# Animal Classification – Individual Report

Shuting Cai

## *Introduction*

Image recognition is a popular technology, which can accurately identify the visual content in the image, and provide a variety of an object, scenes, and concept labels, with the ability of target detection and attribute recognition, to help clients accurately identify and understand the image. The technique has been applied to multiple fields that text recognition, license plate recognition, and face recognition. The team would explore the topic of computer vision primarily, applying the image classification technique to animal detection. The animal classification could be used in animal management and tag the different animal species, for further application, it could be developed and suitable for animal photo recognition, early childhood education science, and image content analysis, to help people get a better idea of animals and the diversity of nature. The team's objectives are to design a CNN model with 90% accuracy, which will be accomplished by exploring the architecture of the CNN model and comparing the state-of-the-art pre-trained models and the model with the customized architecture. The team is using TensorFlow to implement the network, for the reason that TensorFlow is a powerful and mature deep learning library with strong visualization capabilities, and there are multiple options for advanced model development. With the data augmentation of the training, the data will expand up the training dataset, therefore, the model can be fitted better. The optimal hyperparameters that learning rate, number of epochs, batch size also would be determined experimentally to achieve the best performance.

## *Description of the data set*

The team is going to use the “Animal Image Classification Dataset,” a source from Kaggle, which contains 12 classes of animals that are butterflies, cats, cows, dogs, elephants, hens, horses, monkeys, pandas, sheep, spiders, and squirrel. For each class, there are at least 1,400 image files (data resource), which is enough to train the common architectures in the convolutional neural network, besides, data augmentation would be used, which will expand the training dataset.

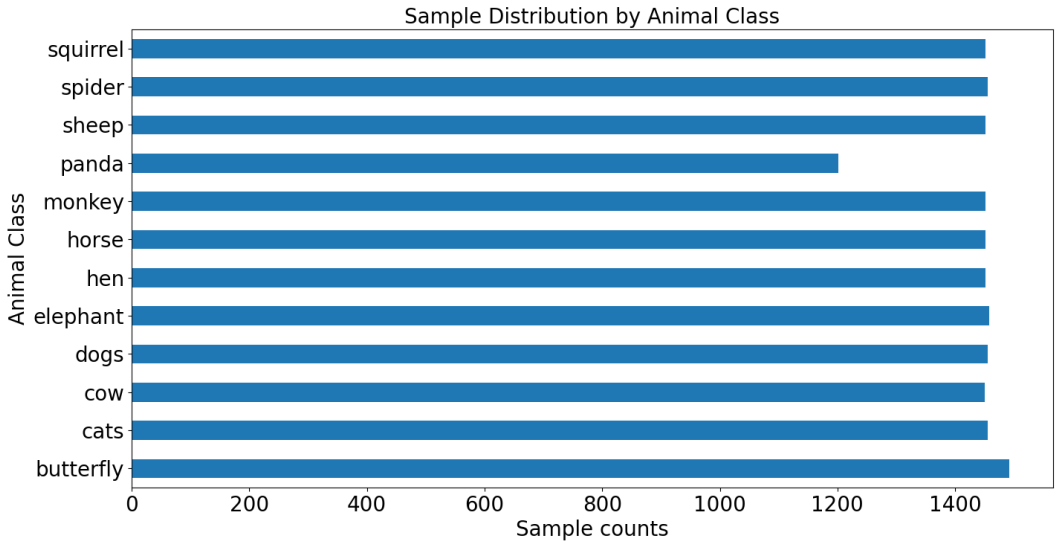


*Figure1. The horizontal count plot for Animal Class.*

*Description of the deep learning network and training algorithm*

ResNet was the winner of the 2015 ImageNet ILSVRC challenge with a top five error rate under 3.6%. ResNet50 can work with 50 deep neural network layers. The key building blocks in ResNet are Deep Residual Networks. Deep residual nets make use of residual blocks to improve the accuracy of the models.

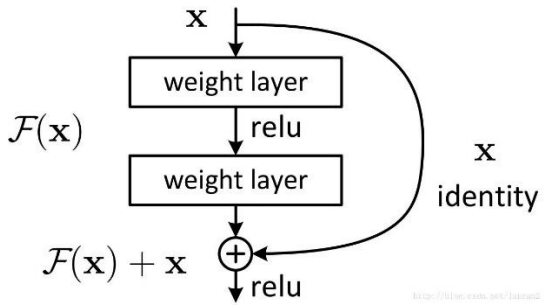


*Figure 2. The skip connection.*

In each residual unit, the number of sublayers in the convolutional layer is doubled, and the height and width of the sublayers are halved with 2 strides. The concept of "skip connections," which lies at the core of the residual blocks, is the strength of this type of neural network.

Compare with Resnet34, a 3-layer bottle-neck block instead of 2-layer blocks, Resnet 50 architecture can save the running time and has much higher accuracy than the 34-layer ResNet model. The architecture of ResNet is using ReLU as the activation function and is followed by a batch normalization layer.

*Experimental setup*

```
import os
import splitfolders
OR_PATH = os.getcwd()
os.chdir("..")  # Change to the parent directory
PATH = os.getcwd()
DATA_DIR = os.getcwd() + os.path.sep + 'Data' + os.path.sep
sep = os.path.sep
os.chdir(OR_PATH)

random_seed = 42
inputfolder = DATA_DIR

splitfolders.ratio(inputfolder, output='train_test', seed=random_seed, ratio=(0.9,0,0.1),group_prefix=None)
```

The original dataset would be split into training, validation, and testing dataset. In training, the loss and accuracy of train and validation would be detected; In testing, the predicted result would be used in judging the performance of the network. I split the train and test dataset

to subfolders with a ratio of 9 to 1, and then I used tf.keras.utils.image_dataset_from_directory to split the training and validation dataset into 7 to 3.

```
# ------------------------------------------------------
#### Image plot
# ------------------------------------------------------
def visualize(original, augmented):
  fig = plt.figure()
  plt.subplot(1,2,1)
  plt.title('Original image')
  plt.imshow(original)
  plt.subplot(1,2,2)
  plt.title('Augmented image')
  plt.imshow(augmented)
  plt.tight_layout()

def flip(data):
    data_flipped = tf.image.flip_left_right(data)
    return data_flipped
def rotation(data):
    data_rotated = tf.image.rot90(data)
    return data_rotated

input_size = [img_height, img_width]
def resize(data):
    data_resized = tf.image.resize(data, input_size)
    return data_resized
```

```
for images, labels in train_ds.take(1):
    img = images[1].numpy().astype("uint8")
    plt.figure(figsize=(6, 6))
    plt.imshow(img)
    plt.title(class_names[labels[1]], fontdict={'fontsize':12})
    plt.axis("off")
    plt.show()
    plt.figure(figsize=(12, 8))
    visualize(img, flip(img))
    plt.show()
    plt.figure(figsize=(12, 8))
    visualize(img, rotation(img))
    plt.show()
```

Before building the training models, I have added some data augmentation to strength our training dataset. I was trying to resize the image to 224 by 224, and did image horizontal flip, rotated the image to 90 degrees, and zoomed the image, but I found that added too many data augmentation, the running time would be slower. The team would randomly select some operations from the above data augmentations then added to the final model. Moreover, there is a disadvantage of data augmentation that is increasing data bias (Xu, 2020). if the augmented data distribution would be significantly different from the original dataset, the performance of the network would be getting worse.

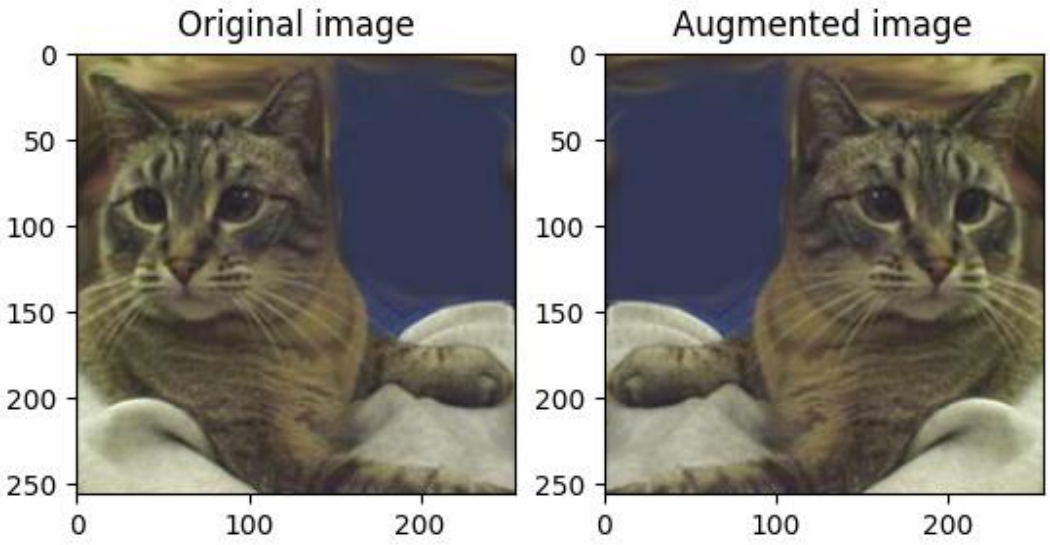


Figure 4. The original image vs. the horizontally flipped images.

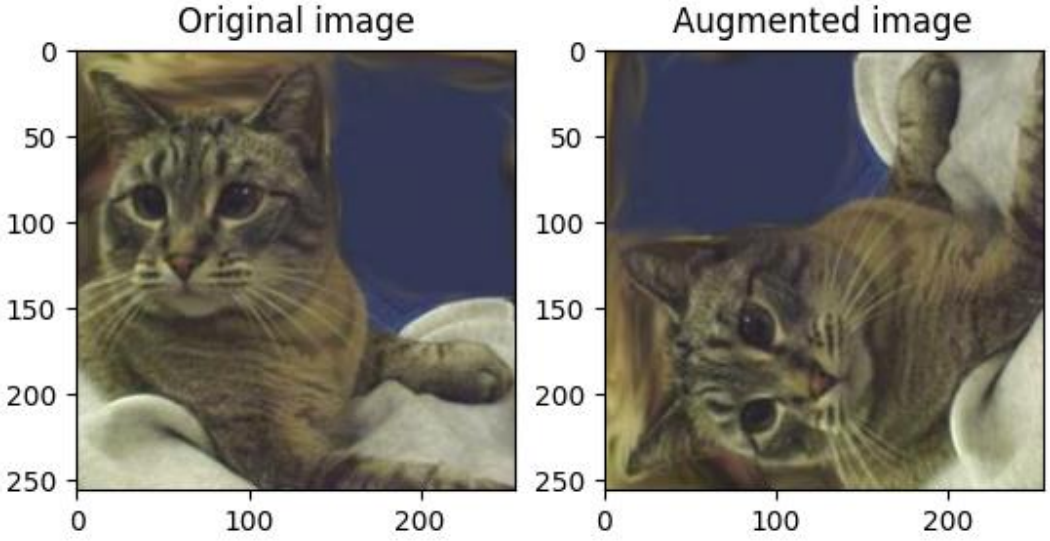


Figure 5. The original image vs. the 90 degrees rotated images.

```python
# ------------------------------------------------------------------------------------------------------------
def model_def():
    model1 = tf.keras.Sequential()
    Resnet = tf.keras.applications.ResNet50(include_top=False, weights='imagenet')
    for layer in Resnet.layers:
        layer.trainable = False
    model1.add(Resnet)
    model1.add(tf.keras.layers.Dense(12))

    # Add the output layer
    average_pooling = keras.layers.GlobalAveragePooling2D()(model1.output)

    output = keras.layers.Dense(12, activation='softmax')(average_pooling)

    # Get the model
    model = keras.Model(inputs=model1.input, outputs=output)

    model.compile(optimizer=keras.optimizers.Adam(learning_rate=lr),
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])

    save_model(model)
    return model


# ------------------------------------------------------------------------------------------------------------
def train_func(train_ds):
    early_stop = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=2, mode='min')
    check_point = tf.keras.callbacks.ModelCheckpoint('model_ResNet50.h5', monitor='val_loss', save_best_only=True, mode='min')
    model = model_def()
    history = model.fit(train_ds, validation_data=val_ds, epochs=epochs, callbacks=[early_stop, check_point])
    return history
```

With the pretrained ResNet50 model, I added global average pooling 2D layer, which applies average pooling on the spatial dimensions until each spatial dimension is one, and leaves other dimensions unchanged, then added one output layer with activation function of SoftMax. In addition, I added freezing layer of the pre-trained architecture, freezing layer is a technique to accelerate neural network training by progressively freezing hidden layers, which can help the model to get better performance. The transferred function of SoftMax is suitable for multiple class problem. A compiled model would be needed to train with the Adam optimizer and learning rate of 0.01, and loss function of “sparse_categorical_crossentropy” that is used as a loss function for multi-class classification model where the output label is assigned integer value (Ajitesh). I added the early stopping and the check point function with the lowest validation loss to save the best model.

The hyperparameters in this model are including learning rate, the number of epochs and the batch sizes. The learning rate defines how quickly a network updates its parameters. A low learning rate slows down the learning process but converges smoothly. A larger learning rate speeds up the learning but may not converge, therefore, a decaying Learning rate is preferred (Radhakrishnan). The number of epochs is the number of times the training data is processed to the network, and the batch size is the number of subsamples given to the network after when the parameters update happens. I found the optimal condition for the hyperparameters that 3 of epochs, 64 of batch size and 0.01 of learning rate.

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

x = np.arange(1, epochs + 1, 1)
fig = plt.figure(figsize=(12, 8))
plt.subplot(1, 2, 1)
plt.plot(x, acc, label='Training Accuracy')
plt.plot(x, val_acc, label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.xticks(x)
plt.legend(loc='upper left')

plt.subplot(1, 2, 2)
plt.plot(x, loss, label='Training Loss')
plt.plot(x, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.xticks(x)
plt.tight_layout()
fig.savefig('train_val_plot_ResNet50.pdf', bbox_inches='tight')
plt.show()
```

Consider the situation of underfitting and overfitting in the training process, underfitting will happen when the dataset is not big enough or the model is not powerful enough, and the network has not learned the relevant patterns in the training data (*Overfit and underfit*). Along with the training process, overfitting might occur, the network has learned the patterns from training data that fails to generalize the testing dataset. The way to avoid overfitting is either by using the appropriate number of epochs or by applying the regularization techniques. To detect

overfitting or underfitting, the team compares the validation metrics to the training metrics that show the trend of accuracy and loss for training and validation. If both metrics are moving in the same direction, everything is fine; If the validation metric begins to stagnate while the training metric continues to improve, you are probably close to overfitting; If the validation metric is going in the wrong direction, the model is clearly overfitting (*Overfit and underfit*).

In the list of evaluation metrics, the team would use the accuracy score, Cohen kappa score, micro F1 score, and hamming loss. The accuracy score is one of the most straightforward metrics in machine learning, which can tell how accurate the model is, and how the predicted result matches the target. Cohen kappa score is one of the evaluation indicators, which can be used for binary or multi-class problems, but it is not suitable for multi-label. It indicates a good classification when the Cohen kappa score is higher than 0.8. The F1 score is a popular metric for evaluating the performance of classification. A higher F1 score shows a better classification. The macro F1 score will compute the metric independently for each class and then take the average, while the micro F1 score aggregates the contributions of all classes to compute the average metric (SHASHANK). In the present case, the micro-averaging F1 score is preferable to be used as an evaluation indicator in this multiple-class problem. The Hamming distance is also suitable for multi-classification problems, it measures the distance between the predicted label and the true label, and the value is between 0 and 1. A distance of 0 means that the predicted result is exactly the same as the real result, whereas, a distance of 1 means that the predicted result is completely contrary to the true target.

*Conclusion*

The final accuracy of ResNet50 is higher than 90% in prediction. After that, the team will compare with the pretrained model of VGG19, ResNet50 and customized model architecture to

get the best model which is suitable for this dataset. In this project, I have learned how to build the convolutional neural networks, and to know how some pretrained model architecture and to get an idea in network algorithms, which can help me to develop the customized CNN model in processing other datasets.

The percentage of the code is 35.8%

*Reference*

Ajitesh, K. (2020, October 28). *Keras - categorical cross entropy loss function*. Data Analytics. Retrieved April 24, 2022, from https://vitalflux.com/keras-categorical-cross-entropy-loss-function/#:~:text=sparse_categorical_crossentropy%3A%20Used%20as%20a%20loss,just%20has%20a%20different%20interface.

Radhakrishnan, P. (2017, October 18). *What are hyperparameters ? and how to tune the hyperparameters in a deep neural network?* Medium. Retrieved April 24, 2022, from https://towardsdatascience.com/what-are-hyperparameters-and-how-to-tune-the-hyperparameters-in-a-deep-neural-network-d0604917584a

Xu, Y., Noy, A., Lin, M., Qian, Q., Li, H., & Jin, R. (2020, October 3). *WeMix: How to better utilize data augmentation*. arXiv.org. Retrieved April 25, 2022, from https://arxiv.org/abs/2010.01267#:~:text=The%20main%20limitation%20of%20data,of%20existing%20data%20augmentation%20methods.