CMP9781M – Big Data Analytics and Modelling

ASSESSMENT COVER PAGE

UNIVERSITY OF LINCOLN

ASSESSMENT NUMBER: 2
Word Count: 1681

NAME OF STUDENT: Edward-John Beazleigh

STUDENT ID NUMBER: BEA15621950

DATE SUBMITTED: 08/05/2023

1.0 Introduction

The purpose of this investigation is to evaluate and compare the efficiency of a multitude of machine learning models in their ability to predict and accurately classify different types of images into 3 separate 'animal' classes. In the specific case of this report, 5 trained machine learning models will be used as multiclass classifiers.

The basic assumption of a multi-class classification task is that each data point will belong to only one of the 'N' (3) classes. When training a machine learning model, it is both advantageous and expected that a data set can provide test data large enough to provide sufficient examples belonging to each class (900) so that the machine learning model can identify and learn the underlying hidden patterns for accurate classification. The dataset used in this report, sufficiently met this criterion, and therefore was selected for this particular task. In this report, 90% of the data was used as training data, and the subsequent 10% for test data, of which is true for each of the five different models.

2.0 Dataset summary

The dataset used in this investigation was sourced from Kaggle, and comprised of 3000 instances, or images, numerically distributed equally between 3 feature classes that each defined a given animal species. The species under investigation were Cats, Dogs and Pandas. The dataset was 400 megabytes in size, this was not just due to the quantity of, but also size of each image, and it should therefore be noted that in the case of each machine learning model used, the images were first resized during the data pre-processing stage, in order to decrease the time taken to run each model.

The data can be found as per the reference link below:

https://www.kaggle.com/datasets/ashishsaxena2209/animal-image-datasetdog-cat-and-panda

3.0 Machine learning models

For the purpose of fluidity in reading this report, coding description has been integrated into the jupyter notebook files, as such making it easier to follow what is being implemented and at what stage by the reader.

In short, all the models followed the same basic structure which commenced with the loading of necessary libraries and the datafile, as well as the pre-processing the data. Then upon completion, the hyperparameters were tuned to find the optimal values. The different hyperparameters of each model being optimised in this report are defined in table 1. Hyperparameter tuning was utilised in this report alongside a 10-fold cross-validation. By using k-fold cross-validation to evaluate the performance of the model with different sets of hyperparameters, we can find the set of hyperparameters that leads to the best performance on average across all folds. These optimised hyperparameters were then used alongside the training data to build an efficient image classification model.

In this report, two methods were applied to employ the 10-fold cross validation which was dependent on the particular training model of use. 'KFold' is one of the methods utilised to split the data into k equally sized folds, where k=10 in this instance. One-fold was used for the validation set and the remaining 9 are used as the training set which allows for an unbiased estimate of a model's performance on new data. 'StratifiedKfold' is variation of Kfold and served as the other method of cross-validation in this report.

Table1: Hyperparameters tuned for each of the image classification models.

Classification Model	*Hyperparameters*
Support Vector machine (SVM)	- C (0.1 or 10), - Gamma (scale or auto)
K-nearest Neighbor (K-NN)	- N_neighbors (3, 5 or 7), - weights (uniform or distance)
Decision tree (DT)	- Max_length (length (1,10))
Convolutional Neural network (CNN)	- Batch size (16, 32 or 64), - Epochs (10, 20 or 30), - Optimizer ('adam' or 'sgd')
Fully connected neural network (FCNN)	- Batch size (16, 32 or 64), - Epochs (10, 20 or 30), - Optimizer ('adam' or 'sgd')

With reference to Table 2, Three scoring metrics were then computed to evaluate and compare the performance of each of the classification models. To further add to the depth of my evaluation, rather than looking simply at the overall accuracy of my model I used the confusion matrix to calculate the individual class accuracy of each image classification model (see table 8).

Table 2: Scoring metrics used to evaluate and compare the image classification models (Chicco and Jurman, 2020).

Scoring Metrics	*Purpose*
Confusion Matrix	- Summarises the performance of a classification model by visualising the number of correct versus incorrect predictions for each class.
Accuracy	- A measure of the proportion of correct predictions made by a model, expressed as a percentage which is used to evaluate the overall performance of a model.
F1 score	- A weighted average of the precision and recall, which are two scoring metrics that quantify the quality of a model's image classification performance.

3.1 Support Vector machine (SVM)

Support Vector machines (SVMs) are a type of supervised machine learning algorithm that can be used for classification and regression tasks. In order to classify an image using an SVM, we first need to extract features from the image, and once extracted, these are utilised as input for the SVM algorithm. The SVM algorithm works by finding the hyperplane that separates the different classes in the feature space (Foody & Mathur, 2004).

3.1.1Data pre-processing & hyperparameter tuning.

```
In [1]: import pandas as pd
        import os
        from skimage.transform import resize
        from skimage.io import imread
        import numpy as np
        from sklearn.model_selection import KFold
        import matplotlib.pyplot as plt
        from sklearn import svm
        from sklearn.svm import SVC
        from sklearn.model_selection import GridSearchCV
        from sklearn.model_selection import train_test_split
        from sklearn.metrics import accuracy_score
        from sklearn.metrics import classification_report

In [2]: # LOAD DATA and preprocess the data
        # animal categories
        categories = ['dog', 'panda', 'cat']
        flat_data_arr=[] #input array
        target_arr=[] #output array
        datadir='/Users/ejbeazleigh/Downloads/Animals/'
        #path which contains all the categories of images
        for i in categories:

            print(f'loading... category : {i}')
            path=os.path.join(datadir,i)
            for img in os.listdir(path):
                img_array=imread(os.path.join(path,img))
                img_resized=resize(img_array,(32,55,3))
                flat_data_arr.append(img_resized.flatten())
                target_arr.append(categories.index(i))
            print(f'loaded category:{i} successfully')
        x=np.array(flat_data_arr)
        y=np.array(target_arr)

        loading... category : dog
```

```
In [7]: from sklearn.model_selection import GridSearchCV
        from sklearn import svm
        from sklearn.svm import SVC
        from sklearn.model_selection import StratifiedKFold, GridSearchCV
        from sklearn.neighbors import KNeighborsClassifier
        from sklearn.metrics import accuracy_score,f1_score, confusion_matrix
        import seaborn as sns

        #initialise the stratfield k-fold 4 crossval with nsplits defining the no. of folds in this case 10
        skf = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
        # defining support vector classifier model
        SVMmodel= SVC(kernel ='rbf')
        # Defining the parameters grid for GridSearchCV
        param_grid = {'C':[0.1,10],'gamma':['scale','auto']}

        # Creating a model using GridSearchCV with the parameters grid SET CV to 2 as only using this for fine tuning so do
        grid_obj = GridSearchCV(SVMmodel,param_grid,cv = 2,n_jobs = -1)
        # checking cell running okay
        print('everythings okay ed!')
        #spltting the data into training and validation sets for each fold of the cross validation
        for fold, (train_index, val_index) in enumerate(skf.split(x,y)):
            trainX, trainY = x[train_index],y[train_index]
            valX, valY = x[val_index], y[val_index]
            print(trainX.shape)
            print(trainY.shape)

        #fit the gridsearch to the training set to help find optimal hyperparamters
            grid_fit = grid_obj.fit(trainX, trainY)
            print('everythings okay ed, gridfit created!')
            SVMopt = grid_fit.best_params_
        #print the best values for each param that will be used moving forward
        print("best params: " + str(grid_obj.best_params_))

        everythings okay ed!
        (2700, 5280)
```

3.1.2 Training the model with 10-fold cross validation and optimised hyperparameters. Printing scoring metrics and computing confusion matrix for evaluation.

```
In [8]: f1List = []
        accuracyList = []
        cmList = []

        #spltting the data into training and validation sets for each fold of the cross valida
        for fold, (train_index, val_index) in enumerate(skf.split(x,y)):
            trainX, trainY = x[train_index],y[train_index]
            valX, valY = x[val_index], y[val_index]
        #intialize the SVM model with new found best hyperparameters
            SVMmodel = SVC(C=SVMopt['C'],gamma=SVMopt['gamma'],kernel='rbf')
        #train thr SVM model based on training set
            SVMmodel.fit(trainX, trainY)
        #use validation set to predict and quantify scoring metrics
            predY = SVMmodel.predict(valX)
            acc = accuracy_score(valY,predY)
            accuracyList.append(acc)
            f1 = f1_score(valY,predY,average='macro')
            f1List.append(f1)
            cm = confusion_matrix(valY,predY)
            cmList.append(cm)

            print(f"Fold{fold+1}:accuracy = {acc}, f1score = {f1}")

        Fold1:accuracy = 0.7033333333333334, f1score = 0.7050957458605455
        Fold2:accuracy = 0.6666666666666666, f1score = 0.6655337735117869
        Fold3:accuracy = 0.6066666666666667, f1score = 0.6067963332658116
        Fold4:accuracy = 0.6566666666666666, f1score = 0.6510292556538507
```

```
In [10]: import matplotlib.pyplot as plt
         from sklearn import svm
         from sklearn.model_selection import GridSearchCV
         from sklearn.model_selection import train_test_split
         from sklearn.metrics import accuracy_score
         from sklearn.metrics import classification_report
         from sklearn.model_selection import StratifiedKFold, GridSearchCV
         from sklearn.neighbors import KNeighborsClassifier
         from sklearn.metrics import accuracy_score,f1_score, confusion_matrix
         import seaborn as sns

         mean_acc = sum(accuracyList)/len(accuracyList)
         mean_f1 = sum(f1List)/len(f1List)
         meanCM = sum(cmList)/len(cmList)

         ax = plt.axes()
         sns.heatmap(meanCM,ax=ax,annot=True,fmt='g',cmap='Purples',square=True)
         fig = plt.figure()
         fig.patch.set_facecolor('xkcd:grey')
         ax.set_title('Support Vector Machine Model confusion matrix')
         ax.set_xticklabels(categories)
         ax.set_yticklabels(categories)
         ax.set_xlabel('Predicted')
         ax.set_ylabel('Actual')
         plt.show()
         print(f"Mean Accuracy: {mean_acc}")
         print(f"Mean f1 Score: {mean_f1}")
```

3.2 K-nearest neighbour (KNN)

It is a non-parametric machine learning and image classification algorithm. As the name seemingly suggests, it finds the “k” nearest data points for a given unknown data point, and the class which is prevailing within those “k” neighbours is predicted as the output class. To find the neighbours, it uses distance metrics like Euclidean distance and the optimal value for k can be found using hyperparameter tuning, in this instance of this report gridsearch.cv will be used (Peterson, 2009). The algorithm directly relies on the distance between feature vectors (In this report, the raw RGB pixel intensities of the images are examined).

3.2.1Data pre-processing

```
In [1]: import os
        import cv2
        import numpy as np
        from sklearn.model_selection import KFold
        from sklearn.neighbors import KNeighborsClassifier
        from sklearn.preprocessing import LabelEncoder
        from sklearn.model_selection import train_test_split
        from sklearn.metrics import classification_report
        from sklearn.model_selection import GridSearchCV

        from imutils import paths #to extract name of each animal-image from there particular di

In [2]: # animal categories
        categories = ['dog', 'panda', 'cat']

        class SimplePreprocessor:
            def __init__(self, width, height, inter=cv2.INTER_AREA):
                # store target image width, height, and interpolation method used when resizing
                self.width = width
                self.height = height
                self.inter = inter

            def preprocess(self, image):
            # resize image to a fixed size, ignoring aspect ratio
                return cv2.resize(image, (self.width, self.height),interpolation=self.inter)
```

```
In [3]: class SimpleDatasetLoader:
            def __init__(self, preprocessors=None):
                # store image preprocessor
                self.preprocessors = preprocessors
        # if preprocessors are None, initialize them as an empty list
                if self.preprocessors is None:
                    self.preprocessors = []

            def load(self, imagePaths, verbose=-1):
                # initialize list of features and labels
                data = []
                labels = []
        # loop over input images
                for (i, imagePath) in enumerate(imagePaths):
                    # load image and extract class label assuming that our path has following format: /path/to/dataset/{clas
                    image = cv2.imread(imagePath)
                    label = imagePath.split(os.path.sep)[-2]
        # check to see if our preprocessors are not None
                    if self.preprocessors is not None:
                        # loop over preprocessors and apply each to image
                        for p in self.preprocessors:
                            image = p.preprocess(image)
        # treat our processed image as a "feature vector"
        # by updating data list followed by the labels
                    data.append(image)
                    labels.append(label)

                    # show an update every `verbose` images
                    if verbose > 0 and i > 0 and (i + 1) % verbose == 0:
                        print("[INFO] processed {}/{}".format(i + 1,
                            len(imagePaths)))

                # return a tuple of data and labels
                return (np.array(data), np.array(labels))
```

```
In [4]: print("[INFO] loading images...")
        imagePaths = list(paths.list_images('/Users/ejbeazleigh/Downloads/Animals/'))

        # initialize image preprocessor, load dataset from disk, and reshape data matrix
        sp = SimplePreprocessor(32, 32)
        sdl = SimpleDatasetLoader(preprocessors=[sp])
        (data, labels) = sdl.load(imagePaths, verbose=500)
        data = data.reshape((data.shape[0], 3072))

        [INFO] loading images...
        [INFO] processed 500/3000
        [INFO] processed 1000/3000
        [INFO] processed 1500/3000
```

3.2.2 Data hyperparameter tuning and training the model with 10-fold cross validation and optimised hyperparameters. Printing scoring metrics and computing confusion matrix for evaluation.

```
In [5]: from sklearn.model_selection import StratifiedKFold, GridSearchCV
        from sklearn.neighbors import KNeighborsClassifier
        from sklearn.metrics import accuracy_score,f1_score, confusion_matrix
        import seaborn as sns

         #initialise the stratfield k-fold 4 crossval with nsplits defining the no. of folds in this case 10
        skf = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
        param_grid = {'n_neighbors':[3,5,7], 'weights': ['uniform','distance']}
        knn = KNeighborsClassifier()
        gridsearch = GridSearchCV(knn, param_grid,cv=skf, n_jobs=1)

        f1List = []
        accuracyList = []
        cmList = []
        #spltting the data into training and validation sets for each fold of the cross validation
        for fold, (train_index, val_index) in enumerate(skf.split(data,labels)):
            trainX, trainY = data[train_index],labels[train_index]
            valX, valY = data[val_index], labels[val_index]
        #fit the gridsearch to the training set to help find optimal hyperparamters
            gridsearch.fit(trainX, trainY)
            best_params = gridsearch.best_params_
        #intialize the KNN model with new found best hyperparameters
            knn = KNeighborsClassifier(n_neighbors=best_params['n_neighbors'],weights=best_params['weights'])
        #train thr knn model based on training set
            knn.fit(trainX, trainY)
        #use validation set to predict and quantify scoring metrics
            predY = knn.predict(valX)
            acc = accuracy_score(valY,predY)
            accuracyList.append(acc)
            f1 = f1_score(valY,predY,average='macro')
            f1List.append(f1)
            cm = confusion_matrix(valY,predY)
            cmList.append(cm)
        #Printing the accuracy and the f1 score for each of the ten folds
            print(f"Fold{fold+1}:accuracy = {acc}, f1score = {f1}")
```

```
In [16]: import matplotlib.pyplot as plt
         from sklearn import svm
         from sklearn.model_selection import GridSearchCV
         from sklearn.model_selection import train_test_split
         from sklearn.metrics import accuracy_score
         from sklearn.metrics import classification_report
         from sklearn.model_selection import StratifiedKFold, GridSearchCV
         from sklearn.neighbors import KNeighborsClassifier
         from sklearn.metrics import accuracy_score,f1_score, confusion_matrix
         import seaborn as sns
         mean_acc = sum(accuracyList)/len(accuracyList)

         mean_f1 = sum(f1List)/len(f1List)
         meanCM = sum(cmList)

         ax = plt.axes()
         sns.heatmap(meanCM,ax=ax,annot=True,fmt='g',cmap='Purples',square=True)
         fig = plt.figure()
         fig.patch.set_facecolor('xkcd:grey')
         ax.set_title('mean K-NN Model confusion matrix')
         ax.set_xticklabels(categories)
         ax.set_yticklabels(categories)
         ax.set_xlabel('Predicted')
         ax.set_ylabel('Actual')
         plt.show()
         print(f"Mean Accuracy: {mean_acc}")
         print(f"Mean f1 Score: {mean_f1}")
```

3.3 *Fully connected neural network (FCNN)*

A fully connected layer refers to a neural network in which every neuron applies a linear transformation to the input vector through a weight matrix. In short, this means every input of the input vector influences every output of the output vector (Hsu, Li & Psaltis, 1990). All possible connections layer to layer are therefore present.

3.3.1Data pre-processing and define the first model.

```
In [3]: # load and preprocess the data, define catergories, images are resized.
        # animal categories
        categories = ['dog', 'panda', 'cat']
        inputArray=[] #input array
        outputArray=[] #output array
        datadir='/Users/ejbeazleigh/Downloads/Animals/'
        #path which contains all the categories of images
        for i in categories:
            print(f'loading... category : {i}')
            path=os.path.join(datadir,i)
            for img in os.listdir(path):
                img_array=imread(os.path.join(path,img))
                img_resized=resize(img_array,(32,55,3))
                inputArray.append(img_resized)
                outputArray.append(categories.index(i))
            print(f'loaded category:{i} successfully')

        loading... category : dog
        loaded category:dog successfully
        loading... category : panda
        loaded category:panda successfully
        loading... category : cat
        loaded category:cat successfully

In [4]: #changing shape of data so that it doesnt corrupt the model,
        # printing shapes and integers to double check everythinf is okay and
        #model will not encounter issues
        from tensorflow.keras.utils import to_categorical
        x=np.array(inputArray, dtype="float")/255.0
        y=np.array(outputArray)
        print(x.shape)
        print(y.shape)
        print(x[0])
        print(y[0])
        y=to_categorical(y,3)
        print(y[0])
```

```
In [8]: #defining the model
        FCNNmodel = Sequential()

        FCNNmodel.add(Flatten(input_shape=(32, 5, 3)))
        FCNNmodel.add(Dense(512, activation='relu'))
        FCNNmodel.add(Dense(256, activation='relu'))
        FCNNmodel.add(Dense(128, activation='relu'))
        FCNNmodel.add(Dropout(0.25))
        FCNNmodel.add(Dense(3, activation='softmax'))

        FCNNmodel.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

        print(FCNNmodel.summary())

        Model: "sequential_2"
        _________________________________________________________________
         Layer (type)                Output Shape              Param #
        =================================================================
         flatten_1 (Flatten)         (None, 480)               0

         dense_4 (Dense)             (None, 512)               246272

         dense_5 (Dense)             (None, 256)               131328

         dense_6 (Dense)             (None, 128)               32896

         dropout_1 (Dropout)         (None, 128)               0

         dense_7 (Dense)             (None, 3)                 387

        =================================================================
        Total params: 410,883
        Trainable params: 410,883
        Non-trainable params: 0
        _________________________________________________________________
```

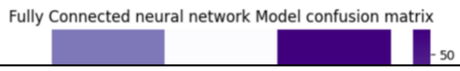
3.3.2 Hyperparameter tuning and redefining the model with optimised hyperparameters.

```
In [17]: # the Sequential object in Keras does not have a get_params() method,
         #which is required by scikit-learn's GridSearchCV to overcome this issue below i have
         #create an instance of KerasClassifier, a wrapper for Keras models that enables them to be used in scikit-learn,
         #using this function.This will then enable me to get the optimal hyperparamaters for my CNN model
         #which i then manually placed into my 'FCNNmodelBest'

         from keras.wrappers.scikit_learn import KerasClassifier
         from sklearn.model_selection import GridSearchCV
         from sklearn.model_selection import KFold
         # Define a function that creates your Keras model
         def create_model(batch_size=16, epochs=10, optimizer='adam'):
             model = Sequential()
             model.add(Flatten(input_shape=(32,55, 3)))
             model.add(Dense(512, activation='relu'))
             model.add(Dense(256, activation='relu'))
             model.add(Dense(128, activation='relu'))
             model.add(Dropout(0.25))
             model.add(Dense(3, activation='softmax'))
             model.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['accuracy'])
             return model

         # Create an instance of KerasClassifier with your create_model function
         keras_model = KerasClassifier(build_fn=create_model)
         # Define the parameters to search over
         param_grid = {'batch_size': [16, 32, 64],'epochs': [10, 20, 30],'optimizer': ['adam', 'sgd']}
         # Create the GridSearchCV object with keras_model as the estimator
         grid = GridSearchCV(estimator=keras_model, param_grid=param_grid, scoring=['accuracy', 'f1'], refit='accuracy', cv=3
         # Fit the GridSearchCV object to your data
         grid_result = grid.fit(x, y)
         # Get the best parameters and the corresponding accuracy and f1 scores
         best_params = grid_result.best_params_
         print("Best parameters: ", best_params)

         188/188 [==============================] - 7s 37ms/step - loss: 0.7383 - accuracy: 0.6173
         Best parameters:  {'batch_size': 16, 'epochs': 10, 'optimizer': 'adam'}

In [13]: from sklearn.model_selection import GridSearchCV
         from sklearn.model_selection import KFold
         #output from last cell : "Best parameters:  {'batch_size': 16, 'epochs': 10, 'optimizer': 'adam'}"

         # Create a new Sequential model and redefining old model with the optimal parameters
         FCNNmodelbest = Sequential()

         FCNNmodelbest.add(Convolution2D(32, (2, 2), activation='relu', input_shape=(32, 55, 3)))
         FCNNmodelbest.add(MaxPooling2D(pool_size=(2, 2)))
         FCNNmodelbest.add(Convolution2D(32, (2, 2), activation='relu'))
         FCNNmodelbest.add(MaxPooling2D(pool_size=(2, 2)))
         FCNNmodelbest.add(Dropout(0.25))
         FCNNmodelbest.add(Flatten())
         FCNNmodelbest.add(Dense(128, activation='relu'))
         FCNNmodelbest.add(Dropout(0.5))
         FCNNmodelbest.add(Dense(3, activation='softmax'))

         FCNNmodelbest.compile(loss='categorical_crossentropy', optimizer='Adam', metrics=['accuracy'])
```

3.3.3 Training the model with 10-fold cross validation and optimised hyperparameters. Printing scoring metrics and computing confusion matrix for evaluation.

```
In [14]: #training the model and printing the scoring metrics
print(x.shape)
print(y.shape)
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import accuracy_score,f1_score, confusion_matrix
import seaborn as sns
from sklearn.model_selection import KFold

f1List = []
accuracyList = []
cmList = []

#initialize the sk-fold 4 crossval with nsplits defining the no. of folds in this case 10
kf = KFold(n_splits=10, shuffle=True, random_state=42)

for fold, (train_index, val_index) in enumerate(kf.split(x, y)):
    trainX, trainY = x[train_index],y[train_index]
    valX, valY = x[val_index], y[val_index]
    #fit model on training data
    FCNNmodelbest.fit(trainX, trainY, batch_size=16, epochs=10, verbose=1)
    # Evaluate the model on the validation set
    loss, acc = FCNNmodelbest.evaluate(valX, valY, verbose=0)
    predY = FCNNmodelbest.predict(valX)
    f1 = f1_score(valY.argmax(axis=1),predY.argmax(axis=1),average='macro')
    accuracyList.append(acc)
    f1List.append(f1)
    cm = confusion_matrix(valY.argmax(axis=1),predY.argmax(axis=1))
    cmList.append(cm)
    print(f"Fold{fold+1}:accuracy = {acc}, f1score = {f1}")
```

```
[16]: #computing the scoring metrics from 10fold results into a confusion matrix
import matplotlib.pyplot as plt
from sklearn import svm
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.model_selection import StratifiedKFold, GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score,f1_score, confusion_matrix
import seaborn as sns

mean_acc = sum(accuracyList)/len(accuracyList)
mean_f1 = sum(f1List)/len(f1List)
meanCM = sum(cmList)/len(cmList)

ax = plt.axes()
sns.heatmap(meanCM,ax=ax,annot=True,fmt='g',cmap='Purples',square=True)
fig = plt.figure()
fig.patch.set_facecolor('xkcd:grey')
ax.set_title('Fully Connected neural network Model confusion matrix')
ax.set_xticklabels(categories)
ax.set_yticklabels(categories)
ax.set_xlabel('Predicted')
ax.set_ylabel('Actual')
plt.show()
print(f"Mean Accuracy: {mean_acc}")
print(f"Mean f1 Score: {mean_f1}")
```



3.4 Convolutional neural network (CNN)

Convolutional networks, also known as convolutional neural networks (CNN) are a specialised kind of neural network for processing data that has a known grid-like topology (LeCun, 1989). Convolutional layers apply to a neural network whereby not all the input nodes in a neuron are connected to the output nodes (Lecun & Bengio, 1995). The number of weights per layer is smaller, which aids high dimensional inputs such as image data, therefore for image classification tasks, convolutional layers provide more flexibility in machine learning (Basha, et al., 2020).

3.4.1 Data pre-processing and define the first model.

```
In [41]: # load the data and preprocess it, set animal class categories
categories = ['dog', 'panda', 'cat']
inputArray=[] #input array
outputArray=[] #output array
datadir='/Users/ejbeazleigh/Downloads/Animals/'
#path which contains all the categories of images
for i in categories:
    print(f'loading... category : {i}')
    path=os.path.join(datadir,i)
    for img in os.listdir(path):
        img_array=imread(os.path.join(path,img))
        img_resized=resize(img_array,(32,55,3))
        inputArray.append(img_resized)
        outputArray.append(categories.index(i))
    print(f'loaded category:{i} successfully')

loading... category : dog
loaded category:dog successfully
loading... category : panda
loaded category:panda successfully
loading... category : cat
loaded category:cat successfully
```

```
In [42]: #changing shape of data so that it doesnt corrupt the model,
# printing shapes and integers to double check everythinf is okay and
#model will not encounter issues
from tensorflow.keras.utils import to_categorical
x=np.array(inputArray, dtype="float")/255.0
y=np.array(outputArray)
print(x.shape)
print(y.shape)
print(x[0])
print(y[0])
y=to_categorical(y,3)
print(y[0])
```

```
In [66]: CNNmodel = Sequential()

CNNmodel.add(Convolution2D(32, (2,2), activation='relu', input_shape=(32, 55, 3)))
CNNmodel.add(MaxPooling2D(pool_size=(2, 2)))
CNNmodel.add(Convolution2D(64, (2, 2), activation='relu'))
CNNmodel.add(MaxPooling2D(pool_size=(2, 2)))
CNNmodel.add(Dropout(0.25))
CNNmodel.add(Flatten())
CNNmodel.add(Dense(128, activation='relu'))
CNNmodel.add(Dropout(0.5))
CNNmodel.add(Dense(3, activation='softmax'))

CNNmodel.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

print(CNNmodel.summary())

Model: "sequential_16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_32 (Conv2D)          (None, 31, 54, 32)        416

 max_pooling2d_32 (MaxPoolin  (None, 15, 27, 32)       0
 g2D)

 conv2d_33 (Conv2D)          (None, 14, 26, 64)        8256

 max_pooling2d_33 (MaxPoolin  (None, 7, 13, 64)        0
 g2D)

 dropout_32 (Dropout)        (None, 7, 13, 64)         0

 flatten_15 (Flatten)        (None, 5824)              0

 dense_32 (Dense)            (None, 128)               745600
```

3.4.2 Hyperparameter tuning and redefining the model with optimised hyperparameters.

```
[71]: # the Sequential object in Keras does not have a get_params() method,
#which is required by scikit-learn's GridSearchCV to overcome this issue below i have
#create an instance of KerasClassifier, a wrapper for Keras models that enables them to be used in scikit-learn,
#using this function.This will then enable me to get the optimal hyperparamaters for my CNN model
#which i then manually placed into my 'CNNmodelBest'

from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import KFold
# Define a function that creates your Keras model
def create_model(batch_size=16, epochs=10, optimizer='adam'):
    model = Sequential()
    model.add(Convolution2D(32, (2, 2), activation='relu', input_shape=(32, 55, 3)))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Convolution2D(32, (2, 2), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))
    model.add(Flatten())
    model.add(Dense(128, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(3, activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['accuracy'])
    return model

# Create an instance of KerasClassifier with your create_model function
keras_model = KerasClassifier(build_fn=create_model)
# Define the parameters to search over
param_grid = {'batch_size': [16, 32, 64],'epochs': [10, 20, 30],'optimizer': ['adam', 'sgd']}
# Create the GridSearchCV object with keras_model as the estimator
grid = GridSearchCV(estimator=keras_model, param_grid=param_grid, scoring=['accuracy', 'f1'], refit='accuracy', cv=3
# Fit the GridSearchCV object to your data
grid_result = grid.fit(x, y)
```

```
188/188 [==============================] - 4s 20ms/step - loss: 1.0987 - accuracy: 0.3347
Epoch 10/10
188/188 [==============================] - 4s 20ms/step - loss: 1.0988 - accuracy: 0.3133
Best parameters:  {'batch_size': 16, 'epochs': 10, 'optimizer': 'adam'}
```

```
77]: from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import KFold
#output from last cell : "Best parameters:  {'batch_size': 16, 'epochs': 10, 'optimizer': 'adam'}"

# Create a new Sequential model with the best parameters
CNNmodelbest = Sequential()

CNNmodelbest.add(Convolution2D(32, (2, 2), activation='relu', input_shape=(32, 55, 3)))
CNNmodelbest.add(MaxPooling2D(pool_size=(2, 2)))
CNNmodelbest.add(Convolution2D(32, (2, 2), activation='relu'))
CNNmodelbest.add(MaxPooling2D(pool_size=(2, 2)))
CNNmodelbest.add(Dropout(0.25))
CNNmodelbest.add(Flatten())
CNNmodelbest.add(Dense(128, activation='relu'))
CNNmodelbest.add(Dropout(0.5))
CNNmodelbest.add(Dense(3, activation='softmax'))

CNNmodelbest.compile(loss='categorical_crossentropy', optimizer='Adam', metrics=['accuracy'])
```

3.4.3 Training the model with 10-fold cross validation and optimised hyperparameters. Printing scoring metrics and computing confusion matrix for evaluation.

```
print(x.shape)
print(y.shape)
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import accuracy_score,f1_score, confusion_matrix
import seaborn as sns
from sklearn.model_selection import KFold

f1List = []
accuracyList = []
cmList = []

#initialize the sk-fold 4 crossval with nsplits defining the no. of folds in this case 10
kf = KFold(n_splits=10, shuffle=True, random_state=42)

for fold, (train_index, val_index) in enumerate(kf.split(x, y)):
    trainX, trainY = x[train_index],y[train_index]
    valX, valY = x[val_index], y[val_index]
    #fit model on training data
    CNNmodelbest.fit(trainX, trainY, batch_size=16, epochs=10)
    # Evaluate the model on the validation set
    loss, acc = CNNmodelbest.evaluate(valX, valY, verbose=0)
    predY = CNNmodelbest.predict(valX)
    f1 = f1_score(valY.argmax(axis=1),predY.argmax(axis=1),average='macro')
    accuracyList.append(acc)
    f1List.append(f1)
    cm = confusion_matrix(valY.argmax(axis=1),predY.argmax(axis=1))
    cmList.append(cm)
    print(f"Fold{fold+1}:accuracy = {acc}, f1score = {f1}")
```

```
Fold10:accuracy = 0.2800000011920929, f1score = 0.14583333333333334
```

```
In [79]: import matplotlib.pyplot as plt
from sklearn import svm
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.model_selection import StratifiedKFold, GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score,f1_score, confusion_matrix
import seaborn as sns

mean_acc = sum(accuracyList)/len(accuracyList)
mean_f1 = sum(f1List)/len(f1List)
meanCM = sum(cmList)/len(cmList)

ax = plt.axes()
sns.heatmap(meanCM,ax=ax,annot=True,fmt='g',cmap='Purples',square=True)
fig = plt.figure()
fig.patch.set_facecolor('xkcd:grey')
ax.set_title('Convolutional neural network Model confusion matrix')
ax.set_xticklabels(categories)
ax.set_yticklabels(categories)
ax.set_xlabel('Predicted')
ax.set_ylabel('Actual')
plt.show()
print(f"Mean Accuracy: {mean_acc}")
print(f"Mean f1 Score: {mean_f1}")
```

3.5 Decision Tree (DT)

A Decision tree consist of nodes and branches. The nodes can further be classified into a root node (starting node of the tree), decision nodes (sub-nodes that split based upon predefined conditions), and leaf nodes (nodes that terminate). A decision tree follows an if-else structure, which means every node uses one and only one independent variable to split into two or more branches. The independent variable which in this instance is categorical (Class), which are used to decide the split of the node (Raschka, Julian and Hearty, 2016, p83-89). The pixel images in this data set are treated as a feature, and the decision tree is trained on a training set of labelled images to predict the class of new unseen data (the validation set).

3.5.1 Data pre-processing and define the model.

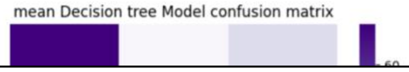
```
In [2]: # LOAD DATA AND and preprocess the data
        # animal categories
        categories = ['dog', 'panda', 'cat']
        flat_data_arr=[] #input array
        target_arr=[] #output array
        datadir='/Users/ejbeazleigh/Downloads/Animals/'
        #path which contains all the categories of images
        for i in categories:

            print(f'loading... category : {i}')
            path=os.path.join(datadir,i)
            for img in os.listdir(path):
                img_array=imread(os.path.join(path,img))
                img_resized=resize(img_array,(32,55,3))
                flat_data_arr.append(img_resized.flatten())
                target_arr.append(categories.index(i))
            print(f'loaded category:{i} successfully')
        x=np.array(flat_data_arr)
        y=np.array(target_arr)

        loading... category : dog
        loaded category:dog successfully
        loading... category : panda
```

```
In [5]: #checking shape is okay
        print(x.shape)
        print(y.shape)

        (3000, 5280)
        (3000,)

In [4]: from sklearn.model_selection import StratifiedKFold, GridSearchCV
        from sklearn.neighbors import KNeighborsClassifier
        from sklearn.metrics import accuracy_score,f1_score, confusion_matrix
        import seaborn as sns

        #initialise the stratfield k-fold 4 crossval
        #with nsplits defining the no. of folds in this case 10
        skf = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
        #Hyperparameters for tuning
        param_grid = [{'max_depth':list(range(1,10))}]
        #define Model
        DTmodel = DecisionTreeClassifier()
        gridsearch = GridSearchCV(DTmodel, param_grid,cv=skf)

        f1List = []
        accuracyList = []
        cmList = []
```

3.5.2 Training the model with 10-fold cross validation and optimised hyperparameters. Printing scoring metrics and computing confusion matrix for evaluation

```
In [5]:
        #SUMMARY ... Iterate through each fold train the model with optimised parameters and calculate metrics
        #splting the data into training and validation sets for each fold of the cross validation
        for fold, (train_index, val_index) in enumerate(skf.split(x,y)):
            trainX, trainY = x[train_index],y[train_index]
            valX, valY = x[val_index], y[val_index]
        ##intialize the DECISON TREE model with new found best hyperparameters
            gridsearch.fit(trainX, trainY)
            best_params = gridsearch.best_params_
            DTmodel = DecisionTreeClassifier(max_depth=best_params['max_depth'])
            #train thr decision tree based on training set
            DTmodel.fit(trainX, trainY)
        #predict on validation set and calculate metrics
            predY = DTmodel.predict(valX)
            acc = accuracy_score(valY,predY)
            accuracyList.append(acc)
            f1 = f1_score(valY,predY,average='macro')
            f1List.append(f1)
            cm = confusion_matrix(valY,predY)
            cmList.append(cm)

            print(f"Fold{fold+1}:accuracy = {acc}, f1score = {f1}")

        Fold1:accuracy = 0.5233333333333333, f1score = 0.523011413487338
        Fold2:accuracy = 0.57, f1score = 0.5729451299919178
        Fold3:accuracy = 0.5533333333333333, f1score = 0.5525428693535143
        Fold4:accuracy = 0.55, f1score = 0.5240332190555949
        Fold5:accuracy = 0.5266666666666666, f1score = 0.5305285755784945
        Fold6:accuracy = 0.53, f1score = 0.5261044176706827
        Fold7:accuracy = 0.5533333333333333, f1score = 0.5570185314016421
        Fold8:accuracy = 0.5433333333333333, f1score = 0.528356703635073
        Fold9:accuracy = 0.5433333333333333, f1score = 0.5329618863049096
        Fold10:accuracy = 0.5366666666666666, f1score = 0.5262368039370557
```

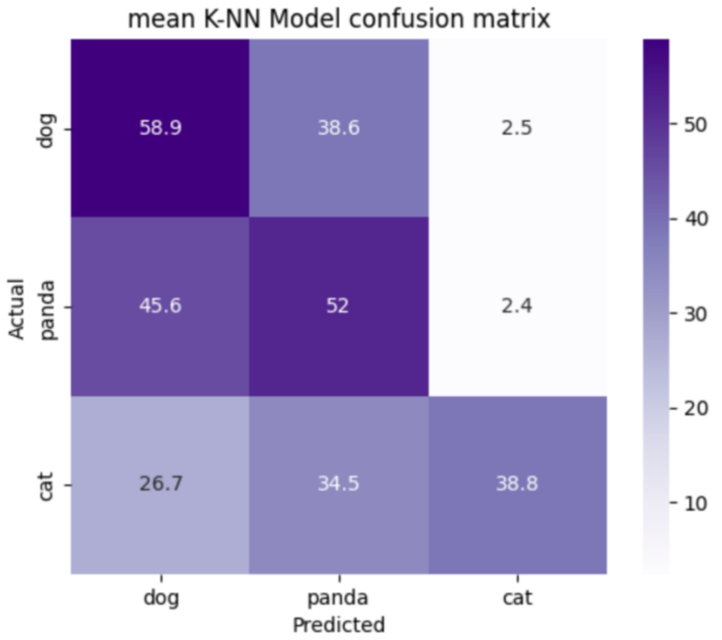
```
In [6]: import matplotlib.pyplot as plt
        from sklearn import svm
        from sklearn.model_selection import GridSearchCV
        from sklearn.model_selection import train_test_split
        from sklearn.metrics import accuracy_score
        from sklearn.metrics import classification_report
        from sklearn.model_selection import StratifiedKFold, GridSearchCV
        from sklearn.neighbors import KNeighborsClassifier
        from sklearn.metrics import accuracy_score,f1_score, confusion_matrix
        import seaborn as sns

        mean_acc = sum(accuracyList)/len(accuracyList)
        mean_f1 = sum(f1List)/len(f1List)
        meanCM = sum(cmList)/len(cmList)

        ax = plt.axes()
        sns.heatmap(meanCM,ax=ax,annot=True,fmt='g',cmap='Purples',square=True)
        fig = plt.figure()
        fig.patch.set_facecolor('xkcd:grey')
        ax.set_title('mean Decision tree Model confusion matrix')
        ax.set_xticklabels(categories)
        ax.set_yticklabels(categories)
        ax.set_xlabel('Predicted')
        ax.set_ylabel('Actual')
        plt.show()
        print(f"Mean Accuracy: {mean_acc}")
        print(f"Mean f1 Score: {mean_f1}")
```



4.0 Evaluation and comparisons of machine learning models for image classification.

4.1 Results

Figure 1: Confusion matrix for mean results of K-nearest neighbour model.

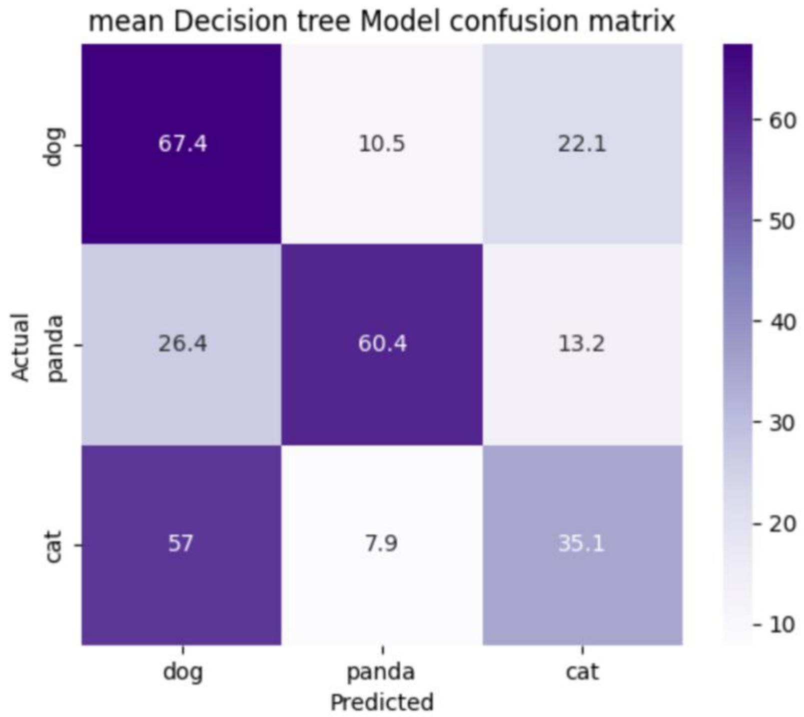



```
<Figure size 640x480 with 0 Axes>

Mean Accuracy: 0.499
Mean f1 Score: 0.5024077598358785
```

Table 3: Performance metrics for model evaluation of K-nearest neighbour classifier

FOLD	ACCURACY (%)	F1 SCORE
1	0.52	0.53
2	0.51	0.52
3	0.51	0.52
4	0.5	0.5
5	0.5	0.5
6	0.48	0.47
7	0.5	0.5
8	0.52	0.52
9	0.49	0.49
10	0.46	0.45
AVERAGE	**0.49**	**0.5**

Figure 2: Confusion matrix for mean results of Decision Tree model




```
<Figure size 640x480 with 0 Axes>

Mean Accuracy: 0.5429999999999999
Mean f1 Score: 0.5373739550416222
```

Table 4: Performance metrics for model evaluation of decision tree classifier

FOLD	ACCURACY (%)	F1 SCORE
1	0.52	0.52
2	0.57	0.57
3	0.55	0.55
4	0.55	0.52
5	0.53	0.53
6	0.55	0.56
7	0.54	0.53
8	0.53	0.52
9	0.54	0.53
10	0.53	0.53
AVERAGE	**0.54**	**0.54**

Figure 3: Confusion matrix for mean results of Support Vector machine model

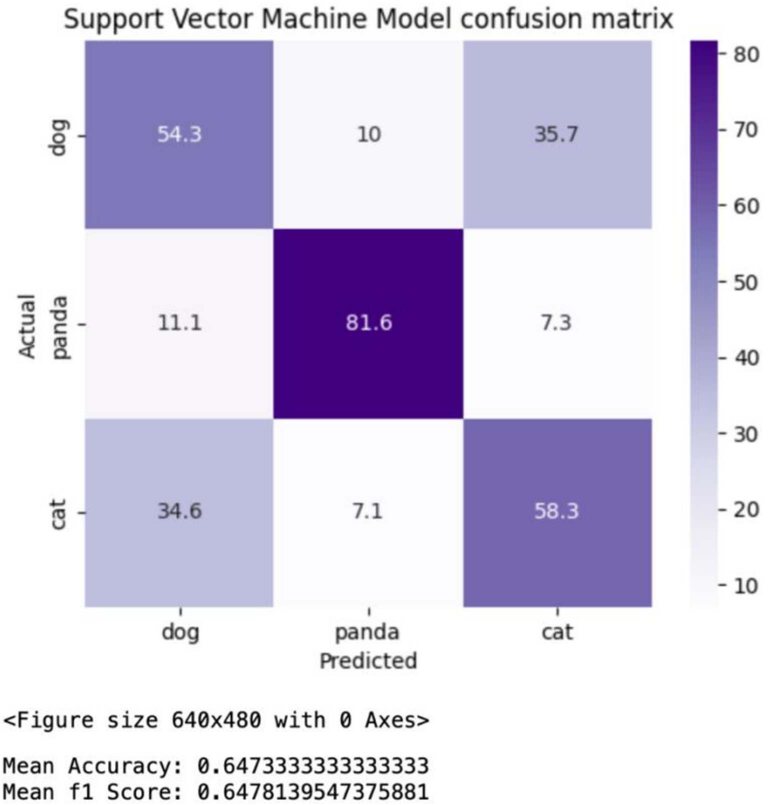
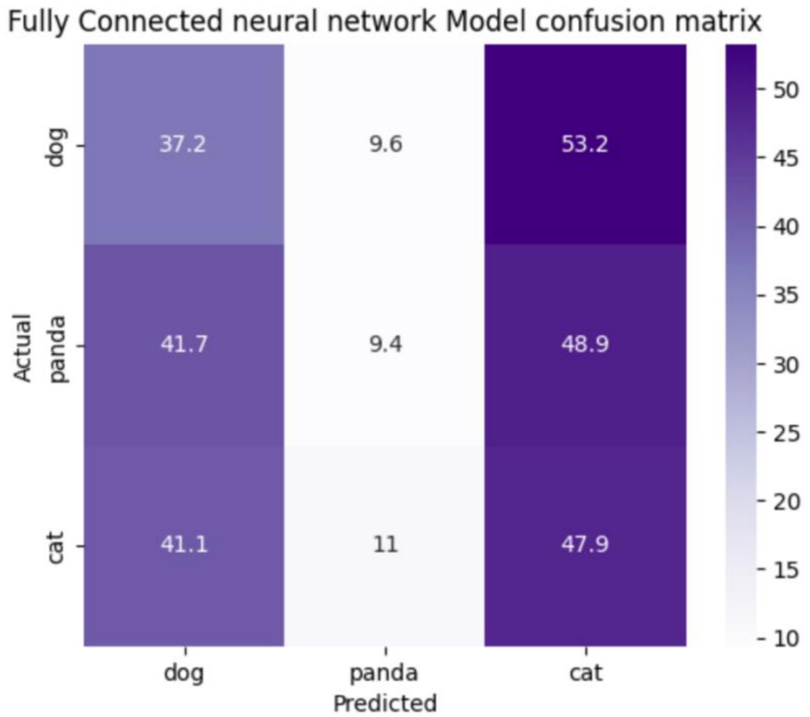


Table 5: Performance metrics for model evaluation of support vector machine classifier

FOLD	ACCURACY (%)	F1 SCORE
1	0.7	0.7
2	0.66	0.67
3	0.6	0.6
4	0.65	0.65
5	0.63	0.63
6	0.61	0.61
7	0.68	0.68
8	0.68	0.68
9	0.62	0.62
10	0.62	0.62
AVERAGE	**0.65**	**0.65**

Figure 4: Confusion matrix for mean results of fully connected neural network model.

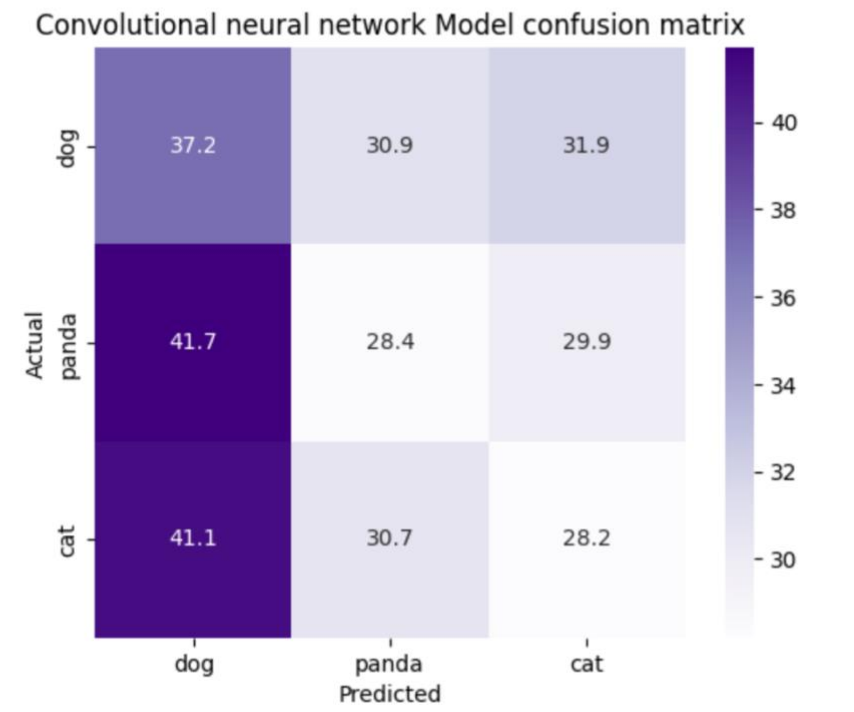



```
<Figure size 640x480 with 0 Axes>

Mean Accuracy: 0.3149999976158142
Mean f1 Score: 0.15964206162127254
```

Table 6: Performance metrics for model evaluation of fully connected neural network classifier.

FOLD	ACCURACY (%)	F1 SCORE
1	0.31	0.16
2	0.31	0.16
3	0.32	0.16
4	0.32	0.16
5	0.32	0.16
6	0.32	0.16
7	0.32	0.16
8	0.33	0.17
9	0.32	0.16
10	0.28	0.15
AVERAGE	**0.31**	**0.16**

Figure 5: Confusion matrix for mean results of Convolutional neural network model




```
<Figure size 640x480 with 0 Axes>

Mean Accuracy: 0.31266666352748873
Mean f1 Score: 0.158754072589611
```

Table 7: Performance metrics for model evaluation of Convolutional neural network classifier.

FOLD	ACCURACY (%)	F1 SCORE
1	0.3	0.16
2	0.31	0.16
3	0.31	0.16
4	0.32	0.16
5	0.33	0.16
6	0.32	0.16
7	0.31	0.16
8	0.32	0.16
9	0.32	0.16
10	0.28	0.15
AVERAGE	**0.32**	**0.16**

Table 8: Individual class accuracy of each image classification model.

Model	Individual Class	Accuracy Scores (%)	
	Dog	**Cat**	**Panda**
K-NN	62.05	59.41	**77.56**
Decision Tree	61.33	80.67	66.67
SVM	**69.33**	**88.33**	71.67
CNN	51.33	55.33	55.33
FCNN	51.33	62.67	48.67
Class Mean	**59.07**	**69.29**	**63.98**

Table 9: Over all image classification model performance

Model	Overall Accuracy (%)	Overall F1 Score (%)
K-NN	0.49	0.5
Decision Tree	0.54	0.54
SVM	**0.65**	**0.65**
CNN	0.31	0.16
FCNN	0.31	0.16
Mean	**0.46**	**0.40**

4.2 Discussion

In terms of overall summary, all the classification models in this report performed poorly relative to the dataset used, with the overall accuracy scores ranging from 0.31 to 0.65 (see table 9). As a generalised rule, when a model achieves below the 70% percentile accuracy threshold, it can be considered as a poor classifier. In short, to improve these scores, as is the case for each of the models, perhaps further feature extractions could be implemented after the pre-processing of data to aid in the training of each classification model.

With reference to table 9, when considering the overall mean accuracy as a representative value of a model's overall performance, the support vector machine (SVM) scored the highest in over accuracy and F1 scoring metrics (0.65 and 0.65) and as such was the best model at classifying the images of the dataset of question in this report. Comparatively, the fully connected neural network (FCNN) scored an overall accuracy of 0.31 and f1 score of 0.16 which classed it as the least favourable method of image classifying. This result is particular alarming as constrastingly there is amassing literature that suggests Convolutional neural networks commonly outperform Support vector machines for image classification task (Hasan, Shafri and Habashi, 2019; Sothe, et al., 2020).The remaining three models (K-NN, DT and CNN) scored an overall accuracy of 0.49%, 0.54% and 0.32% respectively.

With reference to table 8, when considering the specific class efficiency of each model cats were predicted more accurately (69.29%) as a mean across all models than dogs (59.07%) and pandas (63.98). The SVM model was the most efficient model at predicting the dog and cat class (69.33% and 88.33%) whilst the K-NN model was the most efficient model at classifying the panda class (77.56%).

5.0 Possible ways to further improve the classification performance.

In conclusion, from the results of this report the support vector machine (SVM) is suggestibly the most suitable model for the classification task. The following suggestive improvements are written contextually with reference to improving the SVM model, however, may still be applicable to improving the performance of other classification models.

As aforementioned, feature engineering is one such method which could further improve the classification performance of the models used, should this investigation ever be repeated. For image classification, this would involve extracting features such as colour, texture, and shape from the images via techniques such as Harris's corner detection. By using a combination of different feature extraction methods, you can create a better conceptualisation of the images to help the model of use discriminate between the different classes under examination (Lu and Weng, 2007).

To further improve the accuracy of the SVM model, we can take inspiration from research within the last 5 years that has advocated assembling a SVM model alongside another classification model (such as a CNN) can improve model performance (ZHU, et al., 2019; Zhang, et al., 2022).

Furthermore, another advantageous technique that could be adopted to achieve a greater degree of accuracy performance in the SVM model, would be expanding the number of hyperparameter tuning options. For example, in this report due to the nature of how long cells were taking to run, the number of parameter options within the gridsearchCV function was reduced to two for each parameter. To improve the accuracy of SVM model this could therefore be expanded and options for the hyperparameter 'Kernel' could also be consider as per the suggested code example below. It must be noted however this will come at the expense of time, the original code took in excess of 2 hours for the single cell to run through the loop used (see section 3.1.2 of this report for reference).

Quote from original code:

```
param_grid = {'C':[0.1,10],'gamma':['scale', 'auto']}
```

Improved parameter tuning:

```
param_grid = { 'C':[0.1 ,1 ,10 , 100], 'gamma':[ 'scale' , 'auto' ], 'Kernel':[ 'RBF' , 'Linear'}
```

6.0 Referencing

Basha, S.S., Dubey, S.R., Pulabaigari, V. and Mukherjee, S., 2020. Impact of fully connected layers on performance of convolutional neural networks for image classification. *Neurocomputing*, *378*, pp.112-119.

Chicco, D. and Jurman, G., 2020. The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation. *BMC genomics*, *21*, pp.1-13.

Foody, G.M. and Mathur, A., 2004. A relative evaluation of multiclass image classification by support vector machines. *IEEE Transactions on geoscience and remote sensing*, *42*(6), pp.1335-1343.

Hasan, H., Shafri, H.Z. and Habshi, M., 2019, November. A comparison between support vector machine (SVM) and convolutional neural network (CNN) models for hyperspectral image classification. In *IOP Conference Series: Earth and Environmental Science* (Vol. 357, No. 1, p. 012035). IOP Publishing.

Hsu, K.Y., Li, H.Y. and Psaltis, D., 1990. Holographic implementation of a fully connected neural network. *Proceedings of the IEEE*, *78*(10), pp.1637-1645.

https://www.kaggle.com/datasets/ashishsaxena2209/animal-image-datasetdog-cat-and-panda

LeCun, Y. and Bengio, Y., 1995. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, *3361*(10), p.1995.

LeCun, Y., 1989. Generalization and network design strategies. *Connectionism in perspective*, *19*(143-155), p.18.

Lu, D. and Weng, Q., 2007. A survey of image classification methods and techniques for improving classification performance. *International journal of Remote sensing*, *28*(5), pp.823-870.

Peterson, L.E., 2009. K-nearest neighbor. *Scholarpedia*, *4*(2), p.1883.

Raschka, S., Julian, D. and Hearty, J., 2016. *Python: deeper insights into machine learning*. Packt Publishing Ltd.

Sothe, C., De Almeida, C.M., Schimalski, M.B., La Rosa, L.E.C., Castro, J.D.B., Feitosa, R.Q., Dalponte, M., Lima, C.L., Liesenberg, V., Miyoshi, G.T. and Tommaselli, A.M.G., 2020. Comparative performance of convolutional neural network, weighted and conventional support vector machine and random forest for classifying tree species using hyperspectral and photogrammetric data. *GIScience & Remote Sensing*, *57*(3), pp.369-394.

Zhang, W., Li, L., Zhu, Y., Yu, P. and Wen, J., 2022. CNN-LSTM neural network model for fine-grained negative emotion computing in emergencies. *Alexandria Engineering Journal*, *61*(9), pp.6755-6767.

Zhu, C., Song, F., Wang, Y., Dong, H., Guo, Y. and Liu, J., 2019. Breast cancer histopathology image classification through assembling multiple compact CNNs. *BMC medical informatics and decision making*, *19*(1), pp.1-17.

WEBPAGE References:

https://www.kaggle.com/general/197993

https://stackoverflow.com/questions/72101295/python-gridsearchcv-taking-too-long-to-finish-running

https://medium.com/grabngoinfo/support-vector-machine-svm-hyperparameter-tuning-in-python-a65586289bcb

https://towardsdatascience.com/all-the-steps-to-build-your-first-image-classifier-with-code-cf244b015799

https://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html#sphx-glr-auto-examples-classification-plot-classifier-comparison-py

https://www.kaggle.com/code/bygbrains/dog-cat-pandas-image-classifier

https://www.kaggle.com/datasets/ashishsaxena2209/animal-image-datasetdog-cat-and-panda/code

https://pyimagesearch.com/2021/04/17/your-first-image-classifier-using-k-nn-to-classify-images/

https://www.sciencedirect.com/science/article/pii/S2772662222000261

https://www.deeplearningbook.org

https://scikit-learn.org/stable/modules/svm.html#classification

https://confusionmatrixonline.com