



Student Name	Sakthignana Sundaram Somaskandan
Student Number	14346091
Email Address	Sakthignana.somaskandan2@mail.dcu.ie
Program of Study	M.Sc. in Computing (Part-time)
Programme Code	MCM
Project Title	Computer Vision Assignment 22/23
Module code	EE544 Computer Vision
Module Coordinator	Prof. Paul F. Whelan
Project Due Date	22 nd March 2023

I/We declare that this material, which I/we now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I/We understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. I/We have read and understood the Assignment Regulations. I/We have identified and included the source of all facts, ideas, opinions, and viewpoints of other in the assignment references. Direct quotations from books, journal articles, internet sources, module text, or any other source whatsoever are acknowledged, and the sources cited are identified in the assignment references. This assignment, or any part of it, has not been previously submitted by me/us or any other person for assessment on this or any other course of study.

I/We have read and understood the referencing guidelines found at <http://www.dcu.ie/info/regulations/plagiarism.shtml>, <https://www4.dcu.ie/students/az/plagiarism> and/or recommended in the assignment guidelines

Name: Sakthignana Sundaram Somaskandan

Date: 22nd March 2023

Table of Contents

Introduction	4
Multi-class image classification	4
Dog breed classification using fine-tuning based transfer learning.....	4
Techniques	4
Artificial Neural Network	4
Activation function	5
Loss function.....	5
Optimisation function	5
Backpropagation.....	5
Convolutional Neural Network (CNN)	5
Convolution	5
Max pooling	6
Flattening.....	6
Dropout.....	6
Dense layer	6
Transfer Learning.....	6
Design	6
Multi-class image classification	6
Data Split	6
Model Architecture	7
Dog breed classification using fine-tuning based transfer learning.....	8
Model Architecture	9
Implementation.....	9
Multi-class image classification	9
Step 1: Prepare the dataset.....	9
Step 2: Split the data into train, test and validation	10
Step 3: Define model architecture	10
Step 4: Fit the model on the train set	11
Step 5: Evaluate the model on the test set	12
Dog breed classification using fine-tuning based transfer learning.....	12
Step 1: Prepare the dataset.....	12
Step 3: Define model architecture	12
Testing, Results & Analysis.....	13
Multi-class image classification	13
Sample Data.....	13
Baseline.....	14
Experiment 1: Data Split	14
Experiment 2: Dropout with early stopping	15
Experiment 3: Data Augmentation.....	15
Experiment 4: Batch Normalisation	15
Experiment 5: L2 Regularisation.....	16
Dog breed classification using fine-tuning based transfer learning.....	16

Sample Data.....	16
Baseline.....	17
<i>Discussion & Conclusion.....</i>	<i>18</i>
Multi-class image classification	18
Dog breed classification using fine-tuning based transfer learning.....	18
<i>GitHub Repo Link</i>	<i>19</i>
<i>References.....</i>	<i>19</i>
<i>Appendix</i>	<i>19</i>
Multi-class image classification	19
Dog breed classification using fine-tuning based transfer learning.....	24

Introduction

The assignment consists of two parts:

1. Multi-class image classification – training from scratch using ImageNette.
2. Dog breed classification using fine-tuning based transfer learning.

Multi-class image classification

ImageNette is a subset of 10 easily classified classes from the ImageNet dataset: tench, English springer, cassette player, chain saw, church, French horn, garbage truck, gas pump, golf ball, and parachute. The dataset has a train/validation split of 1000/500 images per class. The objective of the task is to design, implement and experiment with ways to improve the baseline performance of a *vgglike* network implemented from scratch using the Keras library [1].

Dog breed classification using fine-tuning based transfer learning

ImageWoof contains 10 dog classes from the ImageNet dataset: Australian terrier, Border terrier, Samoyed, Beagle, Shih-Tzu, English foxhound, Rhodesian ridgeback, Dingo, Golden retriever, Old English sheepdog. The dataset has a train/validation split of 1300/50 images. This task uses the Keras library to implement transfer learning and fine-tune the ResNet50 model.

Techniques

Artificial Neural Network

Artificial neural networks (ANN) are inspired by neural activity in biology. The human brain is composed of cells called neurons that exchange signals with one another, forming a very dense and complex network. ANNs are simpler than the neural networks in the brain, but they share a couple of key components in the architecture. The similarities are depicted in Figure 1. Each node can have many inputs with only one output. A neural network comprises of layers of nodes, where each layer communicates with the subsequent layer.

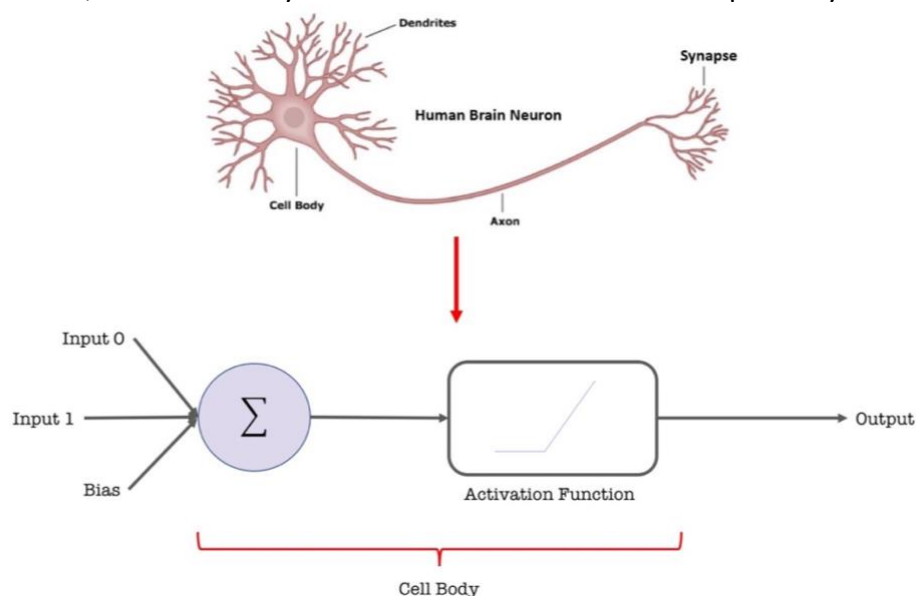


Figure 1: Biological Neuron v Artificial Neuron

Activation function

Activation functions are non-linear functions applied to the summed product of weights and biases of a neuron. This is what enables a neural network to learn a non-linear mapping of the data. There are several different activation functions: sigmoid, tanh and relu.

Loss function

Loss functions are used to quantify the error in the output. It is the difference between the true and the predicted output. The objective of the model is to minimise this measure. There are several different loss functions: cross entropy.

Optimisation function

Optimisation functions are used to update the model parameters (weights and biases) so that the loss function diminishes over epochs. Optimisation functions can be constant learning rate algorithms and adaptive learning rate algorithms. There are several different optimisation functions: rmsprop, adam.

Backpropagation

Backpropagation refers to the feedback process in a neural network after a batch of data is processed. The algorithm updates every parameter in the network depending on its influence on the loss function. The main goal of this process is to find weights and biases that result in the smallest possible error value. The chain rule is used to analytically compute the partial derivatives that are the gradient of the error. For this reason, functions used in a neural network are required to be differentiable.

Convolutional Neural Network (CNN)

Convolutional Neural Networks are also inspired by neurobiology, specifically the visual cortex which processes data only for its local receptive field.

Convolution

Convolution is a technique that slides a kernel across the image and computes the sum of products of each element of the kernel with the corresponding element in the sub-matrix of the image. The process of convolution can be represented mathematically as:

$$(h * w)(x, y) = \sum_{u=-\infty}^{\infty} \sum_{v=-\infty}^{\infty} h(u, v)w(x - u, y - v)$$

$h = \text{image}$

$w = \text{kernel}$

$x, y = \text{discrete variables in two dimensions}$

The size of the output image after convolution is determined by: the size of the input image, the size of the kernel, the stride and the padding. The kernel size defines the field of view of the convolution. The stride refers to the number of steps the kernel takes after each iteration of the convolution process. Each convolution process results in only one output pixel value. Therefore, bigger the stride the smaller the output. Padding is a technique to handle the border of an image in an attempt to retain valid edge pixels.

Max pooling

Pooling is a process of downsampling the input to reduce the computational load and the risk of overfitting. The output size of a pooling layer depends on the strides between the pools. Max pooling is among many other techniques where the maximum value within the pooling window is taken and discards the rest. There are no trainable parameters in a pooling layer. The risk of losing valuable information during this process could impact the model's performance.

Flattening

Flattening is a process of converting a multi-dimensional vector to a one-dimensional vector before passing the extracted features to the classifier.

Dropout

The concept behind dropout is to randomly deactivate a specified percentage of the neurons in the network during training to address the overfitting issue.

Dense layer

A dense layer, also called a fully connected layer, essentially refers to the number of connections a layer has to its subsequent layer. The neurons in a dense layer are connected to every neuron in the subsequent layer and hence do not have a local receptive field like a convolutional layer.

Transfer Learning

Transfer learning is the process of fine-tuning a model for a task that is different to the task on which the model was originally trained. The hyperparameters of a network are saved and stored so that they can be reused for a different task. The advantage of this technique is that an industry-leading model can be trained on a very large dataset, and this model can then be repurposed for a specific task by simply training the last dense layers. Essentially, the idea is to use the pre-trained model as a feature extractor. For this to work, the layers must be left untouched by the new model during training, also known as the freezing of layers.

Design

Multi-class image classification

The system's design to classify images involves using CNNs and Dense layers with activation, loss and optimisation functions to aid learning. The activation, loss and optimisation functions are carefully chosen for the task and the type of images present in the dataset.

Data Split

The data is split into training, validation and test sets. The training set is used to train the model parameters. The validation set is used to fine-tune the model parameters, and the test set is used to evaluate the model on unseen data. The main premise of splitting the data into equally distributed sets is to train and evaluate the model's performance. The ratio of the split depends on the amount of data available. The split ratio used for this task is broken down in

Table 1. Different data splits were explored, and the results can be seen in the Testing, Results & Analysis section.

Table 1: Data Split Information

ImageNette	Train	Validation	Test
Ratio	70	20	10
Samples	3780	1085	535

Model Architecture

The model architecture of a CNN network is depicted in Figure 2.

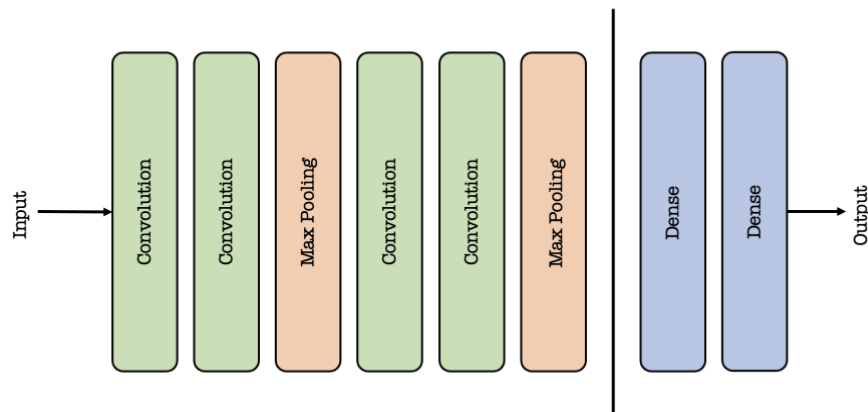


Figure 2: vgg-lite CNN network

The input is an image from the ImageNette dataset. The image gets processed in the following stages:

1. The first convolutional layer with 32 filters and a filter size of 3×3 extracts relevant features from the input and outputs a feature map. Each filter in the layer detects a different feature.
2. The second convolutional layer with 32 filters and a filter size of 3×3 takes the feature map from the first convolutional layer and applies more filters to extract more abstract features.
3. The feature map from the second convolutional layer then gets downsampled by the pooling layer with a pooling size of 2×2 where the maximum value over an input window is taken.
4. The above three steps are repeated once again with 64 filters and filter size of 3×3 for the convolutional layers and downsampled one more time with max pooling with pool size of 2×2 . The aim of this process is to pass only the decisive information to the fully connected (Dense) layer.
5. The three-dimensional image data gets flattened to a one-dimensional vector before the dense layer.
6. The dense layer captures the flattened vector and outputs the probability of the input image belonging to each class using the *softmax* activation function.

Conv2D layers use the following default parameters:

$$strides = (1,1)$$

dilation rate = (1,1)
activation = relu
kernel initializer = glorot_uniform
bias initializer = zero
kernel size = 3

The model is trained on the training dataset for a specified number of epochs and batch size. The batch size refers to the number of samples that propagate through the network at any one time during training. Increasing the batch size decreases the epochs required to train the model but increases the computational resources required. Therefore, it's important to strike a good balance. The batch size chosen is 128.

The objective of the network is to converge and learn a good mapping function that represents the data. In terms of the activation function, ReLU is chosen due to the advantages of computational efficiency and sparse activated network. The loss function chosen is cross entropy. The optimisation function chosen is Adam due to its ability of variable learning rate.

A summary of the model workflow is illustrated in Figure 3.

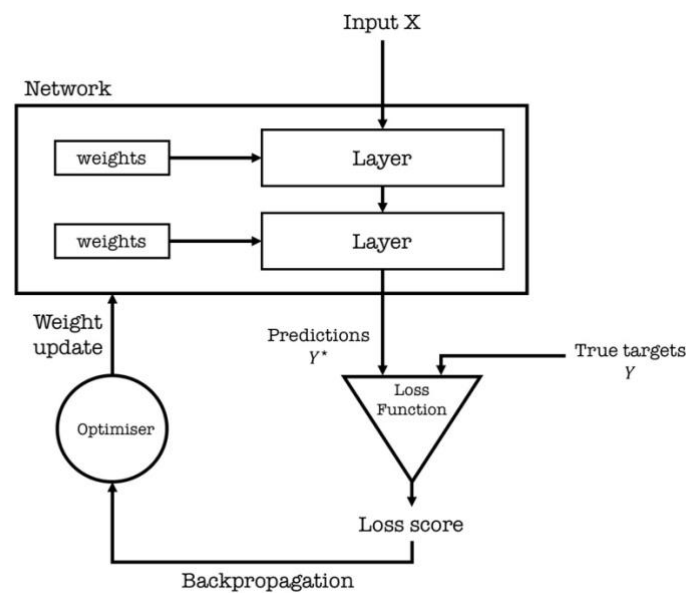


Figure 3: Network workflow

Dog breed classification using fine-tuning based transfer learning

The system's design to classify images for this task involves using the transfer learning technique. The concept of transfer learning is depicted in Figure 4.

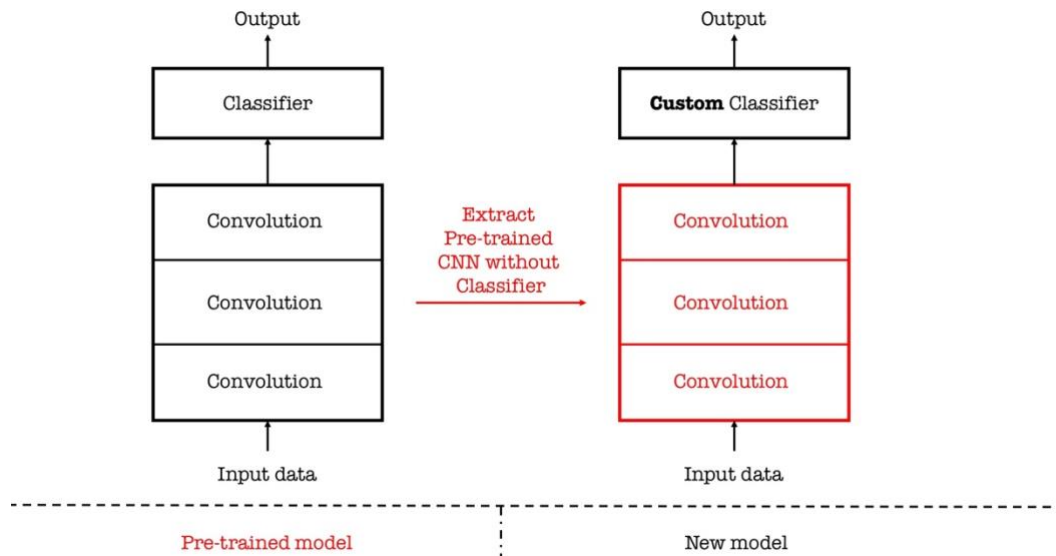


Figure 4: Concept of transfer learning

Model Architecture

The model architecture consists of the pre-trained ResNet50 model and a custom classifier on top of the ResNet50 model. The first 141 layers of the model are frozen, which means the pre-trained weights are used and not fine-tuned. The last 33 layers (res5c block) are unfrozen so that they can be fine-tuned. The model is fine-tuned on the ImageWoof dataset to classify images of different dog breeds present in the dataset. The ResNet50 model takes an input of shape $224 \times 224 \times 3$, where three refers to the number of channels in the image data. The custom classifier consists of a GlobalAveragePooling2D and 3 Dense layers.

Implementation

The full code listing can be found in the Appendix section.

Multi-class image classification

Step 1: Prepare the dataset

Since the dataset contains ten classes, this assignment requires picking four classes as the working dataset. The following four classes were picked [2]:

1. n03445777 – golf ball
2. n03417042 – garbage truck
3. n02979186 – cassette player
4. n03028079 – church

```
def load_dataset(img_folder):
    for dir in os.listdir(img_folder):
        if dir == 'n03445777' or dir == 'n03417042' or dir == 'n02979186' or dir == 'n03028079':
            for file in os.listdir(os.path.join(img_folder, dir)):
                image_path = os.path.join(img_folder, dir, file)
                image = tf.keras.preprocessing.image.load_img(image_path, target_size=(IMG_WIDTH, IMG_HEIGHT))
                image = np.array(image)

                data.append(image)
                labels.append(dir)

load_dataset(PATH_TO_TRAIN_DATA)
load_dataset(PATH_TO_VAL_DATA)
```

Once the classes were selected, the images corresponding to the selected classes had to be loaded into the program to manipulate the data further. The `load_img` function from the Keras library was used to read the images given the `image_path` (the zip file containing the data was uploaded to Google Drive and unzipped using the `tar` command in Colab) and the target size of the images. The target size is 64×64 as the input layer of the model is expected to be an image of size $64 \times 64 \times 3$, where three refers to the number of channels (RGB – colour image).

Step 2: Split the data into train, test and validation

The data was split using the `train_test_split` function from the scikit-learn library, which takes image array, label array, test set size, and the random state as inputs [3]. The random state is for the reproducibility of the output sets. Furthermore, the class distribution of each dataset is also visualised in Colab, as it is a good practice to maintain the same distribution among the datasets for effective model performance.

```
X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.30, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_test, y_test, test_size=0.33, random_state=42)
```

This is a multi-class image classification problem where the model output is the probability value of an input image belonging to each class. To enable this, the labels of the dataset must be one hot encoded representing the true class of an image.

```
def one_hot_encode(y, num_classes):
    le = LabelEncoder()
    return to_categorical(le.fit_transform(y), num_classes)
```

Step 3: Define model architecture

The model architecture outlined in the assignment sheet has been translated to code using the Keras library functions: `Conv2D`, `MaxPooling2D`, `Flatten`, `Dense`, and various other functions were used to define and compile the model, as seen below. The `build_model` function definition takes a few additional parameters to enable experimentation of the model to enhance its performance. The results obtained by running the model can be seen in Step 1: Prepare the dataset

The dataset is prepared the same way, but the dataset is different, and hence the classes picked are also different:

1. n02105641 – Shih-Tzu
2. n02087394 – Beagle
3. n02096294 – English foxhound

Additionally, the target size is 224×224 which is what the `ResNet50` model expects to take as input.

Step 3: Define model architecture

The `ResNet50` model is fetched using the Keras function API with the top of the model excluded and weights associated with the ImageNet dataset. Once the `ResNet50` model is obtained, the initial 143 layers are frozen by setting the trainable parameter to `False`. The custom classifier is defined afterwards using `Dense` and `GlobalAveragePooling2D` layers.

```
def build_model():
    base_model = ResNet50(include_top=False, weights='imagenet')

    for layer in base_model.layers[:143]:
        layer.trainable = False

    # add a global spatial average pooling layer
    x = GlobalAveragePooling2D()(base_model.output)

    # add fully-connected
    x = Dense(1024, activation='relu')(x)
    x = Dense(512, activation='relu')(x)
    out = Dense(3, activation='softmax')(x)

    model = Model(base_model.input, out)

    model.compile(
        optimizer=Adam(),
        loss='categorical_crossentropy',
        metrics=[
            CategoricalAccuracy(),
            Precision(),
            Recall(),
            AUC()
        ]
    )

    return model
```

Once the model is defined, it is trained, and results are visualised. The results can be seen in the Testing, Results & Analysis section.

Testing, Results & Analysis section.

```
def build_model(batchNorm=False, dropOut=False, l2Reg=False):
    model = Sequential()

    if l2Reg:
        model.add(Conv2D(32, kernel_size=3, activation='relu', kernel_regularizer='l2', input_shape=(64,64,3)))
        model.add(Conv2D(32, kernel_size=3, activation='relu', kernel_regularizer='l2'))
    else:
        model.add(Conv2D(32, kernel_size=3, activation='relu', input_shape=(64,64,3)))
        model.add(Conv2D(32, kernel_size=3, activation='relu'))

    model.add(MaxPooling2D(pool_size=(2,2)))

    if batchNorm:
        model.add(BatchNormalization())

    if l2Reg:
        model.add(Conv2D(64, kernel_size=3, activation='relu', kernel_regularizer='l2'))
        model.add(Conv2D(64, kernel_size=3, activation='relu', kernel_regularizer='l2'))
    else:
        model.add(Conv2D(64, kernel_size=3, activation='relu'))
        model.add(Conv2D(64, kernel_size=3, activation='relu'))

    model.add(MaxPooling2D(pool_size=(2,2)))

    if batchNorm:
        model.add(BatchNormalization())

    model.add(Flatten())

    if l2Reg:
        model.add(Dense(512, kernel_regularizer='l2'))
    else:
        model.add(Dense(512))

    if dropOut:
        model.add(Dropout(0.25))

    if l2Reg:
        model.add(Dense(NUM_CLASSES, activation='softmax', kernel_regularizer='l2'))
    else:
        model.add(Dense(NUM_CLASSES, activation='softmax'))

    model.compile(
        optimizer=Adam(),
        loss='categorical_crossentropy',
        metrics=[
            CategoricalAccuracy(),
```

```

        Precision(),
        Recall(),
        AUC()
    ]
)

return model

```

Step 4: Fit the model on the train set

Once the model architecture was defined, the model was fit using the training and validation datasets for 30 epochs. A couple of callback functions were defined to enable `EarlyStopping` and save `ModelCheckpoint`. The checkpoint callback function was used to save the best model weights monitoring the `val_categorical_accuracy`.

```

early_stopping = EarlyStopping(monitor='val_loss', verbose=1, patience=20)
checkpoint      = ModelCheckpoint('/content/drive/MyDrive/EE544    Computer    Vision/task-1-weights.hdf5',    verbose=1,    save_best_only=True,
monitor='val_categorical_accuracy')

baseline_history = baseline_model.fit(
    train_ds,
    epochs=30,
    verbose=1,
    validation_data=val_ds,
    callbacks=[checkpoint]
)

```

The datasets used for model fitting were prepared using the `ImageDataGenerator` function in the Keras library. This was done to rescale the image and optionally add data augmentation techniques to the training dataset.

```

train_datagen = ImageDataGenerator(
    rescale=1./255,
    fill_mode="nearest"
)

val_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

train_ds = train_datagen.flow(
    np.array(X_train), y_train,
    shuffle=True,
    batch_size=128
)

val_ds = val_datagen.flow(
    np.array(X_val), y_val,
    shuffle=True,
    batch_size=128
)

test_ds = test_datagen.flow(
    np.array(X_test), y_test,
    shuffle=False,
    batch_size=128
)

```

Step 5: Evaluate the model on the test set

The evaluation of the model was done on the trained model. The following functions were defined to streamline the evaluation process for the various experiments conducted. The results obtained from executing these functions can be seen in Step 1: Prepare the dataset. The dataset is prepared the same way, but the dataset is different, and hence the classes picked are also different:

4. n02105641 – Shih-Tzu
5. n02087394 – Beagle
6. n02096294 – English foxhound

Additionally, the target size is 224×224 which is what the ResNet50 model expects to take as input.

Step 3: Define model architecture

The ResNet50 model is fetched using the Keras function API with the top of the model excluded and weights associated with the ImageNet dataset. Once the ResNet50 model is obtained, the initial 143 layers are frozen by setting the trainable parameter to False. The custom classifier is defined afterwards using Dense and GlobalAveragePooling2D layers.

```
def build_model():
    base_model = ResNet50(include_top=False, weights='imagenet')

    for layer in base_model.layers[:143]:
        layer.trainable = False

    # add a global spatial average pooling layer
    x = GlobalAveragePooling2D()(base_model.output)

    # add fully-connected
    x = Dense(1024, activation='relu')(x)
    x = Dense(512, activation='relu')(x)
    out = Dense(3, activation='softmax')(x)

    model = Model(base_model.input, out)

    model.compile(
        optimizer=Adam(),
        loss='categorical_crossentropy',
        metrics=[
            CategoricalAccuracy(),
            Precision(),
            Recall(),
            AUC()
        ]
    )

    return model
```

Once the model is defined, it is trained, and results are visualised. The results can be seen in the Testing, Results & Analysis section.

Testing, Results & Analysis section.

```
def plot(train, validation, ylabel, title):
    plt.plot(train, color='red', label='train')
    plt.plot(validation, color='blue', label='validation')
    plt.title(title)
    plt.ylabel(ylabel)
    plt.xlabel('Epoch')
    plt.legend()
    plt.grid(linestyle='-', linewidth=0.5)

def evaluate_and_predict(model):
    # evaluate on test dataset
    eval_results = model.evaluate(test_ds, batch_size=30)

    # print evaluation results
    print("Test loss:", eval_results[0])
    print("Test categorical_accuracy:", eval_results[1])
    print("Test precision:", eval_results[2])
    print("Test recall:", eval_results[3])
    print("Test auc:", eval_results[4])

    # predict
    return model.predict(test_ds)

def draw_confusion_matrix(true, pred):
    cm = confusion_matrix(true.argmax(axis=1), pred.argmax(axis=1))
    sns.heatmap(cm, annot=True, annot_kws={"size": 12, "fmt": "g", "cbar=False"}, cmap="viridis")
    plt.show()

print(classification_report(y_test.argmax(axis=1), pred.argmax(axis=1)))
```

Dog breed classification using fine-tuning based transfer learning

The steps involved in this task are mostly the same as the above task, except for a few changes:

Step 1: Prepare the dataset

The dataset is prepared the same way, but the dataset is different, and hence the classes picked are also different:

7. n02105641 – Shih-Tzu
8. n02087394 – Beagle
9. n02096294 – English foxhound

Additionally, the target size is 224×224 which is what the ResNet50 model expects to take as input.

Step 3: Define model architecture

The ResNet50 model is fetched using the Keras function API with the top of the model excluded and weights associated with the ImageNet dataset. Once the ResNet50 model is obtained, the initial 143 layers are frozen by setting the trainable parameter to False. The custom classifier is defined afterwards using Dense and GlobalAveragePooling2D layers.

```
def build_model():
    base_model = ResNet50(include_top=False, weights='imagenet')

    for layer in base_model.layers[:143]:
        layer.trainable = False

    # add a global spatial average pooling layer
    x = GlobalAveragePooling2D()(base_model.output)

    # add fully-connected
    x = Dense(1024, activation='relu')(x)
    x = Dense(512, activation='relu')(x)
    out = Dense(3, activation='softmax')(x)

    model = Model(base_model.input, out)

    model.compile(
        optimizer=Adam(),
        loss='categorical_crossentropy',
        metrics=[
            CategoricalAccuracy(),
            Precision(),
            Recall(),
            AUC()
        ]
    )

    return model
```





Once the model is defined, it is trained, and results are visualised. The results can be seen in the Testing, Results & Analysis section.

Testing, Results & Analysis

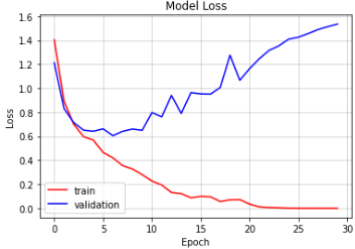
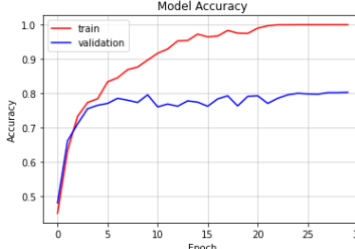
Multi-class image classification

Sample Data

Class	Image
-------	-------


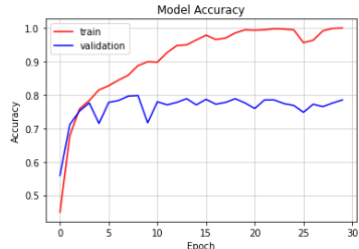
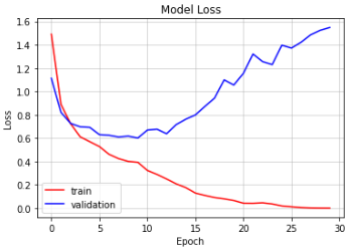
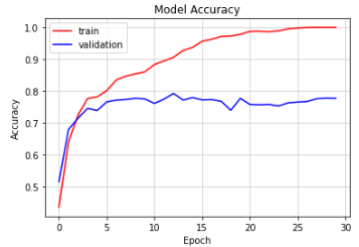
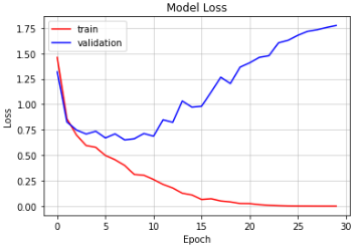
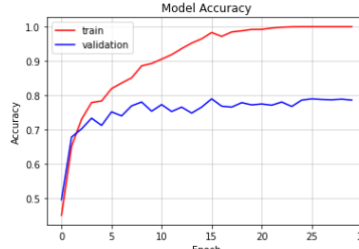
Golf ball		
Garbage truck		
Cassette player		
Church		

Baseline

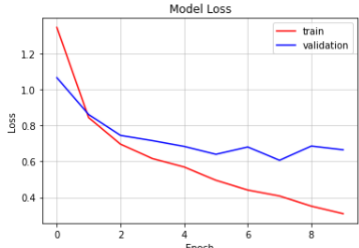
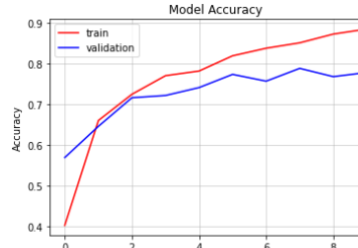
Test Loss	Test Accuracy	Train Loss	Train Accuracy
1.3349	0.8112		

Confusion Matrix					Classification Report				
	0	1	2	3		precision	recall	f1-score	support
0	127	3	6	14	0	0.78	0.85	0.81	150
1	5	100	10	7	1	0.81	0.82	0.82	122
2	14	15	105	2	2	0.85	0.77	0.81	136
3	17	5	3	102	3	0.82	0.80	0.81	127
					accuracy			0.81	535
					macro avg	0.81	0.81	0.81	535
					weighted avg	0.81	0.81	0.81	535

Experiment 1: Data Split

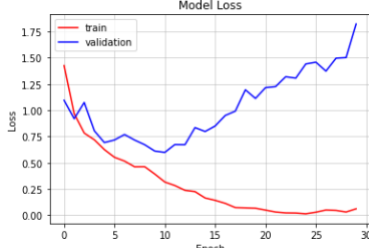
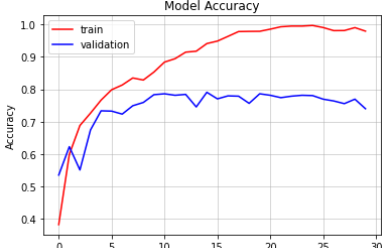
Split	Test Loss	Test Accuracy	Train Loss	Train Accuracy
80/10/10	1.4147	0.7889		
60/30/10	1.4135	0.7722		
70/20/10	1.3286	0.8262		

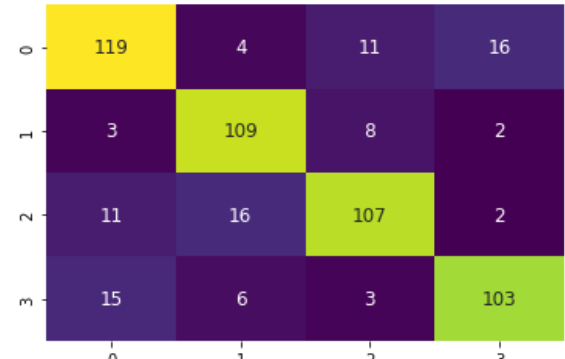
Experiment 2: Dropout with early stopping

Test Loss	Test Accuracy	Train Loss	Train Accuracy
0.5268	0.8262		

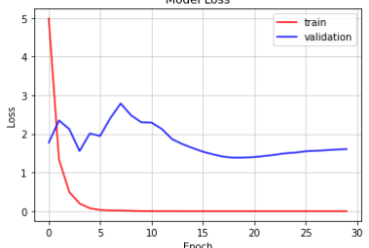
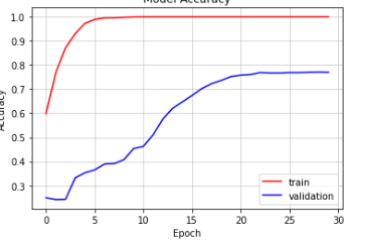
Confusion Matrix		Classification Report			
0	126	3	7	14	
	4	102	9	7	
	11	12	111	2	
	15	6	3	103	
	0	1	2	3	
<pre> precision recall f1-score support 0 0.81 0.84 0.82 150 1 0.83 0.84 0.83 122 2 0.85 0.82 0.83 136 3 0.82 0.81 0.81 127 accuracy 0.83 macro avg 0.83 weighted avg 0.83 </pre>					

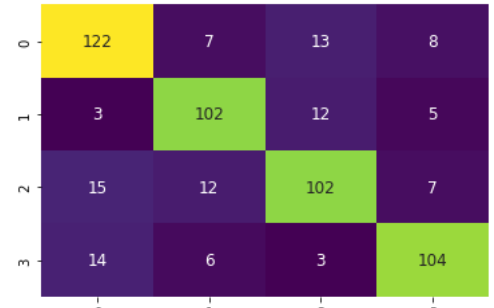
Experiment 3: Data Augmentation

Test Loss	Test Accuracy	Train Loss	Train Accuracy
1.4107	0.7813		

Confusion Matrix	Classification Report
	<pre> precision recall f1-score support 0 0.80 0.79 0.80 150 1 0.81 0.89 0.85 122 2 0.83 0.79 0.81 136 3 0.84 0.81 0.82 127 accuracy 0.82 535 macro avg 0.82 0.82 0.82 535 weighted avg 0.82 0.82 0.82 535 </pre>

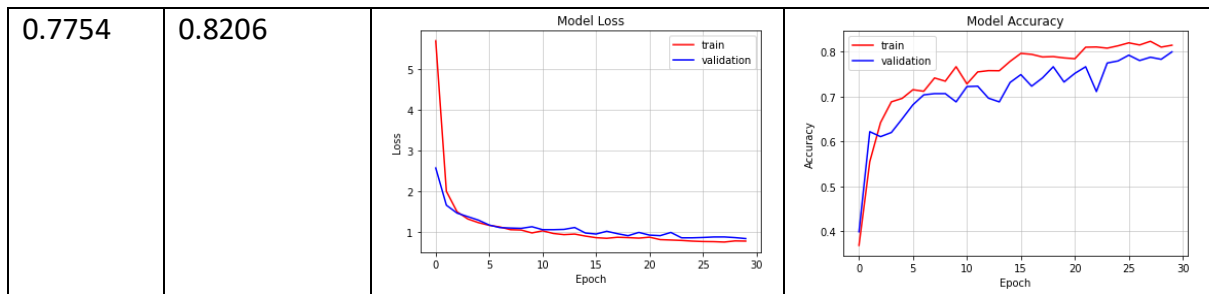
Experiment 4: Batch Normalisation

Test Loss	Test Accuracy	Train Loss	Train Accuracy
1.3470	0.8037		

Confusion Matrix	Classification Report
	<pre> precision recall f1-score support 0 0.79 0.81 0.80 150 1 0.80 0.84 0.82 122 2 0.78 0.75 0.77 136 3 0.84 0.82 0.83 127 accuracy 0.80 535 macro avg 0.80 0.80 0.80 535 weighted avg 0.80 0.80 0.80 535 </pre>

Experiment 5: L2 Regularisation



Test Loss	Test Accuracy	Train Loss	Train Accuracy
-----------	---------------	------------	----------------



Confusion Matrix					Classification Report				
					precision	recall	f1-score	support	
0	123	9	9	9	0	0.84	0.82	0.83	150
1	3	104	8	7	1	0.76	0.85	0.81	122
2	10	16	107	3	2	0.83	0.79	0.81	136
3	11	7	5	104	3	0.85	0.82	0.83	127
	0	1	2	3	accuracy			0.82	535
					macro avg	0.82	0.82	0.82	535
					weighted avg	0.82	0.82	0.82	535

Dog breed classification using fine-tuning based transfer learning

Sample Data

Class	Image
Shih-Tzu	
Beagle	

English foxhound



Baseline

Test Loss	Test Accuracy	Train Loss	Train Accuracy
8.3593	0.3629		

Confusion Matrix	Classification Report
	<pre> precision recall f1-score support 0 0.87 0.19 0.31 139 1 0.38 0.94 0.54 120 2 0.88 0.47 0.62 146 accuracy 0.71 macro avg 0.71 weighted avg 0.73 </pre>

Discussion & Conclusion

Multi-class image classification

Since the baseline model architecture overfitted the data after approximately seven epochs, a few experiments were conducted to overcome the overfitting problem. The following techniques were explored [4]:

1. Dropout with early stopping
2. Data Augmentation
3. Batch Normalisation
4. L2 Regularisation

Dropout is a regularisation technique that prevents overfitting. Dropout randomly drops the specified percentage of neurons from the network during training. When neurons in the network are randomly dropped, it's equivalent to training different networks. The net effect

of different overfit neural networks is to reduce overfitting. Using dropout reduced overfitting with the help of early stopping.

Data augmentation refers to applying augmentation techniques to the dataset: flipping, translation, scaling, adding noise etc. This helps add more variety to the dataset and reduce overfitting by forcing the model to generalise. Though data augmentation didn't fully avoid overfitting, it took more epochs to overfit.

Batch Normalisation normalises the activations of the previous layer at each batch and applies a transformation to maintain mean activation close to 0 and standard deviation close to 1. It speeds up the training process. As seen in the Testing, Results & Analysis section, the training achieves 100% quickly but still overfits the data.

L2 regularisation adds a penalty to the loss function and aims to minimise the squared magnitude of the weights. L2 regularisation was chosen to overcome overfitting because it can learn complex data patterns, and image data is usually considered complex data to be modelled. This is clear in the loss and accuracy plots where the overfitting is solved and provides good data modelling.

Dog breed classification using fine-tuning based transfer learning

Since the baseline model architecture acts strangely, various experiments were conducted using the following techniques:

1. Reduce batch size – 64 instead of 128
2. Data augmentation
3. Try a different optimisation function - rmsprop
4. Try dropout layer
5. Increase training epochs
6. Ensure the ResNet50 model is frozen appropriately
7. Reduce regularisation

The above techniques were explored but didn't improve the baseline model performance. The confusion matrix was always skewed to one of the classes. It could be the amount of data available for training and validation causing the skewed results and inappropriate validation loss curve.

GitHub Repo Link

<https://github.com/sur-sakthy/computer-vision>

References

- [1] "Keras: Deep Learning for humans." <https://keras.io/> (accessed Mar. 19, 2023).
- [2] A. Romanov, "Multiclass Classification with Keras | HackerNoon," *Hackernoon*, 2022. <https://hackernoon.com/multiclass-classification-with-keras> (accessed Mar. 19, 2023).
- [3] "scikit-learn: machine learning in Python — scikit-learn 1.2.2 documentation." <https://scikit-learn.org/stable/index.html> (accessed Mar. 19, 2023).
- [4] A. Sagar, "5 Techniques to Prevent Overfitting in Neural Networks - KDnuggets," *KD Nuggets*, 2019. <https://www.kdnuggets.com/2019/12/5-techniques-prevent->

overfitting-neural-networks.html (accessed Mar. 19, 2023).

Appendix

Multi-class image classification

```
# -*- coding: utf-8 -*-
"""Assignment #1

Automatically generated by Colaboratory.

Original file is located at
https://colab.research.google.com/drive/1FCmz1FvudVnEjvqD0XItgtetbAT8y8xl

# Assignment #1
"""

!nvidia-smi

from google.colab import drive
drive.mount('/content/drive')

!cp '/content/drive/MyDrive/EE544 Computer Vision/imagenette2-160.tgz' ./

# Commented out IPython magic to ensure Python compatibility.
## remove %%capture command to show output from unzipping
## %%capture
# !tar zxvf /content/imagenette2-160.tgz

"""## Imports"""

import os
import glob
import pandas as pd
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
import seaborn as sns

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, Dense, Flatten, MaxPooling2D, BatchNormalization, Dropout
from tensorflow.keras.metrics import CategoricalAccuracy, Precision, AUC, Recall
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import to_categorical, img_to_array
from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_img
from keras.callbacks import EarlyStopping, ModelCheckpoint
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import confusion_matrix, classification_report

"""## Constants"""

PATH_TO_TRAIN_DATA = '/content/imagenette2-160/train'
PATH_TO_VAL_DATA = '/content/imagenette2-160/val'

IMG_WIDTH=64
IMG_HEIGHT=64

NUM_CLASSES=4

all_classes = os.listdir(PATH_TO_TRAIN_DATA)
all_classes

all_classes = os.listdir(PATH_TO_VAL_DATA)
all_classes

"""## Load dataset"""

data = []
labels = []

def load_dataset(img_folder):
    for dir in os.listdir(img_folder):
        if dir == 'n03445777' or dir == 'n03417042' or dir == 'n02979186' or dir == 'n03028079':
            for file in os.listdir(os.path.join(img_folder, dir)):
                image_path = os.path.join(img_folder, dir, file)
                image = tf.keras.preprocessing.image.load_img(image_path, target_size=(IMG_WIDTH, IMG_HEIGHT))
                image = np.array(image)

                data.append(image)
                labels.append(dir)
```

```

load_dataset(PATH_TO_TRAIN_DATA)
load_dataset(PATH_TO_VAL_DATA)

np.unique(labels)

data[0]

"""## Split
Experimenting with the following splits:

Training/Validation/Test:
- 80/10/10
- 60/30/10
- 70/20/10

"""

# X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.20, random_state=42)
# X_val, X_test, y_val, y_test = train_test_split(X_test, y_test, test_size=0.50, random_state=42)

# X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.40, random_state=42)
# X_val, X_test, y_val, y_test = train_test_split(X_test, y_test, test_size=0.25, random_state=42)

X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.30, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_test, y_test, test_size=0.33, random_state=42)

"""## Class distribution"""

df_train = pd.DataFrame(y_train, columns=['label'])
df_train.head()

sns.countplot(data=df_train, x='label')
plt.show()

df_test = pd.DataFrame(y_test, columns=['label'])
df_test.head()

sns.countplot(data=df_test, x='label')
plt.show()

df_val = pd.DataFrame(y_val, columns=['label'])
df_val.head()

sns.countplot(data=df_val, x='label')
plt.show()

"""## Encoding"""

def one_hot_encode(y, num_classes):
    le = LabelEncoder()
    return to_categorical(le.fit_transform(y), num_classes)

y_train[0]

y_train = one_hot_encode(y_train, NUM_CLASSES)

y_train[0]

y_test[0]

y_test = one_hot_encode(y_test, NUM_CLASSES)

y_test[0]

np.array(y_train).shape

y_val[0]

y_val = one_hot_encode(y_val, NUM_CLASSES)

y_val

"""## Create train, validation and test sets"""

train_datagen = ImageDataGenerator(
    rescale=1./255,
    fill_mode="nearest"
)

val_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

# data augmented training data
train_aug_datagen = ImageDataGenerator(
    rescale=1./255,

```

```

        fill_mode="nearest",
        rotation_range=40,
        width_shift_range=0.2,
        height_shift_range=0.2,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True
    )

train_ds = train_datagen.flow(
    np.array(X_train), y_train,
    shuffle=True,
    batch_size=128
)

train_aug_ds = train_datagen.flow(
    np.array(X_train), y_train,
    shuffle=True,
    batch_size=128
)

val_ds = val_datagen.flow(
    np.array(X_val), y_val,
    shuffle=True,
    batch_size=128
)

test_ds = test_datagen.flow(
    np.array(X_test), y_test,
    shuffle=False,
    batch_size=128
)

print('Shape of training samples: ', train_ds.x.shape)
print('Shape of validation samples: ', val_ds.x.shape)
print('Shape of test samples: ', test_ds.x.shape)

print('-----')
print('Shape of augmented training samples: ', train_ds.x.shape)

"""## Build model"""

def build_model(batchNorm=False, dropOut=False, l2Reg=False):
    model = Sequential()

    if l2Reg:
        model.add(Conv2D(32, kernel_size=3, activation='relu', kernel_regularizer='l2', input_shape=(64,64,3)))
        model.add(Conv2D(32, kernel_size=3, activation='relu', kernel_regularizer='l2'))
    else:
        model.add(Conv2D(32, kernel_size=3, activation='relu', input_shape=(64,64,3)))
        model.add(Conv2D(32, kernel_size=3, activation='relu'))

    model.add(MaxPooling2D(pool_size=(2,2)))

    if batchNorm:
        model.add(BatchNormalization())

    if l2Reg:
        model.add(Conv2D(64, kernel_size=3, activation='relu', kernel_regularizer='l2'))
        model.add(Conv2D(64, kernel_size=3, activation='relu', kernel_regularizer='l2'))
    else:
        model.add(Conv2D(64, kernel_size=3, activation='relu'))
        model.add(Conv2D(64, kernel_size=3, activation='relu'))

    model.add(MaxPooling2D(pool_size=(2,2)))

    if batchNorm:
        model.add(BatchNormalization())

    model.add(Flatten())

    if l2Reg:
        model.add(Dense(512, kernel_regularizer='l2'))
    else:
        model.add(Dense(512))

    if dropOut:
        model.add(Dropout(0.5))

    if l2Reg:
        model.add(Dense(NUM_CLASSES, activation='softmax', kernel_regularizer='l2'))
    else:
        model.add(Dense(NUM_CLASSES, activation='softmax'))

    model.compile(
        optimizer=Adam(),

```

```

    loss='categorical_crossentropy',
    metrics=[
        CategoricalAccuracy(),
        Precision(),
        Recall(),
        AUC()
    ]
)

return model

baseline_model = build_model()
baseline_model.summary()

"""## Train model

### Initialise callbacks
"""

early_stopping = EarlyStopping(monitor='val_loss', verbose=1, patience=2)
checkpoint      = ModelCheckpoint('/content/drive/MyDrive/EE544 Computer Vision/task-1-weights.hdf5', verbose=1, save_best_only=True,
monitor='val_categorical_accuracy')

"""### Fit the model"""

baseline_history = baseline_model.fit(
    train_ds,
    epochs=30,
    verbose=1,
    validation_data=val_ds,
    callbacks=[checkpoint]
)

"""## Plot results"""

def plot(train, validation, ylabel, title):
    plt.plot(train, color='red', label='train')
    plt.plot(validation, color='blue', label='validation')
    plt.title(title)
    plt.ylabel(ylabel)
    plt.xlabel('Epoch')
    plt.legend()
    plt.grid(linestyle='-', linewidth=0.5)

plot(baseline_history.history['categorical_accuracy'], baseline_history.history['val_categorical_accuracy'], 'Accuracy', 'Model Accuracy')

plot(baseline_history.history['loss'], baseline_history.history['val_loss'], 'Loss', 'Model Loss')

"""## Evaluate model"""

def evaluate_and_predict(model):
    # evaluate on test dataset
    eval_results = model.evaluate(test_ds, batch_size=30)

    # print evaluation results
    print("Test loss:", eval_results[0])
    print("Test categorical_accuracy:", eval_results[1])
    print("Test precision:", eval_results[2])
    print("Test recall:", eval_results[3])
    print("Test auc:", eval_results[4])

    # predict
    return model.predict(test_ds)

pred = evaluate_and_predict(baseline_model)

pred

def draw_confusion_matrix(true, pred):
    cm = confusion_matrix(true.argmax(axis=1), pred.argmax(axis=1))
    sns.heatmap(cm, annot=True, annot_kws={"size": 12}, fmt='g', cbar=False, cmap="viridis")
    plt.show()

draw_confusion_matrix(y_test, pred)

print(classification_report(y_test.argmax(axis=1), pred.argmax(axis=1)))

"""## Improve the baseline networks performance
1. Batch Normalisation
2. Dropout
3. Data Augmentation
4. Regularisation

### Batch Normalisation
"""

```



```

batchnorm_model = build_model(batchNorm=True)
batchnorm_model.summary()

batchnorm_history = batchnorm_model.fit(
    train_ds,
    epochs=30,
    verbose=1,
    validation_data=val_ds
)

plot(batchnorm_history.history['categorical_accuracy'], batchnorm_history.history['val_categorical_accuracy'], 'Accuracy', 'Model Accuracy')

plot(batchnorm_history.history['loss'], batchnorm_history.history['val_loss'], 'Loss', 'Model Loss')

batchnorm_pred = evaluate_and_predict(batchnorm_model)

draw_confusion_matrix(y_test, batchnorm_pred)

print(classification_report(y_test.argmax(axis=1), batchnorm_pred.argmax(axis=1)))

"""### Dropout"""

dropout_model = build_model(dropOut=True)
dropout_model.summary()

dropout_history = dropout_model.fit(
    train_ds,
    epochs=30,
    verbose=1,
    validation_data=val_ds,
    callbacks=[early_stopping]
)

plot(dropout_history.history['categorical_accuracy'], dropout_history.history['val_categorical_accuracy'], 'Accuracy', 'Model Accuracy')

plot(dropout_history.history['loss'], dropout_history.history['val_loss'], 'Loss', 'Model Loss')

dropout_pred = evaluate_and_predict(dropout_model)

draw_confusion_matrix(y_test, dropout_pred)

print(classification_report(y_test.argmax(axis=1), dropout_pred.argmax(axis=1)))

"""### Data Augmentation"""

dataaug_model = build_model()
dataaug_model.summary()

dataaug_history = dataaug_model.fit(
    train_aug_ds,
    epochs=30,
    verbose=1,
    validation_data=val_ds
)

plot(dataaug_history.history['categorical_accuracy'], dataaug_history.history['val_categorical_accuracy'], 'Accuracy', 'Model Accuracy')

plot(dataaug_history.history['loss'], dataaug_history.history['val_loss'], 'Loss', 'Model Loss')

dataaug_pred = evaluate_and_predict(dataaug_model)

draw_confusion_matrix(y_test, dataaug_pred)

print(classification_report(y_test.argmax(axis=1), dataaug_pred.argmax(axis=1)))

"""### L2 Regularisation"""

l2reg_model = build_model(l2Reg=True)
l2reg_model.summary()

l2reg_history = l2reg_model.fit(
    train_ds,
    epochs=30,
    verbose=1,
    validation_data=val_ds
)

plot(l2reg_history.history['categorical_accuracy'], l2reg_history.history['val_categorical_accuracy'], 'Accuracy', 'Model Accuracy')

plot(l2reg_history.history['loss'], l2reg_history.history['val_loss'], 'Loss', 'Model Loss')

l2reg_pred = evaluate_and_predict(l2reg_model)

draw_confusion_matrix(y_test, l2reg_pred)

print(classification_report(y_test.argmax(axis=1), l2reg_pred.argmax(axis=1)))

```

Dog breed classification using fine-tuning based transfer learning

```
# -*- coding: utf-8 -*-
"""Assignment #2

Automatically generated by Colaboratory.

Original file is located at
https://colab.research.google.com/drive/1UOtifoJPom5yXPWVDypMW0yAGRU0Ue9Q

# Assignment #2
"""

Invidia-smi

from google.colab import drive
drive.mount('/content/drive')

!cp '/content/drive/MyDrive/EE544 Computer Vision/imagewoof2-320.tgz' ./

# Commented out IPython magic to ensure Python compatibility.
# # remove %%capture command to show output from unzipping
# %%capture
# !tar xzvf /content/imagewoof2-320.tgz

"""## Imports"""

import os
import cv2
import numpy as np
import tensorflow as tf
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense, Dropout, AveragePooling2D, Flatten, Input
from tensorflow.keras.models import Model
from tensorflow.keras.metrics import CategoricalAccuracy, Precision, AUC, Recall
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam
from keras.callbacks import EarlyStopping, ModelCheckpoint
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import confusion_matrix, classification_report

"""## Constants"""

PATH_TO_TRAIN_DATA = '/content/imagewoof2-320/train'
PATH_TO_VAL_DATA = '/content/imagewoof2-320/val'

IMG_WIDTH=224
IMG_HEIGHT=224

NUM_CLASSES=3

all_classes = os.listdir(PATH_TO_TRAIN_DATA)
all_classes

all_classes = os.listdir(PATH_TO_VAL_DATA)
all_classes

"""## Load dataset"""

data = []
labels = []

def load_dataset(img_folder):
    for dir in os.listdir(img_folder):
        if dir == 'n02105641' or dir == 'n02087394' or dir == 'n02096294':
            for file in os.listdir(os.path.join(img_folder, dir)):
                image_path = os.path.join(img_folder, dir, file)
                image = tf.keras.preprocessing.image.load_img(image_path, target_size=(IMG_WIDTH, IMG_HEIGHT))
                image = np.array(image)

                data.append(image)
                labels.append(dir)

load_dataset(PATH_TO_TRAIN_DATA)
load_dataset(PATH_TO_VAL_DATA)
```

```

np.unique(labels)

data[0]

np.min(data), np.max(data)

"""## Split dataset
Experimenting with the following splits:

Training/Validation/Test:
- 80/10/10
- 60/30/10
- 70/20/10

"""

X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.20, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_test, y_test, test_size=0.50, random_state=42)

# X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.40, random_state=42)
# X_val, X_test, y_val, y_test = train_test_split(X_test, y_test, test_size=0.25, random_state=42)

# X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.30, random_state=42)
# X_val, X_test, y_val, y_test = train_test_split(X_test, y_test, test_size=0.33, random_state=42)

"""## Class distribution"""

df_train = pd.DataFrame(y_train, columns=['label'])
df_train.head()

sns.countplot(data=df_train, x='label')
plt.title('Train set label distribution')
plt.show()

df_test = pd.DataFrame(y_test, columns=['label'])
df_test.head()

sns.countplot(data=df_test, x='label')
plt.title('Test set label distribution')
plt.show()

df_val = pd.DataFrame(y_val, columns=['label'])
df_val.head()

sns.countplot(data=df_val, x='label')
plt.title('Validation set label distribution')
plt.show()

"""## Encoding"""

def one_hot_encode(y, num_classes):
    le = LabelEncoder()
    return to_categorical(le.fit_transform(y), num_classes)

y_train[0]

y_train = one_hot_encode(y_train, NUM_CLASSES)

y_train[0]

y_test[0]

y_test = one_hot_encode(y_test, NUM_CLASSES)

y_test[0]

np.array(y_train).shape

y_val[0]

y_val = one_hot_encode(y_val, NUM_CLASSES)

y_val

"""## Create train, validation and test sets"""

train_datagen = ImageDataGenerator(
    rescale=1./255,
    fill_mode="nearest"
)

val_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

train_ds = train_datagen.flow(
    np.array(X_train), y_train,

```

```

shuffle=True,
batch_size=128
)

val_ds = val_datagen.flow(
    np.array(X_val), y_val,
    shuffle=True,
    batch_size=128
)

test_ds = test_datagen.flow(
    np.array(X_test), y_test,
    shuffle=False,
    batch_size=128
)

print('Shape of training samples: ', train_ds.x.shape)
print('Shape of validation samples: ', val_ds.x.shape)
print('Shape of test samples: ', test_ds.x.shape)

"""## Build model"""

def build_model():
    base_model = ResNet50(include_top=False, weights='imagenet')

    for layer in base_model.layers[:143]:
        layer.trainable = False

    # add a global spatial average pooling layer
    x = GlobalAveragePooling2D()(base_model.output)

    # add fully-connected
    x = Dense(1024, activation='relu')(x)
    x = Dense(512, activation='relu')(x)
    out = Dense(3, activation='softmax')(x)

    model = Model(base_model.input, out)

    model.compile(
        optimizer=Adam(),
        loss='categorical_crossentropy',
        metrics=[
            CategoricalAccuracy(),
            Precision(),
            Recall(),
            AUC()
        ]
    )

    return model

custom_resnet_model = build_model()

custom_resnet_model.summary()

for i, layer in enumerate(custom_resnet_model.layers):
    print(i, layer.name, "-", layer.trainable)

"""## Train model

### Initialise callbacks
"""

early_stopping = EarlyStopping(monitor='val_loss', verbose=1, patience=20)
checkpoint      = ModelCheckpoint('/content/drive/MyDrive/EE544 Computer Vision/task-2-weights.hdf5', verbose=1, save_best_only=True,
monitor='val_categorical_accuracy')

"""### Fit the model"""

resnet_history = custom_resnet_model.fit(
    train_ds,
    epochs=15,
    verbose=1,
    validation_data=val_ds,
    callbacks=[checkpoint]
)

"""## Plot the results"""

def plot(train, validation, ylabel, title):
    plt.plot(train, color='red', label='train')
    plt.plot(validation, color='blue', label='validation')
    plt.title(title)
    plt.ylabel(ylabel)
    plt.xlabel('Epoch')
    plt.legend()

```

```

plt.grid(linestyle='-', linewidth=0.5)

plot(resnet_history.history['categorical_accuracy'], resnet_history.history['val_categorical_accuracy'], 'Accuracy', 'Model Accuracy')

plot(resnet_history.history['loss'], resnet_history.history['val_loss'], 'Loss', 'Model Loss')

"""## Evaluate model"""

def evaluate_and_predict(model):
    # evaluate on test dataset
    eval_results = model.evaluate(test_ds, batch_size=30)

    # print evaluation results
    print('Test loss:', eval_results[0])
    print('Test categorical_accuracy:', eval_results[1])
    print('Test precision:', eval_results[2])
    print('Test recall:', eval_results[3])
    print('Test auc:', eval_results[4])

    # predict
    return model.predict(test_ds)

pred = evaluate_and_predict(custom_resnet_model)

pred

def draw_confusion_matrix(true, pred):
    cm = confusion_matrix(true.argmax(axis=1), pred.argmax(axis=1))
    sns.heatmap(cm, annot=True, annot_kws={"size": 12}, fmt='g', cbar=False, cmap="viridis")
    plt.show()

draw_confusion_matrix(y_test, pred)

print(classification_report(y_test.argmax(axis=1), pred.argmax(axis=1)))

```