Introduction:

After understanding how regression models and classifications models can be accomplished with neural networks, it is time to understand a different type of neural network. This new type of neural network is mostly used in image processing and it is known as Convolutional Neural Network, (CNN). These neural networks can be used to classify images, hence why this project will focus on two CNN models that will attempt to classify Pandas vs Dogs & Pandas vs Cats. The data for this project will be obtained from one of the kaggle datasets which contains 1000 images for each of the animals (Pandas, Dogs, Cats). The goal of this project is to understand how CNN's perform in image classification tasks and what is the difference in difficulty between the two CNN's.

Procedures:

In this project the first step is to set up the data by importing into google colab from kaggle and making a tensorflow dataset for each of the animals (Pandas, Dogs, Cats). Once this step has been completed then the data will not be used if it is removed from the tensorflow dataset otherwise, the model will not train as intended. Following this step, comes the creation of the CNN model and the training/evaluation of it. After the first model has been completed, an attempt to improve this model will be made by using a regularization method (dropout, L1, L2). Once the second model has been completed, a 3rd improvement will be attempted by using data augmentation. This last step marks the end of a set of CNN's models that can be used to classify Pandas vs Dogs; hence the same steps are repeated but for a set of CNN's models that can be used to classify Pandas vs Cats.

Results:

In this section the plots and final values pertaining to each model will be presented. In addition the structure of the network alongside its complete description (i.e number of layers, activation function, processing elements, etc.) will be encompassed in this section. Finally a brief description will be provided regarding the changes that were made between from one model to the next.

II.1 CNN Model 1 (pvmd1)

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 180, 180, 3)]     0

 rescaling (Rescaling)       (None, 180, 180, 3)       0

 conv2d (Conv2D)             (None, 178, 178, 32)      896

 max_pooling2d (MaxPooling2D  (None, 89, 89, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 87, 87, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 43, 43, 64)       0
 2D)

 conv2d_2 (Conv2D)           (None, 41, 41, 128)       73856

 max_pooling2d_2 (MaxPooling  (None, 20, 20, 128)      0
 2D)

 conv2d_3 (Conv2D)           (None, 18, 18, 256)       295168

 max_pooling2d_3 (MaxPooling  (None, 9, 9, 256)        0
 2D)

 conv2d_4 (Conv2D)           (None, 7, 7, 256)         590080

 flatten (Flatten)           (None, 12544)             0

 dense (Dense)               (None, 1)                 12545

=================================================================
Total params: 991,041
Trainable params: 991,041
Non-trainable params: 0
```

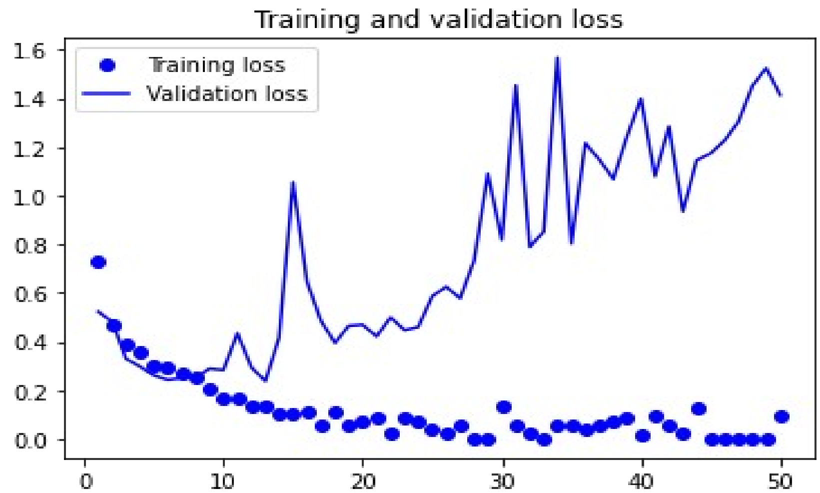

Figure 1: Preliminary Loss (Training and Validation) vs Epochs

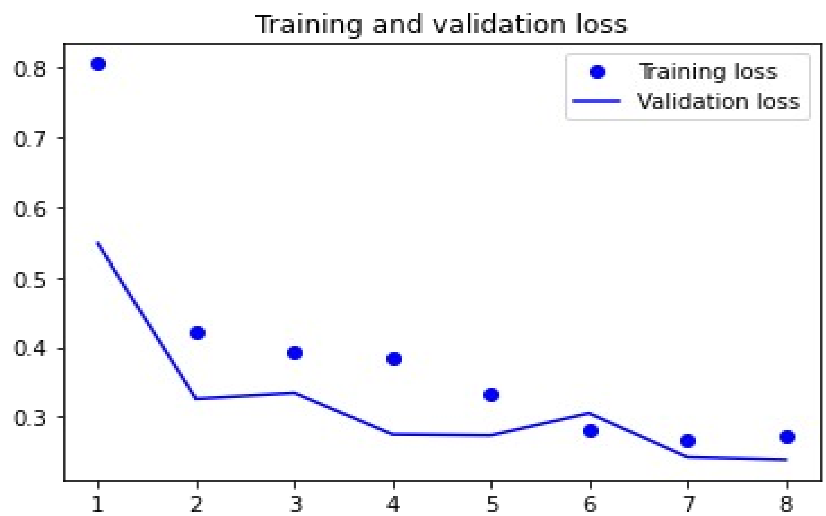

Figure 2: Final Loss (Training and Validation) vs Epochs

```
Final Training loss:  0.27349179983139040
Final Training Accuracy:  0.8964285850524902
Final Validation loss:  0.23817463219165802
Final Validation Accuracy:  0.8949999809265137
7/7 [==============================] - 0s 13ms/step - loss: 0.1917 - accuracy: 0.9050 [0.1916598081588745, 0.9049999713897705]
```

II.2 CNN Model 2 (pvmd2)

```
Model: "model_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_7 (InputLayer)        [(None, 180, 180, 3)]     0

 rescaling_6 (Rescaling)     (None, 180, 180, 3)       0

 conv2d_30 (Conv2D)          (None, 178, 178, 32)      896

 max_pooling2d_24 (MaxPoolin  (None, 89, 89, 32)       0
 g2D)

 conv2d_31 (Conv2D)          (None, 87, 87, 64)        18496

 max_pooling2d_25 (MaxPoolin  (None, 43, 43, 64)       0
 g2D)

 conv2d_32 (Conv2D)          (None, 41, 41, 128)       73856

 max_pooling2d_26 (MaxPoolin  (None, 20, 20, 128)      0
 g2D)

 conv2d_33 (Conv2D)          (None, 18, 18, 256)       295168

 max_pooling2d_27 (MaxPoolin  (None, 9, 9, 256)        0
 g2D)

 conv2d_34 (Conv2D)          (None, 7, 7, 256)         590080

 flatten_6 (Flatten)         (None, 12544)             0

 dense_6 (Dense)             (None, 1)                 12545

=================================================================
Total params: 991,041
Trainable params: 991,041
Non-trainable params: 0
```

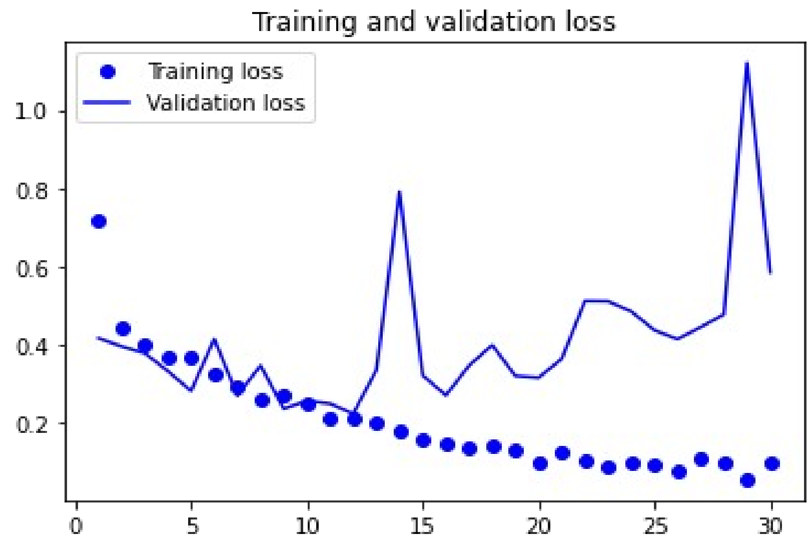

Figure 3: Preliminary Loss (Training and Validation) vs Epochs

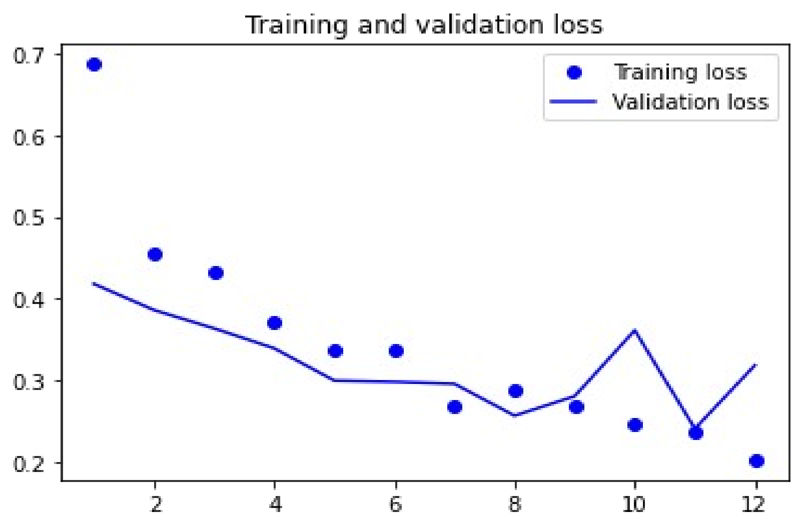

Figure 4: Final Loss (Training and Validation) vs Epochs

```
Final Training loss:  0.20187591016292572
Final Training Accuracy:  0.9314285516738892
Final Validation loss:  0.31808987259864807
Final Validation Accuracy:  0.8899999856948853
7/7 [==============================] - 0s 14ms/step - loss: 0.2729 -
accuracy: 0.9200 [0.2729094624519348, 0.9200000166893005]
```

II.3 CNN Model 3 (pvdm3)

```
Model: "model_11"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_12 (InputLayer)       [(None, 180, 180, 3)]     0         
                                                                 
 sequential_3 (Sequential)   (None, 180, 180, 3)       0         
                                                                 
 rescaling_11 (Rescaling)    (None, 180, 180, 3)       0         
                                                                 
 conv2d_55 (Conv2D)          (None, 178, 178, 32)      896       
                                                                 
 max_pooling2d_44 (MaxPoolin  (None, 89, 89, 32)       0         
 g2D)                                                            
                                                                 
 dropout_3 (Dropout)         (None, 89, 89, 32)        0         
                                                                 
 conv2d_56 (Conv2D)          (None, 87, 87, 64)        18496     
                                                                 
 max_pooling2d_45 (MaxPoolin  (None, 43, 43, 64)       0         
 g2D)                                                            
                                                                 
 dropout_4 (Dropout)         (None, 43, 43, 64)        0         
                                                                 
 conv2d_57 (Conv2D)          (None, 41, 41, 128)       73856     
                                                                 
 max_pooling2d_46 (MaxPoolin  (None, 20, 20, 128)      0         
 g2D)                                                            
                                                                 
 dropout_5 (Dropout)         (None, 20, 20, 128)       0         
                                                                 
 conv2d_58 (Conv2D)          (None, 18, 18, 256)       295168    
                                                                 
 max_pooling2d_47 (MaxPoolin  (None, 9, 9, 256)        0         
 g2D)                                                            
                                                                 
 dropout_6 (Dropout)         (None, 9, 9, 256)         0         
                                                                 
 conv2d_59 (Conv2D)          (None, 7, 7, 256)         590080    
                                                                 
 flatten_11 (Flatten)        (None, 12544)             0         
                                                                 
 dropout_7 (Dropout)         (None, 12544)             0         
                                                                 
 dense_11 (Dense)            (None, 1)                 12545     
                                                                 
=================================================================
Total params: 991,041
Trainable params: 991,041
Non-trainable params: 0
```

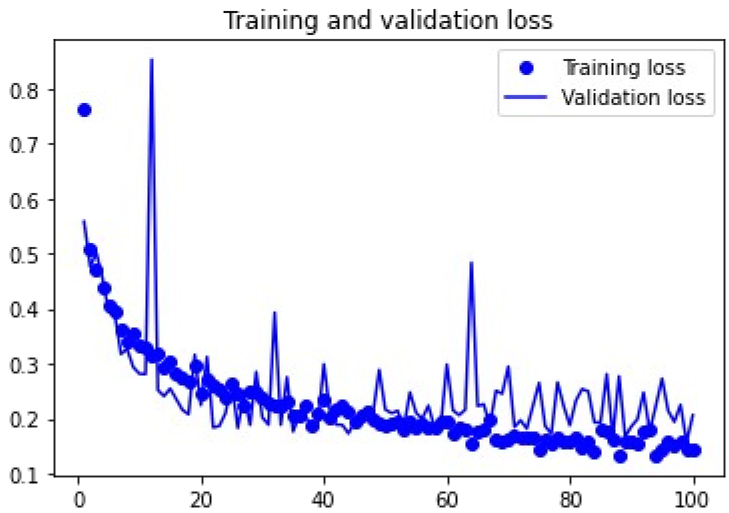

Figure 5: Preliminary Loss (Training and Validation) vs Epochs

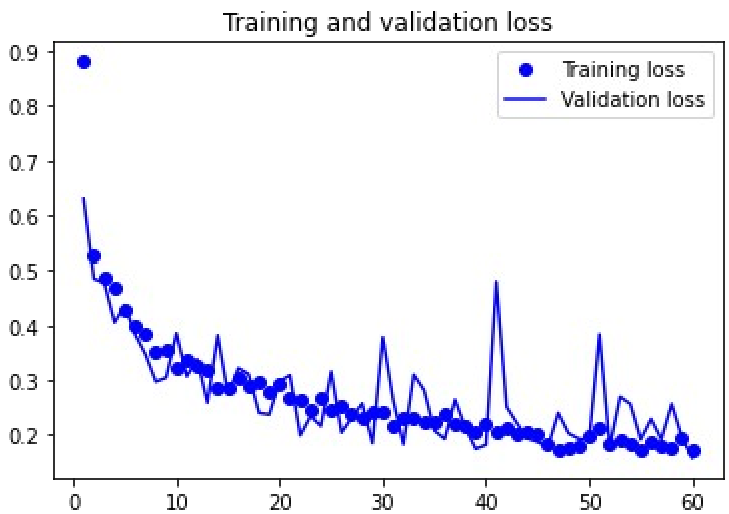

Figure 6: Final Loss (Training and Validation) vs Epochs

```
Final Training loss:  0.17143961787223816
Final Training Accuracy:  0.9457142949104309
Final Validation loss:  0.15801990032196045
Final Validation Accuracy:  0.9524999856948853
7/7 [==============================] - 0s 14ms/step - loss: 0.1511 -
accuracy: 0.9600 [0.15106220543384552, 0.9599999785423279]
```

Discussion/Observations of Panda vs Dog model :

In the first CNN model (pvdm1) the model that was built consisted of no regularization methods and data augmentation. Once the model went through the first 30 epochs (preliminary) it was decided that the best time to stop the training was in epoch 8. The final values obtained from evaluating the test dataset was (`loss: 0.1917 - accuracy: 0.9050`). After obtaining these values it was determined that a second model that would use regularization methods will be implemented in order to improve the accuracy of the CNN model. Hence, the second CNN model (pvdm2) was built. After trying different combinations of regularization methods it was determined that *L2(.0001)* improved the model's accuracy. Once the model went through the first 30 epochs (preliminary) it was decided that the best time to stop the training was in epoch 12. The final values obtained from evaluating the test dataset was (`loss: 0.2729 - accuracy: 0.9200`). Finally, it was determined that one last model would be built in an attempt to improve the accuracy of the second model. The third CNN model (pvdm3) used data augmentation, L2 regularization, and Dropout. The data augmentation used in (pvdm3) consisted of the following parameters and values: *RandomFlip('horizontal'), RandomRotation(.1), and RandomZoom(.2)*. The Dropout layers were implemented after each MaxPool layer and after the flatten layer. The values for the Dropout layers ranged from *(.15 - .5)* and the L2 regularization was kept the same as in the second model. Once the model went through the first 100 epochs (preliminary) it was decided that the best time to stop the training was in epoch 60. The final values obtained from evaluating the test dataset was (`loss: 0.1511 - accuracy: 0.9600`)

* *Further Analysis*

In the midst of training the third model I attempted to use the ModelCheckpoint and callbacks functions to determine the best model. In addition, instead of saving the best model via the validation loss metric, I used the validation accuracy. Then I proceeded to train the model for 200 epochs. Once the 200 epochs were completed I used the callback function to retrieve the best model and evaluated it with the test dataset. The results obtained from this evaluation was (loss: 0.0804 - accuracy: 0.9900). Hence, an interesting question now comes into my mind, for engineers the most important parameter is the loss (specifically validation loss), because it allows us to keep track of how the model is performing from epoch to epoch. However, is it more beneficial to save the model which has the highest validation accuracy?

III.1 CNN Model 1 (pvcm1)

```
Model: "model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 180, 180, 3)]     0

 rescaling_1 (Rescaling)     (None, 180, 180, 3)       0

 conv2d_5 (Conv2D)           (None, 178, 178, 32)      896

 max_pooling2d_4 (MaxPooling  (None, 89, 89, 32)       0
 2D)

 conv2d_6 (Conv2D)           (None, 87, 87, 64)        18496

 max_pooling2d_5 (MaxPooling  (None, 43, 43, 64)       0
 2D)

 conv2d_7 (Conv2D)           (None, 41, 41, 128)       73856

 max_pooling2d_6 (MaxPooling  (None, 20, 20, 128)      0
 2D)

 conv2d_8 (Conv2D)           (None, 18, 18, 256)       295168

 max_pooling2d_7 (MaxPooling  (None, 9, 9, 256)        0
 2D)

 conv2d_9 (Conv2D)           (None, 7, 7, 256)         590080

 flatten_1 (Flatten)         (None, 12544)             0

 dense_1 (Dense)             (None, 1)                 12545

=================================================================
Total params: 991,041
Trainable params: 991,041
Non-trainable params: 0
```

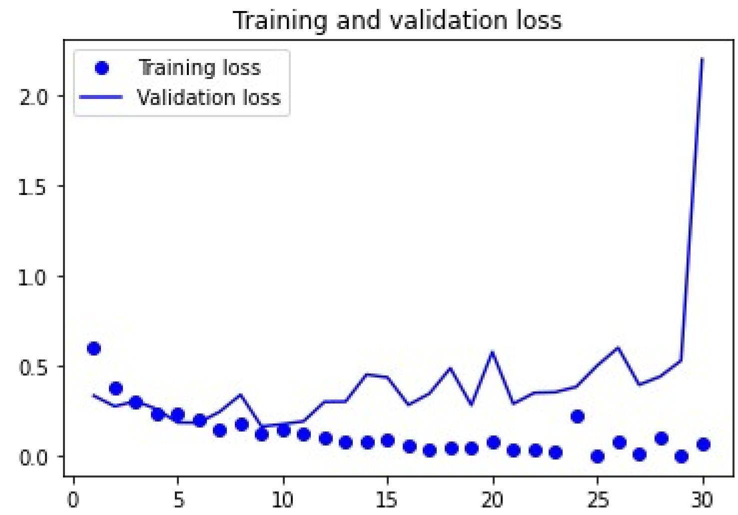

Figure 7: Preliminary Loss (Training and Validation) vs Epochs

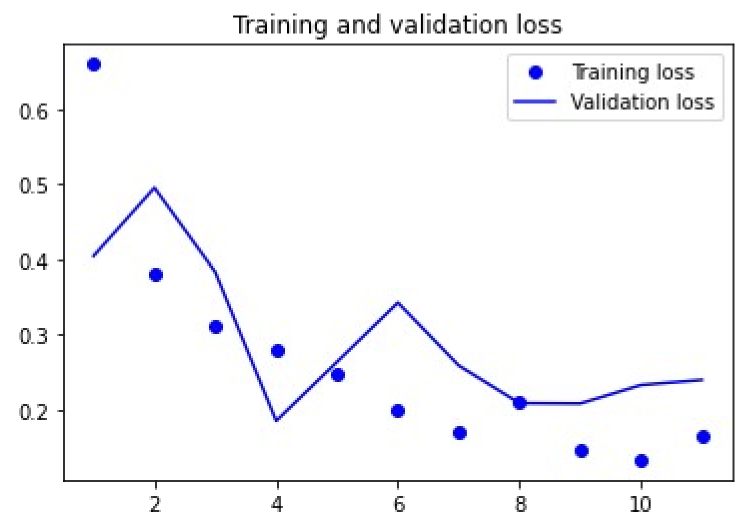

Figure 8: Final Loss (Training and Validation) vs Epochs

```
Final Training loss:  0.16530783474445343
Final Training Accuracy:  0.954285740852356
Final Validation loss:  0.23942400515079498
Final Validation Accuracy:  0.9125000238418579
7/7 [==============================] - 0s 12ms/step - loss: 0.3077 -
accuracy: 0.9000 [0.30769720673561096, 0.8999999761581421]
```

III.2 CNN Model 2 (pvcm2)

```
Model: "model_9"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_10 (InputLayer)       [(None, 180, 180, 3)]     0         
                                                                 
 rescaling_9 (Rescaling)     (None, 180, 180, 3)       0         
                                                                 
 conv2d_45 (Conv2D)          (None, 178, 178, 32)      896       
                                                                 
 max_pooling2d_36 (MaxPoolin  (None, 89, 89, 32)       0         
 g2D)                                                            
                                                                 
 dropout_10 (Dropout)        (None, 89, 89, 32)        0         
                                                                 
 conv2d_46 (Conv2D)          (None, 87, 87, 64)        18496     
                                                                 
 max_pooling2d_37 (MaxPoolin  (None, 43, 43, 64)       0         
 g2D)                                                            
                                                                 
 dropout_11 (Dropout)        (None, 43, 43, 64)        0         
                                                                 
 conv2d_47 (Conv2D)          (None, 41, 41, 128)       73856     
                                                                 
 max_pooling2d_38 (MaxPoolin  (None, 20, 20, 128)      0         
 g2D)                                                            
                                                                 
 dropout_12 (Dropout)        (None, 20, 20, 128)       0         
                                                                 
 conv2d_48 (Conv2D)          (None, 18, 18, 256)       295168    
                                                                 
 max_pooling2d_39 (MaxPoolin  (None, 9, 9, 256)        0         
 g2D)                                                            
                                                                 
 dropout_13 (Dropout)        (None, 9, 9, 256)         0         
                                                                 
 conv2d_49 (Conv2D)          (None, 7, 7, 256)         590080    
                                                                 
 flatten_9 (Flatten)         (None, 12544)             0         
                                                                 
 dropout_14 (Dropout)        (None, 12544)             0         
                                                                 
 dense_9 (Dense)             (None, 1)                 12545     
                                                                 
=================================================================
Total params: 991,041
Trainable params: 991,041
Non-trainable params: 0
```

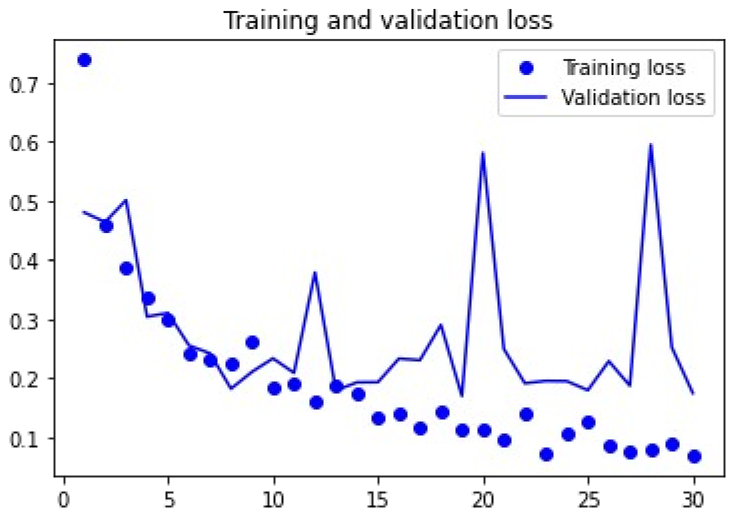

Figure 9: Preliminary Loss (Training and Validation) vs Epochs

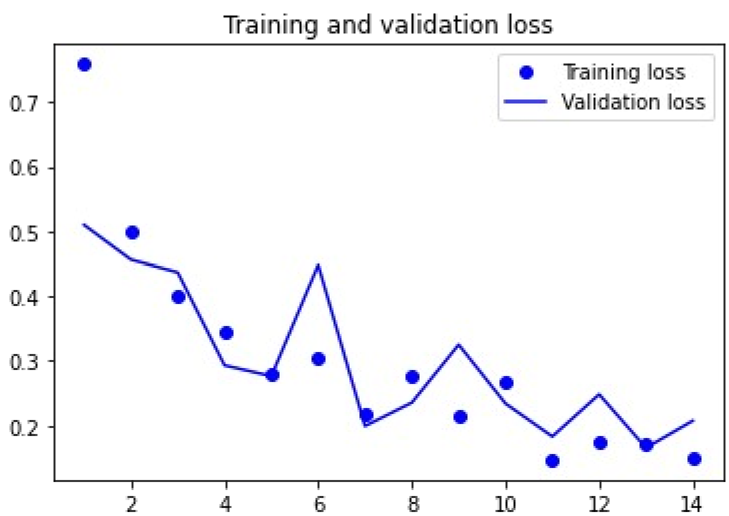

Figure 10: Final Loss (Training and Validation) vs Epochs

```
Final Training loss:  0.14952512085437775
Final Training Accuracy:  0.9557142853736877
Final Validation loss:  0.20736859738826752
Final Validation Accuracy:  0.9325000047683716
7/7 [==============================] - 0s 12ms/step - loss: 0.2329 -
accuracy: 0.9300 [0.23292145133018494, 0.9300000071525574]
```

III.3 CNN Model 3 (pvcm3)

```
Model: "model_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_7 (InputLayer)        [(None, 180, 180, 3)]     0         
                                                                 
 sequential (Sequential)     (None, 180, 180, 3)       0         
                                                                 
 rescaling_6 (Rescaling)     (None, 180, 180, 3)       0         
                                                                 
 conv2d_30 (Conv2D)          (None, 178, 178, 32)      896       
                                                                 
 max_pooling2d_24 (MaxPoolin  (None, 89, 89, 32)       0         
 g2D)                                                            
                                                                 
 dropout (Dropout)           (None, 89, 89, 32)        0         
                                                                 
 conv2d_31 (Conv2D)          (None, 87, 87, 64)        18496     
                                                                 
 max_pooling2d_25 (MaxPoolin  (None, 43, 43, 64)       0         
 g2D)                                                            
                                                                 
 dropout_1 (Dropout)         (None, 43, 43, 64)        0         
                                                                 
 conv2d_32 (Conv2D)          (None, 41, 41, 128)       73856     
                                                                 
 max_pooling2d_26 (MaxPoolin  (None, 20, 20, 128)      0         
 g2D)                                                            
                                                                 
 dropout_2 (Dropout)         (None, 20, 20, 128)       0         
                                                                 
 conv2d_33 (Conv2D)          (None, 18, 18, 256)       295168    
                                                                 
 max_pooling2d_27 (MaxPoolin  (None, 9, 9, 256)        0         
 g2D)                                                            
                                                                 
 dropout_3 (Dropout)         (None, 9, 9, 256)         0         
                                                                 
 conv2d_34 (Conv2D)          (None, 7, 7, 256)         590080    
                                                                 
 flatten_6 (Flatten)         (None, 12544)             0         
                                                                 
 dropout_4 (Dropout)         (None, 12544)             0         
                                                                 
 dense_6 (Dense)             (None, 1)                 12545     
                                                                 
=================================================================
Total params: 991,041
Trainable params: 991,041
Non-trainable params: 0
```

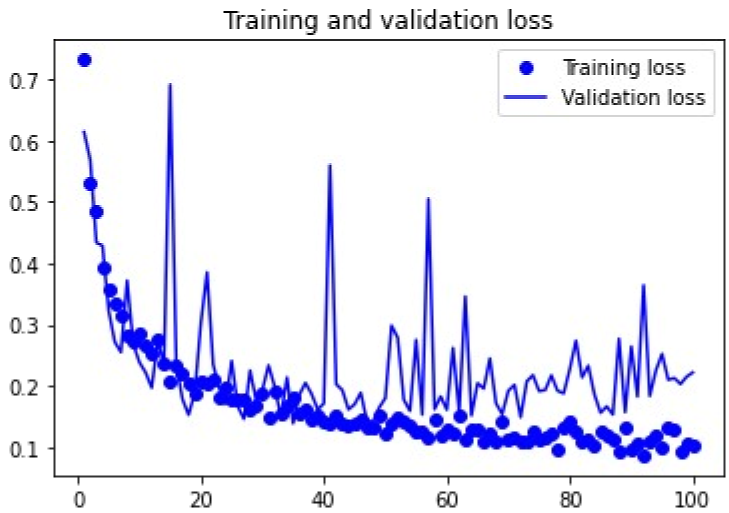

Figure 11: Preliminary Loss (Training and Validation) vs Epochs

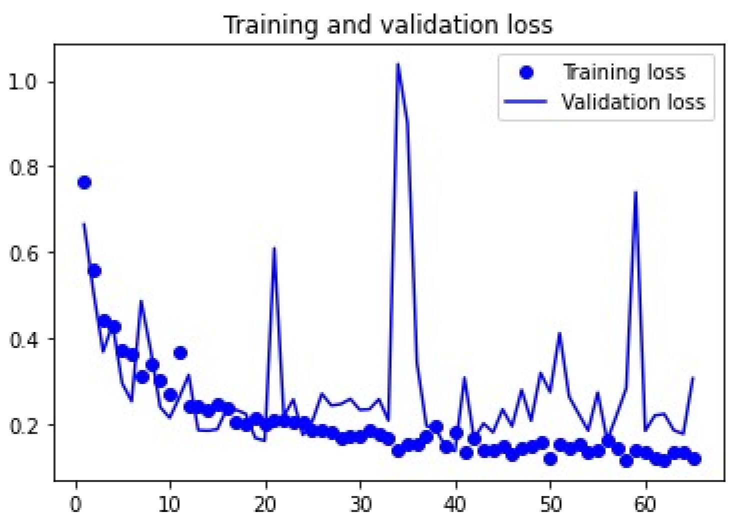

Figure 12: Final Loss (Training and Validation) vs Epochs

```
Final Training loss:  0.12229102849960327
Final Training Accuracy:  0.9628571271896362
Final Validation loss:  0.30623793601989746
Final Validation Accuracy:  0.9424999952316284
7/7 [==============================] - 0s 13ms/step - loss: 0.2660 -
accuracy: 0.9500 [0.26600533723831177, 0.949999988079071]
```

Discussion/Observations of Panda vs Cat models:

In the first CNN model (pvcm1) the model that was built consisted of no regularization methods and data augmentation. Once the model went through the first 30 epochs (preliminary) it was decided that the best time to stop the training was in epoch 11. The final values obtained from evaluating the test dataset was (`loss: 0.3077 - accuracy: 0.9000`). After obtaining these values it was determined that a second model that would use regularization methods will be implemented in order to improve the accuracy of the CNN model. Hence, the second CNN model (pvcm2) was built. After trying different combinations of regularization methods it was determined that *L2(.0001)* and Dropout layers with values of *(.25)* improved the model's accuracy. Once the model went through the first 30 epochs (preliminary) it was decided that the best time to stop the training was in epoch 12. The final values obtained from evaluating the test dataset was (`loss: 0.2329 - accuracy: 0.9300`). Finally, it was determined that one last model would be built in an attempt to improve the accuracy of the second model. The third CNN model (pvdm3) used data augmentation, L2 regularization, and Dropout. The data augmentation used in (pvdm3) consisted of the following parameters and values: *RandomFlip('horizontal'), RandomRotation(.1), and RandomZoom(.2)*. The Dropout layers were implemented after each MaxPool layer and after the flatten layer. The values for the Dropout layers ranged from *(.15 - .5)* and the L2 regularization was kept the same as in the second model. Once the model went through the first 100 epochs (preliminary) it was decided that the best time to stop the training was in epoch 65. The final values obtained from evaluating the test dataset was (`loss: 0.2660 - accuracy: 0.9500`).

Conclusion:

In this project the biggest thing I learned was the impacts of regularization methods and data augmentation on the performance of CNN models, specifically data augmentation seems to always improve the model no matter what. In this project I expected the models to be able to classify images with an accuracy of 90%. However, what I definitely was not expecting was that the first CNN model would automatically be around this range. Even more unexpected was that the model can be improved to 95-96% just by using some basic techniques such as regularization and data augmentation. I believe that one way to improve these models is by increasing the dataset to have more images to classify. In addition, a preprocessing stage can be set up to select random images to have in the training,validation and test split. This will eliminate possible selecting images that are harder to train. I believe that Panda vs Cats seems to be a harder classification task because for the second model I had to use much more techniques to improve the second model, but the last model was lower than the one for the Panda vs Dogs final model.

Appendix

```
# Importing Libraries

import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd
from io import StringIO
from sklearn.preprocessing import LabelEncoder
import sklearn
from sklearn.model_selection import train_test_split
from numpy import array
from numpy import argmax
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
import copy
from keras.engine.input_layer import Input
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neural_network import MLPClassifier
from google.colab import files
import os, shutil, pathlib
from tensorflow.keras.preprocessing import image_dataset_from_directory
from keras import regularizers
from sklearn.utils import validation

# Uploading Kaggle JSon File

files.upload()

# Removing/Creating Directory for Kaggle

!rm -r ~/.kaggle/
!mkdir ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json



# Importing Dataset

!kaggle datasets download -d ashishsaxena2209/animal-image-datasetdog-cat-
and-panda
```

```
# Unzipping Dataset

!unzip -qq animal-image-datasetdog-cat-and-panda.zip

# New Directory Structure

original_dir = pathlib.Path('animals')
new_base_dir = pathlib.Path('newanim')

def make_subset(subset_name, start_index, end_index):
  for category in ('cats','dogs','panda'):
    dir = new_base_dir / subset_name / category
    dirsrc = original_dir / category
    os.makedirs(dir)
    fnames = ['{}_{:05d}.jpg'.format(category,i) for i in range
(start_index, end_index)]
    for fname in fnames:
      shutil.copyfile(src=dirsrc / fname , dst = dir / fname)

make_subset('train', start_index=1, end_index=701)
make_subset('validation', start_index=701, end_index=901)
make_subset('test', start_index=901,end_index=1001)

# Remove the Cat Directories

!rm -r ./newanim/train/cats/
!rm -r ./newanim/validation/cats/
!rm -r ./newanim/test/cats/

# Creating TensorFlow Dataset

train_dataset = image_dataset_from_directory(
    new_base_dir / 'train',
    image_size = (180,180),
    batch_size = 32)
valdiation_dataset = image_dataset_from_directory(
    new_base_dir / 'validation',
    image_size = (180,180),
    batch_size = 32)
test_dataset = image_dataset_from_directory(
    new_base_dir / 'test',
    image_size = (180,180),
    batch_size = 32)

# Displaying shape of Dataset
```

```
for data_batch, labels_batch in train_dataset:
  print("data batch shape:", data_batch.shape)
  print("labels batch shape:", labels_batch.shape)
  break

# CNN Model Panda vs Dog (pvdm1)

inputs = keras.Input(shape=(180, 180, 3))
x = layers.Rescaling(1./255)(inputs)
x = layers.Conv2D(filters=32, kernel_size=3,activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
pvdm1 = keras.Model(inputs=inputs, outputs=outputs)

pvdm1.summary()

## Configure Model for Training

pvdm1.compile(loss = "binary_crossentropy", optimizer="rmsprop",
metrics=["accuracy"])

## Fitting the Model using a Dataset

callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="pvdm1.h5",
        save_best_only = True,
        monitor = "val_loss")
]

history1 = pvdm1.fit(
    train_dataset,
    epochs = 8,
    validation_data = valdiation_dataset,
    callbacks = callbacks
)

## Displaying Curves of Loss and Accuracy during Training
```

```
accuracy = history1.history["accuracy"]
val_accuracy = history1.history["val_accuracy"]
loss = history1.history["loss"]
val_loss = history1.history["val_loss"]
epochs = range(1, len(accuracy) + 1)
plt.plot(epochs, accuracy, "bo", label="Training accuracy")
plt.plot(epochs, val_accuracy, "b", label="Validation accuracy")
plt.title("Training and validation accuracy")
plt.legend()
plt.figure()
plt.plot(epochs, loss, "bo", label="Training loss")
plt.plot(epochs, val_loss, "b", label="Validation loss")
plt.title("Training and validation loss")
plt.legend()
plt.show()
# Final Values
print("Final Training loss: ",history1.history['loss'][-1],"\nFinal Training
Accuracy: ", history1.history['accuracy'][-1])
print("Final Validation loss: ",history1.history['val_loss'][-1],"\nFinal
Validation Accuracy: ", history1.history['val_accuracy'][-1])

## Evaluate Model

pvdm1.evaluate(test_dataset)

# CNN Model 2 Panda vs Dog (w/L2) (pvdm2)

inputs = keras.Input(shape=(180, 180, 3))
x = layers.Rescaling(1./255)(inputs)
x = layers.Conv2D(filters=32, kernel_size=3,
kernel_regularizer=regularizers.l2(.0001),activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3,
kernel_regularizer=regularizers.l2(.0001),activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3,
kernel_regularizer=regularizers.l2(.0001),activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3,
kernel_regularizer=regularizers.l2(.0001),activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3,
kernel_regularizer=regularizers.l2(.0001),activation="relu")(x)
x = layers.Flatten()(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
pvdm2 = keras.Model(inputs=inputs, outputs=outputs)
```

```
pvdm2.summary()

## Configure Model for Training

pvdm2.compile(loss = "binary_crossentropy", optimizer="rmsprop",
metrics=["accuracy"])

## Fitting the Model using a Dataset

callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="pvdm2.h5",
        save_best_only = True,
        monitor = "val_loss")
]

history2 = pvdm2.fit(
    train_dataset,
    epochs = 10,
    validation_data = valdiation_dataset,
    callbacks = callbacks

)

## Displaying Curves of Loss and Accuracy during Training

accuracy = history2.history["accuracy"]
val_accuracy = history2.history["val_accuracy"]
loss = history2.history["loss"]
val_loss = history2.history["val_loss"]
epochs = range(1, len(accuracy) + 1)
plt.plot(epochs, accuracy, "bo", label="Training accuracy")
plt.plot(epochs, val_accuracy, "b", label="Validation accuracy")
plt.title("Training and validation accuracy")
plt.legend()
plt.figure()
plt.plot(epochs, loss, "bo", label="Training loss")
plt.plot(epochs, val_loss, "b", label="Validation loss")
plt.title("Training and validation loss")
plt.legend()
plt.show()
# Final Values
print("Final Training loss: ",history2.history['loss'][-1],"\nFinal Training
Accuracy: ", history2.history['accuracy'][-1])
print("Final Validation loss: ",history2.history['val_loss'][-1],"\nFinal
Validation Accuracy: ", history2.history['val_accuracy'][-1])
```

```
## Evaluate Model

pvdm2.evaluate(test_dataset)

# CNN Model 3 Panda vs Dog (w/Data Augmentation) (pvdm3)

data_augmentation = keras.Sequential(
    [
      layers.RandomFlip("horizontal"),
      layers.RandomRotation(.1),
      layers.RandomZoom(.2)
    ]
)

inputs = keras.Input(shape=(180, 180, 3))
x = data_augmentation(inputs)
x = layers.Rescaling(1./255)(x)
x = layers.Conv2D(filters=32, kernel_size=3,
kernel_regularizer=regularizers.l2(.0001),activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Dropout(.15)(x)
x = layers.Conv2D(filters=64, kernel_size=3,
kernel_regularizer=regularizers.l2(.0001),activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Dropout(.25)(x)
x = layers.Conv2D(filters=128, kernel_size=3,
kernel_regularizer=regularizers.l2(.0001),activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Dropout(.5)(x)
x = layers.Conv2D(filters=256, kernel_size=3,
kernel_regularizer=regularizers.l2(.0001),activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Dropout(.25)(x)
x = layers.Conv2D(filters=256, kernel_size=3,
kernel_regularizer=regularizers.l2(.0001),activation="relu")(x)
x = layers.Flatten()(x)
x = layers.Dropout(.25)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
pvdm3 = keras.Model(inputs=inputs, outputs=outputs)

pvdm3.summary()

## Configure Model for Training

pvdm3.compile(loss = "binary_crossentropy", optimizer="rmsprop",
metrics=["accuracy"])
```

```
## Fitting the Model using a Dataset

callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="pvdm3.h5",
        save_best_only = True,
        monitor = "val_loss")
]

history3 = pvdm3.fit(
    train_dataset,
    epochs = 150,
    validation_data = valdiation_dataset,
    callbacks = callbacks

)

## Displaying Curves of Loss and Accuracy during Training

accuracy = history3.history["accuracy"]
val_accuracy = history3.history["val_accuracy"]
loss = history3.history["loss"]
val_loss = history3.history["val_loss"]
epochs = range(1, len(accuracy) + 1)
plt.plot(epochs, accuracy, "bo", label="Training accuracy")
plt.plot(epochs, val_accuracy, "b", label="Validation accuracy")
plt.title("Training and validation accuracy")
plt.legend()
plt.figure()
plt.plot(epochs, loss, "bo", label="Training loss")
plt.plot(epochs, val_loss, "b", label="Validation loss")
plt.title("Training and validation loss")
plt.legend()
plt.show()
# Final Values
print("Final Training loss: ",history3.history['loss'][-1],"\nFinal Training
Accuracy: ", history3.history['accuracy'][-1])
print("Final Validation loss: ",history3.history['val_loss'][-1],"\nFinal
Validation Accuracy: ", history3.history['val_accuracy'][-1])

## Evaluate Model

pvdm3.evaluate(test_dataset)




 # Remove the Dog DIrectories
```

```
!rm -r ./newanim/train/dogs/
!rm -r ./newanim/validation/dogs/
!rm -r ./newanim/test/dogs/

# CNN Model Panda vs Cat (pvcm1)

inputs = keras.Input(shape=(180, 180, 3))
x = layers.Rescaling(1./255)(inputs)
x = layers.Conv2D(filters=32, kernel_size=3,activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
pvcm1 = keras.Model(inputs=inputs, outputs=outputs)

pvcm1.summary()

## Configure Model for Training

pvcm1.compile(loss = "binary_crossentropy", optimizer="rmsprop",
metrics=["accuracy"])

## Fitting the Model using a Dataset

callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="pvcm1.h5",
        save_best_only = True,
        monitor = "val_loss")
]

history1 = pvcm1.fit(
    train_dataset,
    epochs = 11,
    validation_data = valdiation_dataset,
    callbacks = callbacks
)

## Displaying Curves of Loss and Accuracy during Training
```

```
accuracy = history1.history["accuracy"]
val_accuracy = history1.history["val_accuracy"]
loss = history1.history["loss"]
val_loss = history1.history["val_loss"]
epochs = range(1, len(accuracy) + 1)
plt.plot(epochs, accuracy, "bo", label="Training accuracy")
plt.plot(epochs, val_accuracy, "b", label="Validation accuracy")
plt.title("Training and validation accuracy")
plt.legend()
plt.figure()
plt.plot(epochs, loss, "bo", label="Training loss")
plt.plot(epochs, val_loss, "b", label="Validation loss")
plt.title("Training and validation loss")
plt.legend()
plt.show()
# Final Values
print("Final Training loss: ",history1.history['loss'][-1],"\nFinal Training
Accuracy: ", history1.history['accuracy'][-1])
print("Final Validation loss: ",history1.history['val_loss'][-1],"\nFinal
Validation Accuracy: ", history1.history['val_accuracy'][-1])

## Evaluate Model

pvcm1.evaluate(test_dataset)

# CNN Model 2 Panda vs Cat (w/L2) (pvcm2)

inputs = keras.Input(shape=(180, 180, 3))
x = layers.Rescaling(1./255)(inputs)
x = layers.Conv2D(filters=32, kernel_size=3,
kernel_regularizer=regularizers.l2(.0001),activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Dropout(.25)(x)
x = layers.Conv2D(filters=64, kernel_size=3,
kernel_regularizer=regularizers.l2(.0001),activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Dropout(.25)(x)
x = layers.Conv2D(filters=128, kernel_size=3,
kernel_regularizer=regularizers.l2(.0001),activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Dropout(.25)(x)
x = layers.Conv2D(filters=256, kernel_size=3,
kernel_regularizer=regularizers.l2(.0001),activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Dropout(.25)(x)
x = layers.Conv2D(filters=256, kernel_size=3,
kernel_regularizer=regularizers.l2(.0001),activation="relu")(x)
```

```
x = layers.Flatten()(x)
x = layers.Dropout(.25)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
pvcm2 = keras.Model(inputs=inputs, outputs=outputs)

pvcm2.summary()

## Configure Model for Training

pvcm2.compile(loss = "binary_crossentropy", optimizer="rmsprop",
metrics=["accuracy"])

## Fitting the Model using a Dataset

callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="pvcm2.h5",
        save_best_only = True,
        monitor = "val_loss")
]

history2 = pvcm2.fit(
    train_dataset,
    epochs = 14,
    validation_data = valdiation_dataset,
    callbacks = callbacks

)

## Displaying Curves of Loss and Accuracy during Training

accuracy = history2.history["accuracy"]
val_accuracy = history2.history["val_accuracy"]
loss = history2.history["loss"]
val_loss = history2.history["val_loss"]
epochs = range(1, len(accuracy) + 1)
plt.plot(epochs, accuracy, "bo", label="Training accuracy")
plt.plot(epochs, val_accuracy, "b", label="Validation accuracy")
plt.title("Training and validation accuracy")
plt.legend()
plt.figure()
plt.plot(epochs, loss, "bo", label="Training loss")
plt.plot(epochs, val_loss, "b", label="Validation loss")
plt.title("Training and validation loss")
plt.legend()
plt.show()
# Final Values
```

```
print("Final Training loss: ",history2.history['loss'][-1],"\nFinal Training
Accuracy: ", history2.history['accuracy'][-1])
print("Final Validation loss: ",history2.history['val_loss'][-1],"\nFinal
Validation Accuracy: ", history2.history['val_accuracy'][-1])

## Evaluate Model

pvcm2.evaluate(test_dataset)

# CNN Model 3 Panda vs Cat (w/Data Augmentation) (pvcm3)

data_augmentation = keras.Sequential(
    [
      layers.RandomFlip("horizontal"),
      layers.RandomRotation(.1),
      layers.RandomZoom(.2)
    ]
)

inputs = keras.Input(shape=(180, 180, 3))
x = data_augmentation(inputs)
x = layers.Rescaling(1./255)(x)
x = layers.Conv2D(filters=32, kernel_size=3,
kernel_regularizer=regularizers.l2(.0001),activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Dropout(.15)(x)
x = layers.Conv2D(filters=64, kernel_size=3,
kernel_regularizer=regularizers.l2(.0001),activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Dropout(.25)(x)
x = layers.Conv2D(filters=128, kernel_size=3,
kernel_regularizer=regularizers.l2(.0001),activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Dropout(.5)(x)
x = layers.Conv2D(filters=256, kernel_size=3,
kernel_regularizer=regularizers.l2(.0001),activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Dropout(.25)(x)
x = layers.Conv2D(filters=256, kernel_size=3,
kernel_regularizer=regularizers.l2(.0001),activation="relu")(x)
x = layers.Flatten()(x)
x = layers.Dropout(.25)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
pvcm3 = keras.Model(inputs=inputs, outputs=outputs)

pvcm3.summary()
## Configure Model for Training
```

```
pvcm3.compile(loss = "binary_crossentropy", optimizer="rmsprop",
metrics=["accuracy"])

## Fitting the Model using a Dataset

callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="pvcm3.h5",
        save_best_only = True,
        monitor = "val_loss")
]

history3 = pvcm3.fit(
    train_dataset,
    epochs = 65,
    validation_data = valdiation_dataset,
    callbacks = callbacks
)

## Displaying Curves of Loss and Accuracy during Training

accuracy = history3.history["accuracy"]
val_accuracy = history3.history["val_accuracy"]
loss = history3.history["loss"]
val_loss = history3.history["val_loss"]
epochs = range(1, len(accuracy) + 1)
plt.plot(epochs, accuracy, "bo", label="Training accuracy")
plt.plot(epochs, val_accuracy, "b", label="Validation accuracy")
plt.title("Training and validation accuracy")
plt.legend()
plt.figure()
plt.plot(epochs, loss, "bo", label="Training loss")
plt.plot(epochs, val_loss, "b", label="Validation loss")
plt.title("Training and validation loss")
plt.legend()
plt.show()
# Final Values
print("Final Training loss: ",history3.history['loss'][-1],"\nFinal Training
Accuracy: ", history3.history['accuracy'][-1])
print("Final Validation loss: ",history3.history['val_loss'][-1],"\nFinal
Validation Accuracy: ", history3.history['val_accuracy'][-1])

## Evaluate Model
pvcm3.evaluate(test_dataset)
```