# Overview

We are presented with a food dataset from Kaggle consisting of 101 different types of food , for this assignment we are given a subset of 10 different types of food to train our image classification model on. Our objective is to develop and tune a convolutional neural network that can perform accurate image classification across 10 different image classes. To hit this objective, I will go with the following approach : will first create a base model that will contain basic layers such as the input, relu, fully connected as well as output layers. From here, this base model will be scaled up by increasing the number of epochs until overfitting is observed. Then I will need to experiment with different hyperparameters as well as additional layers to be added in order to help reduce overfitting. Here is the following techniques I will be exploring, average pooling, max pooling, striding, dropout layers, normalization layers, kernel regularizers, kernel initializers ,different optimizers, different batch sizes, different learning rates. After testing all these techniques, I will record down which techniques helped and which techniques didn’t help. When am sure I am done will all the techniques and the model is still not satisfactory, I will then create a second base model. For the second base model, I will make major adjustments to the structure of the neural network from the first base. I will increase node sizes, add additional layers and use the best techniques I observed from the first base model and test the model. If the model still doesn’t satisfy me, I will move on to the third base layers where I will decrease the number of nodes in the neural network from the first base model as well as reduce the number of layers. Then I will apply all the techniques learnt from the first base model and train the model. Moving on from models made from scratch, I will utilize multiple pretrained models available on keras to create new models, I will finetune each of the models by unfreezing them and letting them train on the food data so that they can be more versed with the Kaggle food dataset. Finally, I will apply overfitting techniques mentioned above to the other layers outside the pretrained model to fix overfitting and I will be done with model creation. Next comes model selection. As for model selection, I will be prioritizing non-overfitted models over models with high training accuracy but are still overfitted. After testing all these models, I will choose the best model to perform classification on and that would conclude the assignment

# Data Preprocessing and Data Loading

To prepare the data, I first had to download the entire dataset from Kaggle. Next, I performed sub setting on the data using a python script so that I was only left with the data of the 10 food items I was tasked to predict. The script would divide each food class into their train test validation sets which are in the ratio of 75:20:5 respectively. After preparing the data, I saved the directories of each training validation and testing folders into my main jupyter notebook. I then proceeded to load the data into the notebook.

Training a CNN on a small number of images will result in overfitting. Consequently, the model will make errors in classifying new, unseen images. I decided to implement data augmentation to help avoid this. Data augmentation applies transformation to training images such as rescaling, rotating, shearing images, what this does is artificially increasing the amount of data by adding data points from existing data. This gives the CNN more data to be trained on , reducing overfitting in the process.

Text

Description automatically generated

These are the configurations I used on my image data generator for data augmentation, I firstly rescaled the rgb values by multiplying them by 1/255, this ensures that their values range between 0 and 1 making things easier for the model to process.

Next, I applied rotation to the image, using a value of 40, meaning that keras will randomly choose a value between 0 and 40 degrees to rotate the image clockwise or counter clockwise.

Then, I applied width and height shift of 0.2 each. This means that the image can be moved left, right up or down by 20%, the image is stretched to fill in the empty pixels.

Following that, I sheared the image with a range of 0.2 meaning that the image will be distorted along an axis by a maximum of 20%

Chart, diagram

Description automatically generated

I then used the zoom transformation which zooms into images, this is helpful as the topic of the image is usually at the center, zooming in will reduce noise from background inputs.

Then, I flipped the images horizontally using horizontal flip.

Lastly, I set the fill mode to nearest which means that images are stretched to fill in missing pixels.

Next, I configured my imagedatagenerator for the validation dataset which basically only had configuration to rescale its rgb values so that the values are between 0 and 1.

I applied the image data generator to the datasets. The image data generator generates batches containing the data of tensor images and augments these tensors in the process. This step is needed as it converts images to tensors so that the CNN can interpret it. I used a batch size of 50 as I found it to give the best results. For the class mode since we are doing a categorical classification I set it to categorical.

# Developing Image Classification models from scratch

For the development of the models from scratch decided to go for the following flow:

1. start with a base model
2. perform experiments
3. apply best experiments to new base models
4. select candidates to be scaled until overfit

This was decided because I wanted to save as much time by keeping the average epochs down, if I scaled every single upgraded model till it overfitted it would take too much time.

|  |  |  |
| --- | --- | --- |
| Model name +build+compiler | Overfitting | Description+ overfitting notes |
| **Base1ver1**  Conv2D(32, (3, 3), activation='relu',  input\_shape=(img\_size, img\_size, 3)  MaxPooling2D((2, 2))  Conv2D(64, (3, 3), activation='relu')  MaxPooling2D((2, 2))  Conv2D(128, (3, 3), activation='relu')  MaxPooling2D((2, 2))  Conv2D(128, (3, 3), activation='relu')  MaxPooling2D((2, 2))  Flatten()  Dense(512, activation='relu')  Dense(10, activation='softmax')  model.compile(loss='categorical\_crossentropy',  optimizer=optimizers.RMSprop(learning\_rate=1e-4),  metrics=['acc']) |  | This is the base model I used to start the assignment, hence model  Base1.  I noted that overfitting occurred at early stages of the training around epoch 11-12 |
| **Base1ver2**  Conv2D(32, (3, 3), activation='relu',  input\_shape=(img\_size, img\_size, 3))  MaxPooling2D((2, 2))  Dropout(0.5)  Conv2D(64, (3, 3), activation='relu')  MaxPooling2D((2, 2))  Dropout(0.25)  Conv2D(128, (3, 3), activation='relu')  MaxPooling2D((2, 2))  Dropout(0.25)  Conv2D(128, (3, 3), activation='relu')  MaxPooling2D((2, 2))  Dropout(0.25)  Flatten()  Dense(512, activation='relu')  Dropout(0.5)  Dense(10, activation='softmax')  model.compile(loss='categorical\_crossentropy',  optimizer=optimizers.RMSprop(learning\_rate=1e-4),  metrics=['acc']) |  | Since we observed overfitting in the first version of this base, I created a second version of this base where I added additional dropoutlayers as well as implemented the use of data augmentation.  As we can observe from the  Graph, overfitting occurred at around 50 epochs, the training accuracy also didn’t reach as high as before,.  From the loss graph it was also observed that overfitting was delayed until 50 epochs as compared to the 10 epochs without aumgnetation.We need to continue tuning more parameters and scale up the model |
| **Base1ver3**  Conv2D(32, (3, 3), activation='relu',  input\_shape=(img\_size, img\_size, 3))  Dropout(0.2)  MaxPooling2D((2, 2))  BatchNormalization()  Conv2D(64, (3, 3), activation='relu' ,kernel\_regularizer='l2')  MaxPooling2D((2, 2))  BatchNormalization()  Dropout(0.15)  Conv2D(128, (3, 3), activation='relu' ,kernel\_regularizer='l2')  MaxPooling2D((2, 2))  BatchNormalization()  Dropout(0.15)  Conv2D(128, (3, 3), activation='relu' ,kernel\_regularizer='l2')  MaxPooling2D((2, 2))  BatchNormalization()  Dropout(0.15)  Flatten()  Dense(512, activation='relu' ,kernel\_regularizer='l2')  model.add(BatchNormalization())  Dropout(0.5)  Dense(10, activation='softmax')  model.compile(loss='categorical\_crossentropy',  optimizer=optimizers.RMSprop(learning\_rate=1e-4),  metrics=['acc']) |  | We added some overfitting prevention methods and made a third version of base 1. We tweaked the dropout layers to its optimal values,added kernel regularizers as well as added batchnormalization.  From the graph we can see that overfitting was prevented meaning that the additions of the methods should be kept. |
| **Base1ver3.1**  Conv2D(32, (3, 3), activation='relu',  input\_shape=(img\_size, img\_size, 3))  Dropout(0.2)  MaxPooling2D((2, 2))  BatchNormalization()  Conv2D(64, (3, 3), activation='relu' ,kernel\_regularizer='l2')  MaxPooling2D((2, 2))  BatchNormalization()  Dropout(0.15)  Conv2D(128, (3, 3), activation='relu' ,kernel\_regularizer='l2')  MaxPooling2D((2, 2))  BatchNormalization()  Dropout(0.15)  Conv2D(128, (3, 3), activation='relu' ,kernel\_regularizer='l2')  MaxPooling2D((2, 2))  BatchNormalization()  Dropout(0.15)  Flatten()  Dense(512, activation='relu' ,kernel\_regularizer='l2')  model.add(BatchNormalization())  Dropout(0.5)  Dense(10, activation='softmax')  model.compile(loss='categorical\_crossentropy',  optimizer=optimizers.Adam(learning\_rate=1e-4),  metrics=['acc']) |  | I decided to start exploring the different optimizers adam and adadelta.  For version 3.1 I tried out the adam optimizer.  There is divergence occurring |
| **Base1ver3.2**  Conv2D(32, (3, 3), activation='relu',  input\_shape=(img\_size, img\_size, 3))  Dropout(0.2)  MaxPooling2D((2, 2))  BatchNormalization()  Conv2D(64, (3, 3), activation='relu' ,kernel\_regularizer='l2')  MaxPooling2D((2, 2))  BatchNormalization()  Dropout(0.15)  Conv2D(128, (3, 3), activation='relu' ,kernel\_regularizer='l2')  MaxPooling2D((2, 2))  BatchNormalization()  Dropout(0.15)  Conv2D(128, (3, 3), activation='relu' ,kernel\_regularizer='l2')  MaxPooling2D((2, 2))  BatchNormalization()  Dropout(0.15)  Flatten()  Dense(512, activation='relu' ,kernel\_regularizer='l2')  model.add(BatchNormalization())  Dropout(0.5)  Dense(10, activation='softmax')  model.compile(loss='categorical\_crossentropy',  optimizer=tf.keras.optimizers.legacy.Adadelta(learning\_rate=1.0,rho=0.95,epsilon =None,decay = 0.0),  metrics=['acc']) |  | Continuing with the optimizer experiments, I tried out the adadelta optimizer for version 3.2  As can be seen from the graph, the accuracy didn’t go as high as adam or rmsprop. There was high volatility observed as well as |
| **Base1ver3.1.1**  Conv2D(32, (3, 3), activation='relu',  input\_shape=(img\_size, img\_size, 3))  Dropout(0.2)  MaxPooling2D((2, 2))  BatchNormalization()  Conv2D(64, (3, 3), activation='relu' )  MaxPooling2D((2, 2))  BatchNormalization()  Dropout(0.15)  Conv2D(128, (3, 3), activation='relu')  MaxPooling2D((2, 2))  BatchNormalization()  Dropout(0.15)  Conv2D(128, (3, 3), activation='relu')  MaxPooling2D((2, 2))  BatchNormalization()  Dropout(0.15)  Flatten()  Dense(512, activation='relu')  (BatchNormalization())  Dropout(0.5)  Dense(10, activation='softmax')  model.compile(loss='categorical\_crossentropy',  optimizer=optimizers.Adam(learning\_rate=1e-4),  metrics=['acc']) |  | Through further reading I noted that some regularizers potentially did not pair well with batch normalization, hence I decided to begin my experiment for different regularizer configurations.  For the first experiment (version3.1.1) I decided to remove the r2 regularizers  From the graph it can be seen that the divergence between the validation and the training accuracy is very severe compared to (verion3.1) This means that overfitting would occur earlier if scaled up. |
| **Base1ver3.1.2**  Conv2D(32, (3, 3), activation='relu',  input\_shape=(img\_size, img\_size, 3))  Dropout(0.2)  MaxPooling2D((2, 2))  Conv2D(64, (3, 3), activation='relu' ,kernel\_regularizer='l2')  MaxPooling2D((2, 2))  Dropout(0.15)  Conv2D(128, (3, 3), activation='relu' ,kernel\_regularizer='l2')  MaxPooling2D((2, 2))  Dropout(0.15)  Conv2D(128, (3, 3), activation='relu' ,kernel\_regularizer='l2')  MaxPooling2D((2, 2))  Dropout(0.15)  Flatten()  Dense(512, activation='relu' ,kernel\_regularizer='l2')  Dropout(0.5)  Dense(10, activation='softmax')  model.compile(loss='categorical\_crossentropy',  optimizer=optimizers.Adam(learning\_rate=1e-4),  metrics=['acc']) |  | Next, I decided to remove the batch norm layers and add back the l2 regs.  As can be seen in comparison to version3.1, the removal of batch norm layers resulted in a very nice graph , however the accuracy levels reached were extremely low. Having accuracy levels this low will cause the CNN to be useless. |
| **Base1ver3.1.3**  Conv2D(32, (3, 3), activation='relu',  input\_shape=(img\_size, img\_size, 3))  Dropout(0.2)  MaxPooling2D((2, 2))  Conv2D(64, (3, 3), activation='relu' ,kernel\_regularizer='l1')  MaxPooling2D((2, 2))  BatchNormalization()  Dropout(0.15)  Conv2D(128, (3, 3), activation='relu' ,kernel\_regularizer='l1')  MaxPooling2D((2, 2))  BatchNormalization()  Dropout(0.15)  Conv2D(128, (3, 3), activation='relu' ,kernel\_regularizer='l1')  MaxPooling2D((2, 2))  BatchNormalization()  Dropout(0.15)  Flatten()  Dense(512, activation='relu' ,kernel\_regularizer='l1')  Dropout(0.5)  Dense(10, activation='softmax')  model.compile(loss='categorical\_crossentropy',  optimizer=optimizers.Adam(learning\_rate=1e-4),  metrics=['acc']) |  | Next I decided to test the l1 regularizer with batchnormalization.  As you can see from the graph, although the training and validation accuracy are close to each other, the model’s accuracy was slowing down almost to a straight line within 120 epochs, this mean that that even though the model doesn’t overfit, it isn’t worth it to scale up due to the low accuracy |
| **Base1ver3.1.4**  Conv2D(32, (3, 3), activation='relu',  input\_shape=(img\_size, img\_size, 3))  Dropout(0.2)  MaxPooling2D((2, 2))  Conv2D(64, (3, 3), activation='relu' ,kernel\_regularizer='l1')  MaxPooling2D((2, 2))  Dropout(0.15)  Conv2D(128, (3, 3), activation='relu' ,kernel\_regularizer='l1')  MaxPooling2D((2, 2))  Dropout(0.15)  Conv2D(128, (3, 3), activation='relu' ,kernel\_regularizer='l1')  MaxPooling2D((2, 2))  Dropout(0.15)  Flatten()  Dense(512, activation='relu' ,kernel\_regularizer='l1')  Dropout(0.5)  Dense(10, activation='softmax')  model.compile(loss='categorical\_crossentropy',  optimizer=optimizers.Adam(learning\_rate=1e-4),  metrics=['acc']) |  | Next I decided to test the l1 regularizer without batch normalization.  From the graph it shows pure chaos, this indicates that the model is not even learning from the training set |
| **Base1ver3.1.5**  Conv2D(32, (3, 3), activation='relu',  input\_shape=(img\_size, img\_size, 3))  Dropout(0.2)  MaxPooling2D((2, 2))  Conv2D(64, (3, 3), activation='relu' , kernel\_regularizer=regularizers.l1\_l2(l1=0.001, l2=0.001))  MaxPooling2D((2, 2))  Dropout(0.15)  Conv2D(128, (3, 3), activation='relu' , kernel\_regularizer=regularizers.l1\_l2(l1=0.001, l2=0.001))  MaxPooling2D((2, 2))  Dropout(0.15)  Conv2D(128, (3, 3), activation='relu' , kernel\_regularizer=regularizers.l1\_l2(l1=0.001, l2=0.001))  MaxPooling2D((2, 2))  Dropout(0.15)  Flatten()  Dense(512, activation='relu' , kernel\_regularizer=regularizers.l1\_l2(l1=0.001, l2=0.001))  Dropout(0.5)  Dense(10, activation='softmax')  model.compile(loss='categorical\_crossentropy',  optimizer=optimizers.Adam(learning\_rate=1e-4), |  | Next I decided to try out the l1\_l2 regularizers. First I started off without the batch normalization.  As you can see from the graph the model is underfitting, the validation accuracy is consistently higher than the training accuracy. |
| **Base1ver3.1.7**  Conv2D(32, (3, 3), activation='relu',  input\_shape=(img\_size, img\_size, 3))  Dropout(0.2)  MaxPooling2D((2, 2))  BatchNormalization()  Conv2D(64, (3, 3), kernel\_initializer=keras.initializers.HeNormal(),activation='relu' ,kernel\_regularizer='l2')  MaxPooling2D((2, 2))  BatchNormalization()  Dropout(0.15)  Conv2D(128, (3, 3), kernel\_initializer=keras.initializers.HeNormal(),activation='relu' ,kernel\_regularizer='l2')  MaxPooling2D((2, 2))  BatchNormalization()  Dropout(0.15)  Conv2D(128, (3, 3), kernel\_initializer=keras.initializers.HeNormal(), activation='relu' ,kernel\_regularizer='l2')  MaxPooling2D((2, 2))  BatchNormalization()  Dropout(0.15)  Flatten()  Dense(512, activation='relu' , kernel\_initializer=keras.initializers.HeNormal(),kernel\_regularizer='l2')  model.add(BatchNormalization())  Dropout(0.5)  Dense(10, activation='softmax')  model.compile(loss='categorical\_crossentropy',  optimizer=optimizers.Adam(learning\_rate=1e-4),  metrics=['acc']) |  | Next I decided to try out kernel initializers. For this case I tried he normal initializer. He normal initializer draws samples from a truncated normal dist centered on 0 with std given by std = sqrt(2/num of inputs).  From the accuracy graphs it is observed that the addition of the kernel initializer caused large divergence between validation acc and training acc as compared to without the kernel initializer (base1ver3.1) |
| **Base1ver3.1.8**  Conv2D(32, (3, 3), activation='relu',  input\_shape=(img\_size, img\_size, 3))  Dropout(0.2)  model.add(layers.MaxPooling2D((2, 2)))  BatchNormalization()  Conv2D(64, (3, 3), strides=(2, 2), kernel\_initializer=keras.initializers.HeNormal(),activation='relu' ,kernel\_regularizer='l2')  BatchNormalization()  Dropout(0.15)  Conv2D(128, (3, 3), kernel\_initializer=keras.initializers.HeNormal(),activation='relu' ,kernel\_regularizer='l2')  BatchNormalization()  Dropout(0.15)  Conv2D(128, (3, 3), strides=(2, 2),kernel\_initializer=keras.initializers.HeNormal(), activation='relu' ,kernel\_regularizer='l2')  BatchNormalization()  Dropout(0.15)  Flatten()  Dense(512, activation='relu' , kernel\_initializer=keras.initializers.HeNormal(),kernel\_regularizer='l2')  model.add(BatchNormalization())  Dropout(0.5)  Dense(10, activation='softmax')  model.compile(loss='categorical\_crossentropy',  optimizer=optimizers.Adam(learning\_rate=1e-4),  metrics=['acc']) |  | For the next mode, I wanted to test to affects of striding paired with maxpooling. Striding with steps of 2 moves the filter 2 pixels at a time, for maxpooling, the filters only select the highest value within the segment.  From the graph we can see that the divergence between the training and validation accuracy were reduced with striding |
| **Base1ver3.1.9**  Conv2D(32, (3, 3) ,padding="same", activation='relu',  input\_shape=(img\_size, img\_size, 3))  Dropout(0.2)  model.add(layers.MaxPooling2D((2, 2)))  BatchNormalization()  Conv2D(64, (3, 3) ,padding="same", strides=(2, 2), kernel\_initializer=keras.initializers.HeNormal(),activation='relu' ,kernel\_regularizer='l2')  BatchNormalization()  Dropout(0.15)  Conv2D(128, (3, 3) ,padding="same", kernel\_initializer=keras.initializers.HeNormal(),activation='relu' ,kernel\_regularizer='l2')  BatchNormalization()  Dropout(0.15)  Conv2D(128, (3, 3) ,padding="same", strides=(2, 2),kernel\_initializer=keras.initializers.HeNormal(), activation='relu' ,kernel\_regularizer='l2')  BatchNormalization()  Dropout(0.15)  Flatten()  Dense(512, activation='relu' , kernel\_initializer=keras.initializers.HeNormal(),kernel\_regularizer='l2')  model.add(BatchNormalization())  Dropout(0.5)  Dense(10, activation='softmax')  model.compile(loss='categorical\_crossentropy',  optimizer=optimizers.Adam(learning\_rate=1e-4),  metrics=['acc']) |  | Next I decided to see how padding affected the model, I added padding to each conv layers.  It was observed through the graph that that addition of padding caused the divergence between training accuracy and validation accuracy to increase as compared to the model without padding (previous model) |
| ­**Base1ver4.1**  Conv2D(32, (3, 3), activation='relu',  input\_shape=(img\_size, img\_size, 3))  Dropout(0.2)  MaxPooling2D((2, 2))  BatchNormalization()  Conv2D(64, (3, 3), activation='relu' ,kernel\_regularizer='l2')  MaxPooling2D((2, 2))  BatchNormalization()  Dropout(0.15)  Conv2D(128, (3, 3), activation='relu' ,kernel\_regularizer='l2')  MaxPooling2D((2, 2))  BatchNormalization()  Dropout(0.15)  Conv2D(128, (3, 3), activation='relu' ,kernel\_regularizer='l2')  MaxPooling2D((2, 2))  BatchNormalization()  Dropout(0.15)  Flatten()  Dense(512, activation='relu' ,kernel\_regularizer='l2')  model.add(BatchNormalization())  Dropout(0.5)  Dense(10, activation='softmax')  model.compile(loss='categorical\_crossentropy',  optimizer=optimizers.Adam(learning\_rate= 0.00017),  metrics=['acc']) |  | Next I begin testing how learning rates affected the models. I started by increasing the learning rate  By 70% (0.0001 to 0.00017).  The model above was taken from base1ver3.1.  Comparing to the graph of base1ver3.1, the increase of learning rate helped to reduce the divergence of validation accuracy and training accuracy which means that increasing learning rate is beneficial to the graph. |
| ­**Base1ver4.2**  Conv2D(32, (3, 3), activation='relu',  input\_shape=(img\_size, img\_size, 3))  Dropout(0.2)  MaxPooling2D((2, 2))  BatchNormalization()  Conv2D(64, (3, 3), activation='relu' ,kernel\_regularizer='l2')  MaxPooling2D((2, 2))  BatchNormalization()  Dropout(0.15)  Conv2D(128, (3, 3), activation='relu' ,kernel\_regularizer='l2')  MaxPooling2D((2, 2))  BatchNormalization()  Dropout(0.15)  Conv2D(128, (3, 3), activation='relu' ,kernel\_regularizer='l2')  MaxPooling2D((2, 2))  BatchNormalization()  Dropout(0.15)  Flatten()  Dense(512, activation='relu' ,kernel\_regularizer='l2')  model.add(BatchNormalization())  Dropout(0.5)  Dense(10, activation='softmax')  model.compile(loss='categorical\_crossentropy',  optimizer=optimizers.Adam(learning\_rate= 0.0002),  metrics=['acc']) |  | Next I wanted to check if further increasing the learning rate will further improve the model. Hence I changed the learning rate form 0.00017 to 0.0002 double of the initial learning rate.  As we can see from the graph, compared to the 0.00017 learning rate, the increase in the learning rate helped drastically reduce divergence in the validation accuracy and training accuracy for the most part of the epochs expect for epoch 100 where there was large divergence. |
| ­**Base1ver4.3**  Conv2D(32, (3, 3), activation='relu',  input\_shape=(img\_size, img\_size, 3))  Dropout(0.2)  MaxPooling2D((2, 2))  BatchNormalization()  Conv2D(64, (3, 3), activation='relu' ,kernel\_regularizer='l2')  MaxPooling2D((2, 2))  BatchNormalization()  Dropout(0.15)  Conv2D(128, (3, 3), activation='relu' ,kernel\_regularizer='l2')  MaxPooling2D((2, 2))  BatchNormalization()  Dropout(0.15)  Conv2D(128, (3, 3), activation='relu' ,kernel\_regularizer='l2')  MaxPooling2D((2, 2))  BatchNormalization()  Dropout(0.15)  Flatten()  Dense(512, activation='relu' ,kernel\_regularizer='l2')  model.add(BatchNormalization())  Dropout(0.5)  Dense(10, activation='softmax')  model.compile(loss='categorical\_crossentropy',  optimizer=optimizers.Adam(learning\_rate= 0.00003),  metrics=['acc']) |  | Next, I tried reducing the learning rate by 70%. I changed the learning rate from 0.0001 to 0.00003.  From the graph we can see that there is larger divergence between the training accuracy and validation accuracy as compared to higher learning rates. |

Now, we have finished with experimenting with most of the hyperparameters, lets move on to scaling up/down our models in terms of computational complexity. This means that we start messing with the nodes for each conv layers, add more conv layers and even manipulate nodes for dense layers.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Base2ver1  model = models.Sequential()  model.add(layers.Conv2D(16, (3, 3), activation='relu',  input\_shape=(img\_size, img\_size, 3)))  model.add(layers.MaxPooling2D((2, 2)))  model.add(layers.Conv2D(16, (3, 3), activation='relu'))  model.add(layers.MaxPooling2D((2, 2)))  model.add(layers.Conv2D(32, (3, 3), activation='relu'))  model.add(layers.MaxPooling2D((2, 2)))  model.add(layers.Conv2D(64, (3, 3), activation='relu'))  model.add(layers.MaxPooling2D((2, 2)))  model.add(layers.Flatten())  model.add(layers.Dense(128, activation='relu'))  model.add(layers.Dense(10, activation='softmax'))  model.summary()  model.compile(loss='categorical\_crossentropy',  optimizer=optimizers.Adam(learning\_rate=1e-4),  metrics=['acc']) | |  | I started off the new experiments by creating a simpler model, I reduced number of nodes in each conv layer as week as the dense layer, this results in a simpler mode with much faster training times.  From the graph it can be seen that the model was unable to learn much of the complex features from the training set. This can be seen from the low training and validation accuracy. From the loss table we can also observe overfitting at epoch 100 where the loss values increases in value | |
| base 3  version 1  from tensorflow.keras.layers import Dropout  from tensorflow.keras.layers import BatchNormalization  model = models.Sequential()  model.add(layers.Conv2D(16, (3, 3), activation='relu',  input\_shape=(img\_size, img\_size, 3)))  model.add(Dropout(0.2))  model.add(layers.MaxPooling2D((2, 2)))  model.add(BatchNormalization())  model.add(layers.Conv2D(32, (3, 3), activation='relu', kernel\_regularizer='l2'))  model.add(layers.MaxPooling2D((2, 2)))  model.add(BatchNormalization())  model.add(Dropout(0.15))  model.add(layers.Conv2D(32, (3, 3), activation='relu',kernel\_regularizer='l2'))  model.add(BatchNormalization())  model.add(Dropout(0.15))  model.add(layers.Conv2D(64, (3, 3), activation='relu', kernel\_regularizer='l2'))  model.add(BatchNormalization())  model.add(layers.Conv2D(64, (3, 3), activation='relu', kernel\_regularizer='l2'))  model.add(layers.MaxPooling2D((2, 2)))  model.add(BatchNormalization())  model.add(Dropout(0.15))  model.add(layers.Flatten())  model.add(layers.Dense(256, activation='relu', kernel\_regularizer='l2'))  model.add(BatchNormalization())  model.add(Dropout(0.5))  model.add(layers.Dense(10, activation='softmax'))  model.summary()  model.compile(loss='categorical\_crossentropy',  optimizer=optimizers.Adam(learning\_rate=1e-4),  metrics=['acc']) |  | for the next base, I kept the reduced node size for all of the hidden layers except for 1 (16 changed to 32), I also increased the nodes in the dense layer and added an extra hidden layer.  From the graph it can be seen that the addition of the new layer as well as the increase in dense layer nodes helped the model a lot, compared to base2, the accuracy levels were higher for both training and validation, however it would have been better to stop the training at around 102 epochs. | |
| **base 4 readjusted #addition of 1 256 layer inoput changed to 64**  from tensorflow.keras.layers import Dropout  from tensorflow.keras.layers import BatchNormalization  model = models.Sequential()  model.add(layers.Conv2D(64, (3, 3), activation='relu',  input\_shape=(img\_size, img\_size, 3)))  model.add(Dropout(0.2))  model.add(layers.MaxPooling2D((2, 2)))  model.add(BatchNormalization())  model.add(layers.Conv2D(64, (3, 3), activation='relu', kernel\_regularizer='l2'))  model.add(layers.MaxPooling2D((2, 2)))  model.add(BatchNormalization())  model.add(Dropout(0.15))  model.add(layers.Conv2D(128, (3, 3), activation='relu', kernel\_regularizer='l2'))  model.add(BatchNormalization())  model.add(layers.Conv2D(128, (3, 3), activation='relu', kernel\_regularizer='l2'))  model.add(layers.MaxPooling2D((2, 2)))  model.add(BatchNormalization())  model.add(Dropout(0.15))  model.add(layers.Conv2D(256, (3, 3), activation='relu', kernel\_regularizer='l2'))  model.add(layers.MaxPooling2D((2, 2)))  model.add(layers.Flatten())  model.add(layers.Dense(512, activation='relu', kernel\_regularizer='l2'))  model.add(BatchNormalization())  model.add(Dropout(0.5))  model.add(layers.Dense(10, activation='softmax'))  model.summary()  model.compile(loss='categorical\_crossentropy',  optimizer=optimizers.Adam(learning\_rate=1e-4),  metrics=['acc']) | | |  | For the next base, I decided to increase the complexity of the mode, I added a hidden layer with 256 nodes in hopes to learn more features, I also readjusted the input layers nodes to 64.  From the graphs we can see that the model was able to hit higher accuracies for both training and validation. This means that the model was able to learn more from the training set. |
| base 4  version 2 different epoch  from tensorflow.keras.layers import Dropout  from tensorflow.keras.layers import BatchNormalization  model = models.Sequential()  model.add(layers.Conv2D(64, (3, 3), activation='relu',  input\_shape=(img\_size, img\_size, 3)))  model.add(Dropout(0.2))  model.add(layers.MaxPooling2D((2, 2)))  model.add(BatchNormalization())  model.add(layers.Conv2D(64, (3, 3), activation='relu', kernel\_regularizer='l2'))  model.add(layers.MaxPooling2D((2, 2)))  model.add(BatchNormalization())  model.add(Dropout(0.15))  model.add(layers.Conv2D(128, (3, 3), activation='relu', kernel\_regularizer='l2'))  model.add(BatchNormalization())  model.add(layers.Conv2D(128, (3, 3), activation='relu', kernel\_regularizer='l2'))  model.add(layers.MaxPooling2D((2, 2)))  model.add(BatchNormalization())  model.add(Dropout(0.15))  model.add(layers.Conv2D(256, (3, 3), activation='relu', kernel\_regularizer='l2'))  model.add(layers.MaxPooling2D((2, 2)))  model.add(layers.Flatten())  model.add(layers.Dense(512, activation='relu', kernel\_regularizer='l2'))  model.add(BatchNormalization())  model.add(Dropout(0.5))  model.add(layers.Dense(10, activation='softmax'))  model.summary()  model.compile(loss='categorical\_crossentropy',  optimizer=optimizers.Adam(learning\_rate=1e-4),  metrics=['acc']) | | |  | I continued the experiments with base4 to find the optimal stop point for the model, with reference from the previous graph (base4 ver 1) I noted that validation acc and training acc was at its highest at around the 120th epoch hence I decided to stop training there | |
| #base4 tried l1 l2  #ver3  from tensorflow.keras.layers import Dropout  from tensorflow.keras.layers import BatchNormalization  from tensorflow.keras import regularizers  model = models.Sequential()  model.add(layers.Conv2D(64, (3, 3), activation='relu',  input\_shape=(img\_size, img\_size, 3)))  model.add(Dropout(0.2))  model.add(layers.MaxPooling2D((2, 2)))  model.add(BatchNormalization())  model.add(layers.Conv2D(64, (3, 3), activation='relu',kernel\_regularizer=regularizers.l1\_l2(l1=0.001, l2=0.001)))  model.add(layers.MaxPooling2D((2, 2)))  model.add(BatchNormalization())  model.add(Dropout(0.15))  model.add(layers.Conv2D(128, (3, 3), activation='relu', kernel\_regularizer=regularizers.l1\_l2(l1=0.001, l2=0.001)))  model.add(BatchNormalization())  model.add(layers.Conv2D(128, (3, 3), activation='relu', kernel\_regularizer=regularizers.l1\_l2(l1=0.001, l2=0.001)))  model.add(layers.MaxPooling2D((2, 2)))  model.add(BatchNormalization())  model.add(Dropout(0.15))  model.add(layers.Conv2D(256, (3, 3), activation='relu', kernel\_regularizer=regularizers.l1\_l2(l1=0.001, l2=0.001)))  model.add(layers.MaxPooling2D((2, 2)))  model.add(layers.Flatten())  model.add(layers.Dense(512, activation='relu', kernel\_regularizer=regularizers.l1\_l2(l1=0.001, l2=0.001)))  model.add(BatchNormalization())  model.add(Dropout(0.5))  model.add(layers.Dense(10, activation='softmax'))  model.summary()  model.compile(loss='categorical\_crossentropy',  optimizer=optimizers.Adam(learning\_rate=0.00018),  metrics=['acc']) | | |  | Next, I tried the l1-l2 regularizers on base 4 because it was noted from earlier experimentations that l1\_l2 was the best at reducing divergence between training and validation accuracy.  From the graph it was observed that the divergence between the 2 accuracies was indeed in reduced, however the overall accuracy was also pulled down. | |
| #base 4 readjusted column #addition of 1 256 layer inoput changed to 64  #version 4 increase lr to 0.0002  from tensorflow.keras.layers import Dropout  from tensorflow.keras.layers import BatchNormalization  model = models.Sequential()  model.add(layers.Conv2D(64, (3, 3), activation='relu',  input\_shape=(img\_size, img\_size, 3)))  model.add(Dropout(0.2))  model.add(layers.MaxPooling2D((2, 2)))  model.add(BatchNormalization())  model.add(layers.Conv2D(64, (3, 3), activation='relu', kernel\_regularizer='l2'))  model.add(layers.MaxPooling2D((2, 2)))  model.add(BatchNormalization())  model.add(Dropout(0.15))  model.add(layers.Conv2D(128, (3, 3), activation='relu', kernel\_regularizer='l2'))  model.add(BatchNormalization())  model.add(layers.Conv2D(128, (3, 3), activation='relu', kernel\_regularizer='l2'))  model.add(layers.MaxPooling2D((2, 2)))  model.add(BatchNormalization())  model.add(Dropout(0.15))  model.add(layers.Conv2D(256, (3, 3), activation='relu', kernel\_regularizer='l2'))  model.add(layers.MaxPooling2D((2, 2)))  model.add(layers.Flatten())  model.add(layers.Dense(512, activation='relu', kernel\_regularizer='l2'))  model.add(BatchNormalization())  model.add(Dropout(0.5))  model.add(layers.Dense(10, activation='softmax'))  model.summary()  model.compile(loss='categorical\_crossentropy',  optimizer=optimizers.Adam(learning\_rate=0.0002),  metrics=['acc']) | | |  | Next, I experimented with increasing the learning rate as it was observed to increase the accuracy from earlier experiments.  From the graph it can be seen that the model was not picking up new features at around 40 epochs due to the validation accuracy being relatively stale. | |
| #base 1  #version 4.2hagne lr++  from tensorflow.keras.layers import Dropout  from tensorflow.keras.layers import BatchNormalization  model = models.Sequential()  model.add(layers.Conv2D(32, (3, 3), activation='relu',  input\_shape=(img\_size, img\_size, 3)))  model.add(Dropout(0.2))  model.add(layers.MaxPooling2D((2, 2)))  model.add(BatchNormalization())  model.add(layers.Conv2D(64, (3, 3), activation='relu', kernel\_regularizer='l2'))  model.add(layers.MaxPooling2D((2, 2)))  model.add(BatchNormalization())  model.add(Dropout(0.15))  model.add(layers.Conv2D(128, (3, 3), activation='relu',kernel\_regularizer='l2'))  model.add(layers.MaxPooling2D((2, 2)))  model.add(BatchNormalization())  model.add(Dropout(0.15))  model.add(layers.Conv2D(128, (3, 3), activation='relu', kernel\_regularizer='l2'))  model.add(layers.MaxPooling2D((2, 2)))  model.add(BatchNormalization())  model.add(Dropout(0.15))  model.add(layers.Flatten())  model.add(layers.Dense(512, activation='relu', kernel\_regularizer='l2'))  model.add(BatchNormalization())  model.add(Dropout(0.5))  model.add(layers.Dense(10, activation='softmax'))  model.summary()  model.compile(loss='categorical\_crossentropy',  optimizer=optimizers.Adam(learning\_rate=0.0002),  metrics=['acc']) | | |  | Since we observed an increase in accuracy when the learning rate was increased, I decided to scale it up to 200 epochs to see the maximum potential it could deliver  From the graph it is observed that overfitting occurs at around 123 epochs, the validation accuracy begins dropping. | |
| #base 1  #version 4.2hagne lr++  from tensorflow.keras.layers import Dropout  from tensorflow.keras.layers import BatchNormalization  model = models.Sequential()  model.add(layers.Conv2D(32, (3, 3), activation='relu',  input\_shape=(img\_size, img\_size, 3)))  model.add(Dropout(0.2))  model.add(layers.MaxPooling2D((2, 2)))  model.add(BatchNormalization())  model.add(layers.Conv2D(64, (3, 3), activation='relu', kernel\_regularizer='l2'))  model.add(layers.MaxPooling2D((2, 2)))  model.add(BatchNormalization())  model.add(Dropout(0.15))  model.add(layers.Conv2D(128, (3, 3), activation='relu',kernel\_regularizer='l2'))  model.add(layers.MaxPooling2D((2, 2)))  model.add(BatchNormalization())  model.add(Dropout(0.15))  model.add(layers.Conv2D(128, (3, 3), activation='relu', kernel\_regularizer='l2'))  model.add(layers.MaxPooling2D((2, 2)))  model.add(BatchNormalization())  model.add(Dropout(0.15))  model.add(layers.Flatten())  model.add(layers.Dense(512, activation='relu', kernel\_regularizer='l2'))  model.add(BatchNormalization())  model.add(Dropout(0.5))  model.add(layers.Dense(10, activation='softmax'))  model.summary()  model.compile(loss='categorical\_crossentropy',  optimizer=optimizers.Adam(learning\_rate=0.0002),  metrics=['acc']) | | |  | As noted from the previous model,we should stop the training at 123 epochs, so for this experiment I did exactly that. | |

# Developing Pretrained Image Classification Models

For the development of pretrained image classification models, I will be first picking out the best pretrained network available on keras. Then I will fine-tune the pretrained model by unfreezing the last few layers. I will also be adding dropout prevention techniques to the model.

|  |  |  |
| --- | --- | --- |
| **Basept1ver1**  from tensorflow.keras.applications import InceptionV3  img\_size =150  conv\_base = InceptionV3(weights='imagenet',  include\_top=False,    input\_shape=(img\_size, img\_size, 3))  model = models.Sequential()  model.add(conv\_base)  model.add(layers.Flatten())  model.add(layers.Dense(256, activation='relu'))  model.add(layers.Dense(10, activation='softmax'))    model.compile(loss='categorical\_crossentropy',  optimizer=optimizers.Adam(learning\_rate=2e-5),  metrics=['acc'])  model.summary()  conv\_base.summary() |  | I started off with using the inceptionv3 pretrained model with the weights set as those obtained when training on ImageNet. I made sure to make all layers untrainable so that the weights would not change  As we can see from the graph, overfitting occurs at around 13 epochs, the validation loss starts increasing from its all time low at the 13th epoch indicating overfitting |
| **Basept2ver1**  from tensorflow.keras.applications import ResNet50  img\_size =150  conv\_base = ResNet50(weights='imagenet',  include\_top= False,  input\_shape=(img\_size, img\_size, 3))  model = models.Sequential()  model.add(conv\_base)  model.add(layers.Flatten())  model.add(layers.Dense(256, activation='relu'))  model.add(layers.Dense(10, activation='softmax'))    model.compile(loss='categorical\_crossentropy',  optimizer=optimizers.Adam(learning\_rate=2e-5),  metrics=['acc'])  model.summary()  conv\_base.summary()  for layer in conv\_base.layers:  layer.trainable = False |  | Next, I tested out the resnet50 pretrained model with weights trained on the imagenet dataset. I made sure I froze all the layers before training the model.  From the graph we can observe that overfitting occurs at the 8th epoch. The loss graph shows us an increase in validation loss from its lowest point at the 8th epoch which indicated overfitting. |

I concluded that the better pretrained model was the inceptionv3 model, hence I will be attempting to finetune the model using the base of inceptionv3

|  |  |  |
| --- | --- | --- |
| **Baseptver2**  from tensorflow.keras.applications import InceptionV3  img\_size =150  conv\_base = InceptionV3(weights='imagenet',  include\_top=False,    input\_shape=(img\_size, img\_size, 3))  model = models.Sequential()  model.add(conv\_base)  model.add(layers.Flatten())  model.add(layers.Dense(256, activation='relu'))  model.add(layers.Dense(10, activation='softmax'))    model.compile(loss='categorical\_crossentropy',  optimizer=optimizers.Adam(learning\_rate=2e-5),  metrics=['acc'])  conv\_base.trainable = True  set\_trainable = False  for layer in conv\_base.layers:  if layer.name == 'mixed9':  set\_trainable = True  if set\_trainable:  layer.trainable = True  else:  layer.trainable = False  model.summary()  conv\_base.summary() |  | I first unfroze layers starting from mixed 9 onwards so that it could be trained and have the weights updated base on the Kaggle dataset. This allows the model to pick up features unique to the dataset, inhopes of achieving better accuracy.  From the graph we can observe that unfreezing the layers did not help fix the overfitting issues as we can still overfitting occurring at around the 10th epoch from the loss graph |
| **Basept1ver3**  from tensorflow.keras.applications import InceptionV3  img\_size =150  conv\_base = InceptionV3(weights='imagenet',  include\_top=False,    input\_shape=(img\_size, img\_size, 3))  model = models.Sequential()  model.add(conv\_base)  model.add(layers.Flatten())  model.add(layers.Dense(512, activation='relu'))  model.add(layers.Dense(10, activation='softmax'))    model.compile(loss='categorical\_crossentropy',  optimizer=optimizers.Adam(learning\_rate=2e-5),  metrics=['acc'])  conv\_base.trainable = True  set\_trainable = False  for layer in conv\_base.layers:  if layer.name == 'mixed9':  set\_trainable = True  if set\_trainable:  layer.trainable = True  else:  layer.trainable = False  model.summary()  conv\_base.summary() |  | Next , I increased the dense filters to 512.  From the graph we can observe that the increase of filters in the dense layers caused an increase in volatility of the validation accuracy. |
| **Basept1ver4**  from tensorflow.keras.applications import InceptionV3  img\_size =150  conv\_base = InceptionV3(weights='imagenet',  include\_top=False,    input\_shape=(img\_size, img\_size, 3))  model = models.Sequential()  model.add(conv\_base)  model.add(layers.Flatten())  model.add(layers.Dense(256, activation='relu',kernel\_regularizer ="l2"))  model.add(layers.Dense(10, activation='softmax'))    model.compile(loss='categorical\_crossentropy',  optimizer=optimizers.Adam(learning\_rate=2e-5),  metrics=['acc'])  model.summary()  conv\_base.summary() |  | It was concluded earlier that unfreezing the pretrained model did not help at all so I will stick to using the pre-set weights from the imagenet dataset. I will also attempt to use overfitting prevention techniques to try to delay the overfitting  From the graph we can see that overfitting was greatly reduced,  The loss graph did not show any signs of overfitting, the model could have been trained for 10-20 more epochs before overfitting would be observed |

# Evaluate models using Test images

For the evaluation of models using test images, I will be picking 4 candidates from the models built from scratch to be compared against 1 pretrained model.

When choosing a model in real life, there are many other factors that should be taken into consideration such as the total training time as well as time for prediction (inference time. Some use cases for a neural network require instant predictions, some examples include stock market predictions where every second the market is open can lead to drastic changes in stock prices. In this case there was no use case specified hence we do not need to consider the total training time or the inference time. Hence, we will be selecting the best model purely based on accuracy.

Text

Description automatically generated with medium confidence

The best pretrained model was candidatept.h5 (basept1ver4) and the best model made from scratch was candidate5.h5 (base4ver2)having a score of 86% and 73.6% respectively. This makes the pretrained model candidatept.h5 to be the best model that I will be using to make predictions later.

# Use the Best Model to perform classification

A close-up of some food

Description automatically generated with medium confidence

A bowl of food

Description automatically generated with medium confidence

As discussed in the earlier section, I used basept1ver4 for prediction. For the prediction I decided to do 2 rounds of predictions, one image taken from google search and another image taken from a youtube video. The reason why I took an image from the youtube video is because I can be 100% certain that the image was not included in the training set and can be as authentic as possible. The image take from google could have also existed in the dataset depending on how the dataset was build, hence I needed to do 2 rounds of predictions. In this case, I got the porkchop image from google search and the ramen picture from a video by mikechen from strictly dumplings as can be seen below (exact image at 11:10 of the video) From the prediction results it can be seen that the model performed well for both images, for the porkchop image it was 99.96% confident that it was porkchop, whereas for the ramen the model was 100% sure that the image was ramen.

A pan of food

Description automatically generated with low confidence

# Summary

Overall, I think that the developed model did a good job in predicting images, for the 2 images it had an accuracy of 100% with an average confidence of 99.98% which is very good considering the use case of this model.

**Further Improvements**

There are many aspects of this project that could have been improved on my part. I wanted to improve the project by doing model ensembling. Because we created so many models with similar accuracy, by performing ensemble learning we can combine predictions from these well-performing models to create better predictions.

Also, Due to time constraint, I couldn’t follow the ideal flow when developing the model from scratch. I took a short cut and tested all the overfitting prevention techniques on 1 base model and not on all the new base models created later on. This meant that I assumed that all of the techniques would work the same on different models which is incorrect. I also would have liked to change the ratio of the train test validation data to 80:15:5 so that the dataset had more data to be trained on. It would have also been better if I studied about the mathematics behind the neural network so that I would be able to better pick out certain parameters without testing them.