

Deep Learning Food Classifier

Project Overview

Millions of images are uploaded to the internet daily and with the explosion of accessible data one problem still remains to be solved confidently by any of the major companies. The problem is image classification and despite a growing number of companies providing this service the accuracy of the services are overall still no match for the human eye.

Deep learning is a subset of machine learning techniques that are based on feature learning and in recent years have been used heavily in the machine learning industry including image classification. This is a series of techniques that extract useful representations from datasets and is often heard with neural networks which is a model which loosely represents the neurons within the human brain. A neural network consists of a series of connected nodes which each takes a weighted input and produces an output. The accuracy of the neural network is improved by updating its weights during training through a technique known as backpropagation.

Although neural networks such as multi layered perceptrons have had success in the past with image classifications, they suffer from a major problem with considering the spatial structure of the data. Hence, pixels that are close to each in the image have the same relevance as pixels far away.

Convolutional neural networks solves these problems by using a 3D representations of the image rather than a one dimensional input that MLPs take. The representation would consists of channels (in image recognition this could be the 3 colours: red, green and blue) and each channel would be a 2D matrix. The channels are passed through different types of hidden layers before the final fully connected layer which produces the final result for the classification, an example would be softmax. Hidden layers such as pooling layers takes a convolutional layer as input and can be used to reduce the size of the dataset to avoid overfitting and the computational complexity of the CNN. Examples include AlexNet¹, ResNET² and VGG³, which are used in competitions such as the Large Scale Visual Recognition Challenge (ILSVRC).

Problem Statement

Modern smartphones have lead to an explosion of images on the internet. In recent years the number of photos uploaded to social media sites have grown exponentially to the rate

¹ [AlexNet](#) winner of ILSVRC 2012, used 8 layers

² [ResNET](#) winner of ILSVRC 2015, used 152 layers

³ [VGG](#) used 19 layers

where millions are now uploaded daily onto the internet. The increase in data however brings new challenges to the users and companies.

Organising and search images have become more challenging as photos are lost in a mess of unstructured data dumps on the net, images unlike documents cannot be searched without meta data. Social media and photo sharing companies have allowed users to manually tag or label photos but this process is usually tedious and laborious, a more automated solution was needed.

Whilst the scope of image classification is vast, I will be focusing on one particular use case and that is the classification of food. A significant proportion of images being uploaded are food and in particular photos of people's lunches. However, users are still having to add tags or labels manually to describe their dish. If this problem can be solved, users on sites such as Instagram would no longer have to hashtag their dish manually and business could also utilize this service to provide all sorts of applications for customers. For example, imagine an app that could tell you what recipes you can cook from a photo of your ingredients!

Data Exploration

The model is performing supervised learning to classify images into specific labels using a convolutional neural network. In order to train the model weights, we need to obtain a set of images and labels as the input. CNNs will require a suitable number of images to create enough filters to provide accurate results, given the problem at hand which is to classify foods into categories we will need thousands of images to reach our target of 50% top 5 accuracy. Datasets needs to be clean i.e. the training image needs to match up with the label and then able to be processed into the correct input size and shape for the CNN. This would include transforming the image to a square and potentially reduce the resolution to improve training times (but not too much that it'll affect performance).

The dataset I have chosen is UECFOOD-256⁴ which was used for research in The University of Electro-Communication in Tokyo and is free to use for non-commercial activities. The dataset has 256 labels for different kinds of dishes. The data is distributed almost evenly across the datasets (with approximately 130-180 images per folder) with the exception of a three labels that have an abnormally larger dataset. The largest 3 labels have 729, 621 and 234 images. Our pre-processing step should include image re-sizing as the current data vary from 200 to 900 pixels for both the height and width. All images have colour layers which allow us to add extra channels to our CNN.

If the model becomes accurate, we could potentially add more categories and data to the training set to classify more dishes. The dataset contains exactly 31,651 images, which should hopefully be enough to train a suitable model and reach the performance target.

⁴ <http://foodcam.mobi/dataset256.html>

Algorithms and Techniques

In the last decade the rise of available data and increased computing power lead to companies investing in machine learning. These techniques allowed companies to perform analysis on large amounts data that would otherwise be impossible to do manually, one of which was called deep learning.

Deep learning is a series of techniques which extracted features from datasets to solve problems such as classification, optimisation or decision making. The term deep learning refers to the layout of the model which includes an input layer, multiple hidden layers and a final activation layer. The more hidden layers that a model has, the deeper it is going to extract features. Each layer consists of a series of nodes which resembles the neurons within a brain and hence the name neural network.

Convolutional neural networks

Convolutional neural networks (CNN) were specifically designed to extract features from images. Unlike multi layered perceptrons in other neural networks, CNN takes a multi dimensional input thus allowing it to extract features with spatial awareness. The input can also be comprised of multiple channels which can be used to represent colours e.g. red, green blue.

After the input layer, the model consists of the hidden layer(s) and the activation layer. The hidden layers comprises of convolutional layers which extract features from the previous layer and pooling layers which compresses the data from the previous layer. Both uses a sliding frame for the calculations and there are options to pad the data if the frame goes beyond the bounds of the input.

The data flows through the model, calculating the weights of the nodes as it reaches to the activation layer which produces the final result. Once the result is produced, the margin of error is calculated and fed backwards in a technique known as back propagation. The weights of the nodes are updated in the process and the margin of error reduced, several iterations are performed until the weights do not change any more.

When training our model it is important follow the cross validation process. Cross validation enables us to split the dataset into three parts: training data, validation data and testing data. During the training process training data is used as input to train the model whilst the validation dataset is used to calculate the accuracy of the model in each iteration. Testing data is kept separate from the training process and used to calculate the final accuracy of the model.

Convolutional layers will use the ReLU activation function to avoid the vanishing gradient problem because its gradient is always 1 when positive. It has a performance boost⁵ when handling negative values (negative values are represented as 0) which makes calculations much simpler. The padding for the sliding window will be 'same' which automatically pads the window if it reaches the edge of the image rather than discarding the information, thus reduce data loss.

The model will be compiled with RMSprop as the optimizer which adjusts the rate decay with the change in the gradient as it approaches the optimum. Categorical entropy loss will be used as the loss function which is more performant for back propagation over means squared error.

Transfer Learning

If the accuracy of the results are below the target specified earlier, transfer learning will be used for feature extraction from a well known CNN. This will require the weights of an existing CNN to be loaded before applying my model as the final layers. The weights of the exiting CNN will be frozen as the my model is training.

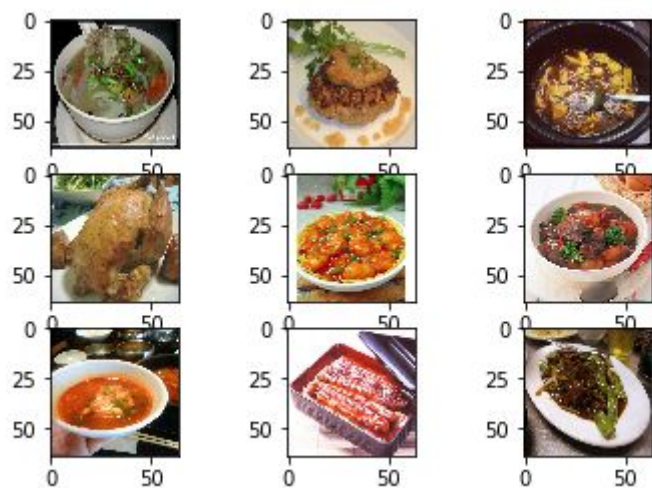
After training and running the model a few times, there's the option to fine tune individual layers of the base model. This would involve looking at each individual layers and deciding which ones to unfreeze or even remove from the model. This step will be out of scope for this project.

Data Augmentation

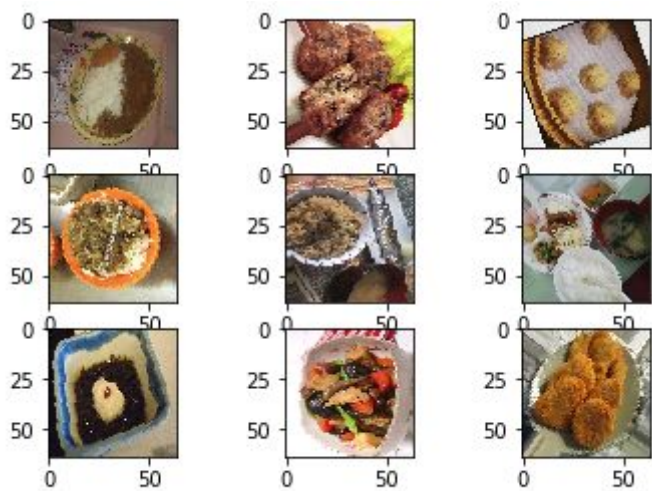
Data is the key component to the success of any deep learning classifier and data augmentation is a technique that allows us to build models with little data. The image augmentation process involves taking the existing training data and making copies of it after applying a series of transformations on it. This helps generalize the existing training/validation set and the extra data helps prevent overfitting in the model. Here are examples of the augmentation process on our training data:

⁵ [AlexNet](#) uses ReLU which reaches the 25% training error rate ten times faster than tanh neurons

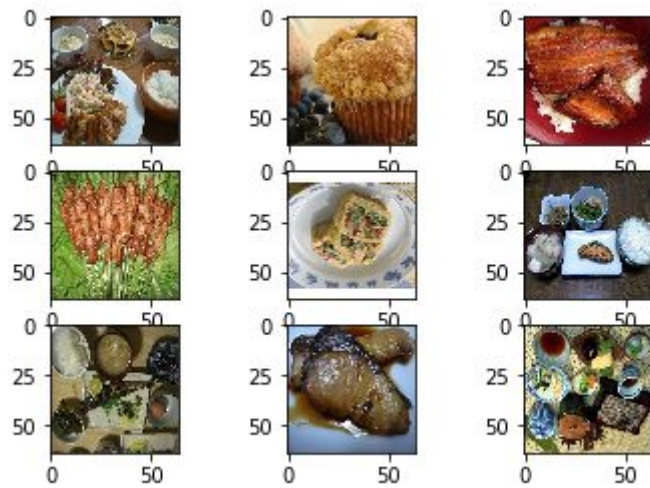
No Augmentation



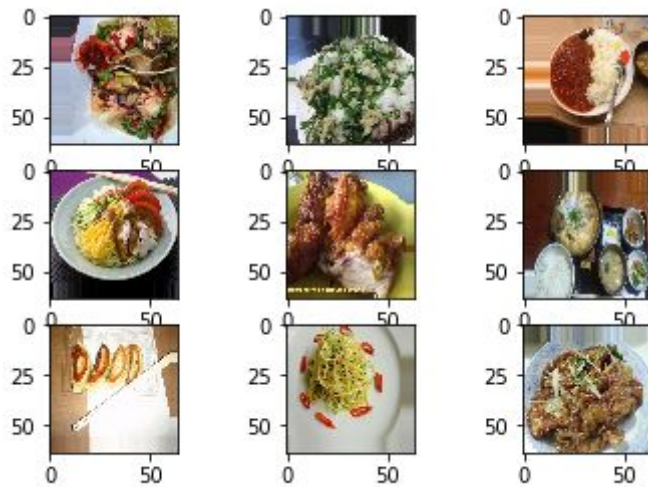
Random Rotations



Random Flips

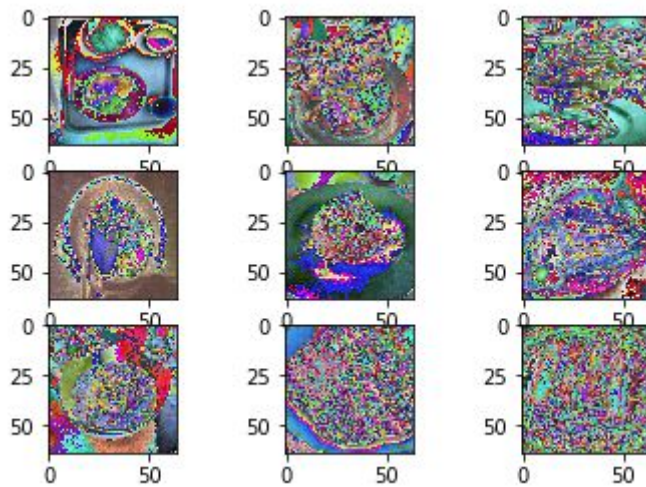


Random Shifts



Feature Standardization

This is applied to ensure they have the properties of a standard normal distribution of mean 0 and standard deviation of 1. This is important to ensure the data is not affine transformed invariant i.e. there's no linear function that can transform one image onto another.



ZCA Whitening

The ZCA whitening process is applied to reduce redundancies in the data and find features that strong differentiators for the classification process. Hence data that are highly correlated with each other will be merged together (in PCA this will be an eigenvector). In this project we will choose to use ZCA over PCA as it retains the spatial qualities of the original image during the whitening⁶.

Data Preprocessing

The dataset I have chosen is UECFOOD-256⁷ which was used for research in The University of Electro-Communication in Tokyo and is free to use for non-commercial activities. The dataset has 256 labels for different kinds of dishes and each folder contains the images corresponding to the numerical value of the label. For example, folder 1 corresponds to rice.

The mapping between numbers and labels were stored in a text file on one line which had to be manually edited to be comma delimited. Once properly formatted, the labels can be read by pandas.

Cross validation was performed on the images to split into training (70%), validation (20%) and testing (10%) sets. This was performed using the sklearn libraries and running the cross validation function twice. The images were then transferred to the sub folders train, valid, test within food-images.

Model Design and Tuning Strategy

During the project I wish to experiment with the following parameters to see which makes a positive effect on the test results:

- Image size

⁶ [Learning Multiple Layers of Features from Tiny Images](#)

⁷ <http://foodcam.mobi/dataset256.html>

- Number of hidden layers
- Image augmentation with random shifts, rotations, normalization
- ZCA Whitening
- Dropouts
- Nodes per layer

In the project I will start with implementing the benchmark design which will be a simple CNN with one convolutional layer and one pooling layer with images resized to 64 pixels. I will then increase the number of hidden layers and note the effect on the results before increasing the number of pixels.

Afterwards I'll run the same tests with image augmentation, dropouts, zcva and testing different layers/nodes to find which features are having a positive effect. Once I've found a model that provides the best results given the hardware limitations I shall proceed with transfer learning and note the results as well.

The models will have a verbose level of 1, which will provide information regarding the training loss and accuracy and the validation loss and accuracy. An indication of overfitting will be a decreasing loss per epoch but also a decrease in validation accuracy.

Metrics

Once the models are trained against the validation set they will run against the isolated test. For each test data, the model will return an array of numbers. Each index corresponds to the label within our dataset and the value within the index represents the likelihood of being the correct result. For example, if we had three labels and the result from a prediction was [0,0.1,0.9] then the model is 90% certain label 3 is the correct result.

Success is measured using with the classifier being able to determine the top 1 and top 5 scores. A top 1 score is the percentage of outcomes when the top class (i.e. the predicted label with the highest probability) matches the target label. A top 5 score is the same but one of the top 5 predicted labels with the highest probabilities matches the target label. The results will be displayed further on in a confusion matrix.

Implementation

The project experiments are separated into three categories: basic classifiers without augmentation, classifiers with augmentation on the training set, classifiers with standardization augmentation on both training and validation set and finally classifiers using transfer learning. The strategy is to continually improve our test results by running models that quick to train on our hardware first to get feedback before creating more complicated models that take longer to train.

Basic classifiers without image augmentation

These models are simplest of our experiments and aim to investigate the effect of the following properties on our test results:

- Pixel size
- Number of convolutional layers

Basic model

Our simplest model will be a simple CNN consisting of one convolutional layer and one pooling layer on a pixel size of 64.

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 64, 64, 16)	208
max_pooling2d_1 (MaxPooling2)	(None, 32, 32, 16)	0
global_average_pooling2d_1 (GlobalAveragePooling2D)	(None, 16)	0
dense_1 (Dense)	(None, 256)	4352
Total params: 4,560.0		
Trainable params: 4,560.0		
Non-trainable params: 0.0		

Pixel size 64, channels 3, number of conv layers 1, number of pool layers 1
Checkpoint at basic.pixel.64.conv.1.pool.1.hdf5

```
Epoch 4/5
Epoch 00003: val_loss improved from 5.43231 to 5.39899, saving model to saved_weights/basic.pixel.64.conv.1.pool.1.hdf5
35s - loss: 5.4181 - acc: 0.0234 - val_loss: 5.3990 - val_acc: 0.0243
Epoch 5/5
Epoch 00004: val_loss improved from 5.39899 to 5.35789, saving model to saved_weights/basic.pixel.64.conv.1.pool.1.hdf5
35s - loss: 5.3818 - acc: 0.0243 - val_loss: 5.3579 - val_acc: 0.0236
Top 1 test accuracy: 2.3598%
Top 5 test accuracy: 7.6310%
CPU times: user 7min 18s, sys: 1min 45s, total: 9min 4s
Wall time: 3min 4s
```

The logs show a constant decline on the training loss yet an increase on the validation accuracy (up to the 4th epoch). This is an encouraging indication that the model is not overfitting.

Benchmark model

In the next model I've kept the number of pixels the same but added another convolutional and pooling layer and this form the basic benchmark I use in the other experiments.

Layer (type)	Output Shape	Param #
conv2d_2 (Conv2D)	(None, 64, 64, 16)	208
max_pooling2d_2 (MaxPooling2D)	(None, 32, 32, 16)	0
conv2d_3 (Conv2D)	(None, 32, 32, 32)	2080
max_pooling2d_3 (MaxPooling2D)	(None, 16, 16, 32)	0
conv2d_4 (Conv2D)	(None, 16, 16, 64)	8256
max_pooling2d_4 (MaxPooling2D)	(None, 8, 8, 64)	0
global_average_pooling2d_2 (GlobalAveragePooling2D)	(None, 64)	0
dense_2 (Dense)	(None, 256)	16640
Total params: 27,184.0		
Trainable params: 27,184.0		
Non-trainable params: 0.0		

Pixel size 64, channels 3, number of conv layers 3, number of pool layers 3

Epoch 4/5

Epoch 00003: val_loss improved from 4.97392 to 4.91378, saving model to saved_weights/basic.pixel.64.conv.3.pool.3.hdf5

66s - loss: 4.9186 - acc: 0.0439 - val_loss: 4.9138 - val_acc: 0.0461

Epoch 5/5

Epoch 00004: val_loss improved from 4.91378 to 4.81903, saving model to saved_weights/basic.pixel.64.conv.3.pool.3.hdf5

66s - loss: 4.8207 - acc: 0.0524 - val_loss: 4.8190 - val_acc: 0.0529

Top 1 test accuracy: 5.6696%

Top 5 test accuracy: 16.6411%

CPU times: user 14min 55s, sys: 3min 30s, total: 18min 26s

Wall time: 5min 42s

Adding the extra layers have improved the top 1 accuracy to 5.67% which is a 3.31% from the previous model of 2.41%, top 5 also more than doubled in performance to 16.64%. The extra has helped the model extra more features from the images which improves the classification process.

Benchmark model using 128 pixels

For the next experiment I've kept the model the same and increased the pixel size of the image to 128 with the expectation that more detail would increase the classification process.

```

Epoch 4/5
Epoch 00003: val_loss improved from 4.67917 to 4.60851, saving model to saved_weights/basic.pixel.128.conv.3.pool.3.hdf5
119s - loss: 4.6439 - acc: 0.0490 - val_loss: 4.6085 - val_acc: 0.0501
Epoch 5/5
Epoch 00004: val_loss improved from 4.60851 to 4.56674, saving model to saved_weights/basic.pixel.128.conv.3.pool.3.hdf5
117s - loss: 4.5757 - acc: 0.0531 - val_loss: 4.5667 - val_acc: 0.0501
Top 1 test accuracy: 5.8262%
Top 5 test accuracy: 18.1471%
Wall time: 10min 26s

```

The results are encouraging with top1 accuracy of 5.83% and 18.14% for top 5, both of which were slightly better than the 64 pixel model. and was steadily improving with each epoch thus showing no signs of overfitting. This model might even improve further if we increased the number of epochs.

Benchmark model using 256 pixels

For the next experiment I shall increase the number of pixels to 256.

```

Epoch 3/5
Epoch 00002: val_loss improved from 4.82432 to 4.70545, saving model to saved_weights/basic.pixel.256.conv.3.pool.3.hdf5
456s - loss: 4.7429 - acc: 0.0422 - val_loss: 4.7054 - val_acc: 0.0416
Epoch 4/5
Epoch 00003: val_loss improved from 4.70545 to 4.66186, saving model to saved_weights/basic.pixel.256.conv.3.pool.3.hdf5
458s - loss: 4.6649 - acc: 0.0470 - val_loss: 4.6619 - val_acc: 0.0487
Epoch 5/5
Epoch 00004: val_loss improved from 4.66186 to 4.58268, saving model to saved_weights/basic.pixel.256.conv.3.pool.3.hdf5
456s - loss: 4.5996 - acc: 0.0525 - val_loss: 4.5827 - val_acc: 0.0525
Top 1 test accuracy: 6.1127%
Top 5 test accuracy: 17.5263%
Wall time: 39min 15s

```

A slight improvement to the top 1 accuracy and no signs of overfitting either after 5 epochs which suggests the model could continue training, However, since the training took 39 mins and over 35GB of memory and thus I've decided to not pursue experiments with pixel sizes greater than 128.

Classifiers with basic image augmentation

As discussion earlier in the project, image augmentation allows us to build more powerful classifiers by reusing our existing dataset in different ways. The ImageDataGenerator I used allows us to generate a larger dataset with using random rotations, flips and shifts on the

existing training set. I've set the rotation range to 90 degrees and the flips and shifts parameter to 0.2 which I will keep consistent throughout the project.

Basic augmentation of 3 convolutional layers

The experiments started with the same benchmark mode of 3 convolutional layers.

```
Checkpoint at nozca.nostd.pixel.128.conv.3.pool.3.hdf5
Epoch 1/5
700/700 [=====>.] - ETA: 0s - loss: 5.4835 - acc: 0.0227Epoch 00000: val_loss improved from inf to 5.419
84, saving model to saved_models/nozca.nostd.pixel.128.conv.3.pool.3.hdf5
701/700 [=====] - 248s - loss: 5.4839 - acc: 0.0227 - val_loss: 5.4198 - val_acc: 0.0230
Epoch 2/5
700/700 [=====>.] - ETA: 0s - loss: 5.3568 - acc: 0.0247Epoch 00001: val_loss improved from 5.41984 to
5.30336, saving model to saved_models/nozca.nostd.pixel.128.conv.3.pool.3.hdf5
701/700 [=====] - 248s - loss: 5.3566 - acc: 0.0247 - val_loss: 5.3034 - val_acc: 0.0220
Epoch 3/5
700/700 [=====>.] - ETA: 0s - loss: 5.2441 - acc: 0.0267Epoch 00002: val_loss improved from 5.30336 to
5.30116, saving model to saved_models/nozca.nostd.pixel.128.conv.3.pool.3.hdf5
701/700 [=====] - 248s - loss: 5.2431 - acc: 0.0266 - val_loss: 5.3012 - val_acc: 0.0284
Epoch 4/5
700/700 [=====>.] - ETA: 0s - loss: 5.1607 - acc: 0.0313Epoch 00003: val_loss improved from 5.30116 to
5.25499, saving model to saved_models/nozca.nostd.pixel.128.conv.3.pool.3.hdf5
701/700 [=====] - 249s - loss: 5.1616 - acc: 0.0313 - val_loss: 5.2550 - val_acc: 0.0267
Epoch 5/5
700/700 [=====>.] - ETA: 0s - loss: 5.0813 - acc: 0.0344Epoch 00004: val_loss improved from 5.25499 to
5.05010, saving model to saved_models/nozca.nostd.pixel.128.conv.3.pool.3.hdf5
701/700 [=====] - 248s - loss: 5.0825 - acc: 0.0344 - val_loss: 5.0501 - val_acc: 0.0333

CPU times: user 1h 6min 8s, sys: 7min 47s, total: 1h 13min 56s
Wall time: 20min 57s
```

The results are very promising with a steady increase with validation accuracy after each epoch. The top 1 accuracy of 3.89% is lower than our basic model with image augmentation, however, I believed we could push the results higher using dropouts of 0.5.

Basic augmentation of 3 convolutional layers with 0.5 dropouts

Layer (type)	Output Shape	Param #
conv2d_4 (Conv2D)	(None, 128, 128, 16)	208
dropout_1 (Dropout)	(None, 128, 128, 16)	0
max_pooling2d_4 (MaxPooling2D)	(None, 64, 64, 16)	0
conv2d_5 (Conv2D)	(None, 64, 64, 32)	2080
dropout_2 (Dropout)	(None, 64, 64, 32)	0
max_pooling2d_5 (MaxPooling2D)	(None, 32, 32, 32)	0
conv2d_6 (Conv2D)	(None, 32, 32, 64)	8256
dropout_3 (Dropout)	(None, 32, 32, 64)	0
max_pooling2d_6 (MaxPooling2D)	(None, 16, 16, 64)	0
global_average_pooling2d_2 (GlobalAveragePooling2D)	(None, 64)	0
dense_2 (Dense)	(None, 256)	16640
Total params: 27,184.0		
Trainable params: 27,184.0		
Non-trainable params: 0.0		

Pixel size 128, channels 3, number of conv layers 3, number of pool layers 3
Checkpoint at nozca.nostd.dropouts.pixel.128.conv.3.pool.3.hdf5

```

Epoch 3/5
700/700 [=====>.] - ETA: 0s - loss: 5.1011 - acc: 0.0358
Epoch 00002: val_loss improved from 5.36983 to 5.30049, saving model to saved_m
odels/nozca.nostd.dropouts.pixel.128.conv.3.pool.3.hdf5
701/700 [=====] - 361s - loss: 5.0997 - acc: 0.0357 -
val_loss: 5.3005 - val_acc: 0.0337
Epoch 4/5
700/700 [=====>.] - ETA: 0s - loss: 4.9863 - acc: 0.0426
Epoch 00003: val_loss improved from 5.30049 to 5.26305, saving model to saved_m
odels/nozca.nostd.dropouts.pixel.128.conv.3.pool.3.hdf5
701/700 [=====] - 362s - loss: 4.9841 - acc: 0.0433 -
val_loss: 5.2630 - val_acc: 0.0304
Epoch 5/5
700/700 [=====>.] - ETA: 0s - loss: 4.8923 - acc: 0.0488
Epoch 00004: val_loss improved from 5.26305 to 5.16554, saving model to saved_m
odels/nozca.nostd.dropouts.pixel.128.conv.3.pool.3.hdf5
701/700 [=====] - 360s - loss: 4.8893 - acc: 0.0495 -
val_loss: 5.1655 - val_acc: 0.0316

CPU times: user 1h 29min 17s, sys: 19min 59s, total: 1h 49min 16s
Wall time: 30min 18s

```

Adding the dropouts lowered the overall top 1 accuracy of the model to 3.46% and top 5 to 11.77% which wasn't surprising considering the number of layers and epochs remained the same.

Basic augmentation of 3 convolutional layers with 0.3 dropouts and 10 epochs

For the next model I was less afraid of over fitting and increased the number of epochs to 10 whilst making the dropouts less extreme with a value of 0.3.

Layer (type)	Output Shape	Param #
conv2d_7 (Conv2D)	(None, 128, 128, 16)	208
dropout_4 (Dropout)	(None, 128, 128, 16)	0
max_pooling2d_7 (MaxPooling2D)	(None, 64, 64, 16)	0
conv2d_8 (Conv2D)	(None, 64, 64, 32)	2080
dropout_5 (Dropout)	(None, 64, 64, 32)	0
max_pooling2d_8 (MaxPooling2D)	(None, 32, 32, 32)	0
conv2d_9 (Conv2D)	(None, 32, 32, 64)	8256
dropout_6 (Dropout)	(None, 32, 32, 64)	0
max_pooling2d_9 (MaxPooling2D)	(None, 16, 16, 64)	0
global_average_pooling2d_3 (GlobalAveragePooling2D)	(None, 64)	0
dense_3 (Dense)	(None, 256)	16640
Total params: 27,184.0		
Trainable params: 27,184.0		
Non-trainable params: 0.0		

Pixel size 128, channels 3, number of conv layers 3, number of pool layers 3
Checkpoint at nozca.nostd.dropouts.10.epochs.pixel.128.conv.3.pool.3.hdf5

```
Epoch 7/10
700/700 [=====>.] - ETA: 0s - loss: 4.8153 - acc: 0.0563Epoch 00006: val_loss improved from 5.07412 to
4.98084, saving model to saved_models/nozca.nostd.dropouts.10.epochs.pixel.128.conv.3.pool.3.hdf5
701/700 [=====] - 363s - loss: 4.8152 - acc: 0.0563 - val_loss: 4.9808 - val_acc: 0.0527
Epoch 8/10
700/700 [=====>.] - ETA: 0s - loss: 4.7559 - acc: 0.0594Epoch 00007: val_loss did not improve
701/700 [=====] - 362s - loss: 4.7552 - acc: 0.0593 - val_loss: 5.0504 - val_acc: 0.0380
Epoch 9/10
700/700 [=====>.] - ETA: 0s - loss: 4.6974 - acc: 0.0639Epoch 00008: val_loss improved from 4.98084 to
4.92358, saving model to saved_models/nozca.nostd.dropouts.10.epochs.pixel.128.conv.3.pool.3.hdf5
701/700 [=====] - 362s - loss: 4.6983 - acc: 0.0638 - val_loss: 4.9236 - val_acc: 0.0508
Epoch 10/10
700/700 [=====>.] - ETA: 0s - loss: 4.6495 - acc: 0.0675Epoch 00009: val_loss improved from 4.92358 to
4.86330, saving model to saved_models/nozca.nostd.dropouts.10.epochs.pixel.128.conv.3.pool.3.hdf5
701/700 [=====] - 363s - loss: 4.6495 - acc: 0.0674 - val_loss: 4.8633 - val_acc: 0.0579

CPU times: user 2h 58min 42s, sys: 40min 59s, total: 3h 39min 42s
Wall time: 1h 55s
```

Our experiments finally hit pass the 5% milestone and ended with a top 1 accuracy of 5.33% and validation accuracy of 18.73%. The logs showed very positive signs that the model was performing well with a constant rise of validation accuracy and decreasing training accuracy. The models are taking longer to train and uses roughly 10GB of RAM during the training process (which is 20GB less than the data augmentation with normalization).

Basic augmentation of 4 convolutional layers with 0.3 dropouts and 10 epochs

With a dropout strategy that I was comfortable with, I increased the number of convolutional layers to 4.

Layer (type)	Output Shape	Param #
conv2d_14 (Conv2D)	(None, 128, 128, 32)	416
dropout_11 (Dropout)	(None, 128, 128, 32)	0
max_pooling2d_14 (MaxPooling)	(None, 64, 64, 32)	0
conv2d_15 (Conv2D)	(None, 64, 64, 64)	8256
dropout_12 (Dropout)	(None, 64, 64, 64)	0
max_pooling2d_15 (MaxPooling)	(None, 32, 32, 64)	0
conv2d_16 (Conv2D)	(None, 32, 32, 64)	16448
dropout_13 (Dropout)	(None, 32, 32, 64)	0
max_pooling2d_16 (MaxPooling)	(None, 16, 16, 64)	0
conv2d_17 (Conv2D)	(None, 16, 16, 128)	32896
dropout_14 (Dropout)	(None, 16, 16, 128)	0
max_pooling2d_17 (MaxPooling)	(None, 8, 8, 128)	0
global_average_pooling2d_5 (GlobalAveragePooling2D)	(None, 128)	0
dense_5 (Dense)	(None, 256)	33024

Total params: 91,040.0
 Trainable params: 91,040.0
 Non-trainable params: 0.0

Pixel size 128, channels 3, number of conv layers 4, number of pool layers 4
 Checkpoint at nozca.nostd.morenodes.dropouts.10.epochs.pixel.128.conv.4.pool.4.hdf5

```

Epoch 7/10
700/700 [=====>.] - ETA: 0s - loss: 4.5333 - acc: 0.0804Epoch 00006: val_loss did not improve
701/700 [=====] - 454s - loss: 4.5346 - acc: 0.0802 - val_loss: 5.0331 - val_acc: 0.0389
Epoch 8/10
700/700 [=====>.] - ETA: 0s - loss: 4.4744 - acc: 0.0848Epoch 00007: val_loss improved from 4.98213 to
4.86061, saving model to saved_models/nozca.nostd.dropouts.10.epochs.pixel.128.conv.4.pool.4.hdf5
701/700 [=====] - 455s - loss: 4.4759 - acc: 0.0847 - val_loss: 4.8606 - val_acc: 0.0654
Epoch 9/10
700/700 [=====>.] - ETA: 0s - loss: 4.4092 - acc: 0.0947Epoch 00008: val_loss did not improve
701/700 [=====] - 453s - loss: 4.4112 - acc: 0.0946 - val_loss: 5.0988 - val_acc: 0.0405
Epoch 10/10
700/700 [=====>.] - ETA: 0s - loss: 4.3583 - acc: 0.0978Epoch 00009: val_loss improved from 4.86061 to
4.78247, saving model to saved_models/nozca.nostd.dropouts.10.epochs.pixel.128.conv.4.pool.4.hdf5
701/700 [=====] - 446s - loss: 4.3595 - acc: 0.0977 - val_loss: 4.7825 - val_acc: 0.0623

CPU times: user 3h 20min 29s, sys: 1h 5min 18s, total: 4h 25min 47s
Wall time: 1h 14min 45s

```

The extra layer made a positive change to our top 1 and top 5 accuracies bringing them to 6.10% and 20.63%. The training time doubled with the extra complexity and for the next experiments I decided to vary the number of nodes.

Basic augmentation of 4 convolutional layers with 0.3 dropouts, more nodes and 10 epochs

For the next model, the number of layers were kept the same but the number of nodes were increased in the first layers.

Layer (type)	Output Shape	Param #
conv2d_14 (Conv2D)	(None, 128, 128, 32)	416
dropout_11 (Dropout)	(None, 128, 128, 32)	0
max_pooling2d_14 (MaxPooling)	(None, 64, 64, 32)	0
conv2d_15 (Conv2D)	(None, 64, 64, 64)	8256
dropout_12 (Dropout)	(None, 64, 64, 64)	0
max_pooling2d_15 (MaxPooling)	(None, 32, 32, 64)	0
conv2d_16 (Conv2D)	(None, 32, 32, 64)	16448
dropout_13 (Dropout)	(None, 32, 32, 64)	0
max_pooling2d_16 (MaxPooling)	(None, 16, 16, 64)	0
conv2d_17 (Conv2D)	(None, 16, 16, 128)	32896
dropout_14 (Dropout)	(None, 16, 16, 128)	0
max_pooling2d_17 (MaxPooling)	(None, 8, 8, 128)	0
global_average_pooling2d_5 ((None, 128)		0
dense_5 (Dense)	(None, 256)	33024
Total params: 91,040.0		
Trainable params: 91,040.0		
Non-trainable params: 0.0		

Pixel size 128, channels 3, number of conv layers 4, number of pool layers 4
Checkpoint at nozca.nostd.morenodes.dropouts.10.epochs.pixel.128.conv.4.pool.4.hdf5

```
Epoch 7/10
700/700 [=====>.] - ETA: 0s - loss: 4.4428 - acc: 0.0901Epoch 00006: val_loss improved from 4.93847 to 4.72674, saving model to saved_models/nozca.nostd.morenodes.dropouts.10.epochs.pixel.128.conv.4.pool.4.hdf5
701/700 [=====>.] - 781s - loss: 4.4435 - acc: 0.0900 - val_loss: 4.7267 - val_acc: 0.0803
Epoch 8/10
700/700 [=====>.] - ETA: 0s - loss: 4.3655 - acc: 0.0992Epoch 00007: val_loss did not improve
701/700 [=====>.] - 779s - loss: 4.3635 - acc: 0.0997 - val_loss: 4.7523 - val_acc: 0.0674
Epoch 9/10
700/700 [=====>.] - ETA: 0s - loss: 4.3033 - acc: 0.1070Epoch 00008: val_loss did not improve
701/700 [=====>.] - 773s - loss: 4.3021 - acc: 0.1069 - val_loss: 4.9446 - val_acc: 0.0520
Epoch 10/10
700/700 [=====>.] - ETA: 0s - loss: 4.2313 - acc: 0.1154Epoch 00009: val_loss did not improve
701/700 [=====>.] - 775s - loss: 4.2321 - acc: 0.1152 - val_loss: 4.8067 - val_acc: 0.0595

CPU times: user 6h 18min 20s, sys: 1h 45min 32s, total: 8h 3min 52s
Wall time: 2h 10min 32s
```

Increasing the number of nodes managed to boost the top 1 accuracy to 9.19% and top 5 accuracy to 25.62%. Validation accuracy did drop after the 7th epoch whilst the training accuracy continued to climb thus giving an indication of overfitting.

Advanced image augmentation on validation set with normalization

Feature wise normalization and standardization are turned on to set the mean of the dataset to 0 and to divide the data by its standard deviation. Another ImageDataGenerator was also created to perform augmentation on the validation set which not only standardizes the validation tensors but also expand the validation data set which helps reduce overfitting.

Advanced augmentation of 3 convolutional layers

I started off with the simple 3 convolutional layer model:

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 128, 128, 16)	208
max_pooling2d_1 (MaxPooling2D)	(None, 64, 64, 16)	0
conv2d_2 (Conv2D)	(None, 64, 64, 32)	2080
max_pooling2d_2 (MaxPooling2D)	(None, 32, 32, 32)	0
conv2d_3 (Conv2D)	(None, 32, 32, 64)	8256
max_pooling2d_3 (MaxPooling2D)	(None, 16, 16, 64)	0
global_average_pooling2d_1 (GlobalAveragePooling2D)	(None, 64)	0
dense_1 (Dense)	(None, 256)	16640
Total params: 27,184.0		
Trainable params: 27,184.0		
Non-trainable params: 0.0		

Pixel size 128, channels 3, number of conv layers 3, number of pool layers 3
Checkpoint at nozca.pixel.128.conv.3.pool.3.hdf5

Checkpoint at nozca.pixel.128.conv.3.pool.3.hdf5

Epoch 1/5

Epoch 00000: val_loss improved from inf to 4.90287, saving model to saved_weights/nozca.pixel.128.conv.3.pool.3.hdf5

129s - loss: 5.0285 - acc: 0.0346 - val_loss: 4.9029 - val_acc: 0.0356

Epoch 2/5

Epoch 00001: val_loss improved from 4.90287 to 4.86432, saving model to saved_weights/nozca.pixel.128.conv.3.pool.3.hdf5

129s - loss: 4.8326 - acc: 0.0369 - val_loss: 4.8643 - val_acc: 0.0405

Epoch 3/5

Epoch 00002: val_loss improved from 4.86432 to 4.63306, saving model to saved_weights/nozca.pixel.128.conv.3.pool.3.hdf5

130s - loss: 4.6988 - acc: 0.0450 - val_loss: 4.6331 - val_acc: 0.0422

Epoch 4/5

Epoch 00003: val_loss improved from 4.63306 to 4.51280, saving model to saved_weights/nozca.pixel.128.conv.3.pool.3.hdf5

128s - loss: 4.5676 - acc: 0.0544 - val_loss: 4.5128 - val_acc: 0.0585

Epoch 5/5

Epoch 00004: val_loss improved from 4.51280 to 4.49072, saving model to saved_weights/nozca.pixel.128.conv.3.pool.3.hdf5

128s - loss: 4.4689 - acc: 0.0656 - val_loss: 4.4907 - val_acc: 0.0604

Standardizing data

Top 1 test accuracy: 6.6380%

Standardizing data

Top 5 test accuracy: 14.8520%

Wall time: 11min 4s

There were no signs of overfitting and the accuracy results were 6.64% for top 1 accuracy and 14.85% for top 5 accuracy. These are better than the same model with basic augmentation which had a top 1 hit rate of 3.89% and a top 5 hit rate of 13.48%.

Advanced augmentation of 3 convolutional layers with 0.5 dropouts

I decided to add dropout layers (with a setting of 0.5) to see the effect on our current models with the expectation of a reduction in overfitting. The number of epochs for the next set of experiments was increased to 10 to ensure we get the best set of weights possible.

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 128, 128, 16)	208
dropout_1 (Dropout)	(None, 128, 128, 16)	0
max_pooling2d_1 (MaxPooling2D)	(None, 64, 64, 16)	0
conv2d_2 (Conv2D)	(None, 64, 64, 32)	2080
dropout_2 (Dropout)	(None, 64, 64, 32)	0
max_pooling2d_2 (MaxPooling2D)	(None, 32, 32, 32)	0
conv2d_3 (Conv2D)	(None, 32, 32, 64)	8256
dropout_3 (Dropout)	(None, 32, 32, 64)	0
max_pooling2d_3 (MaxPooling2D)	(None, 16, 16, 64)	0
global_average_pooling2d_1 (GlobalAveragePooling2D)	(None, 64)	0
dense_1 (Dense)	(None, 256)	16640
Total params: 27,184.0		
Trainable params: 27,184.0		
Non-trainable params: 0.0		

Pixel size 128, channels 3, number of conv layers 3, number of pool layers 3
Checkpoint at nozca.multi.nodes.dropout.10.epochs.pixel.128.conv.3.pool.3.hdf5

```

Checkpoint at nozca.multi.nodes.dropout.10.epochs.pixel.128.conv.3.pool.3.hdf5
Epoch 1/10
Epoch 00000: val_loss improved from inf to 5.04254, saving model to saved_weights/nozca.multi.nodes.dropout.10.epochs.pixel.128.conv.3.pool.3.hdf5
257s - loss: 5.0317 - acc: 0.0327 - val_loss: 5.0425 - val_acc: 0.0392
Epoch 2/10
Epoch 00001: val_loss improved from 5.04254 to 4.89634, saving model to saved_weights/nozca.multi.nodes.dropout.10.epochs.pixel.128.conv.3.pool.3.hdf5
249s - loss: 4.7243 - acc: 0.0460 - val_loss: 4.8963 - val_acc: 0.0427
Epoch 3/10
Epoch 00002: val_loss improved from 4.89634 to 4.83447, saving model to saved_weights/nozca.multi.nodes.dropout.10.epochs.pixel.128.conv.3.pool.3.hdf5
247s - loss: 4.5617 - acc: 0.0595 - val_loss: 4.8345 - val_acc: 0.0471
Epoch 4/10
Epoch 00003: val_loss improved from 4.83447 to 4.77578, saving model to saved_weights/nozca.multi.nodes.dropout.10.epochs.pixel.128.conv.3.pool.3.hdf5
249s - loss: 4.4439 - acc: 0.0722 - val_loss: 4.7758 - val_acc: 0.0484
Epoch 5/10
Epoch 00004: val_loss improved from 4.77578 to 4.74961, saving model to saved_weights/nozca.multi.nodes.dropout.10.epochs.pixel.128.conv.3.pool.3.hdf5
250s - loss: 4.3512 - acc: 0.0786 - val_loss: 4.7496 - val_acc: 0.0424
Epoch 6/10
Epoch 00005: val_loss improved from 4.74961 to 4.68993, saving model to saved_weights/nozca.multi.nodes.dropout.10.epochs.pixel.128.conv.3.pool.3.hdf5
245s - loss: 4.2814 - acc: 0.0869 - val_loss: 4.6899 - val_acc: 0.0751
Epoch 7/10
Epoch 00006: val_loss did not improve
249s - loss: 4.2334 - acc: 0.0909 - val_loss: 4.6929 - val_acc: 0.0533
Epoch 8/10
Epoch 00007: val_loss improved from 4.68993 to 4.64696, saving model to saved_weights/nozca.multi.nodes.dropout.10.epochs.pixel.128.conv.3.pool.3.hdf5
248s - loss: 4.1824 - acc: 0.0977 - val_loss: 4.6470 - val_acc: 0.0683
Epoch 9/10
Epoch 00008: val_loss improved from 4.64696 to 4.62042, saving model to saved_weights/nozca.multi.nodes.dropout.10.epochs.pixel.128.conv.3.pool.3.hdf5
248s - loss: 4.1418 - acc: 0.1054 - val_loss: 4.6204 - val_acc: 0.0721
Epoch 10/10
Epoch 00009: val_loss did not improve
247s - loss: 4.1093 - acc: 0.1079 - val_loss: 4.6371 - val_acc: 0.0653
Standardizing data
Top 1 test accuracy: 7.7841%
Standardizing data
Top 5 test accuracy: 14.1834%
Wall time: 41min 53s

```

Adding dropout layers and increase the number of epochs has improved the top 1 and top 5 accuracies to 7.78% and 14.18%. We can see from the logs that the best weights were calculated at epoch 6 which would've been missed had we kept the epoch limit to 5.

Advanced augmentation of 4 convolutional layers

For next experiment we add another convolutional layer and remove the dropout layers:

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 128, 128, 16)	208
max_pooling2d_1 (MaxPooling2D)	(None, 64, 64, 16)	0
conv2d_2 (Conv2D)	(None, 64, 64, 32)	2080
max_pooling2d_2 (MaxPooling2D)	(None, 32, 32, 32)	0
conv2d_3 (Conv2D)	(None, 32, 32, 64)	8256
max_pooling2d_3 (MaxPooling2D)	(None, 16, 16, 64)	0
conv2d_4 (Conv2D)	(None, 16, 16, 128)	32896
max_pooling2d_4 (MaxPooling2D)	(None, 8, 8, 128)	0
global_average_pooling2d_1 (GlobalAveragePooling2D)	(None, 128)	0
dense_1 (Dense)	(None, 256)	33024
Total params: 76,464.0		
Trainable params: 76,464.0		
Non-trainable params: 0.0		

Pixel size 128, channels 3, number of conv layers 4, number of pool layers 4
Checkpoint at nozca.10.epochs.pixel.128.conv.4.pool.4.hdf5

Checkpoint at nozca.10.epochs.pixel.128.conv.4.pool.4.hdf5
Epoch 1/10
Epoch 0000: val_loss improved from inf to 5.04271, saving model to saved_weights/nozca.10.epochs.pixel.128.conv.4.pool.4.hdf5
135s - loss: 5.0013 - acc: 0.0352 - val_loss: 5.0427 - val_acc: 0.0313
Epoch 2/10
Epoch 0001: val_loss improved from 5.04271 to 4.88181, saving model to saved_weights/nozca.10.epochs.pixel.128.conv.4.pool.4.hdf5
136s - loss: 4.7469 - acc: 0.0401 - val_loss: 4.8818 - val_acc: 0.0411
Epoch 3/10
Epoch 0002: val_loss improved from 4.88181 to 4.45437, saving model to saved_weights/nozca.10.epochs.pixel.128.conv.4.pool.4.hdf5
134s - loss: 4.5500 - acc: 0.0566 - val_loss: 4.4544 - val_acc: 0.0612
Epoch 4/10
Epoch 0003: val_loss did not improve
134s - loss: 4.4044 - acc: 0.0739 - val_loss: 4.5181 - val_acc: 0.0650
Epoch 5/10
Epoch 0004: val_loss improved from 4.45437 to 4.36447, saving model to saved_weights/nozca.10.epochs.pixel.128.conv.4.pool.4.hdf5
134s - loss: 4.2824 - acc: 0.0865 - val_loss: 4.3645 - val_acc: 0.0871
Epoch 6/10
Epoch 0005: val_loss improved from 4.36447 to 4.30311, saving model to saved_weights/nozca.10.epochs.pixel.128.conv.4.pool.4.hdf5
134s - loss: 4.1879 - acc: 0.0960 - val_loss: 4.3031 - val_acc: 0.0906
Epoch 7/10
Epoch 0006: val_loss did not improve
132s - loss: 4.1050 - acc: 0.1045 - val_loss: 4.4628 - val_acc: 0.0854
Epoch 8/10
Epoch 0007: val_loss improved from 4.30311 to 4.20705, saving model to saved_weights/nozca.10.epochs.pixel.128.conv.4.pool.4.hdf5
134s - loss: 4.0336 - acc: 0.1160 - val_loss: 4.2071 - val_acc: 0.1004
Epoch 9/10
Epoch 0008: val_loss did not improve
134s - loss: 3.9649 - acc: 0.1208 - val_loss: 4.3137 - val_acc: 0.0808
Epoch 10/10
Epoch 0009: val_loss did not improve
133s - loss: 3.9119 - acc: 0.1290 - val_loss: 4.5181 - val_acc: 0.1045
Standardizing data
Top 1 test accuracy: 9.6466%
Standardizing data
Top 5 test accuracy: 20.3438%
Wall time: 22min 42s

Both top 1 accuracy improved from 7.78% to 14.18% and top 5 improved from 14.85% to 20.34% when compared to the previous model.

Advanced augmentation of 4 convolutional layers, with 64 nodes

So far the results have shown that adding more layers have shown a positive influence on the results and for the next experiment the number of nodes will increase to 64 in the current layers we have.

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 128, 128, 64)	832
max_pooling2d_1 (MaxPooling2D)	(None, 64, 64, 64)	0
conv2d_2 (Conv2D)	(None, 64, 64, 64)	16448
max_pooling2d_2 (MaxPooling2D)	(None, 32, 32, 64)	0
conv2d_3 (Conv2D)	(None, 32, 32, 64)	16448
max_pooling2d_3 (MaxPooling2D)	(None, 16, 16, 64)	0
conv2d_4 (Conv2D)	(None, 16, 16, 64)	16448
max_pooling2d_4 (MaxPooling2D)	(None, 8, 8, 64)	0
global_average_pooling2d_1 (GlobalAveragePooling2D)	(None, 64)	0
dense_1 (Dense)	(None, 256)	16640
Total params: 66,816.0		
Trainable params: 66,816.0		
Non-trainable params: 0.0		

Pixel size 128, channels 3, number of conv layers 4, number of pool layers 4
Checkpoint at nozca.64.nodes.10.epochs.pixel.128.conv.4.pool.4.hdf5

```

Checkpoint at nozca.64.nodes.10.epochs.pixel.128.conv.4.pool.4.hdf5
Epoch 1/10
Epoch 00000: val_loss improved from inf to 4.83574, saving model to saved_weights/nozca.64.nodes.10.epochs.pixel.128.conv.4.pool.4.hdf5
391s - loss: 5.0161 - acc: 0.0363 - val_loss: 4.8357 - val_acc: 0.0351
Epoch 2/10
Epoch 00001: val_loss did not improve
382s - loss: 4.7596 - acc: 0.0419 - val_loss: 4.8601 - val_acc: 0.0441
Epoch 3/10
Epoch 00002: val_loss improved from 4.83574 to 4.58296, saving model to saved_weights/nozca.64.nodes.10.epochs.pixel.128.conv.4.pool.4.hdf5
375s - loss: 4.5841 - acc: 0.0529 - val_loss: 4.5830 - val_acc: 0.0514
Epoch 4/10
Epoch 00003: val_loss improved from 4.58296 to 4.46207, saving model to saved_weights/nozca.64.nodes.10.epochs.pixel.128.conv.4.pool.4.hdf5
377s - loss: 4.4093 - acc: 0.0738 - val_loss: 4.4621 - val_acc: 0.0615
Epoch 5/10
Epoch 00004: val_loss did not improve
374s - loss: 4.2675 - acc: 0.0853 - val_loss: 4.8006 - val_acc: 0.0713
Epoch 6/10
Epoch 00005: val_loss did not improve
374s - loss: 4.1618 - acc: 0.1009 - val_loss: 4.7836 - val_acc: 0.0596
Epoch 7/10
Epoch 00006: val_loss improved from 4.46207 to 4.40521, saving model to saved_weights/nozca.64.nodes.10.epochs.pixel.128.conv.4.pool.4.hdf5
398s - loss: 4.0701 - acc: 0.1085 - val_loss: 4.4052 - val_acc: 0.0844
Epoch 8/10
Epoch 00007: val_loss did not improve
401s - loss: 4.0008 - acc: 0.1191 - val_loss: 4.7117 - val_acc: 0.0762
Epoch 9/10
Epoch 00008: val_loss improved from 4.40521 to 4.16932, saving model to saved_weights/nozca.64.nodes.10.epochs.pixel.128.conv.4.pool.4.hdf5
374s - loss: 3.9323 - acc: 0.1264 - val_loss: 4.1693 - val_acc: 0.1135
Epoch 10/10
Epoch 00009: val_loss did not improve
389s - loss: 3.8697 - acc: 0.1354 - val_loss: 4.2742 - val_acc: 0.0893
Standardizing data
Top 1 test accuracy: 11.4613%
Standardizing data
Top 5 test accuracy: 15.6638%
Wall time: 1h 4min 43s

```

Adding more nodes pushed the top 1 accuracy to 11.46% but the top 5 accuracy dropped to 15.66%. Validation results were improving slowly from each epoch but is starting to show signs of overfitting at the end. I decided to increase the number of nodes further to 128 for each convolutional layer. The drop in top 5 results was unexpected but this could just be an anomaly.

Advanced augmentation of 4 convolutional layers, with 128 nodes

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 128, 128, 128)	1664
max_pooling2d_1 (MaxPooling2D)	(None, 64, 64, 128)	0
conv2d_2 (Conv2D)	(None, 64, 64, 128)	65664
max_pooling2d_2 (MaxPooling2D)	(None, 32, 32, 128)	0
conv2d_3 (Conv2D)	(None, 32, 32, 128)	65664
max_pooling2d_3 (MaxPooling2D)	(None, 16, 16, 128)	0
conv2d_4 (Conv2D)	(None, 16, 16, 128)	65664
max_pooling2d_4 (MaxPooling2D)	(None, 8, 8, 128)	0
global_average_pooling2d_1 (GlobalAveragePooling2D)	(None, 128)	0
dense_1 (Dense)	(None, 256)	33024
Total params: 231,680.0		
Trainable params: 231,680.0		
Non-trainable params: 0.0		

Pixel size 128, channels 3, number of conv layers 4, number of pool layers 4
Checkpoint at nozca.128.nodes.10.epochs.pixel.128.conv.4.pool.4.hdf5


```

Checkpoint at nozca.128.nodes.10.epochs.pixel.128.conv.4.pool.4.hdf5
Epoch 1/10
Epoch 00000: val_loss improved from inf to 4.94386, saving model to saved_weights/nozca.128.nodes.10.epochs.pixel.128.conv.4.pool.4.hdf5
905s - loss: 4.9826 - acc: 0.0339 - val_loss: 4.9439 - val_acc: 0.0381
Epoch 2/10
Epoch 00001: val_loss improved from 4.94386 to 4.58281, saving model to saved_weights/nozca.128.nodes.10.epochs.pixel.128.conv.4.pool.4.hdf5
905s - loss: 4.6266 - acc: 0.0506 - val_loss: 4.5828 - val_acc: 0.0577
Epoch 3/10
Epoch 00002: val_loss improved from 4.58281 to 4.37873, saving model to saved_weights/nozca.128.nodes.10.epochs.pixel.128.conv.4.pool.4.hdf5
880s - loss: 4.4006 - acc: 0.0695 - val_loss: 4.3787 - val_acc: 0.0751
Epoch 4/10
Epoch 00003: val_loss did not improve
871s - loss: 4.2212 - acc: 0.0905 - val_loss: 5.5519 - val_acc: 0.0561
Epoch 5/10
Epoch 00004: val_loss improved from 4.37873 to 4.22916, saving model to saved_weights/nozca.128.nodes.10.epochs.pixel.128.conv.4.pool.4.hdf5
870s - loss: 4.0808 - acc: 0.1065 - val_loss: 4.2292 - val_acc: 0.0941
Epoch 6/10
Epoch 00005: val_loss did not improve
878s - loss: 3.9669 - acc: 0.1220 - val_loss: 4.3382 - val_acc: 0.0811
Epoch 7/10
Epoch 00006: val_loss did not improve
863s - loss: 3.8689 - acc: 0.1332 - val_loss: 5.2276 - val_acc: 0.0748
Epoch 8/10
Epoch 00007: val_loss did not improve
865s - loss: 3.7912 - acc: 0.1469 - val_loss: 4.7949 - val_acc: 0.0805
Epoch 9/10
Epoch 00008: val_loss improved from 4.22916 to 4.17294, saving model to saved_weights/nozca.128.nodes.10.epochs.pixel.128.conv.4.pool.4.hdf5
863s - loss: 3.7142 - acc: 0.1597 - val_loss: 4.1729 - val_acc: 0.1031
Epoch 10/10
Epoch 00009: val_loss improved from 4.17294 to 4.15282, saving model to saved_weights/nozca.128.nodes.10.epochs.pixel.128.conv.4.pool.4.hdf5
861s - loss: 3.6207 - acc: 0.1744 - val_loss: 4.1528 - val_acc: 0.1249
Standardizing data
Top 1 test accuracy: 12.7507%
Standardizing data
Top 5 test accuracy: 18.0038%
Wall time: 2h 27min 39s

```

The training time for 4 layers of 128 nodes more than doubled for this model from 1hrs to 2.5hrs which highlights the additional complexity we have added when increasing the number of nodes. Both the top 1 and top 5 accuracy improved slightly from the previous model to 12.75% and 18% and the validation accuracy was improving slowly towards the last epoch didn't seem to show signs of overfitting yet. However, I decided to add dropouts with a setting of 0.3 to see the effect it has on the results.

Advanced augmentation of 4 convolutional layers, with 128 nodes, 0.3 dropouts

Layer (type)	Output Shape	Param #
conv2d_2 (Conv2D)	(None, 128, 128, 128)	1664
dropout_1 (Dropout)	(None, 128, 128, 128)	0
max_pooling2d_1 (MaxPooling2D)	(None, 64, 64, 128)	0
conv2d_3 (Conv2D)	(None, 64, 64, 128)	65664
dropout_2 (Dropout)	(None, 64, 64, 128)	0
max_pooling2d_2 (MaxPooling2D)	(None, 32, 32, 128)	0
conv2d_4 (Conv2D)	(None, 32, 32, 128)	65664
dropout_3 (Dropout)	(None, 32, 32, 128)	0
max_pooling2d_3 (MaxPooling2D)	(None, 16, 16, 128)	0
conv2d_5 (Conv2D)	(None, 16, 16, 128)	65664
dropout_4 (Dropout)	(None, 16, 16, 128)	0
max_pooling2d_4 (MaxPooling2D)	(None, 8, 8, 128)	0
global_average_pooling2d_1 (GlobalAveragePooling2D)	(None, 128)	0
dense_1 (Dense)	(None, 256)	33024
Total params: 231,680.0		
Trainable params: 231,680.0		
Non-trainable params: 0.0		

Pixel size 128, channels 3, number of conv layers 4, number of pool layers 4
Checkpoint at nozca.128.nodes.10.dropout.epochs.pixel.128.conv.4.pool.4.hdf5

Checkpoint at nozca.128.nodes.10.dropout.epochs.pixel.128.conv.4.pool.4.hdf5
Epoch 1/10
Epoch 00000: val_loss improved from inf to 4.89473, saving model to saved_weights/nozca.128.nodes.10.dropout.epochs.pixel.128.conv.4.pool.4.hdf5
1708s - loss: 4.9492 - acc: 0.0365 - val_loss: 4.8947 - val_acc: 0.0484
Epoch 2/10
Epoch 00001: val_loss improved from 4.89473 to 4.63473, saving model to saved_weights/nozca.128.nodes.10.dropout.epochs.pixel.128.conv.4.pool.4.hdf5
1606s - loss: 4.5620 - acc: 0.0603 - val_loss: 4.6347 - val_acc: 0.0585
Epoch 3/10
Epoch 00002: val_loss did not improve
1606s - loss: 4.3290 - acc: 0.0784 - val_loss: 4.8085 - val_acc: 0.0452
Epoch 4/10
Epoch 00003: val_loss improved from 4.63473 to 4.48736, saving model to saved_weights/nozca.128.nodes.10.dropout.epochs.pixel.128.conv.4.pool.4.hdf5
1575s - loss: 4.1634 - acc: 0.0978 - val_loss: 4.4874 - val_acc: 0.0759
Epoch 5/10
Epoch 00004: val_loss improved from 4.48736 to 4.37604, saving model to saved_weights/nozca.128.nodes.10.dropout.epochs.pixel.128.conv.4.pool.4.hdf5
1582s - loss: 4.0207 - acc: 0.1194 - val_loss: 4.3760 - val_acc: 0.0756
Epoch 6/10
Epoch 00005: val_loss did not improve
1576s - loss: 3.9132 - acc: 0.1318 - val_loss: 4.4039 - val_acc: 0.0688
Epoch 7/10
Epoch 00006: val_loss improved from 4.37604 to 4.11383, saving model to saved_weights/nozca.128.nodes.10.dropout.epochs.pixel.128.conv.4.pool.4.hdf5
1550s - loss: 3.8086 - acc: 0.1477 - val_loss: 4.1138 - val_acc: 0.1219
Epoch 8/10
Epoch 00007: val_loss did not improve
1558s - loss: 3.7288 - acc: 0.1623 - val_loss: 4.1206 - val_acc: 0.0974
Epoch 9/10
Epoch 00008: val_loss improved from 4.11383 to 4.08646, saving model to saved_weights/nozca.128.nodes.10.dropout.epochs.pixel.128.conv.4.pool.4.hdf5
1558s - loss: 3.6571 - acc: 0.1725 - val_loss: 4.0865 - val_acc: 0.1184
Epoch 10/10
Epoch 00009: val_loss improved from 4.08646 to 4.08007, saving model to saved_weights/nozca.128.nodes.10.dropout.epochs.pixel.128.conv.4.pool.4.hdf5
1570s - loss: 3.5633 - acc: 0.1877 - val_loss: 4.0801 - val_acc: 0.1230
Standardizing data
Top 1 test accuracy: 10.9838%
Standardizing data
Top 5 test accuracy: 25.9790%
Wall time: 4h 26min 28s

Training time nearly doubled and the top 1 result decreased slightly from 12.75% to 10.98% whilst the top 5 results rose from 18% to 25.98%. The results didn't appear to be overfitting much. Overall having dropouts is improving the test results especially for top 5 accuracy (our aim is to reach 50%).

Advanced augmentation of 5 convolutional layers, with 128 nodes, 0.3 dropouts

For the final model of this category, a 5th convolutional layer was added with dropout layers added to prevent overfitting. The number of epochs was increased to 20 to ensure we get the best possible weights, unlike the other models this one was only ran once since it took an extremely long time to run.

Due to the length of the model and log statements, I've only grabbed a screenshot of the last few lines.

```
Epoch 17/20
Epoch 00016: val_loss did not improve
1504s - loss: 3.2331 - acc: 0.2362 - val_loss: 3.9294 - val_acc: 0.1429
Epoch 18/20
Epoch 00017: val_loss did not improve
1495s - loss: 3.1699 - acc: 0.2477 - val_loss: 3.8514 - val_acc: 0.1592
Epoch 19/20
Epoch 00018: val_loss did not improve
1496s - loss: 3.1441 - acc: 0.2573 - val_loss: 3.9795 - val_acc: 0.1214
Epoch 20/20
Epoch 00019: val_loss did not improve
1522s - loss: 3.1031 - acc: 0.2607 - val_loss: 3.8464 - val_acc: 0.1521
Standardizing data
Top 1 test accuracy: 18.1948%
Standardizing data
Top 5 test accuracy: 38.6342%
Wall time: 8h 25min
```

The model took 8h25 mins to run 20 epochs but didn't show significant signs of overfitting. The top 1 accuracy was 18.19% and the top 5 accuracy was an incredible 38.63% which is just 11.37% away from the project target. Adding more layers have certainly improved our results and the dropouts have helped prevent overfitting in our models. I stopped experiments in this category due to the length of time it takes to train models and I felt we had enough data to draw our conclusion for this category.

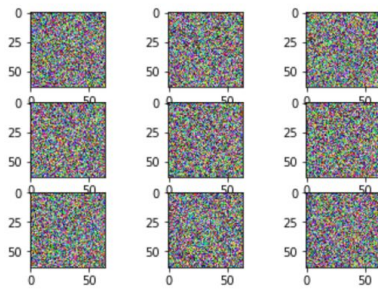
Image Augmentation with ZCA

ZCA can be a powerful tool in feature extraction but I quickly learnt from the experiments that it is computationally expensive to run. Running ZCA on 64 pixel images uses more than 30GB RAM on the machine with the augmentation to process.

```

: %time
datagen = datagen_zca(train_tensors)
display_first_batch(train_tensors, train_targets, datagen)

```



CPU times: user 36min 25s, sys: 1min 35s, total: 38min
Wall time: 9min 45s

A simple model of 3 layers takes 1h 16mins to train and although I would to take the experiments further, the ZCA process is too complicated for images with 3 channels on the hardware I have.

Transfer Learning

There are several award winning neural networks that have competed in the annual ILSVRC competitions using the ImageNet database. The models are trained over hundreds of thousands of images on the world's best hardware over several days. Although an individual can't do this on their personal equipment, Keras has the pre-trained models available to use. Transfer learning involves loading the pre-training weights and model and removing the final layer of the network. A custom layer is added to perform the bespoke classification and the weights in the base layers are frozen to avoid re-training.

In this experiment I will use VGG16⁸ since I am worried about the training times and it has few layers than some of the other CNNs, it also allows smaller images which reduces the training time.

In the first experiment decided to set the pixel size to 48x48 to reduce the training complexity. Next the custom layers were a global pooling layer and a dense layer with 1024 nodes. I deliberately select a larger node size compared to our normal experiments since I believe the base model would've extracted many features at this point.

The first model took 47 mins to train and had top 1 accuracy of 12.07% and a top 5 accuracy of 30.49%. This isn't beat the top model we currently have but considering the model ran without augmentation on smaller pixel images it was a good result.

For the last transfer learning experiment I increased the pixel size to 128x128, the model took 13hrs and 30 mins to train. However, the result was 21.73% for top 1 accuracy and 49% for the top 5 accuracy. This was a 3.53% improvement for top 1 and 10.37% improvement over the our best model running with augmentation and standardization. The

⁸ http://www.robots.ox.ac.uk/~vgg/research/very_deep/

top 5 accuracy was 1% short of our target set out in our proposal but I decided to stop since the training process was becoming too intensive and expensive.

Final Results

Model	File Name	Top 1 Accuracy	Top 5 Accuracy
Basic model	basic.pixel.64.conv.1.pool.1.hdf5	2.36%	7.63%
Benchmark model	basic.pixel.64.conv.3.pool.3.hdf5	5.67%	16.64%
Benchmark model using 128 pixels	basic.pixel.128.conv.3.pool.3.hdf5	5.83%	18.15%
Benchmark model using 256 pixels	basic.pixel.256.conv.3.pool.3.hdf5	6.11%	17.53%
Basic augmentation of 3 convolutional layers	nozca.nostd.pixel.128.conv.3.pool.3.hdf5	3.89%	13.48%
Basic augmentation of 3 convolutional layers with 0.5 dropouts	nozca.nostd.dropouts.pixel.128.conv.3.pool.3.hdf5	3.46%	11.77%
Basic augmentation of 3 convolutional layers with 0.3 dropouts and 10 epochs	nozca.nostd.dropouts.10.epochs.pixel.128.conv.3.pool.3.hdf5	5.33%	18.73%
Basic augmentation of 4 convolutional layers with 0.3 dropouts and 10 epochs	nozca.nostd.dropouts.10.epochs.pixel.128.conv.4.pool.4.hdf5	6.10%	20.63%
Basic augmentation of 4 convolutional layers with 0.3 dropouts, more nodes and 10 epochs	nozca.nostd.morenodes.dropouts.10.epochs.pixel.128.conv.4.pool.4.hdf5	9.19%	25.62%
Advanced augmentation of 3 convolutional layers	nozca.pixel.128.conv.3.pool.3.hdf5	6.64%	14.85%
Advanced augmentation of 3 convolutional layers with 0.5 dropouts	nozca.multi.nodes.dropout.10.epochs.pixel.128.conv.3.pool.3.hdf5	7.78%	14.18%
Advanced augmentation of 4 convolutional layers	nozca.10.epochs.pixel.128.conv.4.pool.4.hdf5	9.65%	20.34%
Advanced augmentation of 4 convolutional layers, with 64 nodes	nozca.64.nodes.10.epochs.pixel.128.conv.4.pool.4.hdf5	11.46%	15.66%
Advanced augmentation of 4 convolutional layers, with 128 nodes	nozca.128.nodes.10.epochs.pixel.128.conv.4.pool.4.hdf5	12.75%	18.00%
Advanced augmentation of 4 convolutional layers, with 128 nodes, 0.3 dropouts	nozca.128.nodes.10.dropout.epochs.pixel.128.conv.4.pool.4.hdf5	10.98%	25.98%
Advanced augmentation of 5 convolutional layers, with 128 nodes, 0.3 dropouts	nozca.128.nodes.10.dropout.20.epochs.pixel.128.conv.5.pool.5.hdf5	18.19%	38.63%
Transfer learning with VGG16 on 48 pixels	vgg16.pixel.48.conv.1.pool.1.hdf5	12.07%	30.49%
Transfer learning with VGG16 on 128 pixels	vgg16.pixel.128.conv.1.pool.1.hdf5	21.73%	49.00%

Conclusion

The objective of the project was to investigate how to build a custom convolutional neural network to classify food images into different categories with the aim of reach a top 5 accuracy (or top 5 error rate) or 50%.

From the experiments conducted I can first conclude that increasing the pixel size of the inputs has a significant effect on your training time but also a positive effect on the test accuracies. Increasing the pixels increases the amount of data that the models take hence allows the models more opportunities to extract better features from the images. This was very apparent in the final stage of the project when using transfer learning where both results nearly doubled when the pixel size was increased from 48 to 128 pixels. However, the training time went up exponentially to from 47 mins to 13 hours which definitely limits the experiments you can run.

The experiments in general, have highlighted how increasing the number of layers and nodes will increase the accuracy of your results with an increase in training times. Overfitting can also be an issue as the training error rate continues to decrease but the validation accuracy also decreases. One method to mitigate this, dropout layers were added to ensure the weights are training correctly but to benefit from this, the number of epochs must increase as the model takes more iterations to train.

The project investigated how augmentation can influence the results and how different settings can greatly change the results. The basic augmentation methods only performed simple random flips, shifts on the training set but this effectively gives the model free data to train with and reduce the chance of overfitting.

I decided to take augmentation further by applying it to the validation set as well, which also greatly expands our validation set without finding new images. Feature wide normalization and standardization were also added to help enhance the model and the results showed a clear improvement from the basic augmentation methods with the simplest model increasing from 3.89% to 6.64% in top 1 accuracy. The process, however, did become more increasingly expensive and memory intensive to train.

The results gave me confidence in my augmentation methods and I decided to experiment with the other parameters of the models: number of layers, number nodes, number of epochs and adding dropouts. The experiments became increasingly longer to train but I noticed adding more layers and increasing the number of nodes will improve the accuracy of the model. However, dropouts were needed to ensure the model does not overfit and ensure the weights are more balanced throughout the network. Of course using more dropout layers requires more epochs as the model takes longer to find its optimal weights. However, I

managed to reach a sound accuracy of 18.19% for top 1 accuracy and 38.63% for top 5 accuracy before I embarked on the next step.

Building a neural network from scratch is difficult because it requires a large amount of data and even with augmentation, it's really difficult to train deep models and raise the accuracy rates. Hence transfer learning and I used award winning models that were pre-trained on ImageNet to improve my classification results. At first the results were disappointing but when the number of pixels were increased to 128 the model successfully achieved a 49% top 5 accuracy rate which is 1% short from the proposed target. The model could be fine tuned to improve the accuracy but since it took 13hrs to train, I decided the 49% accuracy rate was sufficient.

Going further, the project could also investigate feature extraction with existing models from ImageNet competitions to improve results, although that too requires time to train as the base model needs to make a prediction for each of the training, validation and test sets to extract the features. We could also experiment with more layers and nodes with different dropout rates to improve results. I was also hoping to use ZCA whitening in my augmentation process, but unfortunately the process was extremely expensive to run. Even the p2 EC2 instance that I ran from AWS (with 64 GB of RAM) struggled to process images of 64 pixels. This is due to the size of the covariance matrix which grows exponentially with image size.

Final Words

I really enjoyed this project and I think the best way to learn about neural networks is to try and build one yourself. There definitely were challenges in the project that were difficult to overcome and this included debugging incorrect results, running models that didn't use all the memory on the instance and refactoring code as the notebook got bigger (you really miss an IDE like PyCharm at this point).

There's also no opportunity to write unit tests and hence it is important to check results as you progress within the project (I realised sklearn was reading my dataset folders in an unexpected order at one point). It is also important to start with smaller models to get faster feedback and results as rerunning models are very time consuming and expensive if you are hosting on a cloud provider (I've had to re-run my models several times).

Overall I am pleased with the results and understanding I have obtained from the course and will be eager to explore other techniques or frameworks (such as PyTorch).