# EE6770_F24_HW5

October 28, 2024

#

EE6770 Fall 2024 Homework 5: Cats and Dogs Classifier

1. **Write comments** in the code to explain your thoughts.
2. **Important: Execute the codes and show the results**.
3. **Do your own work.**

#### 0.0.1 Submission:

- **Submit this notebook file and the pdf version** - remember to add your name in the filename.
- Deadline: 11:59 pm, 10/28 (Monday)

## 0.1 Assignment Objectives:

### 0.1.1 In this assignment, you will develop a CNN model for the cat-and-dog classifer.

**You will create `at least two models`, applying the various techniques we discussed for improving the performance.**

1. Deeper Conv layers and/or FC layers
2. Image augmentation
3. Transfer learning
4. Regularization: L1/L2, Batch Normalization, Dropout, Max Norm
5. Increasing image size
6. Increasing size of the train/validation/test dataset

- You will compare the performance of your models with the baseline VGG-5 model that we discussed in class.
- Image size is limited to 128-by-128 or smaller
- Performance requirement: the accuracy on the test data needs to be better than 87.5% for at least one of your models

### 0.1.2 Cats & Dogs Dataset

- 
- 
- 

### 0.1.3 Load tool modules

[119]:
```
import tensorflow as tf
from tensorflow import keras
from keras import layers, models

print(tf.config.list_physical_devices())
```

```
[PhysicalDevice(name='/physical_device:CPU:0', device_type='CPU'),
PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]
```

### 0.1.4 Load CNN models

[120]:
```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten
from keras.layers import Dropout
```

### 0.1.5 Load the image processing tools

[121]:
```
from keras.preprocessing.image import load_img, img_to_array
from keras.utils import image_dataset_from_directory
```

### 0.1.6 Load and Process the dataset

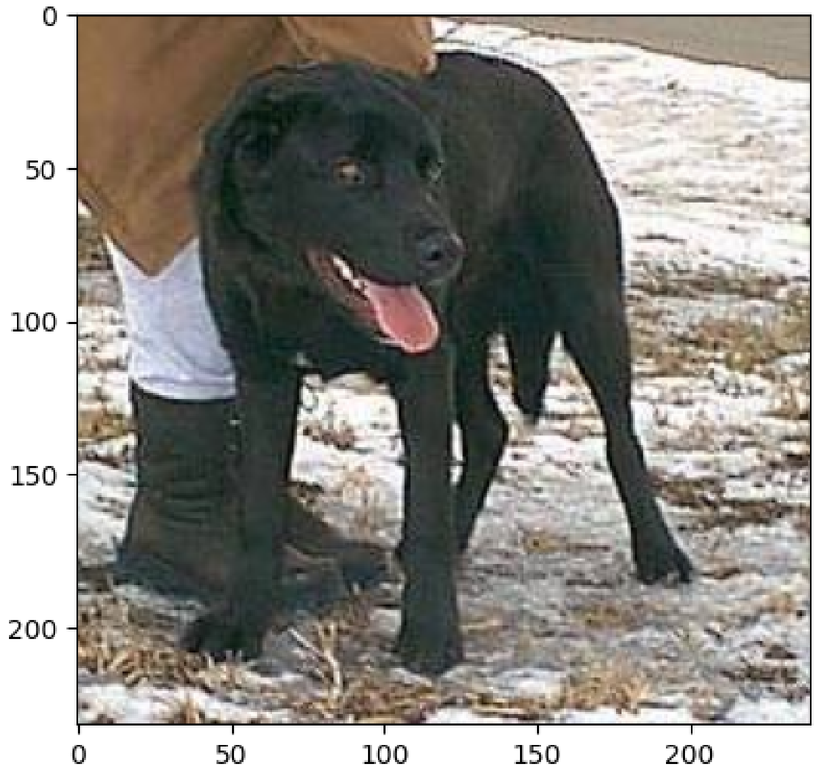
**Create the subdirectory structures per the requirement.**

[122]:
```
import os
import shutil

#--------------------------------------#
# Select the number of files to train  #
N = 25000                              #
#--------------------------------------#

source_dir = "./train"
train_dir = "./cats_dogs"
```

```
os.makedirs(train_dir, exist_ok=True)

def organize_images(name, start, stop):
    for pet in ("cat", "dog"):
        dir = train_dir + "/" + name + "/" + pet
        os.makedirs(dir, exist_ok=True)

        images = [f"{pet}.{i}.jpg" for i in range(start, stop)]
        for file in images:
            shutil.copy(src=source_dir + '/' + file, dst=dir + '/ '+ file)


N_split = int(25000 / 2)
train_perc = int(N_split * 0.6)
val_percent = train_perc + int(N_split * 0.15)
test_percent = val_percent + int(N_split * 0.25)

organize_images("train", start=0, stop=train_perc)
organize_images("validation", start=train_perc, stop=val_percent)
organize_images("test", start=val_percent, stop=test_percent)
```

### 0.1.7 Display 2 input images: one for dog, and one for cat

[123]:
```
import matplotlib.pyplot as plt
import random


dog_dir = os.listdir(train_dir+'/test'+'/dog')
cat_dir = os.listdir(train_dir+'/test'+'/cat')
dog_sample = random.choice(dog_dir)
cat_sample = random.choice(cat_dir)


dog_img = load_img(os.path.join(train_dir+'/test'+'/dog', dog_sample))
cat_img = load_img(os.path.join(train_dir+'/test'+'/cat', cat_sample))

plt.imshow(dog_img)
plt.show()

plt.imshow(cat_img)
plt.show()
```


0
50
100
150
200
0
50
100
150
200

## 1 Baseline CNN Model: VGG-5

[124]:
```
# create data generator

# ----------- Target Picture Size ------------ #
target_pic_size = (64, 64)                     #
target_pic_shape = target_pic_size + (3,)      #
# -------------------------------------------- #



train_data = image_dataset_from_directory(train_dir + '/train',
            color_mode='rgb', batch_size=64, image_size=target_pic_size)

val_data = image_dataset_from_directory(train_dir + '/validation',
           color_mode='rgb', batch_size=64, image_size=target_pic_size)

test_data = image_dataset_from_directory(train_dir + '/test',
            color_mode='rgb', batch_size=64, image_size=target_pic_size)
```

```python
model = Sequential()

# Layer 1
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=target_pic_shape))
model.add(MaxPooling2D((2, 2)))

# Layer 2
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))

# Layer 3
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))

model.add(Flatten())

# FC Layers
model.add(Dense(128, activation='relu'))
model.add(Dense(1, activation='sigmoid'))


model.summary()
```

```
Found 15000 files belonging to 2 classes.
Found 3750 files belonging to 2 classes.
Found 6250 files belonging to 2 classes.
```

**Model: "sequential_20"**

| Layer (type) | Output Shape | Param # |
|---|---|---|
| conv2d_90 (Conv2D) | (None, 62, 62, 32) | 896 |
| max_pooling2d_89 (MaxPooling2D) | (None, 31, 31, 32) | 0 |
| conv2d_91 (Conv2D) | (None, 29, 29, 64) | 18,496 |
| max_pooling2d_90 (MaxPooling2D) | (None, 14, 14, 64) | 0 |
| conv2d_92 (Conv2D) | (None, 12, 12, 128) | 73,856 |

```
max_pooling2d_91 (MaxPooling2D)   (None, 6, 6, 128)                  0

flatten_19 (Flatten)              (None, 4608)                       0

dense_48 (Dense)                  (None, 128)                  589,952

dense_49 (Dense)                  (None, 1)                        129
```

```
Total params: 683,329 (2.61 MB)
```

```
Trainable params: 683,329 (2.61 MB)
```

```
Non-trainable params: 0 (0.00 B)
```

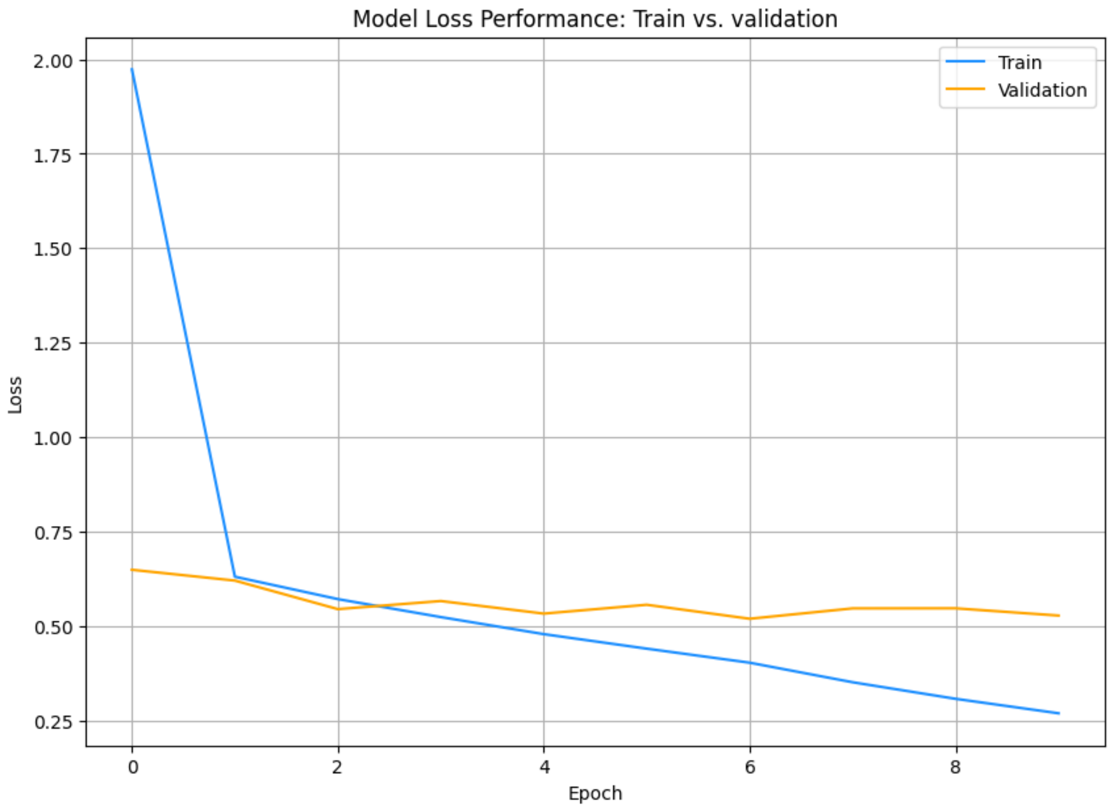
[125]:
```
model.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics =␣
↪['accuracy'])

history = model.fit(train_data, epochs=10, batch_size=64,␣
↪validation_data=val_data, verbose=1)
```

```
Epoch 1/10
235/235              5s 18ms/step -
accuracy: 0.5443 - loss: 6.9580 - val_accuracy: 0.6072 - val_loss: 0.6492
Epoch 2/10
235/235              3s 12ms/step -
accuracy: 0.6418 - loss: 0.6380 - val_accuracy: 0.6557 - val_loss: 0.6206
Epoch 3/10
235/235              3s 12ms/step -
accuracy: 0.6859 - loss: 0.5860 - val_accuracy: 0.7277 - val_loss: 0.5451
Epoch 4/10
235/235              3s 11ms/step -
accuracy: 0.7318 - loss: 0.5274 - val_accuracy: 0.7061 - val_loss: 0.5665
Epoch 5/10
235/235              3s 11ms/step -
accuracy: 0.7676 - loss: 0.4899 - val_accuracy: 0.7437 - val_loss: 0.5332
Epoch 6/10
235/235              3s 11ms/step -
```

```
accuracy: 0.7933 - loss: 0.4487 - val_accuracy: 0.7405 - val_loss: 0.5566
Epoch 7/10
235/235              3s 11ms/step -
accuracy: 0.8144 - loss: 0.4129 - val_accuracy: 0.7581 - val_loss: 0.5196
Epoch 8/10
235/235              3s 11ms/step -
accuracy: 0.8448 - loss: 0.3584 - val_accuracy: 0.7693 - val_loss: 0.5470
Epoch 9/10
235/235              3s 11ms/step -
accuracy: 0.8587 - loss: 0.3177 - val_accuracy: 0.7715 - val_loss: 0.5472
Epoch 10/10
235/235              3s 11ms/step -
accuracy: 0.8766 - loss: 0.2791 - val_accuracy: 0.7808 - val_loss: 0.5280
```

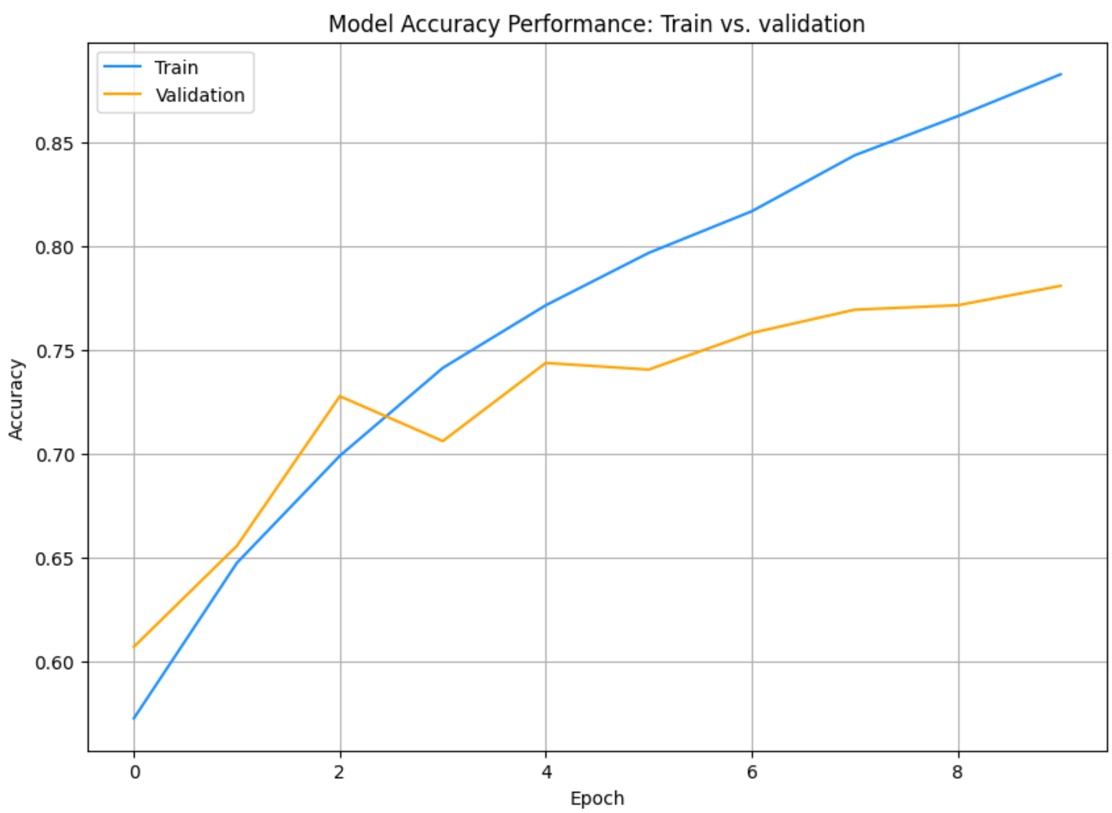
[126]:

```
J = history.history['loss']  # Loss data for Training
J_val = history.history['val_loss']

plt.figure(figsize=(10,7))

plt.title('Model Loss Performance: Train vs. validation')
plt.plot(J, color='DodgerBlue', label='Train')
plt.plot(J_val, color='orange', label='Validation')

plt.ylabel('Loss')
plt.xlabel('Epoch')

plt.legend()
plt.grid()
plt.show()
```




[127]:

```
accu = history.history['accuracy']  # Loss data for Training
accu_val = history.history['val_accuracy']

plt.figure(figsize=(10,7))

plt.title('Model Accuracy Performance: Train vs. validation')
plt.plot(accu, color='DodgerBlue', label='Train')
plt.plot(accu_val, color='orange', label='Validation')

plt.ylabel('Accuracy')
plt.xlabel('Epoch')

plt.legend()
plt.grid()
plt.show()

loss, accuracy = model.evaluate(test_data, verbose=1)

y_pred = model.predict(test_data)
```




```
98/98              1s 10ms/step -
accuracy: 0.7832 - loss: 0.5104
98/98              1s 9ms/step
```

[128]:
```
print("Accuracy = ", accuracy *100, "%")
print("Loss = ", loss, "%")
```

```
Accuracy =  78.33600044250488 %
Loss =  0.5049804449081421 %
```

# 2 Build CNN Model One

## 2.1 Define the CNN model

**Use CONV, POOL and FC layers to construct your CNN model. You can also load pre-trained model, if transfer learning is used. You will train and test the model after this step.**

[129]:
```
# ----------- Target Picture Size ------------ #
target_pic_size = (128, 128)                    #
target_pic_shape = target_pic_size + (3,)      #
# -------------------------------------------- #

train_data = image_dataset_from_directory(train_dir + '/train',
            color_mode='rgb', batch_size=64, image_size=target_pic_size)

val_data = image_dataset_from_directory(train_dir + '/validation',
           color_mode='rgb', batch_size=64, image_size=target_pic_size)

test_data = image_dataset_from_directory(train_dir + '/test',
            color_mode='rgb', batch_size=64, image_size=target_pic_size)



cnn1 = Sequential()


# Input layer
cnn1.add(keras.Input(shape=target_pic_shape))

# Layer 1
cnn1.add(Conv2D(32, (3, 3), activation='relu', kernel_regularizer='l2'))
cnn1.add(MaxPooling2D((2, 2)))

# Layer 2
cnn1.add(Conv2D(64, (3, 3), activation='relu', kernel_regularizer='l2'))
cnn1.add(MaxPooling2D((2, 2)))

# Layer 3
cnn1.add(Conv2D(128, (3, 3), activation='relu', kernel_regularizer='l2'))
cnn1.add(MaxPooling2D((2, 2)))

# Layer 4
cnn1.add(Conv2D(256, (3, 3), activation='relu', kernel_regularizer='l2'))
cnn1.add(MaxPooling2D((2, 2)))

# Layer 5
cnn1.add(Conv2D(512, (3, 3), activation='relu', kernel_regularizer='l2'))
cnn1.add(MaxPooling2D((2, 2)))

cnn1.add(Flatten())
cnn1.add(Dropout(rate=0.25))

# FC Layers
cnn1.add(Dense(128, activation=keras.activations.gelu))
```

```
cnn1.add(Dense(128, activation=keras.activations.gelu))
cnn1.add(Dense(1, activation=keras.activations.sigmoid))
```

```
Found 15000 files belonging to 2 classes.
Found 3750 files belonging to 2 classes.
Found 6250 files belonging to 2 classes.
```

### 2.1.1 Print the model summary that shows the output shape and # of parameters for each layer.

[130]:
```
cnn1.summary()
```

**Model: "sequential_21"**

| Layer (type) | Output Shape | Param # |
|---|---|---|
| conv2d_93 (Conv2D) | (None, 126, 126, 32) | 896 |
| max_pooling2d_92 (MaxPooling2D) | (None, 63, 63, 32) | 0 |
| conv2d_94 (Conv2D) | (None, 61, 61, 64) | 18,496 |
| max_pooling2d_93 (MaxPooling2D) | (None, 30, 30, 64) | 0 |
| conv2d_95 (Conv2D) | (None, 28, 28, 128) | 73,856 |
| max_pooling2d_94 (MaxPooling2D) | (None, 14, 14, 128) | 0 |
| conv2d_96 (Conv2D) | (None, 12, 12, 256) | 295,168 |
| max_pooling2d_95 (MaxPooling2D) | (None, 6, 6, 256) | 0 |
| conv2d_97 (Conv2D) | (None, 4, 4, 512) | 1,180,160 |
| max_pooling2d_96 (MaxPooling2D) | (None, 2, 2, 512) | 0 |
| flatten_20 (Flatten) | (None, 2048) | 0 |
| dropout_10 (Dropout) | (None, 2048) | 0 |
| dense_50 (Dense) | (None, 128) | 262,272 |

```
 dense_51 (Dense)                    (None, 128)                        16,512

 dense_52 (Dense)                    (None, 1)                             129
```

```
 Total params: 1,847,489 (7.05 MB)
```

```
 Trainable params: 1,847,489 (7.05 MB)
```

```
 Non-trainable params: 0 (0.00 B)
```

#### 2.1.2 Question: What are the total number of parameters for the model?

**Answer:** This model has 1,844,641 parameters and takes up 7.04 MB

### 2.2 Train the CNN Model

**Note: Display the history when running model.fit( )**

[131]:
```
cnn1.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics =␣
 ↪['accuracy'])

history = cnn1.fit(train_data, epochs=20, batch_size=64,␣
 ↪validation_data=val_data, verbose=1)
```

```
Epoch 1/20
235/235              16s 57ms/step -
accuracy: 0.5476 - loss: 4.6775 - val_accuracy: 0.6291 - val_loss: 1.6942
Epoch 2/20
235/235              10s 41ms/step -
accuracy: 0.6406 - loss: 1.5152 - val_accuracy: 0.7421 - val_loss: 1.0645
Epoch 3/20
235/235              10s 41ms/step -
accuracy: 0.7330 - loss: 1.0161 - val_accuracy: 0.7656 - val_loss: 0.8439
Epoch 4/20
235/235              10s 41ms/step -
```

```
accuracy: 0.7792 - loss: 0.7855 - val_accuracy: 0.8117 - val_loss: 0.6680
Epoch 5/20
235/235             10s 41ms/step -
accuracy: 0.8017 - loss: 0.6721 - val_accuracy: 0.8037 - val_loss: 0.6435
Epoch 6/20
235/235             10s 40ms/step -
accuracy: 0.8285 - loss: 0.5897 - val_accuracy: 0.8347 - val_loss: 0.5577
Epoch 7/20
235/235             10s 41ms/step -
accuracy: 0.8448 - loss: 0.5292 - val_accuracy: 0.7392 - val_loss: 0.7796
Epoch 8/20
235/235             10s 41ms/step -
accuracy: 0.8425 - loss: 0.5317 - val_accuracy: 0.8341 - val_loss: 0.5501
Epoch 9/20
235/235             10s 41ms/step -
accuracy: 0.8603 - loss: 0.4892 - val_accuracy: 0.8507 - val_loss: 0.4961
Epoch 10/20
235/235             10s 41ms/step -
accuracy: 0.8748 - loss: 0.4576 - val_accuracy: 0.8581 - val_loss: 0.4931
Epoch 11/20
235/235             10s 41ms/step -
accuracy: 0.8782 - loss: 0.4473 - val_accuracy: 0.8723 - val_loss: 0.4829
Epoch 12/20
235/235             10s 41ms/step -
accuracy: 0.8864 - loss: 0.4272 - val_accuracy: 0.8787 - val_loss: 0.4440
Epoch 13/20
235/235             10s 40ms/step -
accuracy: 0.8926 - loss: 0.4165 - val_accuracy: 0.8808 - val_loss: 0.4494
Epoch 14/20
235/235             10s 41ms/step -
accuracy: 0.8941 - loss: 0.4092 - val_accuracy: 0.8229 - val_loss: 0.6158
Epoch 15/20
235/235             9s 40ms/step -
accuracy: 0.9090 - loss: 0.3781 - val_accuracy: 0.8893 - val_loss: 0.4421
Epoch 16/20
235/235             9s 40ms/step -
accuracy: 0.9094 - loss: 0.3764 - val_accuracy: 0.8792 - val_loss: 0.4616
Epoch 17/20
235/235             10s 40ms/step -
accuracy: 0.9139 - loss: 0.3704 - val_accuracy: 0.8589 - val_loss: 0.4787
Epoch 18/20
235/235             10s 41ms/step -
accuracy: 0.9116 - loss: 0.3684 - val_accuracy: 0.8883 - val_loss: 0.4395
Epoch 19/20
235/235             10s 41ms/step -
accuracy: 0.9186 - loss: 0.3584 - val_accuracy: 0.8909 - val_loss: 0.4354
Epoch 20/20
235/235             10s 40ms/step -
```

```
accuracy: 0.9221 - loss: 0.3425 - val_accuracy: 0.8813 - val_loss: 0.4558
```

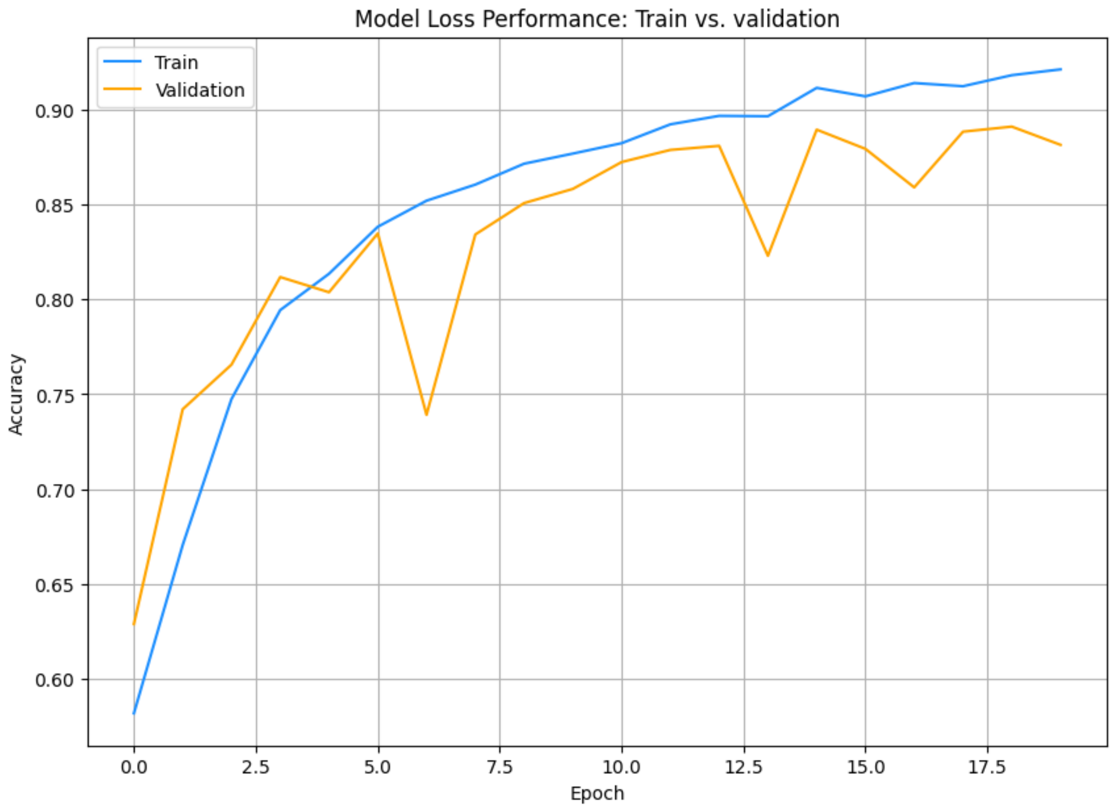
### 2.2.1 Question: What is the estimated total model training time?

**Answer:** With the CPU/GPU combination I have it takes about 10 minutes to run

### 2.2.2 Compare Loss and Accuracy Performance for train and validation data

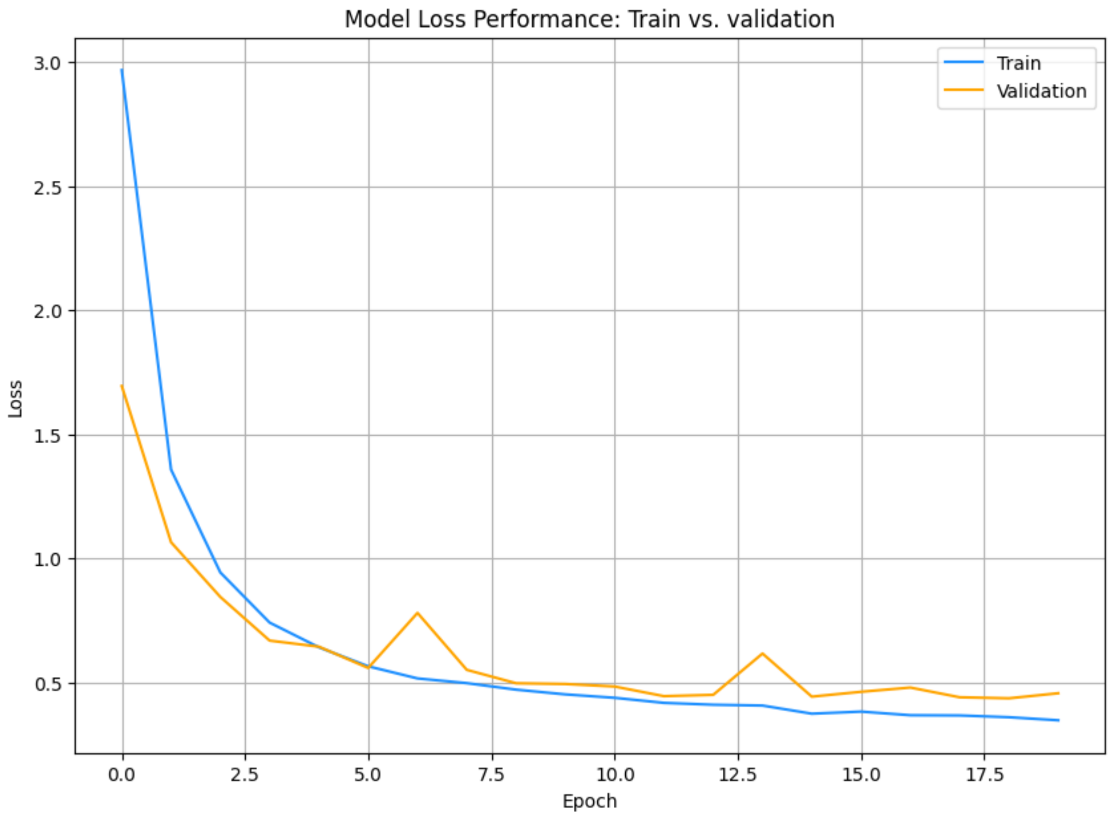
**Plot the loss data, for both train and validation data**

[132]:
```python
J = history.history['accuracy']  # Loss data for Training
J_val = history.history['val_accuracy']

plt.figure(figsize=(10,7))

plt.title('Model Loss Performance: Train vs. validation')
plt.plot(J, color='DodgerBlue', label='Train')
plt.plot(J_val, color='orange', label='Validation')

plt.ylabel('Accuracy')
plt.xlabel('Epoch')

plt.legend()
plt.grid()
plt.show()
```




**Plot the accuracy data, for both train and validation data**

[133]:
```
J = history.history['loss']   # Loss data for Training
J_val = history.history['val_loss']

plt.figure(figsize=(10,7))

plt.title('Model Loss Performance: Train vs. validation')
plt.plot(J, color='DodgerBlue', label='Train')
plt.plot(J_val, color='orange', label='Validation')

plt.ylabel('Loss')
plt.xlabel('Epoch')

plt.legend()
plt.grid()
plt.show()
```




### 2.3 Test the CNN Model

**Note: Display the history when running model.evaluate( )**

#### 2.3.1 Question: What is the estimated inference (testing) time on test dataset?

**Answer:** It took 14 seconds to run the inference

#### 2.3.2 Print the final loss and accuracy of the test data

[134]:
```
loss, accuracy = cnn1.evaluate(test_data, verbose=1)

y_pred = cnn1.predict(test_data)

print("Accuracy: ", round(accuracy * 100, 2), "%")
print("Loss: ", round(loss, 2))
```

```
98/98              1s 14ms/step -
accuracy: 0.8831 - loss: 0.4641
98/98              1s 10ms/step
Accuracy:  88.34 %
Loss:  0.45
```

#### 2.3.3 Save the CNN model parameters

[135]:
```
cnn1.save('./cnn1.keras')
```

# 3 Build CNN Model Two

**For your second and subsequent models, follow the same set of instructions provided for Model One**

[136]:
```
# ----------- Target Picture Size ------------ #
target_pic_size = (128, 128)                     #
target_pic_shape = target_pic_size + (3,)      #
# -------------------------------------------- #

train_data = image_dataset_from_directory(train_dir + '/train',
            color_mode='rgb', batch_size=64, image_size=target_pic_size)

val_data = image_dataset_from_directory(train_dir + '/validation',
           color_mode='rgb', batch_size=64, image_size=target_pic_size)

test_data = image_dataset_from_directory(train_dir + '/test',
            color_mode='rgb', batch_size=64, image_size=target_pic_size)



cnn2 = Sequential()

cnn2.add(keras.Input(shape=target_pic_shape))

cnn2.add(Conv2D(16, (3,3), activation= 'relu'))
cnn2.add(MaxPooling2D((2,2)))

cnn2.add(Conv2D(32, (3,3), activation= 'relu'))
cnn2.add(MaxPooling2D((2,2)))

cnn2.add(Conv2D(64, (3,3), activation= 'relu'))
cnn2.add(MaxPooling2D((2,2)))

cnn2.add(Conv2D(128, (3,3), activation= 'relu'))
cnn2.add(MaxPooling2D((2,2)))

cnn2.add(Conv2D(256, (3,3), activation= 'relu'))
cnn2.add(MaxPooling2D((2,2)))

cnn2.add(Flatten())

cnn2.add(Dense(128, activation='relu'))
```

```python
cnn2.add(Dense(1, activation='sigmoid'))
```

```
Found 15000 files belonging to 2 classes.
Found 3750 files belonging to 2 classes.
Found 6250 files belonging to 2 classes.
```

[137]:
```python
cnn2.summary()
```

**Model: "sequential_22"**

| Layer (type) | Output Shape | Param # |
|---|---|---|
| conv2d_98 (Conv2D) | (None, 126, 126, 16) | 448 |
| max_pooling2d_97 (MaxPooling2D) | (None, 63, 63, 16) | 0 |
| conv2d_99 (Conv2D) | (None, 61, 61, 32) | 4,640 |
| max_pooling2d_98 (MaxPooling2D) | (None, 30, 30, 32) | 0 |
| conv2d_100 (Conv2D) | (None, 28, 28, 64) | 18,496 |
| max_pooling2d_99 (MaxPooling2D) | (None, 14, 14, 64) | 0 |
| conv2d_101 (Conv2D) | (None, 12, 12, 128) | 73,856 |
| max_pooling2d_100 (MaxPooling2D) | (None, 6, 6, 128) | 0 |
| conv2d_102 (Conv2D) | (None, 4, 4, 256) | 295,168 |
| max_pooling2d_101 (MaxPooling2D) | (None, 2, 2, 256) | 0 |
| flatten_21 (Flatten) | (None, 1024) | 0 |
| dense_53 (Dense) | (None, 128) | 131,200 |
| dense_54 (Dense) | (None, 1) | 129 |

**Total params:** 523,937 (2.00 MB)

**Trainable params:** 523,937 (2.00 MB)

Non-trainable params: 0 (0.00 B)

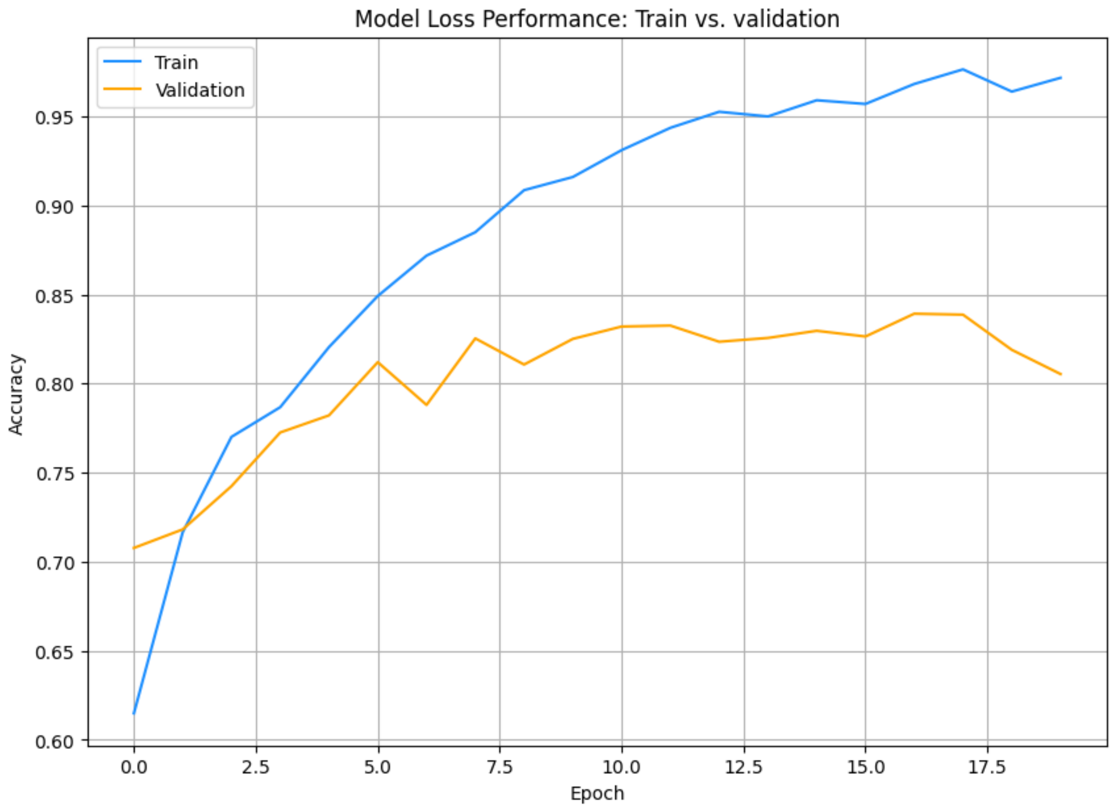
[138]:
```
cnn2.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics =␣
↪['accuracy'])

history = cnn2.fit(train_data, epochs=20, batch_size=64,␣
↪validation_data=val_data, verbose=1)
```

```
Epoch 1/20
235/235              11s 42ms/step -
accuracy: 0.5615 - loss: 2.1807 - val_accuracy: 0.7077 - val_loss: 0.5720
Epoch 2/20
235/235              5s 20ms/step -
accuracy: 0.6976 - loss: 0.5778 - val_accuracy: 0.7181 - val_loss: 0.5522
Epoch 3/20
235/235              5s 20ms/step -
accuracy: 0.7680 - loss: 0.4919 - val_accuracy: 0.7424 - val_loss: 0.5108
Epoch 4/20
235/235              5s 21ms/step -
accuracy: 0.7875 - loss: 0.4441 - val_accuracy: 0.7725 - val_loss: 0.4759
Epoch 5/20
235/235              5s 21ms/step -
accuracy: 0.8178 - loss: 0.3958 - val_accuracy: 0.7821 - val_loss: 0.4669
Epoch 6/20
235/235              5s 21ms/step -
accuracy: 0.8490 - loss: 0.3482 - val_accuracy: 0.8120 - val_loss: 0.4265
Epoch 7/20
235/235              5s 21ms/step -
accuracy: 0.8616 - loss: 0.3162 - val_accuracy: 0.7880 - val_loss: 0.5032
Epoch 8/20
235/235              5s 20ms/step -
accuracy: 0.8692 - loss: 0.3023 - val_accuracy: 0.8253 - val_loss: 0.4574
Epoch 9/20
235/235              5s 20ms/step -
accuracy: 0.9049 - loss: 0.2298 - val_accuracy: 0.8107 - val_loss: 0.5244
Epoch 10/20
235/235              5s 20ms/step -
accuracy: 0.9184 - loss: 0.1989 - val_accuracy: 0.8251 - val_loss: 0.5084
Epoch 11/20
235/235              5s 21ms/step -
accuracy: 0.9321 - loss: 0.1735 - val_accuracy: 0.8320 - val_loss: 0.5269
Epoch 12/20
235/235              5s 20ms/step -
accuracy: 0.9426 - loss: 0.1466 - val_accuracy: 0.8325 - val_loss: 0.5389
Epoch 13/20
235/235              5s 20ms/step -
accuracy: 0.9470 - loss: 0.1333 - val_accuracy: 0.8235 - val_loss: 0.5994
```

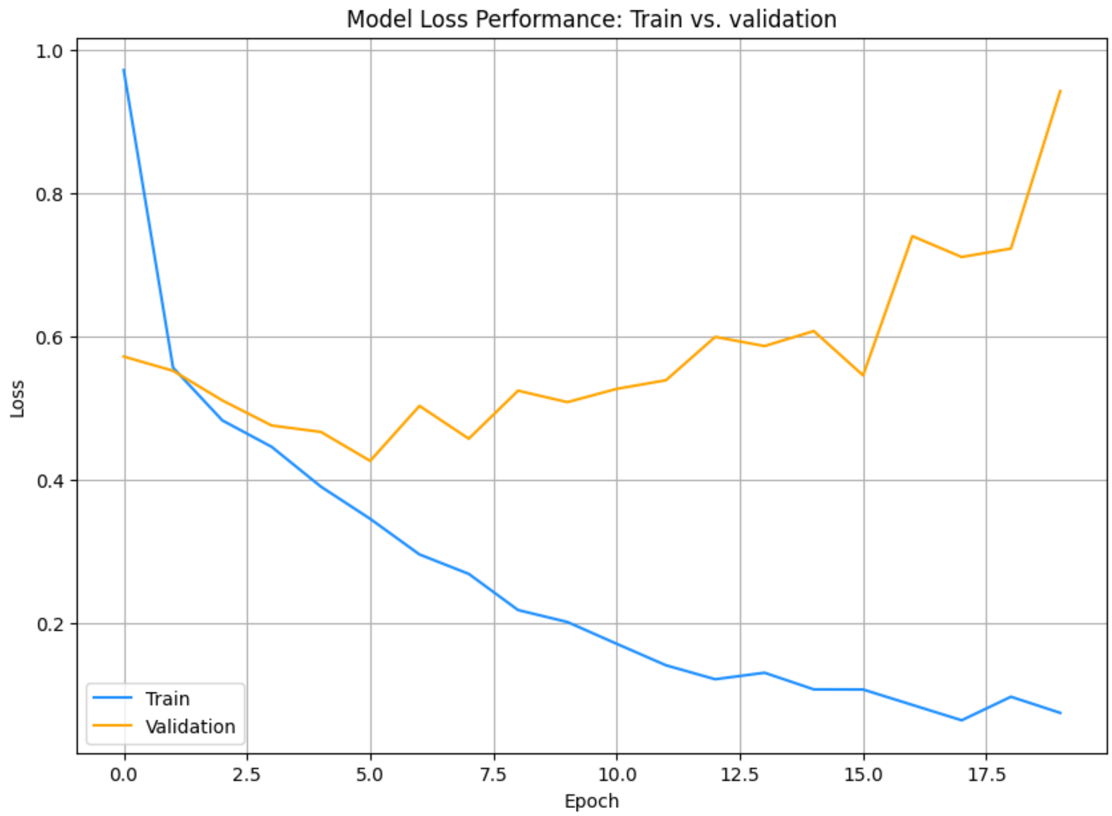
```
Epoch 14/20
235/235              5s 20ms/step -
accuracy: 0.9442 - loss: 0.1377 - val_accuracy: 0.8256 - val_loss: 0.5866
Epoch 15/20
235/235              5s 21ms/step -
accuracy: 0.9591 - loss: 0.1131 - val_accuracy: 0.8296 - val_loss: 0.6075
Epoch 16/20
235/235              5s 21ms/step -
accuracy: 0.9616 - loss: 0.0995 - val_accuracy: 0.8264 - val_loss: 0.5457
Epoch 17/20
235/235              5s 21ms/step -
accuracy: 0.9653 - loss: 0.0938 - val_accuracy: 0.8392 - val_loss: 0.7399
Epoch 18/20
235/235              5s 20ms/step -
accuracy: 0.9752 - loss: 0.0703 - val_accuracy: 0.8387 - val_loss: 0.7108
Epoch 19/20
235/235              5s 20ms/step -
accuracy: 0.9649 - loss: 0.0953 - val_accuracy: 0.8189 - val_loss: 0.7226
Epoch 20/20
235/235              5s 21ms/step -
accuracy: 0.9674 - loss: 0.0806 - val_accuracy: 0.8053 - val_loss: 0.9422
```

[139]:

```
J = history.history['accuracy']  # Loss data for Training
J_val = history.history['val_accuracy']

plt.figure(figsize=(10,7))

plt.title('Model Loss Performance: Train vs. validation')
plt.plot(J, color='DodgerBlue', label='Train')
plt.plot(J_val, color='orange', label='Validation')

plt.ylabel('Accuracy')
plt.xlabel('Epoch')

plt.legend()
plt.grid()
plt.show()
```




[140]:
```
J = history.history['loss']  # Loss data for Training
J_val = history.history['val_loss']

plt.figure(figsize=(10,7))

plt.title('Model Loss Performance: Train vs. validation')
plt.plot(J, color='DodgerBlue', label='Train')
plt.plot(J_val, color='orange', label='Validation')

plt.ylabel('Loss')
plt.xlabel('Epoch')

plt.legend()
plt.grid()
plt.show()
```




[141]:
```
loss, accuracy = cnn2.evaluate(test_data, verbose=1)

y_pred = cnn2.predict(test_data)

print("Accuracy: ", round(accuracy * 100, 2), "%")
print("Loss: ", round(loss, 2))
```

```
98/98              1s 11ms/step -
accuracy: 0.8028 - loss: 0.9424
98/98              1s 13ms/step
Accuracy:  80.45 %
Loss:  0.92
```

[142]:
```
cnn2.save('./cnn2.keras')
```

## 3.1 Conclusion

### 3.1.1 You will fill out information in this table:

| Model | Accuracy | Number of Parameters | Training Time | Inference Speed |
|---|---|---|---|---|
| Baseline VGG-5 | 74.04% | 683,329 | 0:30 | 02.3 |
| CNN1 | 88.34% | 1,847,489 | 3:15 | 0:2.4 |

| Model | Accuracy | Number of Parameters | Training Time | Inference Speed |
|---|---|---|---|---|
| CNN2 | 80.45% | 523,937 | 1:41 | 0:2.2 |

**Please add comments on what you tried and observed while working on the assignment.**

The first thing I realized is that CNNs of any note are computationally demanding. The first couple of architectures I put together pinned my CPU at 100% and took around 1 minute per epoch. I set out then to get Tensorflow to recognize my GPU. This created a lot of headach. After days of struggling, I realized that the core of the issue is that the version of CUDA and cuDNN I am on are newer then the Tensorflow version I originally was using had. After chaning to the latest version of Tensorflow, my GPU was automatically detected. I then ran into the issue that certain features used in the provided codes were depricated. I had to code these features out and replace them with up-to-date equivalents.

I was then able to focus on iterating my architectures in earnest. At first I tried to replicate VGG-16. This was not possible because we are limeted to using a 128x128 picture size and a 3x3 filter would decrease the matrix size too fast. So I then went on to capture the spirit of VGG-16 and started out with a small number of filters, and gradually increased the number until the final CNN layer before flattening. This did a good job but I was getting low to mid 80% on my testinging accuracy. I also tried making fewer layers, but multiple filters, but that provided no advantage. I added a second hidden layer but that only increased my success marginally. The true key was adding a dropout layer after flattening.

The dropout layer that I added was extrememly successful and allowed for my training and validation accuracy to increase in tandom together for longer numbers of epochs. Preventing over training was one of the hardest things about creating a good CNN architecture. A lot of the architectures I created would see divergence between the training and validation accuracy fairly quickly, around the 10 epoch mark. But the dropout allowed for the delay of the divergence for longer in a lot of my testing. It appears that the dropout layer allows the neural network to generalize the features a lot more and really helps prevent over training.

##

Remember to turn in both the notebook and the pdf version.