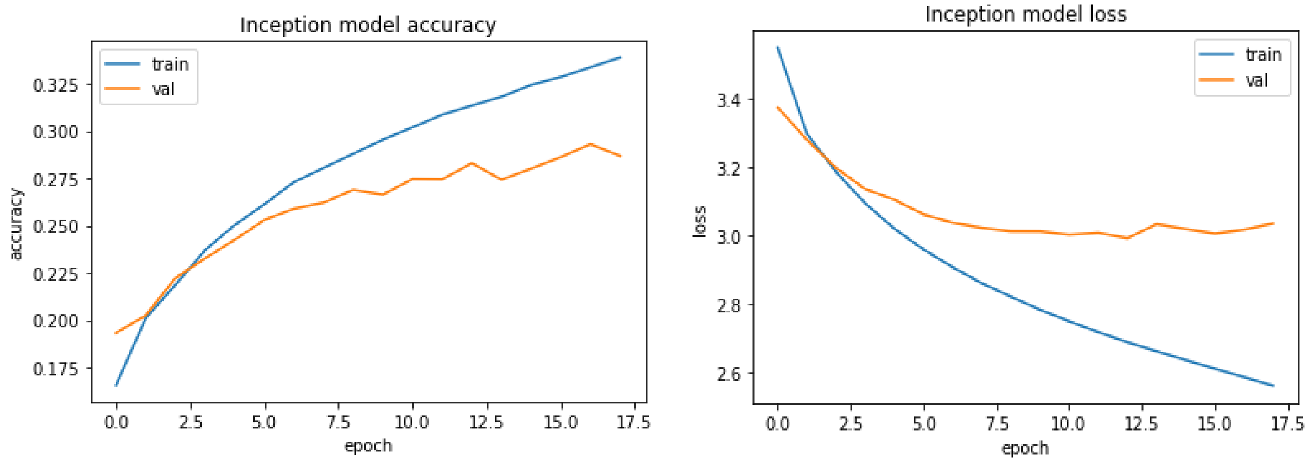


Figure 18. Left Inception Resnet v2 accuracy and loss for both training and validation sets

MobileNet v3 Results

This completed the 20 epochs and finished training significantly quicker than the inception model. The final accuracy was a much more impressive 48.60%. Additional epochs may have led to an additional accuracy improvements. Looking at the confusion matrix, there were only 27 misclassifications of spaghetti bolognaise for spaghetti carbonara. The largest misclassified category was 86 misclassifications of steak for prime rib which is unfair as it is a cut of beef.

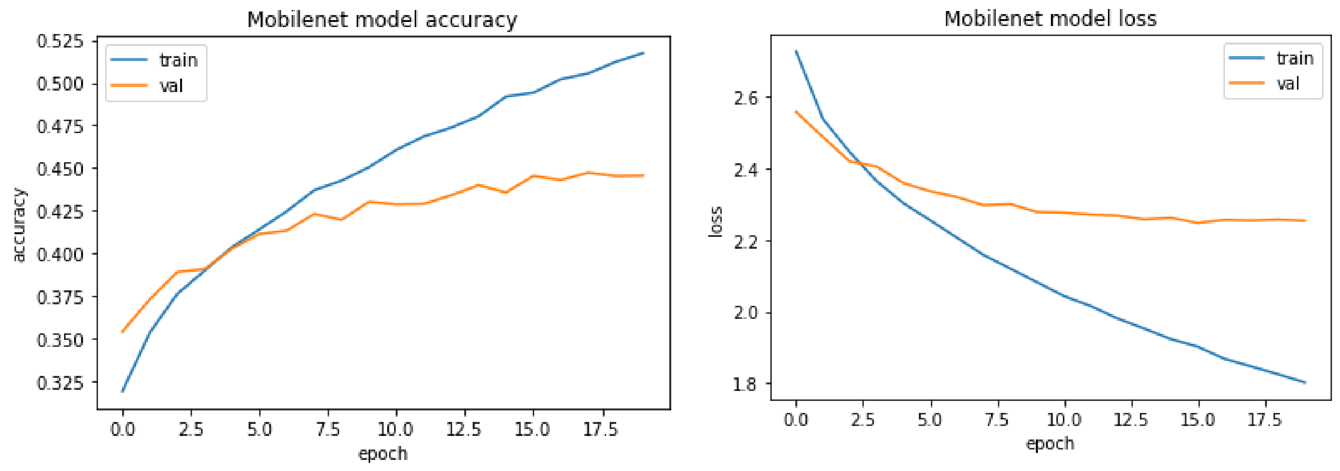


Figure 19. MobileNet v3 accuracy and loss for both training and validation sets

NASNet Results

NASNet only needed 13 epochs before early stopping occurred. It also showed a favorable accuracy score of 45.25%. From a quick glance of the confusion matrix, there didn't seem to be as many extreme misclassifications for any particular class but a handful of misclassifications across a range of classes. The most notable was 58 misclassified pulled pork sandwiches for hamburgers.

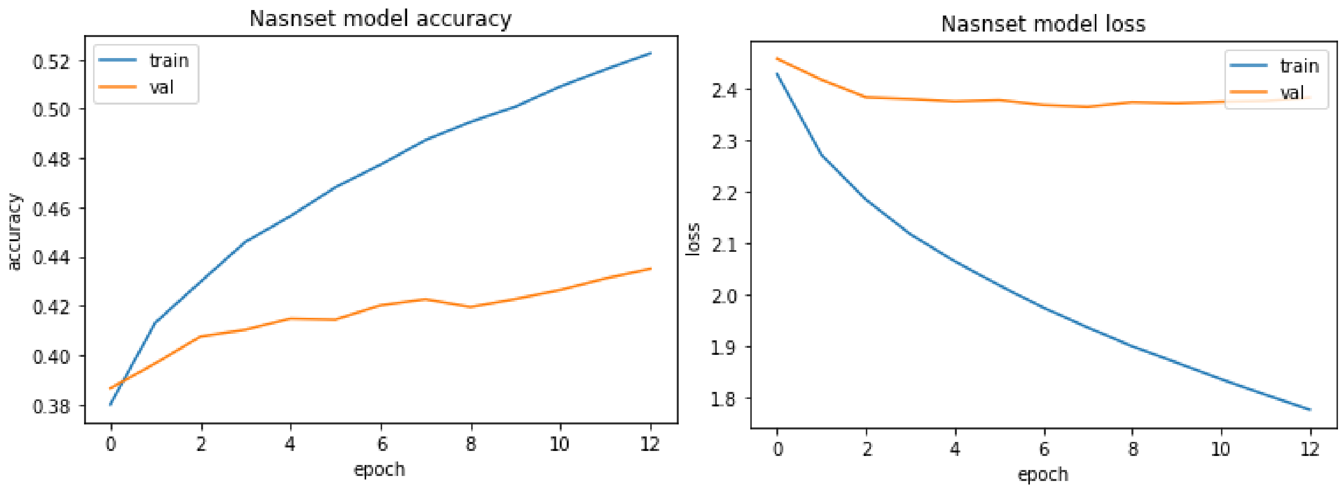


Figure 20. NASNet accuracy and loss for both training and validation sets

The two tables below show the fod categories that had the highest and lowest F1 scores for each model. The high scoring categoreis such as edamame are likely high because its unique within the datase whereas a poor performing category such as steak would get confused with fillet mignonand prime rib.

Best categories by F1 score		
Inception ResNet v2	MobileNet v3	NASNet
Hot dog	Edamame	Edamame
Edamame	Dumplings	Spaghetti carbonara
Guacamole	Guacamole	Guacamole
Spaghetti carbonara	Macarons	Dmplings
French fries	Pizza	hotdog

Table 1. Best F1 scores Common classes edamame, spaghetti carbonara, dumplings, pizza, guacamole where all common the best 10 categories

Worst categories by F1 score		
Inception ResNet v2	MobileNet v3	NASNet
Tuna tartare	Fellt mignon	Steak
Pork chop	Tuna tartare	Huevos rancheros
Apple pie	Crab cakes	Tuna tartare
Ceviche	Steak	Apple pie
Grilled cheese sandwich	Bread pudding	Grilled cheese sandwich

Table 2. Worst F1 scores Common classes tuna tartare, huevos rancheros, apple pie, filet mignon where all common the worst 10 categories

Overall results

Table 3 shows the accuracy, F1 score, recall and precision for each model. The metrics were calculated using the macro value as there is no class imbalance within the datasets. The highest values are in bold and its clear that the MobileNet v3 model outperformed the other two models. From figure 21, if we had set the epochs to a lower value, we may have ended up with NASNet as the best model however with extended epochs, MobileNet v3 was able to continue learning.

	Inception ResNet v2	MobileNet v3	NASNet
Accuracy	30.98	**48.60**	45.25
Macro avg F1 score	0.29	**0.48**	0.45
Macro avg Recall	0.31	**0.49**	0.45
Macro avg Precision	0.30	**0.49**	0.46

Table 3. Metrics of the final models' predictions on the test set

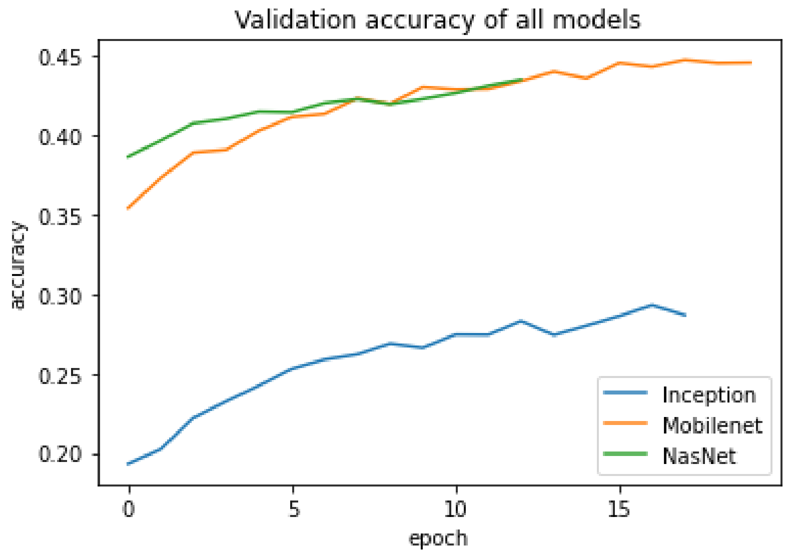


Figure 21. Plot of all models accuracy on the validation set

Examples of similar looking images for various classes

Figure 22 shows that there are some categories that appear identical. These are not limited to clam chowder and lobster bisque, pullerd pork sanwiches and hamburgers and spaghetti bolognaise and spaghetti carbonara. These would be difficult for a human to determine.

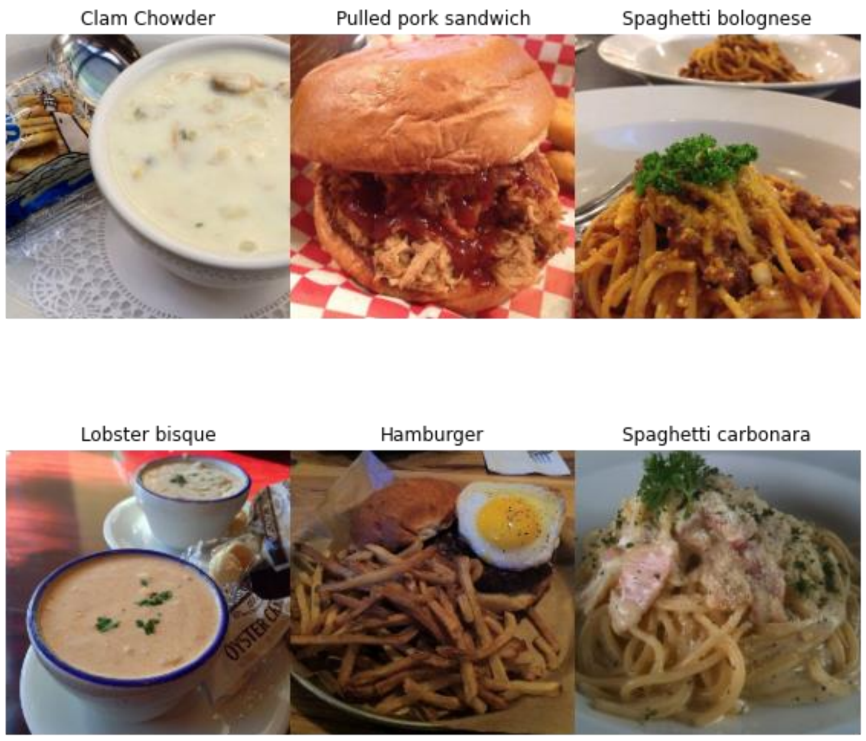


Figure 22. 3x3 plot of various categories that had a lot of misclassifications

MobileNet v3 Fine tuning on Food 101 dataset

The same process was used to generate the train/validation/test splits as described earlier. The MobileNet v3 was again implemented with the include_top set to False and the previously mentioened top layers of 2 x (256 densely connected layers both with ReLu activation function) and a final 101 densely connected layer using the SoftMax function for final prediction. As were fine tuning, the Adam learning rate was decreased to 0.0001.

To fine tune a model, layers are unfrozen from the base model to allow for additional learning to occur. In this report the number of layers that are unfrozen are recorded in table

The total number of parameters was 2,902,973 for all models where 2554968 came from the MobileNet v3 model and the remaining 348,005 come from the added top layers used for fine tuning.

The layers are unfrozen starting from the last layer of the base model working backwards e.g. for the model with an unfrozen layers value of 5 in table 4, only the last 5 layers are trainable not including the additional prediction layers that were added for predictions. The number of trainable parameters for each model are shown below.
It can be seen that more unfrozen layers results in more trainable parameters which would increase training time significantly. Additional callbacks were implemented. Early stopping was again applied but set to a patience of 10 with restore_best_weights set to true. Reduce_lr with a patience of 5, decrease factor of 0.2 and min_lr of 0.00000001. Finally model checkpoint was also implemented incase the training failed at any point.

Model (unfrozen layers)	Trainable	Non trainable
All	2,890,861	12,112
50	2,536,493	366,480
40	2,314,397	588,576
30	2,242,397	660,576
20	2,075,789	827,184
10	1,963,853	939,120
5	1,373,005	1,529,968

Table 4. Number of model parameters trainable an non trainaible for mode.

Table 5 below shows the accuracy of each model on the test set. All models show an improvement upon the initial 48.6% accuracy showing that fine tuning has occurred. The model with the best performance is that with the last 20 layers unfrozen with an accuracy of 60.83%. This is a 12% increase on the initial model and demonstrates the power of transfer learning and fine tuning.

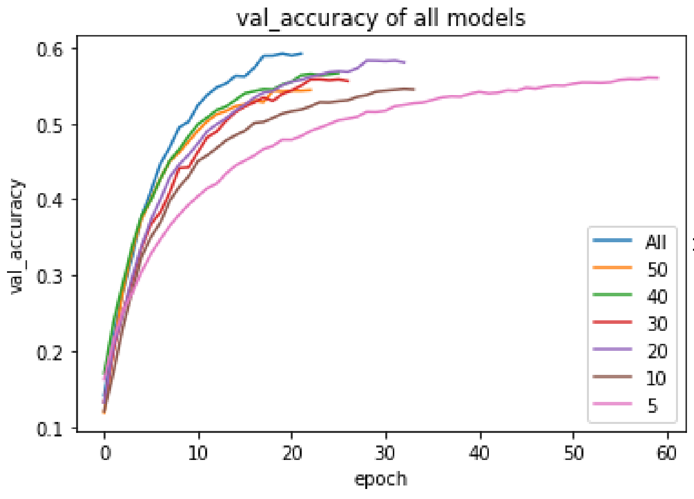
Model (unfrozen layers)	Epochs	Accuracy on test set %
All	22	58.63
50	23	55.66
40	26	58.61
30	27	57.52
20	33	**60.83**
10	34	57.10
5	60	59.91

Table 5. Number of traniing epochs before early stopping and accuracy on test set

Table 6 displays the various metrics of all models calculated using the macro average values. Most models appear to perform very simillarly having a tight cluster of values for each metric. It can be seen the the model with the last 20 layers unfrozen again performs best with marginally better results for each category.

Model (unfrozen layers)	Precision	Recall	F1
All	0.59	0.59	0.58
50	0.56	0.56	0.58
40	0.59	0.59	0.59
30	0.58	0.58	0.57
20	**0.62**	**0.61**	**0.61**
10	0.57	0.57	0.57
5	0.60	0.60	0.60

Table 6. Metrics of all models. The values were calculated using the macro average results.



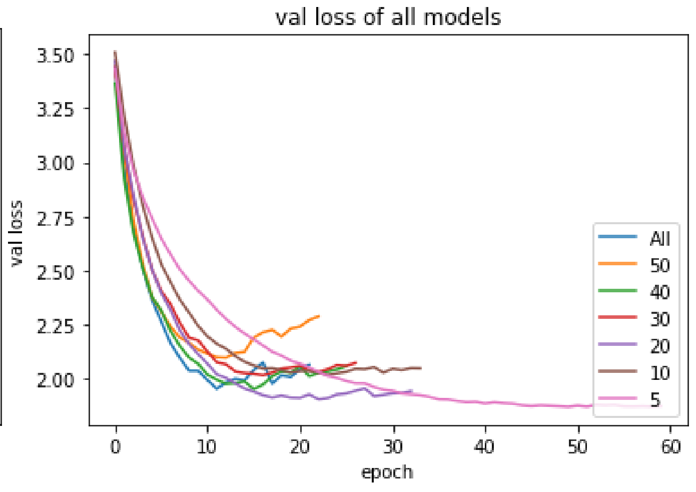


Figure 23. Validation accuracy and loss for all models

The validation accuracy and loss can be seen in figure 23. Its interesting how much faster the layer with the last 20 layers unfrozen trained reaching a higher accuracy than other models in a shorter amount of epochs. This may have something to do with the structure of those layers. More analysis would be need to draw a meaningful conclusion. Further testing around this result may provide insight but the training time may be too prohibitive.

Issues and Future work

Some of the issues encountered in this project include large training times per epoch and potential limitations of the architecture of the added top layers for prediction. Its not known if the chosen top layer is optimum and whether choosing a different architecture would result in increase accuracy.

Another issue is the dataset itself with numerous categories appearing the have identical images identified earlier. These would be very difficult for the models to discriminate between and its likely that the models reverted to predicting one class over the other as spaghetti carbonara had more incorrect predictions compared to spaghetti bolognaise.

Through a quick overview of the 60,000 images, some images are very noise, and the object of interest is overlayed with other food items, cutlery or sides on the plate which is common in the steak category with additional vegetables or sauces obscuring the steak. One alternative would be to include a category of steak with vegetables, steak with sauce etc., to let the model know there is a steak as well as some other items in the image. This has the downside of necessitating more training images which again may not be ideal to train on.

The models used are not the most state-of-the-art models on the ImageNet database. Transfer learning using one of the better performing models may lead to improved accuracy.

Additionally fine tuning is a long process, and it was chosen to unfreeze model layers in batches of 10. It may be the case that the optimum value exists within the range chosen but its not worth the extra computation to explore this possibility.

Conclusion

Transfer learning using the Inception ResNet v2, MobileNet v3 and NASNet were applied to the Food 101 dataset. The top layer of the models was removed and replaced with three additional layers, 2x256 densely connected layers with ReLu followed by a third 101 densely connected layer with SoftMax for predictions. MobileNet v3 had the best accuracy of 48.6% compared to 45.3% for NASNet and 31% for Inception ResNet v2.

The MobileNet v2 model was modified so that layers nearest the output could be trainable in steps of 10. The model was the last 20 layers set to trainable had the highest accuracy of 60.83%, an improvement of over 12% compared to the out of the box implementation used in the initial stage.

References

1. Link to download Food 101 dataset, https://www.vision.ee.ethz.ch/datasets_extra/food-101/
2. APA Psychnet, The perceptron: A probabilistic model for information storage and organization in the brain by Frank Rosenblatt
3. Cybernetic Predicting Devices by Ivakhnenko, A. G. and Lapa, V. G
4. ARS Journal, Gradient Theory of Optimal Flight Paths by H. J. Kelley
5. ARC, Artificial Neural Networks, Back Propagation, and the Kelley-Bryson Gradient Procedure by Stuart Dreyfus
6. BIT Numerical Mathematics, The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors by S. Linnainmaa
7. Beyond regression: new tools for prediction and analysis in the behavioral sciences by Paul Werbos
8. Nature, Learning Representations by Back-propagating Errors, by David E. Rumelhart, Geoffrey E. Hinton & Ronald J. Williams
9. Biological Cybernetics, Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position by Kunihiko Fukushima
10. Psychology, Receptive fields of single neurones in the cat's striate cortex by D. H. Hubel and T. N. Wiesel
11. MIT Press, Backpropagation applied to handwritten zip code recognition by Y. LeCun; B. Boser; J. S. Denker; D. Henderson; R. E. Howard; W. Hubbard; L. D. Jackel
12. First International Conference on Spoken Language Processing, A Neural Network for Speaker-Independent Isolated Word Recognition by Yamaguchi, Kouichi; Sakamoto, Kenji; Akabane, Toshio; Fujimoto, Yoshiji
13. International Journal of Uncertainty Fuzziness and Knowledge-Based Systems, The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions by Sepp Hochreiter
14. Computer Vision and Pattern Recognition, Deep Residual Learning for Image Recognition by Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun
15. Nvidia hardware blog, https://blogs.nvidia.com/blog/2016/01/12/accelerating-ai-artificial-intelligence-gpus/
16. IEEE, ImageNet: A large-scale hierarchical image database by Jia Deng; Wei Dong; Richard Socher; Li-Jia Li; Kai Li; Li Fei-Fei
17. Advances in Neural Information Processing Systems, ImageNet Classification with Deep Convolutional Neural Networks by A. Krizhevsky, S. Ilya, and G. E. Hinton
18. Computer Vision and Pattern Recognition, Multi-column Deep Neural Networks for Image Classification by Dan Cireşan, Ueli Meier, Juergen Schmidhuber
19. Computer Vision and Pattern Recognition, Going Deeper with Convolutions by Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich
20. Computer Vision and Pattern Recognition, Rethinking the Inception Architecture for Computer Vision by Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, Zbigniew Wojna

21. Computer Vision and Pattern Recognition, Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning by Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, Alex Alemi
22. Medium.com, InceptionResNetV2 Simple Introduction by Zahra Elhamraoui, https://medium.com/@zahraelhamraoui1997/inceptionresnetv2-simple-introduction-9a2000edcdb6
23. Computer Science and Pattern Recognition, Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning by Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, Alex Alemi
24. ImageNet leaderboard, https://paperswithcode.com/sota/image-classification-on-imagenet
25. Computer Vision and Pattern Recognition, MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications by Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, Hartwig Adam
26. Computer Vision and Pattern Recognition, MobileNetV2: Inverted Residuals and Linear Bottlenecks by Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, Liang-Chieh Chen
27. KDNuggets.com, Google Open Sources MobileNetV3 with New Ideas to Improve Mobile Computer Vision Models by Jesus Rodriguez, https://www.kdnuggets.com/2019/12/google-open-sources-mobilenetv3-improve-mobile-computer-vision-models.html
28. Computer Vision and Pattern Recognition, Searching for MobileNetV3 by Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, Hartwig Adam
29. Computer Vision and Pattern Recognition, Squeeze-and-Excitation Networks by Jie Hu, Li Shen, Samuel Albanie, Gang Sun, Enhua Wu
30. Computer Vision and Pattern Recognition, Learning Transferable Architectures for Scalable Image Recognition, Barret Zoph, Vijay Vasudevan, Jonathon Shlens, Quoc V. Le
31. Computer Vision and Pattern Recognition, Learning Transferable Architectures for Scalable Image Recognition by Barret Zoph, Vijay Vasudevan, Jonathon Shlens, Quoc V. Le
32. Machine Learning, EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks by Mingxing Tan, Quoc V. Le
33. Computer Vision and Pattern Recognition, EfficientNetV2: Smaller Models and Faster Training by Mingxing Tan, Quoc V. Le
34. Computation and Language, Attention Is All You Need by Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin
35. Journal of Vision, Using deep learning to probe the neural code for images in primary visual cortex by William F. Kindel; Elijah D. Christensen; Joel Zylberberg
36. Artificial Intelligence, Analytics of Explainers of Black Box Deep Neural Networks for Computer Vision: A survey by Vanessa Buhrmester, David Münch, Michael Arens
37. Statistics, Can Everyday AI be Ethical? Machine Learning Algorithm Fairness by Philippe Besse, Celine Castets-Renard, Aurelien Garivier, Jean-Michel Loubes
38. Towards Data Science, A 2022-Ready Deep Learning Hardware Guide by Nir Ben-Zvi, https://towardsdatascience.com/another-deep-learning-hardware-guide-73a4c35d3e86

39. Machine Learning, Accelerating SLIDE Deep Learning on Modern CPUs: Vectorization, Quantizations, Memory Optimizations, and More by Shabnam Daghaghi, Nicholas Meisburger, Mengnan Zhao, Yong Wu, Sameh Gobriel, Charlie Tai, Anshumali Shrivastava
40. Machine Learning, AutoML-Zero: Evolving Machine Learning Algorithms From Scratch by Esteban Real, Chen Liang, David R. So, Quoc V. Le
41. Symposium on Edge Computing, Exploring the Capabilities of Mobile Devices in Supporting Deep Learning by Yitao Chen, Saman Biookaghazadeh, Ming Zhao
42. Computer Vision and Pattern Recognition, YOLObile: Real-Time Object Detection on Mobile Devices via Compression-Compilation Co-Design by Yuxuan Cai, Hongjia Li, Geng Yuan, Wei Niu, Yanyu Li, Xulong Tang, Bin Ren, Yanzhi Wang
43. TensorFlow Core v2.8.0, https://www.tensorflow.org/api_docs/python/tf/keras/applications/inception_resnet_v2/preprocess_input
44. TensorFlow Core v2.8.0, https://www.tensorflow.org/api_docs/python/tf/keras/applications/mobilenet_v3/preprocess_input
45. TensorFlow Core v2.8.0, https://www.tensorflow.org/api_docs/python/tf/keras/applications/nasnet/NASNetMobile