Convolutional Neural Network for Image Classification

```
import tensorflow as tf
from tensorflow import keras
import tensorflow_datasets as tf_data
```

Data Preprocessing

```
# Load in the data and view patch_camelyon dataset information
# Setting as_supervised to True allows us to read in the rows of the dataset as tuples
# Otherwise, each row would be a dictionary with the features
patch_cam, pc_info = tf_data.load("patch_camelyon", with_info=True, as_supervised=True
print(pc_info)

# Split the training, validation, and testing data
train_data = patch_cam['train']
valid_data = patch_cam['validation']
test_data = patch_cam['test']

# Pixel values are between 0 and 225, so we will normalize these values to be between
# This will allow for better performance of the neural network while it is learning ho
# classifying the image, as large values can slow this process
def normalize(input, label):
  newval = tf.cast(input, tf.float32)
  newval /= 225
  return newval, label

train_data = train_data.map(normalize)
valid_data = valid_data.map(normalize)
test_data = test_data.map(normalize)

# Shuffle training data
train_data = train_data.shuffle(1000)

# Set batch size
train_data = train_data.batch(64).prefetch(1)
valid_data = valid_data.batch(64).prefetch(1)
test_data = test_data.batch(64).prefetch(1)
```

Downloading and preparing dataset 7.48 GiB (download: 7.48 GiB, generated: Unknow

Dl Completed...: 100% 6/6 [10:27<00:00, 136.33s/ url]

Dl Size...: 100% 7654/7654 [10:27<00:00, 14.31 MiB/s]

Extraction completed...: 100% 6/6 [10:26<00:00, 160.18s/ file]

Dataset patch_camelyon downloaded and prepared to ~/tensorflow_datasets/patch_ca

```
tfds.core.DatasetInfo(
    name='patch_camelyon',
    full_name='patch_camelyon/2.0.0',
    description="""
    The PatchCamelyon benchmark is a new and challenging image classification
    dataset. It consists of 327.680 color images (96 x 96px) extracted from
    histopathologic scans of lymph node sections. Each image is annoted with a
    binary label indicating presence of metastatic tissue. PCam provides a new
    benchmark for machine learning models: bigger than CIFAR10, smaller than
    Imagenet, trainable on a single GPU.
    """,
    homepage='https://patchcamelyon.grand-challenge.org/',
    data_path='~/tensorflow_datasets/patch_camelyon/2.0.0',
    file_format=tfrecord,
    download_size=7.48 GiB,
    dataset_size=7.06 GiB,
    features=FeaturesDict({
        'id': Text(shape=(), dtype=tf.string),
        'image': Image(shape=(96, 96, 3), dtype=tf.uint8),
        'label': ClassLabel(shape=(), dtype=tf.int64, num_classes=2),
    }),
    supervised_keys=('image', 'label'),
    disable_shuffling=False,
    splits={
        'test': <SplitInfo num_examples=32768, num_shards=8>,
        'train': <SplitInfo num_examples=262144, num_shards=64>,
        'validation': <SplitInfo num_examples=32768, num_shards=8>,
    },
    citation="""@misc{b_s_veeling_j_linmans_j_winkens_t_cohen_2018_2546921,
      author       = {B. S. Veeling, J. Linmans, J. Winkens, T. Cohen, M. Wellin
      title        = {Rotation Equivariant CNNs for Digital Pathology},
      month        = sep,
```

Building the Model: Architecture 1

- This model will utilize the ReLu activation function for the hidden layers after performing the convolution
- The output layer will use the sigmoid activation function due to the binary classification of the outcome

```
# Build the model
from tensorflow.keras import layers, models, optimizers, regularizers

# Layers early in the model will learn fewer convolutional filters
# Layers deeper in the model will learn more
# Input images not too large, so 3x3 kernel size will be used
# Padding will be set to 'same' to preserve spatial dimensions of input volume size wh
# Pooling will be performed between layers of convolution to reduce size of images but
# A uniform distribution will be used to generate the weights for kernel initializatic
model1 = models.Sequential()
model1.add(layers.Conv2D(16, (3,3), padding='same', kernel_initializer='he_uniform', a
model1.add(layers.MaxPooling2D(pool_size=(2,2)))
model1.add(layers.Conv2D(32, (3,3), padding='same', kernel_initializer='he_uniform', a
model1.add(layers.MaxPooling2D(pool_size=(2,2)))
model1.add(layers.Conv2D(32, (3,3), padding='same', kernel_initializer='he_uniform', a
model1.add(layers.MaxPooling2D(pool_size=(2,2)))
model1.add(layers.Flatten())
model1.add(layers.Dense(64, activation='relu'))
model1.add(layers.Dense(1, activation='sigmoid'))

# Summarize the model
model1.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 96, 96, 16)        448

 max_pooling2d (MaxPooling2D  (None, 48, 48, 16)       0
 )

 conv2d_1 (Conv2D)           (None, 48, 48, 32)        4640

 max_pooling2d_1 (MaxPooling  (None, 24, 24, 32)       0
 2D)

 conv2d_2 (Conv2D)           (None, 24, 24, 32)        9248

 max_pooling2d_2 (MaxPooling  (None, 12, 12, 32)       0
 2D)

 flatten (Flatten)           (None, 4608)              0
```

```
 dense (Dense)               (None, 64)                294976

 dense_1 (Dense)             (None, 1)                 65

=================================================================
Total params: 309,377
Trainable params: 309,377
Non-trainable params: 0
_________________________________________________________________
```

Compiling and Fitting Our First Model

```
# Stochastic gradient descent using the Adam (Adaptive Moment Estimation) algorithm wi
# Default learning rate of 0.001 for Adam will be used for this architecture
# Binary cross-entropy loss function will be used due to the binary classification of
model1.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Early stopping will be implemented, using validation loss as the metric
# If validation loss does not improve after 10 epochs, stop training
# stopping = keras.callbacks.EarlyStopping(monitor='val_loss', patience=10)
# can add "callbacks=[stopping]" after epochs to implement stopping

# Fitting our model, seeing one line per epoch
history = model1.fit(train_data, steps_per_epoch=10, epochs=100, validation_data=valic
```

```
Epoch 72/100
10/10 - 13s - loss: 0.4664 - accuracy: 0.7703 - val_loss: 0.4915 - val_accuracy:
Epoch 73/100
10/10 - 12s - loss: 0.4731 - accuracy: 0.7906 - val_loss: 0.4580 - val_accuracy:
Epoch 74/100
10/10 - 13s - loss: 0.4574 - accuracy: 0.7937 - val_loss: 0.4839 - val_accuracy:
Epoch 75/100
10/10 - 12s - loss: 0.4729 - accuracy: 0.7812 - val_loss: 0.4519 - val_accuracy:
Epoch 76/100
10/10 - 12s - loss: 0.4595 - accuracy: 0.7937 - val_loss: 0.4621 - val_accuracy:
Epoch 77/100
10/10 - 12s - loss: 0.4916 - accuracy: 0.7750 - val_loss: 0.4451 - val_accuracy:
Epoch 78/100
10/10 - 13s - loss: 0.4617 - accuracy: 0.7984 - val_loss: 0.4580 - val_accuracy:
Epoch 79/100
10/10 - 12s - loss: 0.4425 - accuracy: 0.7969 - val_loss: 0.4432 - val_accuracy:
Epoch 80/100
10/10 - 12s - loss: 0.4757 - accuracy: 0.7891 - val_loss: 0.4581 - val_accuracy:
Epoch 81/100
10/10 - 12s - loss: 0.4268 - accuracy: 0.8000 - val_loss: 0.4623 - val_accuracy:
Epoch 82/100
10/10 - 13s - loss: 0.4622 - accuracy: 0.7781 - val_loss: 0.4396 - val_accuracy:
Epoch 83/100
10/10 - 12s - loss: 0.4187 - accuracy: 0.8313 - val_loss: 0.4616 - val_accuracy:
Epoch 84/100
10/10 - 12s - loss: 0.4410 - accuracy: 0.7859 - val_loss: 0.4492 - val_accuracy:
Epoch 85/100
10/10 - 12s - loss: 0.4602 - accuracy: 0.7828 - val_loss: 0.4601 - val_accuracy:
```

```
Epoch 86/100
10/10 - 12s - loss: 0.4375 - accuracy: 0.8062 - val_loss: 0.4551 - val_accuracy:
Epoch 87/100
10/10 - 12s - loss: 0.4277 - accuracy: 0.8000 - val_loss: 0.4952 - val_accuracy:
Epoch 88/100
10/10 - 12s - loss: 0.4662 - accuracy: 0.7797 - val_loss: 0.4694 - val_accuracy:
Epoch 89/100
10/10 - 12s - loss: 0.4480 - accuracy: 0.8031 - val_loss: 0.4399 - val_accuracy:
Epoch 90/100
10/10 - 12s - loss: 0.4522 - accuracy: 0.7953 - val_loss: 0.4407 - val_accuracy:
Epoch 91/100
10/10 - 13s - loss: 0.4178 - accuracy: 0.8172 - val_loss: 0.4810 - val_accuracy:
Epoch 92/100
10/10 - 13s - loss: 0.4111 - accuracy: 0.8250 - val_loss: 0.5148 - val_accuracy:
Epoch 93/100
10/10 - 12s - loss: 0.4432 - accuracy: 0.8000 - val_loss: 0.4989 - val_accuracy:
Epoch 94/100
10/10 - 12s - loss: 0.4460 - accuracy: 0.7984 - val_loss: 0.4483 - val_accuracy:
Epoch 95/100
10/10 - 12s - loss: 0.4626 - accuracy: 0.7828 - val_loss: 0.4371 - val_accuracy:
Epoch 96/100
10/10 - 12s - loss: 0.4518 - accuracy: 0.7922 - val_loss: 0.4682 - val_accuracy:
Epoch 97/100
10/10 - 12s - loss: 0.4401 - accuracy: 0.7922 - val_loss: 0.4580 - val_accuracy:
Epoch 98/100

10/10 - 12s - loss: 0.3991 - accuracy: 0.8219 - val_loss: 0.5050 - val_accuracy:
Epoch 99/100
10/10 - 12s - loss: 0.4564 - accuracy: 0.8062 - val_loss: 0.4658 - val_accuracy:
Epoch 100/100
10/10 - 13s - loss: 0.4414 - accuracy: 0.7953 - val_loss: 0.4510 - val_accuracy:
```

Plotting the Accuracy and Loss Curve

```
import matplotlib.pyplot as plt
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)

plt.plot(acc, label='train_accuracy')
plt.plot(val_acc, label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')
plt.show()

plt.plot(loss, label='train_loss')
plt.plot(val_loss, label = 'val_loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
```

```
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```

Evaluating the Model

- Best accuracy for validation set occurs roughly around 70-80 epochs
- Model may be overfitting slightly after 80 epochs as we can see the training and validation accuracy curves begin to diverge slightly, as well as loss begin to slowly increase for the validation set
- Best validation accuracy is 0.7910 at epoch 70

Building the Model: Architecture 2

- This model will add more convolutional hidden layers and filters, increasing the complexity of the overall model

- The learning rate will be lowered from 0.001 to 0.0001 to change the weights less drastically

```
model2 = models.Sequential()
model2.add(layers.Conv2D(128, (3,3), padding='same', kernel_initializer='he_uniform',
model2.add(layers.MaxPooling2D(pool_size=(2,2)))
model2.add(layers.Conv2D(256, (3,3), padding='same', kernel_initializer='he_uniform',
model2.add(layers.MaxPooling2D(pool_size=(2,2)))
model2.add(layers.Conv2D(512, (3,3), padding='same', kernel_initializer='he_uniform',
model2.add(layers.MaxPooling2D(pool_size=(2,2)))
model2.add(layers.Conv2D(1024, (3,3), padding='same', kernel_initializer='he_uniform',
model2.add(layers.MaxPooling2D(pool_size=(2,2)))
model2.add(layers.Flatten())
model2.add(layers.Dense(1024, activation='relu'))
model2.add(layers.Dense(128, activation='relu'))
model2.add(layers.Dense(1, activation='sigmoid'))

model2.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_13 (Conv2D)          (None, 96, 96, 128)       3584

 max_pooling2d_13 (MaxPoolin  (None, 48, 48, 128)      0
 g2D)

 conv2d_14 (Conv2D)          (None, 48, 48, 256)       295168

 max_pooling2d_14 (MaxPoolin  (None, 24, 24, 256)      0
 g2D)

 conv2d_15 (Conv2D)          (None, 24, 24, 512)       1180160

 max_pooling2d_15 (MaxPoolin  (None, 12, 12, 512)      0
 g2D)

 conv2d_16 (Conv2D)          (None, 12, 12, 1024)      4719616

 max_pooling2d_16 (MaxPoolin  (None, 6, 6, 1024)       0
 g2D)

 flatten_3 (Flatten)         (None, 36864)             0

 dense_8 (Dense)             (None, 1024)              37749760

 dense_9 (Dense)             (None, 128)               131200

 dense_10 (Dense)            (None, 1)                 129

=================================================================
Total params: 44,079,617
Trainable params: 44,079,617
```

```
Non-trainable params: 0
```

Compiling and Fitting Our Second Model

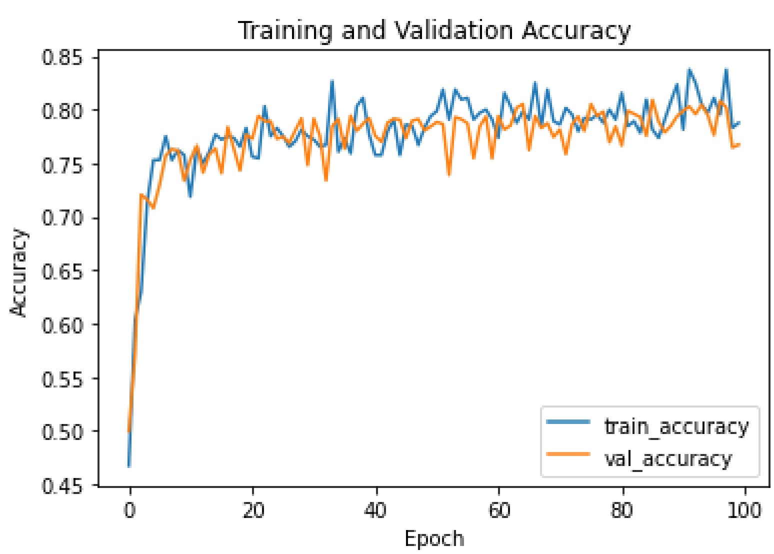
```
model2.compile(optimizer=optimizers.Adam(0.0001), loss='binary_crossentropy', metrics=

history = model2.fit(train_data, steps_per_epoch=10, epochs=100, validation_data=valic
```

```
Epoch 72/100
10/10 - 22s - loss: 0.4543 - accuracy: 0.8016 - val_loss: 0.4952 - val_accuracy:
Epoch 73/100
10/10 - 22s - loss: 0.4642 - accuracy: 0.7953 - val_loss: 0.4544 - val_accuracy:
Epoch 74/100
10/10 - 22s - loss: 0.4610 - accuracy: 0.7797 - val_loss: 0.4246 - val_accuracy:
Epoch 75/100
10/10 - 22s - loss: 0.4839 - accuracy: 0.7922 - val_loss: 0.4540 - val_accuracy:
Epoch 76/100
10/10 - 22s - loss: 0.4907 - accuracy: 0.7906 - val_loss: 0.4284 - val_accuracy:
Epoch 77/100
10/10 - 22s - loss: 0.4769 - accuracy: 0.7953 - val_loss: 0.4296 - val_accuracy:
Epoch 78/100
10/10 - 22s - loss: 0.4805 - accuracy: 0.7875 - val_loss: 0.4252 - val_accuracy:
Epoch 79/100
10/10 - 22s - loss: 0.4572 - accuracy: 0.8000 - val_loss: 0.4868 - val_accuracy:

Epoch 80/100
10/10 - 22s - loss: 0.4713 - accuracy: 0.7906 - val_loss: 0.4548 - val_accuracy:
Epoch 81/100
10/10 - 22s - loss: 0.4331 - accuracy: 0.8156 - val_loss: 0.5278 - val_accuracy:
Epoch 82/100
10/10 - 22s - loss: 0.4784 - accuracy: 0.7844 - val_loss: 0.4340 - val_accuracy:
Epoch 83/100
10/10 - 22s - loss: 0.4505 - accuracy: 0.7891 - val_loss: 0.4556 - val_accuracy:
Epoch 84/100
10/10 - 22s - loss: 0.4637 - accuracy: 0.7781 - val_loss: 0.4334 - val_accuracy:
Epoch 85/100
10/10 - 22s - loss: 0.4156 - accuracy: 0.8094 - val_loss: 0.4850 - val_accuracy:
Epoch 86/100
10/10 - 22s - loss: 0.4763 - accuracy: 0.7812 - val_loss: 0.4271 - val_accuracy:
Epoch 87/100
10/10 - 22s - loss: 0.4687 - accuracy: 0.7734 - val_loss: 0.4615 - val_accuracy:
Epoch 88/100
10/10 - 24s - loss: 0.4475 - accuracy: 0.7906 - val_loss: 0.4688 - val_accuracy:
Epoch 89/100
10/10 - 22s - loss: 0.4386 - accuracy: 0.8078 - val_loss: 0.4827 - val_accuracy:
Epoch 90/100
10/10 - 22s - loss: 0.4016 - accuracy: 0.8234 - val_loss: 0.4430 - val_accuracy:
Epoch 91/100
10/10 - 22s - loss: 0.4508 - accuracy: 0.7812 - val_loss: 0.4301 - val_accuracy:
Epoch 92/100
10/10 - 42s - loss: 0.3852 - accuracy: 0.8375 - val_loss: 0.4249 - val_accuracy:
Epoch 93/100
10/10 - 42s - loss: 0.4196 - accuracy: 0.8250 - val_loss: 0.4435 - val_accuracy:
Epoch 94/100
10/10 - 23s - loss: 0.4413 - accuracy: 0.8047 - val_loss: 0.4198 - val_accuracy:
Epoch 95/100
```

```
10/10 - 22s - loss: 0.4567 - accuracy: 0.7969 - val_loss: 0.4866 - val_accuracy:
Epoch 96/100
10/10 - 22s - loss: 0.4399 - accuracy: 0.8109 - val_loss: 0.5020 - val_accuracy:
Epoch 97/100
10/10 - 42s - loss: 0.4568 - accuracy: 0.7953 - val_loss: 0.4205 - val_accuracy:
Epoch 98/100
10/10 - 22s - loss: 0.4118 - accuracy: 0.8375 - val_loss: 0.4331 - val_accuracy:
Epoch 99/100
10/10 - 22s - loss: 0.4636 - accuracy: 0.7828 - val_loss: 0.4962 - val_accuracy:
Epoch 100/100
10/10 - 22s - loss: 0.4826 - accuracy: 0.7875 - val_loss: 0.4909 - val_accuracy:
```

Plotting the Accuracy and Loss Curve

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)

# Accuracy
plt.plot(acc, label='train_accuracy')
plt.plot(val_acc, label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')
plt.show()
```




```
# Loss
plt.plot(loss, label='train_loss')
plt.plot(val_loss, label = 'val_loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
```

```
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```

Evaluating the Model

- Model has similar performance to first model and there does not appear to be any significant over/under fitting
- Best accuracy for validation set is 0.8091 which occurs at epoch 87

Building the Model: Architecture 3

- This model will be more complex overall (more layers/nodes), however it will utilize dropout training on the hidden layers
- This will be done as an attempt to prevent the model from overfitting the training data

```
model3 = models.Sequential()
model3.add(layers.Conv2D(128, (3,3), padding='same', kernel_initializer='he_uniform',
model3.add(layers.MaxPooling2D(pool_size=(2,2)))
model3.add(layers.Dropout(0.1))
model3.add(layers.Conv2D(128, (3,3), padding='same', kernel_initializer='he_uniform',
model3.add(layers.MaxPooling2D(pool_size=(2,2)))
model3.add(layers.Dropout(0.1))
model3.add(layers.Conv2D(256, (3,3), padding='same', kernel_initializer='he_uniform',
model3.add(layers.MaxPooling2D(pool_size=(2,2)))
model3.add(layers.Dropout(0.2))
model3.add(layers.Conv2D(256, (3,3), padding='same', kernel_initializer='he_uniform',
model3.add(layers.MaxPooling2D(pool_size=(2,2)))
model3.add(layers.Dropout(0.2))
model3.add(layers.Conv2D(512, (3,3), padding='same', kernel_initializer='he_uniform',
model3.add(layers.MaxPooling2D(pool_size=(2,2)))
model3.add(layers.Dropout(0.2))
```

```
model3.add(layers.Conv2D(1024, (3,3), padding='same', kernel_initializer='he_uniform',
model3.add(layers.MaxPooling2D(pool_size=(2,2)))
model3.add(layers.Dropout(0.3))
model3.add(layers.Flatten())
model3.add(layers.Dense(1024, activation='relu'))
model3.add(layers.Dropout(0.2))
model3.add(layers.Dense(128, activation='relu'))
model3.add(layers.Dense(1, activation='sigmoid'))

# Summarize the model
model3.summary()
```

```
=================================================================
 conv2d_7 (Conv2D)           (None, 96, 96, 128)       3584

 max_pooling2d_7 (MaxPooling  (None, 48, 48, 128)      0
 2D)

 dropout (Dropout)           (None, 48, 48, 128)       0

 conv2d_8 (Conv2D)           (None, 48, 48, 128)       147584

 max_pooling2d_8 (MaxPooling  (None, 24, 24, 128)      0
 2D)

 dropout_1 (Dropout)         (None, 24, 24, 128)       0

 conv2d_9 (Conv2D)           (None, 24, 24, 256)       295168

 max_pooling2d_9 (MaxPooling  (None, 12, 12, 256)      0
 2D)

 dropout_2 (Dropout)         (None, 12, 12, 256)       0

 conv2d_10 (Conv2D)          (None, 12, 12, 256)       590080

 max_pooling2d_10 (MaxPoolin  (None, 6, 6, 256)        0
 g2D)

 dropout_3 (Dropout)         (None, 6, 6, 256)         0

 conv2d_11 (Conv2D)          (None, 6, 6, 512)         1180160

 max_pooling2d_11 (MaxPoolin  (None, 3, 3, 512)        0
 g2D)

 dropout_4 (Dropout)         (None, 3, 3, 512)         0

 conv2d_12 (Conv2D)          (None, 3, 3, 1024)        4719616

 max_pooling2d_12 (MaxPoolin  (None, 1, 1, 1024)       0
 g2D)

 dropout_5 (Dropout)         (None, 1, 1, 1024)        0

 flatten_2 (Flatten)         (None, 1024)              0
```

```
 dense_5 (Dense)             (None, 1024)              1049600

 dropout_6 (Dropout)         (None, 1024)              0

 dense_6 (Dense)             (None, 128)               131200

 dense_7 (Dense)             (None, 1)                 129

=================================================================
Total params: 8,117,121
Trainable params: 8,117,121
Non-trainable params: 0
_________________________________________________________________
```

Compiling and Fitting our Third Model

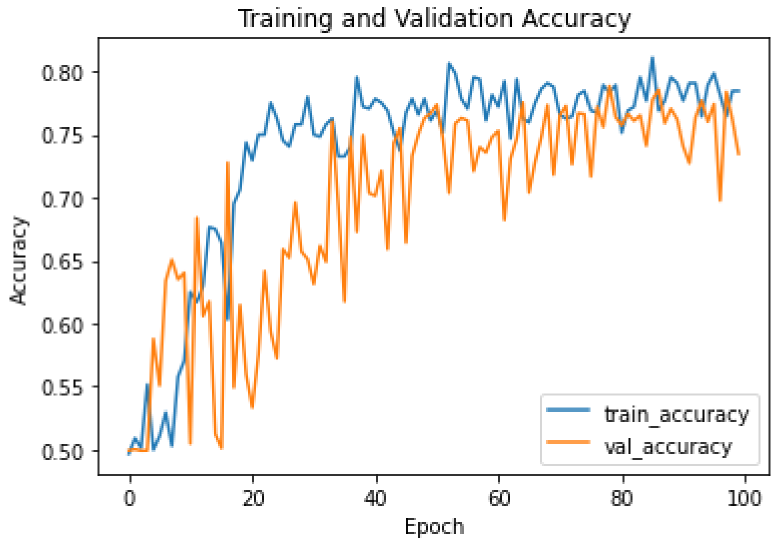
```
model3.compile(optimizer=optimizers.Adam(0.0001), loss='binary_crossentropy', metrics=

history = model3.fit(train_data, steps_per_epoch=10, epochs=100, validation_data=valic
```

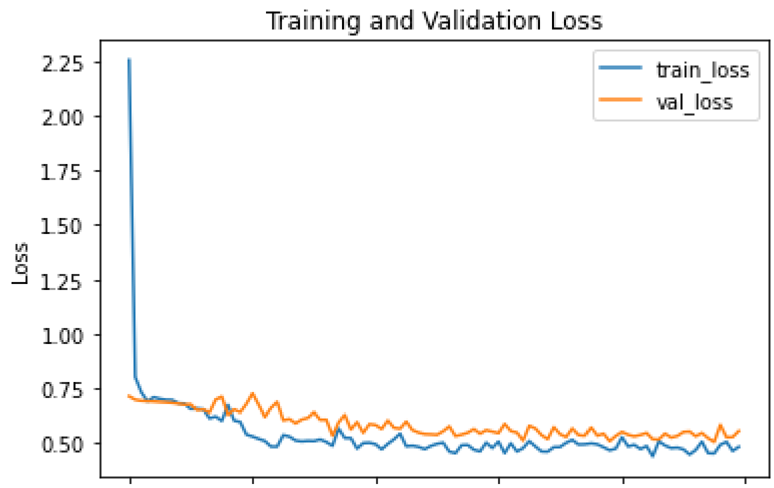
```
10/10 - 15s - loss: 0.5026 - accuracy: 0.7625 - val_loss: 0.5295 - val_accuracy:
Epoch 73/100
10/10 - 15s - loss: 0.5162 - accuracy: 0.7641 - val_loss: 0.5684 - val_accuracy:
Epoch 74/100
10/10 - 15s - loss: 0.4950 - accuracy: 0.7812 - val_loss: 0.5388 - val_accuracy:
Epoch 75/100
10/10 - 15s - loss: 0.4959 - accuracy: 0.7844 - val_loss: 0.5350 - val_accuracy:
Epoch 76/100
10/10 - 15s - loss: 0.5002 - accuracy: 0.7688 - val_loss: 0.5729 - val_accuracy:
Epoch 77/100
10/10 - 15s - loss: 0.4958 - accuracy: 0.7672 - val_loss: 0.5355 - val_accuracy:
Epoch 78/100
10/10 - 15s - loss: 0.4825 - accuracy: 0.7891 - val_loss: 0.5447 - val_accuracy:
Epoch 79/100
10/10 - 15s - loss: 0.4684 - accuracy: 0.7828 - val_loss: 0.5103 - val_accuracy:
Epoch 80/100
10/10 - 15s - loss: 0.4763 - accuracy: 0.7891 - val_loss: 0.5344 - val_accuracy:
Epoch 81/100
10/10 - 15s - loss: 0.5286 - accuracy: 0.7516 - val_loss: 0.5529 - val_accuracy:
Epoch 82/100
10/10 - 15s - loss: 0.4860 - accuracy: 0.7688 - val_loss: 0.5397 - val_accuracy:
Epoch 83/100
10/10 - 15s - loss: 0.4937 - accuracy: 0.7719 - val_loss: 0.5325 - val_accuracy:
Epoch 84/100
10/10 - 15s - loss: 0.4742 - accuracy: 0.7953 - val_loss: 0.5388 - val_accuracy:
Epoch 85/100
10/10 - 15s - loss: 0.4878 - accuracy: 0.7766 - val_loss: 0.5477 - val_accuracy:
Epoch 86/100
10/10 - 15s - loss: 0.4408 - accuracy: 0.8109 - val_loss: 0.5192 - val_accuracy:
Epoch 87/100
10/10 - 15s - loss: 0.5097 - accuracy: 0.7688 - val_loss: 0.5180 - val_accuracy:
Epoch 88/100
```

```
10/10 - 15s - loss: 0.4904 - accuracy: 0.7766 - val_loss: 0.5450 - val_accuracy:
Epoch 89/100
10/10 - 15s - loss: 0.4785 - accuracy: 0.7953 - val_loss: 0.5267 - val_accuracy:
Epoch 90/100
10/10 - 15s - loss: 0.4807 - accuracy: 0.7906 - val_loss: 0.5341 - val_accuracy:
Epoch 91/100
10/10 - 15s - loss: 0.4727 - accuracy: 0.7766 - val_loss: 0.5524 - val_accuracy:
Epoch 92/100
10/10 - 15s - loss: 0.4491 - accuracy: 0.7906 - val_loss: 0.5545 - val_accuracy:
Epoch 93/100
10/10 - 15s - loss: 0.4701 - accuracy: 0.7906 - val_loss: 0.5327 - val_accuracy:
Epoch 94/100
10/10 - 15s - loss: 0.5083 - accuracy: 0.7641 - val_loss: 0.5478 - val_accuracy:
Epoch 95/100
10/10 - 15s - loss: 0.4557 - accuracy: 0.7891 - val_loss: 0.5241 - val_accuracy:
Epoch 96/100
10/10 - 15s - loss: 0.4540 - accuracy: 0.7984 - val_loss: 0.5066 - val_accuracy:
Epoch 97/100
10/10 - 15s - loss: 0.4939 - accuracy: 0.7812 - val_loss: 0.5852 - val_accuracy:
Epoch 98/100
10/10 - 15s - loss: 0.5069 - accuracy: 0.7641 - val_loss: 0.5280 - val_accuracy:
Epoch 99/100
10/10 - 15s - loss: 0.4654 - accuracy: 0.7844 - val_loss: 0.5290 - val_accuracy:
Epoch 100/100
10/10 - 15s - loss: 0.4844 - accuracy: 0.7844 - val_loss: 0.5570 - val_accuracy:
```

Plotting the Accuracy and Loss Curve

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)

plt.plot(acc, label='train_accuracy')
plt.plot(val_acc, label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')
plt.show()

plt.plot(loss, label='train_loss')
plt.plot(val_loss, label = 'val_loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```





Evaluating the Model

- Dropout training did seem to help overfitting at later epochs, as accuracy and loss do not diverge drastically
- Model did not offer an improvement on model 2 in terms of validation accuracy

Evaluating Model With Highest Validation Accuracy on Test Data

- Overall, the second model architecture had the best accuracy to the validation set between 90-100 epochs

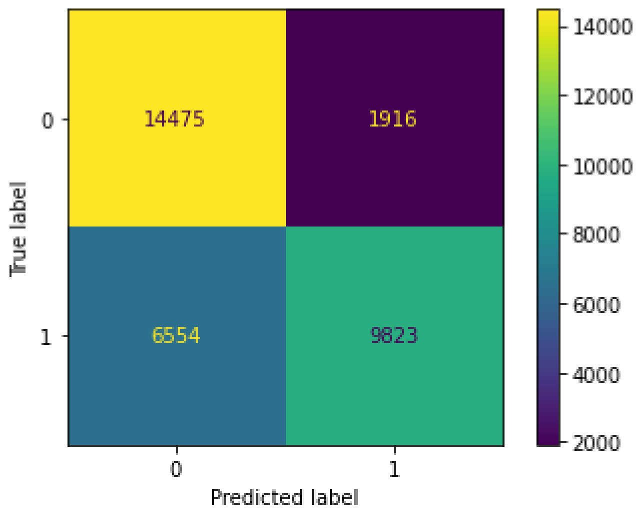
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, roc_curve, auc,
import numpy as np

# Model Accuracy on Test Data
test_loss, test_acc = model2.evaluate(test_data, verbose=2)
```

```
512/512 - 21s - loss: 0.5339 - accuracy: 0.7415 - 21s/epoch - 42ms/step
```

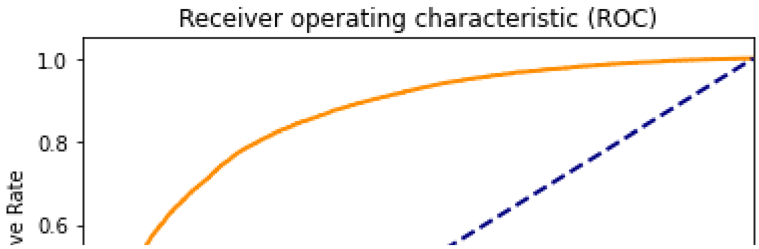
```
# Plotting the Confusion Matrix
```

```
y_prob = model2.predict(test_data)
y_pred = (y_prob >= .5).astype(int)

y = np.concatenate([y for x, y in test_data], axis=0)

conf = confusion_matrix(y, y_pred)
conf1 = ConfusionMatrixDisplay(conf).plot()
print(conf1)
```

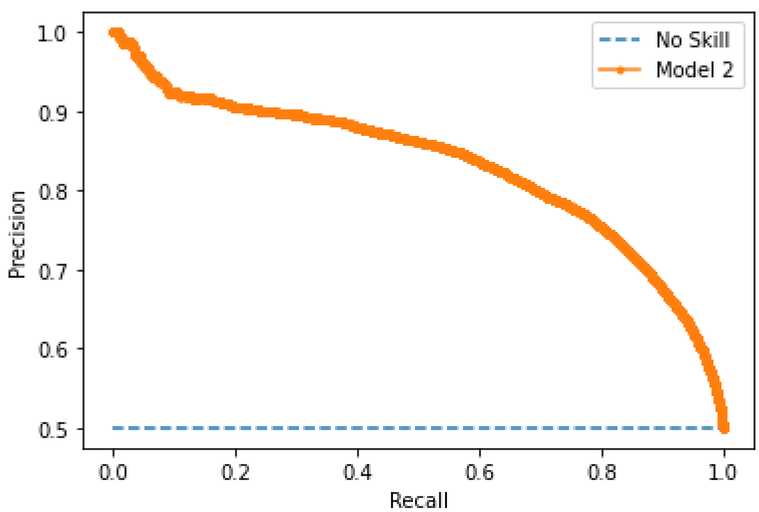
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay object at 0x7f8dfa
```




```
# Plotting ROC Curve
fpr, tpr, thresholds = roc_curve(y, y_prob)
roc_auc = auc(fpr, tpr)

plt.figure()
lw = 2
plt.plot(fpr, tpr, color='darkorange',
 lw=lw, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic (ROC)')
plt.legend(loc="lower right")
plt.show()
```




```
# Plotting Precision Recall Curve
lr_precision, lr_recall, lr_threshholds = precision_recall_curve(y, y_prob)

no_skill = len(y[y==1]) / len(y)
plt.plot([0, 1], [no_skill, no_skill], linestyle='--', label='No Skill')
plt.plot(lr_recall, lr_precision, marker='.', label='Model 2')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.legend()
plt.show()

# Calculating the F1-Score
f1 = f1_score(y, y_pred)
print(f1)
```




```
0.6987480438184664
```

Interpretation of Results

Model Accuracy

- The model had approximately 74% accuracy on the test data.

Confusion Matrix

- There were 14475 true positives and 9823 true negatives predicted by the model. There were 1916 false positives and 6554 false negatives predicted by the model.

ROC Curve

- ROC curve is an improvement over model with no ability to discriminate. It has an area of about 0.84, so it is able to predict more true positives/negatives than false positives/negatives.

Precision-Recall Curve

- Precision-recall curve is an improvement over model with no ability to discriminate.

F1-Score

- F1-score is approximately 0.7.