CECS 456
Machine Learning

Adrian Seth
Kelly Duangrudeeswat
Luke Sunaoka
Matthew Quinn

Introduction and Dataset

We chose to do this project because we had an interest in classifying animals and love cute cat pictures.

Our dataset consists of 28,000 medium quality images of 10 different classifications of animals(dog, cat, horse, spyder, butterfly, chicken, sheep, cow, squirrel, elephant).

The different classifications have between 2-5 thousand images each.

There are intentionally some incorrectly classified data in the set, the intent is to simulate real like conditions.

We retrieved our data from https://www.kaggle.com/datasets/alessiocorrado99/animals10

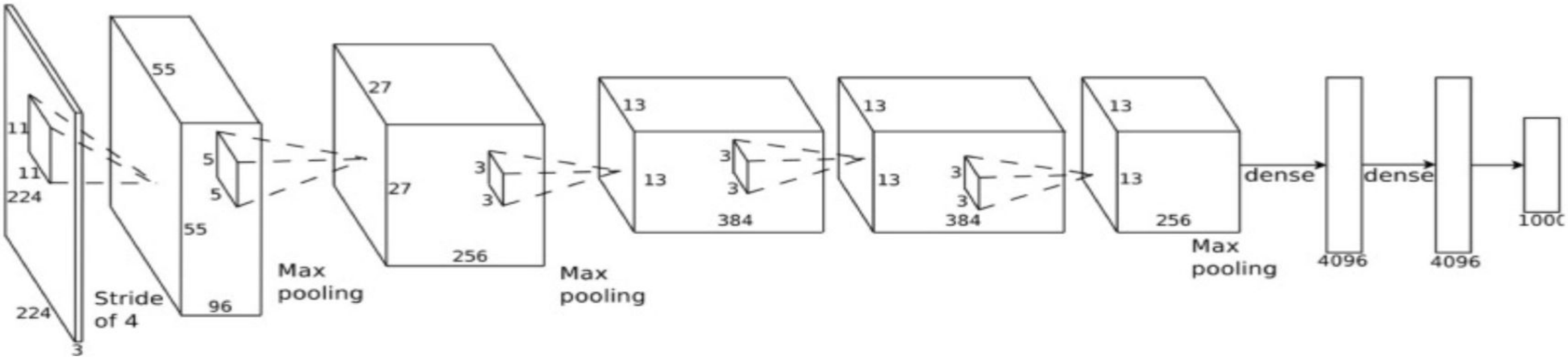
Description and Related Work

On Kaggle’s web site there are a lot of related works that use convolutional neural networks as well as other neural network models in order to classify the animal data set. Some import, famous models such as VGG, AlexNet, and ResNet which are all used to classify their images while others build their models from the ground up.

For our project we decided to implement different variations of CNN and see how well they perform.

Definitions

- Convolution Layer
  - CONV2D:
    - Filter
    - Kernel size
    - Activation
    - padding:
- MaxPool2D
- Flattening
- Dense



Implementation Methodology

- My model was based on the CNN demo we did in class
- I added additional layers and tweaked parameters to see what would improve or worsen accuracy
- A big part of my methodology was testing many different combinations of inputs and layers.
- My model ended with 18 layers as well as 3 drop out layers
- A batch size of 100 and epochs of 5 seemed to be the most consistent. When increasing the epochs I ran into “out of resource errors” This was while the model on my own hardware.
- Activation function Relu
- Total params 1.8m

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 222, 222, 16)      448

max_pooling2d (MaxPooling2D  (None, 111, 111, 16)      0
)

conv2d_1 (Conv2D)            (None, 109, 109, 32)      4640

conv2d_2 (Conv2D)            (None, 107, 107, 32)      9248

max_pooling2d_1 (MaxPooling  (None, 53, 53, 32)        0
2D)

conv2d_3 (Conv2D)            (None, 51, 51, 64)        18496

conv2d_4 (Conv2D)            (None, 49, 49, 64)        36928

max_pooling2d_2 (MaxPooling  (None, 24, 24, 64)        0
2D)

conv2d_5 (Conv2D)            (None, 22, 22, 128)       73856

conv2d_6 (Conv2D)            (None, 20, 20, 128)       147584

max_pooling2d_3 (MaxPooling  (None, 10, 10, 128)       0
2D)

conv2d_7 (Conv2D)            (None, 8, 8, 256)         295168

conv2d_8 (Conv2D)            (None, 6, 6, 256)         590080

max_pooling2d_4 (MaxPooling  (None, 3, 3, 256)         0
2D)

flatten (Flatten)            (None, 2304)              0

dense (Dense)                (None, 256)               590080

dense_1 (Dense)              (None, 128)               32896

dense_2 (Dense)              (None, 128)               16512

dense_3 (Dense)              (None, 10)                1290

=================================================================
Total params: 1,817,226
Trainable params: 1,817,226
Non-trainable params: 0
```

MQ

Results

- Accuracy of 58%
- There is a lot of room for improvement.
- Total loss: 1.23
- Validation loss: 1.32
- Validation accuracy 0.5675

```
Epoch 1/5
655/655 [==============================] - 111s 154ms/step - loss: 1.9750 - accuracy: 0.3131 - val_loss: 1.7947 - val_accuracy: 0.3802
Epoch 2/5
655/655 [==============================] - 98s 149ms/step - loss: 1.6921 - accuracy: 0.4213 - val_loss: 1.5221 - val_accuracy: 0.4786
Epoch 3/5
655/655 [==============================] - 98s 149ms/step - loss: 1.4872 - accuracy: 0.4997 - val_loss: 1.4437 - val_accuracy: 0.5166
Epoch 4/5
655/655 [==============================] - 97s 148ms/step - loss: 1.3558 - accuracy: 0.5467 - val_loss: 1.3398 - val_accuracy: 0.5501
Epoch 5/5
655/655 [==============================] - 97s 148ms/step - loss: 1.2381 - accuracy: 0.5853 - val_loss: 1.3258 - val_a[illegible]cy: 0.5675
```

MQ

Adrian’s Model

Design

I followed the CNN D
the foundation for my
smaller strides and s
amounts and slowly

he reason behind my decisions was to
ave my model slowly acquire
arameters while avoiding having too
any which i believed would reduce
verfitting and decrease training time.

```
cnn = tf.keras.models.Sequential() #initi
cnn.add(tf.keras.layers.Conv2D(filters=16
cnn.add(tf.keras.layers.Conv2D(filters=16
cnn.add(tf.keras.layers.MaxPool2D(pool_si

cnn.add(tf.keras.layers.Conv2D(filters=32
cnn.add(tf.keras.layers.Conv2D(filters=32
cnn.add(tf.keras.layers.MaxPool2D(pool_si

cnn.add(tf.keras.layers.Conv2D(filters=64
cnn.add(tf.keras.layers.Conv2D(filters=64
cnn.add(tf.keras.layers.MaxPool2D(pool_si

cnn.add(tf.keras.layers.Conv2D(filters=12
cnn.add(tf.keras.layers.Conv2D(filters=12
cnn.add(tf.keras.layers.MaxPool2D(pool_si

cnn.add(tf.keras.layers.Flatten()) #Flate
cnn.add(tf.keras.layers.Dense(units=128,
cnn.add(tf.keras.layers.Dense(units=64, a
cnn.add(tf.keras.layers.Dense(units=32, a
cnn.add(tf.keras.layers.Dense(units= 10,
```

```
ut_shape=[128, 128, 3])) #Convolution 1

onvolution 4
onvolution 5
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_16 (Conv2D)          (None, 128, 128, 16)      448

 conv2d_17 (Conv2D)          (None, 128, 128, 16)      2320

 max_pooling2d_8 (MaxPooling  (None, 64, 64, 16)       0
 2D)

 conv2d_18 (Conv2D)          (None, 64, 64, 32)        4640

 conv2d_19 (Conv2D)          (None, 64, 64, 32)        9248

 max_pooling2d_9 (MaxPooling  (None, 32, 32, 32)       0
 2D)

 conv2d_20 (Conv2D)          (None, 32, 32, 64)        18496

 conv2d_21 (Conv2D)          (None, 32, 32, 64)        36928

 max_pooling2d_10 (MaxPoolin  (None, 16, 16, 64)       0
 g2D)

 conv2d_22 (Conv2D)          (None, 16, 16, 128)       73856

 conv2d_23 (Conv2D)          (None, 16, 16, 128)       147584

 max_pooling2d_11 (MaxPoolin  (None, 8, 8, 128)        0
 g2D)

 flatten_2 (Flatten)         (None, 8192)              0

 dense_7 (Dense)             (None, 128)               1048704

 dense_8 (Dense)             (None, 64)                8256

 dense_9 (Dense)             (None, 32)                2080

 dense_10 (Dense)            (None, 10)                330

=================================================================
Total params: 1,352,890
Trainable params: 1,352,890
Non-trainable params: 0
_________________________________________________________________
```

Adrian

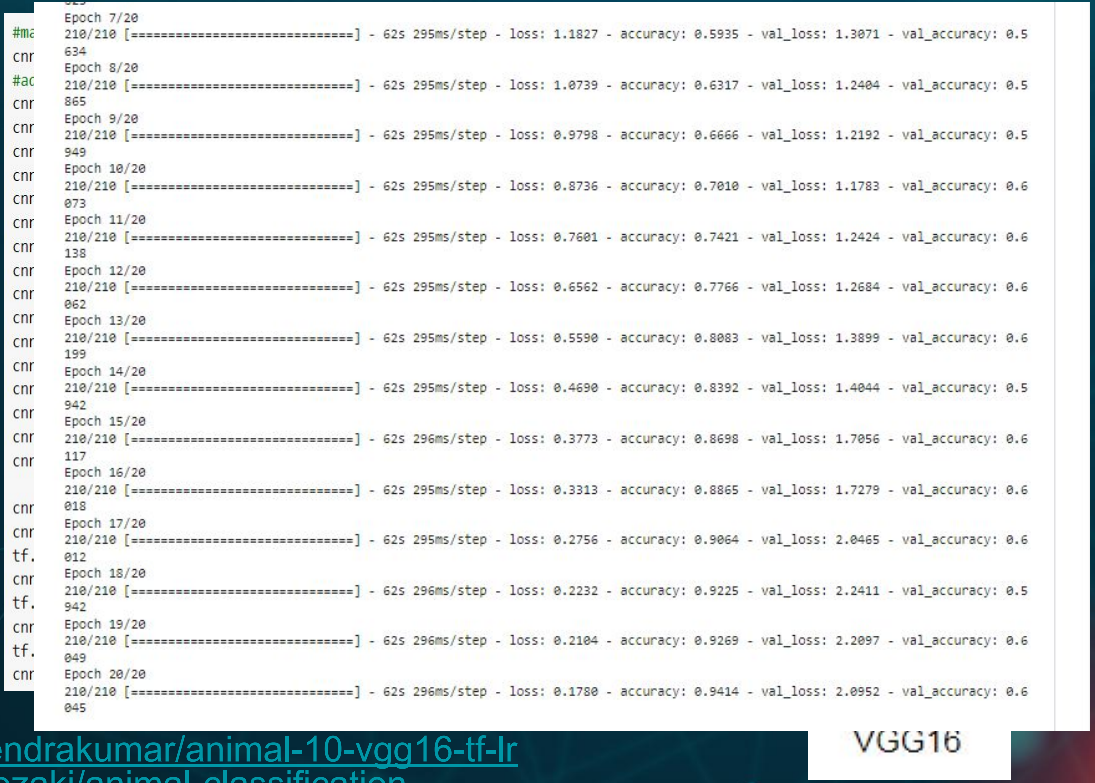
Training accuracy - 81%
Testing accuracy - 66%

```
Epoch 1/10
84/84 [==============================] - 9s 102ms/step - loss: 2.2487 - accuracy: 0.2200 - val_loss: 1.9873 - val_accuracy: 0.3435
Epoch 2/10
84/84 [==============================] - 7s 88ms/step - loss: 1.8074 - accuracy: 0.3872 - val_loss: 1.6525 - val_accuracy: 0.4363
Epoch 3/10
84/84 [==============================] - 7s 88ms/step - loss: 1.4918 - accuracy: 0.4911 - val_loss: 1.4768 - val_accuracy: 0.4956
Epoch 4/10
84/84 [==============================] - 7s 89ms/step - loss: 1.3178 - accuracy: 0.5452 - val_loss: 1.3313 - val_accuracy: 0.5520
Epoch 5/10
84/84 [==============================] - 7s 88ms/step - loss: 1.1573 - accuracy: 0.6023 - val_loss: 1.1898 - val_accuracy: 0.5999
Epoch 6/10
84/84 [==============================] - 7s 88ms/step - loss: 1.0157 - accuracy: 0.6558 - val_loss: 1.2301 - val_accuracy: 0.5752
Epoch 7/10
84/84 [==============================] - 7s 89ms/step - loss: 0.9153 - accuracy: 0.6903 - val_loss: 1.0807 - val_accuracy: 0.6417
Epoch 8/10
84/84 [==============================] - 7s 89ms/step - loss: 0.7815 - accuracy: 0.7332 - val_loss: 1.1224 - val_accuracy: 0.6417
Epoch 9/10
84/84 [==============================] - 7s 89ms/step - loss: 0.6776 - accuracy: 0.7685 - val_loss: 1.1660 - val_accuracy: 0.6393
Epoch 10/10
84/84 [==============================] - 8s 90ms/step - loss: 0.5539 - accuracy: 0.8097 - val_loss: 1.1643 - val_accuracy: 0.6484
```

```
164/164 [==============================] - 1s 8ms/step - loss: 1.1186 - accuracy: 0.6596
Total loss on Testing Set: 1.118599534034729
Accuracy of Testing Set: 0.6595866680145264
```

Implementation Methodology

- Based on VGG16
- Extracting data
- Splitting the dataset
- Creating the model
  - 16 layers
  - Activation = relu
  - Batch size = 100
  - Epoch = 20



Luke Sunaoka's Model

References:

- https://www.kaggle.com/code/bhupendrakumar/animal-10-vgg16-tf-lr
- https://www.kaggle.com/code/min4tozaki/animal-classification

Results

Luke Sunaoka's Model

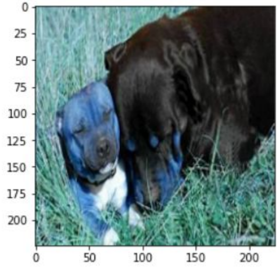
- Total Loss on Testing Set: 0.533
- Accuracy of Testing Set: 0.883
- Did surprisingly well!
- May be due to implementation

```
score = cnn.evaluate(x_test, y_test)
print('Total loss on Testing Set:', score[0])
print('Accuracy of Testing Set:', score[1])
```

```
164/164 [==============================] - 5s 33ms/step - loss: 0.5330 - accuracy: 0.8831
Total loss on Testing Set: 0.5330036282539368
Accuracy of Testing Set: 0.8831169009208679
```

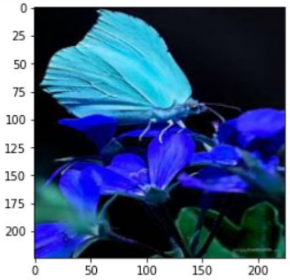
```
#prediction 1
plt.imshow(x_new[0]/255.) #displays the image
print(names[y_pred[0]])
```

dog



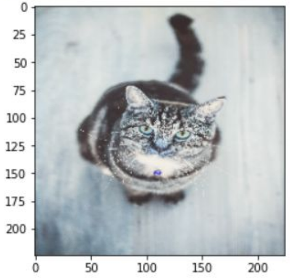

```
#prediction 2
plt.imshow(x_new[1]/255.) #displays the image
print(names[y_pred[1]])
```

butterfly




```
#prediction 3
plt.imshow(x_new[2]/255.) #displays the image
print(names[y_pred[2]])
```

cat



Model: AlexNet

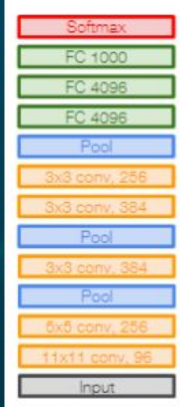
Reasoning

Achieves better results for image classification

- Relu activation function is used

  $f(x) = \max(0, x)$

- Standardization
  - Dropout method prevents overfitting
- Data Enhancement
  - Dropout method: randomly setting output of some neurons = 0



AlexNet

Kelly Duangrudeeswat

Implementation Methodology

```python
# Layer 1
alexnet.add(Conv2D(96, (11, 11), input_shape=(100, 100, 3), padding='same', kernel_regularizer=l2(0)))
alexnet.add(BatchNormalization())
alexnet.add(Activation('relu'))
alexnet.add(MaxPooling2D(pool_size=(2, 2)))

# Layer 2
alexnet.add(Conv2D(256, (5, 5), padding='same'))
alexnet.add(BatchNormalization())
alexnet.add(Activation('relu'))
alexnet.add(MaxPooling2D(pool_size=(2, 2)))

# Layer 3
alexnet.add(ZeroPadding2D((1, 1)))
alexnet.add(Conv2D(512, (3, 3), padding='same'))
alexnet.add(BatchNormalization())
alexnet.add(Activation('relu'))
alexnet.add(MaxPooling2D(pool_size=(2, 2)))

# Layer 4
alexnet.add(ZeroPadding2D((1, 1)))
alexnet.add(Conv2D(1024, (3, 3), padding='same'))
alexnet.add(BatchNormalization())
alexnet.add(Activation('relu'))

# Layer 5
alexnet.add(ZeroPadding2D((1, 1)))
alexnet.add(Conv2D(1024, (3, 3), padding='same'))
alexnet.add(BatchNormalization())
alexnet.add(Activation('relu'))
alexnet.add(MaxPooling2D(pool_size=(2, 2)))

# Layer 6
alexnet.add(Flatten())
alexnet.add(Dense(3072))
alexnet.add(BatchNormalization())
alexnet.add(Activation('relu'))
alexnet.add(Dropout(0.5))

# Layer 7
alexnet.add(Dense(4096))
alexnet.add(BatchNormalization())
alexnet.add(Activation('relu'))
alexnet.add(Dropout(0.5))

# Layer 8
alexnet.add(Dense(10))
alexnet.add(BatchNormalization())
alexnet.add(Activation('softmax'))
```

Design

- Based on AlexNet
- Data extraction
- Split data into Training and Testing datasets
  - Training: 20,858
  - Testing: 5,320
- Training parameters:
  - 8 layers (5 convolutional, 3 fully connected)
  - Activation func = relu
  - Batch size = 53
  - Epoch = 69

Kelly Duangrudeeswat

Results

```
Epoch 1/69
53/53 [==============================] - 31s 375ms/step - loss: 2.2115 - accuracy: 0.2360 - val_loss: 4.9016 - val_accuracy: 0.1205
Epoch 2/69
53/53 [==============================] - 14s 270ms/step - loss: 1.8616 - accuracy: 0.3528 - val_loss: 3.0363 - val_accuracy: 0.2145
Epoch 3/69
53/53 [==============================] - 14s 271ms/step - loss: 1.7204 - accuracy: 0.4087 - val_loss: 2.3990 - val_accuracy: 0.2472
Epoch 4/69
53/53 [==============================] - 14s 270ms/step - loss: 1.5958 - accuracy: 0.4564 - val_loss: 1.9446 - val_accuracy: 0.3402
Epoch 5/69
53/53 [==============================] - 14s 270ms/step - loss: 1.5386 - accuracy: 0.4817 - val_loss: 3.6854 - val_accuracy: 0.2083
Epoch 6/69
53/53 [==============================] - 14s 269ms/step - loss: 1.4388 - accuracy: 0.5173 - val_loss: 2.8521 - val_accuracy: 0.1889
```

```
Epoch 64/69
53/53 [==============================] - 16s 295ms/step - loss: 0.4615 - accuracy: 0.9306 - val_loss: 1.0443 - val_accuracy: 0.7256
Epoch 65/69
53/53 [==============================] - 16s 295ms/step - loss: 0.4727 - accuracy: 0.9206 - val_loss: 1.0382 - val_accuracy: 0.7310
Epoch 66/69
53/53 [==============================] - 16s 295ms/step - loss: 0.4807 - accuracy: 0.9206 - val_loss: 1.1581 - val_accuracy: 0.6562
Epoch 67/69
53/53 [==============================] - 15s 294ms/step - loss: 0.4648 - accuracy: 0.9228 - val_loss: 1.2414 - val_accuracy: 0.6188
Epoch 68/69
53/53 [==============================] - 16s 295ms/step - loss: 0.4468 - accuracy: 0.9292 - val_loss: 1.0275 - val_accuracy: 0.7261
Epoch 69/69
53/53 [==============================] - 16s 296ms/step - loss: 0.4307 - accuracy: 0.9349 - val_loss: 1.0156 - val_accuracy: 0.7530
```

Epoch 69/69

- Accuracy: 0.9349
- Loss: 0.4307
- Validation accuracy: 0.7530
- Validation loss: 1.0156

Total params: 229,985,586

Trainable params: 229,965,406

Non-trainable params: 20,180

```
max_pooling2d_10 (MaxPoolin  (None, 13, 13, 512)      0
g2D)

zero_padding2d_7 (ZeroPaddi  (None, 15, 15, 512)      0
ng2D)

conv2d_13 (Conv2D)          (None, 15, 15, 1024)     4719616

batch_normalization_19 (Bat  (None, 15, 15, 1024)     4096
chNormalization)

activation_19 (Activation)  (None, 15, 15, 1024)     0

zero_padding2d_8 (ZeroPaddi  (None, 17, 17, 1024)     0
ng2D)

conv2d_14 (Conv2D)          (None, 17, 17, 1024)     9438208

batch_normalization_20 (Bat  (None, 17, 17, 1024)     4096
chNormalization)

activation_20 (Activation)  (None, 17, 17, 1024)     0

max_pooling2d_11 (MaxPoolin  (None, 8, 8, 1024)       0
g2D)

flatten_2 (Flatten)         (None, 65536)            0

dense_6 (Dense)             (None, 3072)             201329664

batch_normalization_21 (Bat  (None, 3072)             12288
chNormalization)

activation_21 (Activation)  (None, 3072)             0

dropout_4 (Dropout)         (None, 3072)             0

dense_7 (Dense)             (None, 4096)             12587008

batch_normalization_22 (Bat  (None, 4096)             16384
chNormalization)

activation_22 (Activation)  (None, 4096)             0

dropout_5 (Dropout)         (None, 4096)             0

dense_8 (Dense)             (None, 10)               40970

batch_normalization_23 (Bat  (None, 10)               40
chNormalization)

activation_23 (Activation)  (None, 10)               0

=================================================================
Total params: 229,985,586
Trainable params: 229,965,406
Non-trainable params: 20,180
_________________________________________________________________
```

Kelly Duangrudeeswat

Results, Analysis, and Conclusion

- Kelly (69) and Luke(20) used a higher number of epoch while Matt(5) and Adrian(10) used a lower number of epochs
- Accuracy scores were as follows:
  Matthew : 0.58
  Adrian:0.65
  Luke: 0.88
  Kelly: 0.93

- Although VGG should be having a higher accuracy, because Kelly had higher number of epochs with her AlexNet model, her accuracy had the highest
- Furthermore, Matt and Adrian implemented their own CNN which maybe why its performance is limited in accuracy and results.
- The project was overall an accumulation of everything we’ve learned and it was great to see it being applied.

Thank you