In [1]:

```
import numpy as np
import tensorflow as tf
from keras.preprocessing.image import ImageDataGenerator
tf.__version__
```

```
Using TensorFlow backend.
```

Out[1]:

```
'2.2.0'
```

In [2]:

```
# Image augmentation to avoid over training of the model and feature scaling
train_datagen = ImageDataGenerator(rescale = 1./255, shear_range = 0.2, zoom_range = 0.
2, horizontal_flip = True)
training_set = train_datagen.flow_from_directory('New Plant Diseases Dataset(Augmente
d)/New Plant Diseases Dataset(Augmented)/train', target_size = (224,224), batch_size =
128, class_mode = 'categorical')
```

```
Found 68653 images belonging to 37 classes.
```

In [3]:

```
test_datagen = ImageDataGenerator(rescale = 1./255)
test_set = test_datagen.flow_from_directory('New Plant Diseases Dataset(Augmented)/New
 Plant Diseases Dataset(Augmented)/valid', target_size = (224,224), batch_size = 85, cl
ass_mode = 'categorical')
```

```
Found 17162 images belonging to 37 classes.
```

In [4]:

```
train_num = training_set.samples
test_num = test_set.samples
print(train_num)
print(test_num)
```

```
68653
17162
```

In [5]:

```
cnn = tf.keras.models.Sequential()
cnn.add(tf.keras.layers.Conv2D(filters = 96, kernel_size = 11, strides = (4,4), padding
='valid', activation = 'relu', input_shape = [224,224,3]))
cnn.add(tf.keras.layers.MaxPool2D(pool_size = 2, strides = 2, padding='valid'))
cnn.add(tf.keras.layers.Conv2D(filters = 256, kernel_size = 11, strides = (1,1), paddin
g='valid', activation = 'relu'))
cnn.add(tf.keras.layers.MaxPool2D(pool_size = 2, strides = 2, padding='valid'))
cnn.add(tf.keras.layers.Conv2D(filters = 384, kernel_size = 3, strides = (1,1), padding
='valid', activation = 'relu'))
cnn.add(tf.keras.layers.Conv2D(filters = 384, kernel_size = 3, strides = (1,1), padding
='valid', activation = 'relu'))
cnn.add(tf.keras.layers.Conv2D(filters = 256, kernel_size = 3, strides = (1,1), padding
='valid', activation = 'relu'))
cnn.add(tf.keras.layers.MaxPool2D(pool_size = 2, strides = 2, padding='valid'))


cnn.add(tf.keras.layers.Flatten())
```

In [6]:

```
#input layer and first hidden layer
cnn.add(tf.keras.layers.Dense(units = 4096, activation = 'relu'))
# second and third hidden layer
cnn.add(tf.keras.layers.Dense(units = 4096, activation = 'relu'))
cnn.add(tf.keras.layers.Dense(units = 1000, activation = 'relu'))
cnn.add(tf.keras.layers.Dense(units = 37, activation = 'softmax'))
```

In [7]:

```
cnn.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d (Conv2D)              (None, 54, 54, 96)        34944     
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 27, 27, 96)        0         
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 17, 17, 256)       2973952   
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 8, 8, 256)         0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 6, 6, 384)         885120    
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 4, 4, 384)         1327488   
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 2, 2, 256)         884992    
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 1, 1, 256)         0         
_________________________________________________________________
flatten (Flatten)            (None, 256)               0         
_________________________________________________________________
dense (Dense)                (None, 4096)              1052672   
_________________________________________________________________
dense_1 (Dense)              (None, 4096)              16781312  
_________________________________________________________________
dense_2 (Dense)              (None, 1000)              4097000   
_________________________________________________________________
dense_3 (Dense)              (None, 37)                37037     
=================================================================
Total params: 28,074,517
Trainable params: 28,074,517
Non-trainable params: 0
_________________________________________________________________
```

In [8]:

```
cnn.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accurac
y'])
```

In [9]:

```
checkpoint_filepath = '/tmp/checkpoint'
model_checkpoint_callback = tf.keras.callbacks.ModelCheckpoint(
    filepath=checkpoint_filepath,
    save_weights_only=True,
    monitor='val_acc',
    mode='max',
    save_best_only=True)
```

In [11]:

```
cnn.fit(x = training_set, validation_data = test_set, epochs = 25, steps_per_epoch = tr
ain_num // 128, validation_steps = test_num // 128, callbacks=[model_checkpoint_callbac
k])
```

```
Epoch 1/25
536/536 [==============================] - ETA: 0s - loss: 1.8749 - accura
cy: 0.4247WARNING:tensorflow:Can save best model only with val_acc availab
le, skipping.
536/536 [==============================] - 3653s 7s/step - loss: 1.8749 -
accuracy: 0.4247 - val_loss: 1.4192 - val_accuracy: 0.5443
Epoch 2/25
536/536 [==============================] - ETA: 0s - loss: 1.4068 - accura
cy: 0.5596WARNING:tensorflow:Can save best model only with val_acc availab
le, skipping.
536/536 [==============================] - 2224s 4s/step - loss: 1.4068 -
accuracy: 0.5596 - val_loss: 1.1777 - val_accuracy: 0.6197
Epoch 3/25
536/536 [==============================] - ETA: 0s - loss: 1.1336 - accura
cy: 0.6405WARNING:tensorflow:Can save best model only with val_acc availab
le, skipping.
536/536 [==============================] - 1038s 2s/step - loss: 1.1336 -
accuracy: 0.6405 - val_loss: 0.9601 - val_accuracy: 0.6954
Epoch 4/25
536/536 [==============================] - ETA: 0s - loss: 0.9401 - accura
cy: 0.6983WARNING:tensorflow:Can save best model only with val_acc availab
le, skipping.
536/536 [==============================] - 3196s 6s/step - loss: 0.9401 -
accuracy: 0.6983 - val_loss: 0.7954 - val_accuracy: 0.7397
Epoch 5/25
536/536 [==============================] - ETA: 0s - loss: 0.7999 - accura
cy: 0.7409WARNING:tensorflow:Can save best model only with val_acc availab
le, skipping.
536/536 [==============================] - 2178s 4s/step - loss: 0.7999 -
accuracy: 0.7409 - val_loss: 0.6870 - val_accuracy: 0.7753
Epoch 6/25
536/536 [==============================] - ETA: 0s - loss: 0.7087 - accura
cy: 0.7706WARNING:tensorflow:Can save best model only with val_acc availab
le, skipping.
536/536 [==============================] - 969s 2s/step - loss: 0.7087 - a
ccuracy: 0.7706 - val_loss: 0.6650 - val_accuracy: 0.7802
Epoch 7/25
536/536 [==============================] - ETA: 0s - loss: 0.6501 - accura
cy: 0.7880WARNING:tensorflow:Can save best model only with val_acc availab
le, skipping.
536/536 [==============================] - 1970s 4s/step - loss: 0.6501 -
accuracy: 0.7880 - val_loss: 0.5813 - val_accuracy: 0.8105
Epoch 8/25
536/536 [==============================] - ETA: 0s - loss: 0.5783 - accura
cy: 0.8125WARNING:tensorflow:Can save best model only with val_acc availab
le, skipping.
536/536 [==============================] - 1714s 3s/step - loss: 0.5783 -
accuracy: 0.8125 - val_loss: 0.5991 - val_accuracy: 0.8101
Epoch 9/25
536/536 [==============================] - ETA: 0s - loss: 0.5505 - accura
cy: 0.8223WARNING:tensorflow:Can save best model only with val_acc availab
le, skipping.
536/536 [==============================] - 990s 2s/step - loss: 0.5505 - a
ccuracy: 0.8223 - val_loss: 0.5214 - val_accuracy: 0.8320
Epoch 10/25
536/536 [==============================] - ETA: 0s - loss: 0.4991 - accura
cy: 0.8393WARNING:tensorflow:Can save best model only with val_acc availab
le, skipping.
536/536 [==============================] - 974s 2s/step - loss: 0.4991 - a
ccuracy: 0.8393 - val_loss: 0.4963 - val_accuracy: 0.8340
Epoch 11/25
```

```
536/536 [==============================] - ETA: 0s - loss: 0.4670 - accura
cy: 0.8484WARNING:tensorflow:Can save best model only with val_acc availab
le, skipping.
536/536 [==============================] - 972s 2s/step - loss: 0.4670 - a
ccuracy: 0.8484 - val_loss: 0.4302 - val_accuracy: 0.8608
Epoch 12/25
536/536 [==============================] - ETA: 0s - loss: 0.4313 - accura
cy: 0.8586WARNING:tensorflow:Can save best model only with val_acc availab
le, skipping.
536/536 [==============================] - 967s 2s/step - loss: 0.4313 - a
ccuracy: 0.8586 - val_loss: 0.4484 - val_accuracy: 0.8552
Epoch 13/25
536/536 [==============================] - ETA: 0s - loss: 0.4082 - accura
cy: 0.8670WARNING:tensorflow:Can save best model only with val_acc availab
le, skipping.
536/536 [==============================] - 2817s 5s/step - loss: 0.4082 -
accuracy: 0.8670 - val_loss: 0.3504 - val_accuracy: 0.8857
Epoch 14/25
536/536 [==============================] - ETA: 0s - loss: 0.4123 - accura
cy: 0.8652WARNING:tensorflow:Can save best model only with val_acc availab
le, skipping.
536/536 [==============================] - 967s 2s/step - loss: 0.4123 - a
ccuracy: 0.8652 - val_loss: 0.3591 - val_accuracy: 0.8812
Epoch 15/25
536/536 [==============================] - ETA: 0s - loss: 0.3783 - accura
cy: 0.8771WARNING:tensorflow:Can save best model only with val_acc availab
le, skipping.
536/536 [==============================] - 2061s 4s/step - loss: 0.3783 -
accuracy: 0.8771 - val_loss: 0.3486 - val_accuracy: 0.8875
Epoch 16/25
536/536 [==============================] - ETA: 0s - loss: 0.3666 - accura
cy: 0.8818WARNING:tensorflow:Can save best model only with val_acc availab
le, skipping.
536/536 [==============================] - 1007s 2s/step - loss: 0.3666 -
accuracy: 0.8818 - val_loss: 0.3584 - val_accuracy: 0.8838
Epoch 17/25
536/536 [==============================] - ETA: 0s - loss: 0.5031 - accura
cy: 0.8510WARNING:tensorflow:Can save best model only with val_acc availab
le, skipping.
536/536 [==============================] - 972s 2s/step - loss: 0.5031 - a
ccuracy: 0.8510 - val_loss: 0.3443 - val_accuracy: 0.8923
Epoch 18/25
536/536 [==============================] - ETA: 0s - loss: 0.3384 - accura
cy: 0.8907WARNING:tensorflow:Can save best model only with val_acc availab
le, skipping.
536/536 [==============================] - 973s 2s/step - loss: 0.3384 - a
ccuracy: 0.8907 - val_loss: 0.3249 - val_accuracy: 0.8960
Epoch 19/25
536/536 [==============================] - ETA: 0s - loss: 0.3226 - accura
cy: 0.8956WARNING:tensorflow:Can save best model only with val_acc availab
le, skipping.
536/536 [==============================] - 967s 2s/step - loss: 0.3226 - a
ccuracy: 0.8956 - val_loss: 0.3630 - val_accuracy: 0.8831
Epoch 20/25
536/536 [==============================] - ETA: 0s - loss: 0.3217 - accura
cy: 0.8956WARNING:tensorflow:Can save best model only with val_acc availab
le, skipping.
536/536 [==============================] - 982s 2s/step - loss: 0.3217 - a
ccuracy: 0.8956 - val_loss: 0.3598 - val_accuracy: 0.8861
Epoch 21/25
536/536 [==============================] - ETA: 0s - loss: 0.3095 - accura
```

```
cy: 0.8978WARNING:tensorflow:Can save best model only with val_acc availab
le, skipping.
536/536 [==============================] - 2369s 4s/step - loss: 0.3095 -
accuracy: 0.8978 - val_loss: 0.3547 - val_accuracy: 0.8924
Epoch 22/25
536/536 [==============================] - ETA: 0s - loss: 0.3101 - accura
cy: 0.8991WARNING:tensorflow:Can save best model only with val_acc availab
le, skipping.
536/536 [==============================] - 1190s 2s/step - loss: 0.3101 -
accuracy: 0.8991 - val_loss: 0.3419 - val_accuracy: 0.8957
Epoch 23/25
536/536 [==============================] - ETA: 0s - loss: 0.2986 - accura
cy: 0.9031WARNING:tensorflow:Can save best model only with val_acc availab
le, skipping.
536/536 [==============================] - 999s 2s/step - loss: 0.2986 - a
ccuracy: 0.9031 - val_loss: 0.3080 - val_accuracy: 0.9021
Epoch 24/25
536/536 [==============================] - ETA: 0s - loss: 0.2867 - accura
cy: 0.9074WARNING:tensorflow:Can save best model only with val_acc availab
le, skipping.
536/536 [==============================] - 970s 2s/step - loss: 0.2867 - a
ccuracy: 0.9074 - val_loss: 0.2958 - val_accuracy: 0.9042
Epoch 25/25
536/536 [==============================] - ETA: 0s - loss: 0.2843 - accura
cy: 0.9095WARNING:tensorflow:Can save best model only with val_acc availab
le, skipping.
536/536 [==============================] - 1957s 4s/step - loss: 0.2843 -
accuracy: 0.9095 - val_loss: 0.2885 - val_accuracy: 0.9084
```

Out[11]:

```
<tensorflow.python.keras.callbacks.History at 0x25d189ed048>
```

In [14]:

```
class_distinct = training_set.class_indices
print(class_distinct)

list_class = list(class_distinct.keys())
print(list_class)
```

```
{'Apple___Apple_scab': 0, 'Apple___Black_rot': 1, 'Apple___Cedar_apple_rus
t': 2, 'Apple___healthy': 3, 'Blueberry___healthy': 4, 'Cherry_(including_
sour)___Powdery_mildew': 5, 'Cherry_(including_sour)___healthy': 6, 'Corn_
(maize)___Common_rust_': 7, 'Corn_(maize)___Northern_Leaf_Blight': 8, 'Cor
n_(maize)___healthy': 9, 'Grape___Black_rot': 10, 'Grape___Esca_(Black_Mea
sles)': 11, 'Grape___Leaf_blight_(Isariopsis_Leaf_Spot)': 12, 'Grape___hea
lthy': 13, 'Orange___Haunglongbing_(Citrus_greening)': 14, 'Peach___Bacter
ial_spot': 15, 'Peach___healthy': 16, 'Pepper,_bell___Bacterial_spot': 17,
'Pepper,_bell___healthy': 18, 'Potato___Early_blight': 19, 'Potato___Late_
blight': 20, 'Potato___healthy': 21, 'Raspberry___healthy': 22, 'Soybean__
_healthy': 23, 'Squash___Powdery_mildew': 24, 'Strawberry___Leaf_scorch':
25, 'Strawberry___healthy': 26, 'Tomato___Bacterial_spot': 27, 'Tomato___E
arly_blight': 28, 'Tomato___Late_blight': 29, 'Tomato___Leaf_Mold': 30, 'T
omato___Septoria_leaf_spot': 31, 'Tomato___Spider_mites Two-spotted_spider
_mite': 32, 'Tomato___Target_Spot': 33, 'Tomato___Tomato_Yellow_Leaf_Curl_
Virus': 34, 'Tomato___Tomato_mosaic_virus': 35, 'Tomato___healthy': 36}
['Apple___Apple_scab', 'Apple___Black_rot', 'Apple___Cedar_apple_rust', 'A
pple___healthy', 'Blueberry___healthy', 'Cherry_(including_sour)___Powdery
_mildew', 'Cherry_(including_sour)___healthy', 'Corn_(maize)___Common_rust
_', 'Corn_(maize)___Northern_Leaf_Blight', 'Corn_(maize)___healthy', 'Grap
e___Black_rot', 'Grape___Esca_(Black_Measles)', 'Grape___Leaf_blight_(Isar
iopsis_Leaf_Spot)', 'Grape___healthy', 'Orange___Haunglongbing_(Citrus_gre
ening)', 'Peach___Bacterial_spot', 'Peach___healthy', 'Pepper,_bell___Bact
erial_spot', 'Pepper,_bell___healthy', 'Potato___Early_blight', 'Potato___
Late_blight', 'Potato___healthy', 'Raspberry___healthy', 'Soybean___health
y', 'Squash___Powdery_mildew', 'Strawberry___Leaf_scorch', 'Strawberry___h
ealthy', 'Tomato___Bacterial_spot', 'Tomato___Early_blight', 'Tomato___Lat
e_blight', 'Tomato___Leaf_Mold', 'Tomato___Septoria_leaf_spot', 'Tomato___
Spider_mites Two-spotted_spider_mite', 'Tomato___Target_Spot', 'Tomato___T
omato_Yellow_Leaf_Curl_Virus', 'Tomato___Tomato_mosaic_virus', 'Tomato___h
ealthy']
```

In [20]:

```
from keras.preprocessing import image
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()

image_path = "test/test/AppleCedarRust1.jpg"
new_image = image.load_img(image_path, target_size = (224,224))
img = image.img_to_array(new_image)
img = np.expand_dims(img, axis = 0)
img = img/255

print("Predicted class")
prediction = cnn.predict(img)
d = prediction.flatten()
j = d.max()
for index,item in enumerate(d):
    if item == j:
        class_name = list_class[index]


#plt.figure(figsize(4,4))
plt.imshow(new_image)
plt.axis('off')
plt.title(class_name)
plt.show()
```

```
Predicted class
```

In [22]:

```
from keras.preprocessing import image
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()

image_path = "test/test/TomatoEarlyBlight1.jpg"
new_image = image.load_img(image_path, target_size = (224,224))
img = image.img_to_array(new_image)
img = np.expand_dims(img, axis = 0)
img = img/255

print("Predicted class")
prediction = cnn.predict(img)
d = prediction.flatten()
j = d.max()
for index,item in enumerate(d):
    if item == j:
        class_name = list_class[index]


#plt.figure(figsize(4,4))
plt.imshow(new_image)
plt.axis('off')
plt.title(class_name)
plt.show()
```

```
Predicted class
```

Tomato___Late_blight

In [23]:

```
from keras.preprocessing import image
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()

image_path = "test/test/PotatoEarlyBlight2.jpg"
new_image = image.load_img(image_path, target_size = (224,224))
img = image.img_to_array(new_image)
img = np.expand_dims(img, axis = 0)
img = img/255

print("Predicted class")
prediction = cnn.predict(img)
d = prediction.flatten()
j = d.max()
for index,item in enumerate(d):
    if item == j:
        class_name = list_class[index]


#plt.figure(figsize(4,4))
plt.imshow(new_image)
plt.axis('off')
plt.title(class_name)
plt.show()
```

```
Predicted class
```

Potato___Early_blight

In [24]:

```
from keras.preprocessing import image
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()

image_path = "test/test/TomatoYellowCurlVirus2.jpg"
new_image = image.load_img(image_path, target_size = (224,224))
img = image.img_to_array(new_image)
img = np.expand_dims(img, axis = 0)
img = img/255

print("Predicted class")
prediction = cnn.predict(img)
d = prediction.flatten()
j = d.max()
for index,item in enumerate(d):
    if item == j:
        class_name = list_class[index]


#plt.figure(figsize(4,4))
plt.imshow(new_image)
plt.axis('off')
plt.title(class_name)
plt.show()
```

```
Predicted class
```

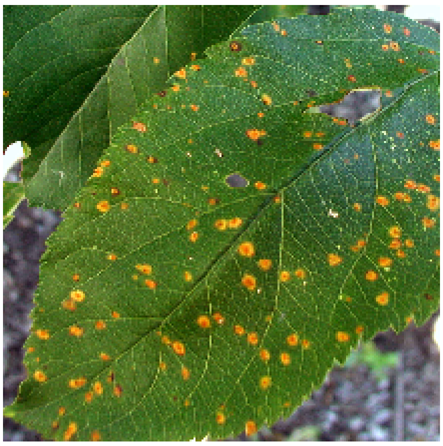
Tomato___Tomato_Yellow_Leaf_Curl_Virus

In [25]:

```
from keras.preprocessing import image
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()

image_path = "test/test/spots-Orange-cedar-apple-rust-disease-apple-leaf.jpg"
new_image = image.load_img(image_path, target_size = (224,224))
img = image.img_to_array(new_image)
img = np.expand_dims(img, axis = 0)
img = img/255

print("Predicted class")
prediction = cnn.predict(img)
d = prediction.flatten()
j = d.max()
for index,item in enumerate(d):
    if item == j:
        class_name = list_class[index]


#plt.figure(figsize(4,4))
plt.imshow(new_image)
plt.axis('off')
plt.title(class_name)
plt.show()
```

```
Predicted class
```

Tomato___Early_blight



In [27]:

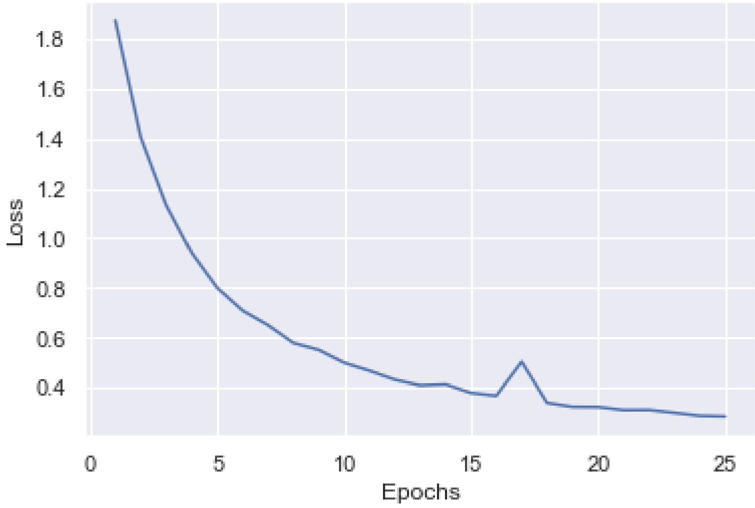
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()

results = pd.read_csv("Train_results.csv")
results
```

Out[27]:

	Epochs	Loss	Accuracy	Time	Time per step	Val_loss	Val_accuracy
0	1	1.8749	0.4247	3653	7	1.4192	0.5443
1	2	1.4068	0.5596	2224	4	1.1777	0.6197
2	3	1.1336	0.6405	1038	2	0.9601	0.6954
3	4	0.9401	0.6983	3196	6	0.7954	0.7397
4	5	0.7999	0.7409	2178	4	0.6870	0.7753
5	6	0.7087	0.7706	969	2	0.6650	0.7802
6	7	0.6501	0.7880	1970	4	0.5813	0.8105
7	8	0.5783	0.8125	1714	3	0.5991	0.8101
8	9	0.5505	0.8223	990	2	0.5214	0.8320
9	10	0.4991	0.8393	974	2	0.4963	0.8340
10	11	0.4670	0.8484	972	2	0.4302	0.8608
11	12	0.4313	0.8586	967	2	0.4484	0.8552
12	13	0.4082	0.8670	2817	5	0.3504	0.8857
13	14	0.4123	0.8652	967	2	0.3591	0.8812
14	15	0.3783	0.8771	2061	4	0.3486	0.8875
15	16	0.3666	0.8818	1007	2	0.3584	0.8838
16	17	0.5031	0.8510	972	2	0.3443	0.8923
17	18	0.3384	0.8907	973	2	0.3249	0.8960
18	19	0.3226	0.8956	967	2	0.3630	0.8831
19	20	0.3217	0.8956	982	2	0.3598	0.8861
20	21	0.3095	0.8978	2369	4	0.3547	0.8924
21	22	0.3101	0.8991	1190	2	0.3419	0.8957
22	23	0.2986	0.9031	999	2	0.3080	0.9021
23	24	0.2867	0.9074	970	2	0.2958	0.9042
24	25	0.2843	0.9095	1957	4	0.2885	0.9084

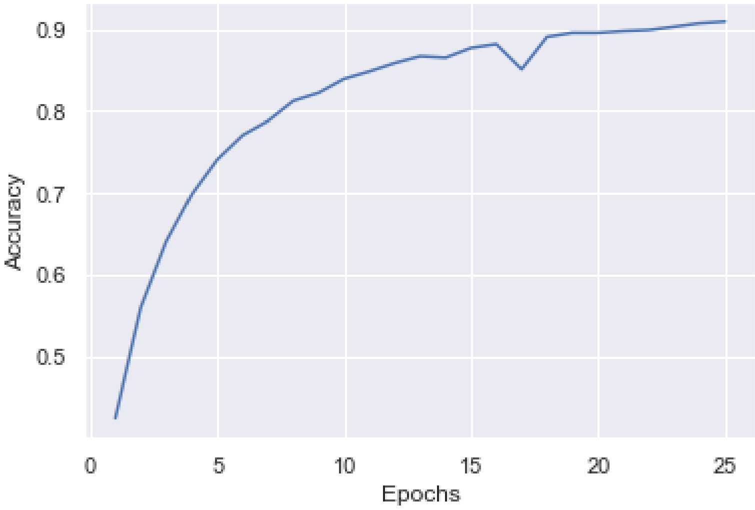
In [28]:

```
ax = sns.lineplot(x="Epochs", y="Loss", data = results)
```



In [29]:

```
ax = sns.lineplot(x="Epochs", y="Accuracy", data = results)
```

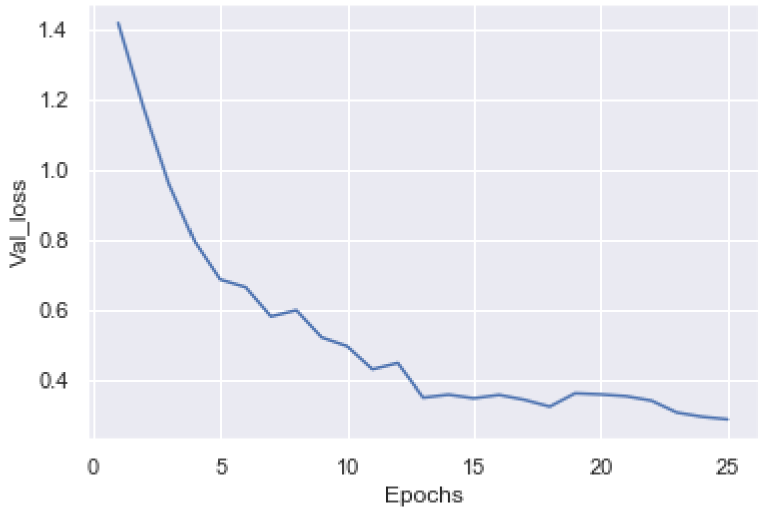


In [30]:

```
ax = sns.lineplot(x="Time", y="Accuracy", data = results)
```

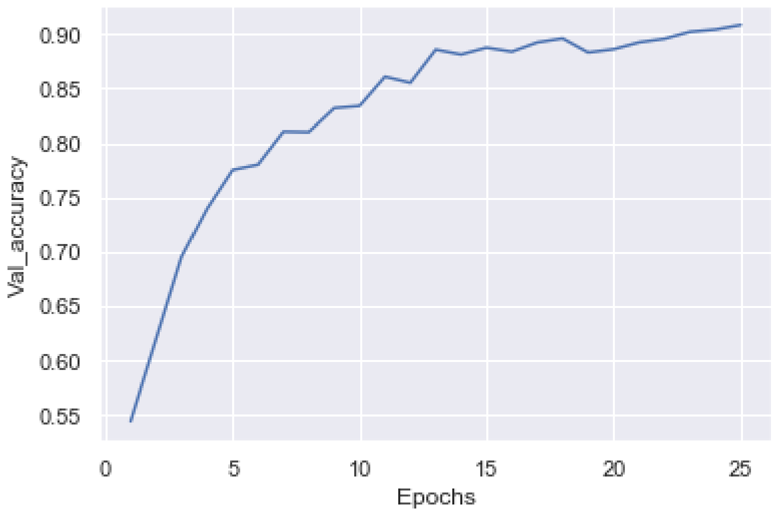
In [31]:

```
bx = sns.lineplot(x="Epochs", y="Val_loss", data = results)
```



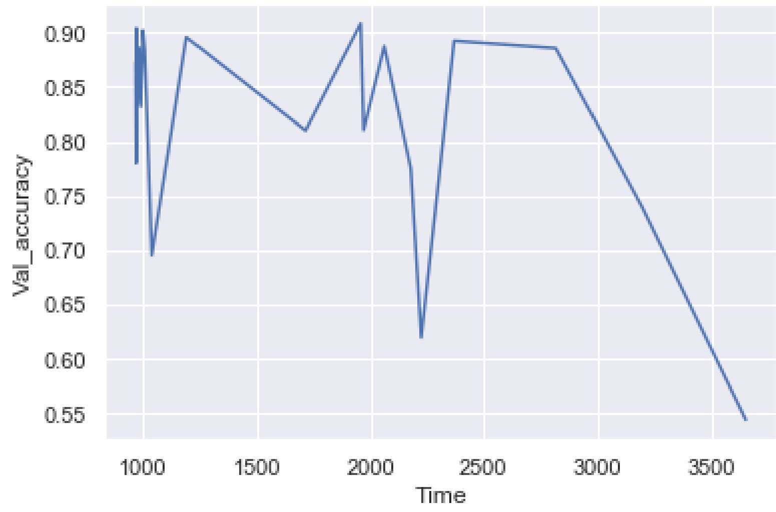
In [32]:

```
bx = sns.lineplot(x="Epochs", y="Val_accuracy", data = results)
```



In [33]:

```
bx = sns.lineplot(x="Time", y="Val_accuracy", data = results)
```

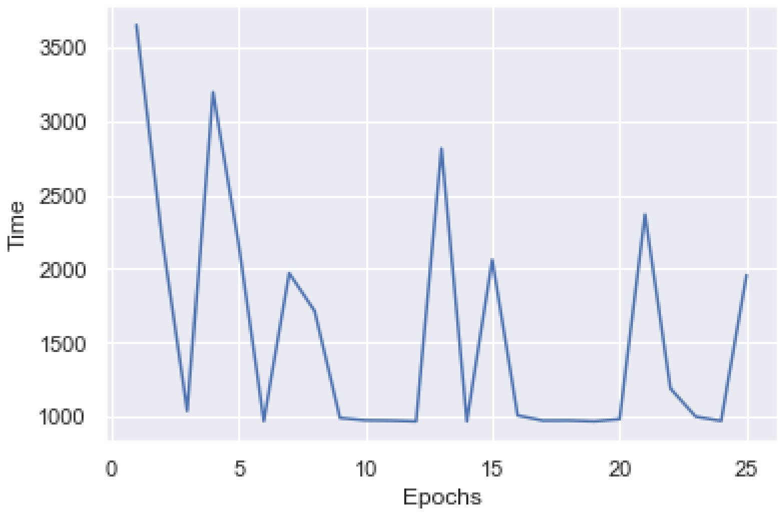


In [34]:

```
bx = sns.lineplot(x="Time", y="Loss", data = results)
```

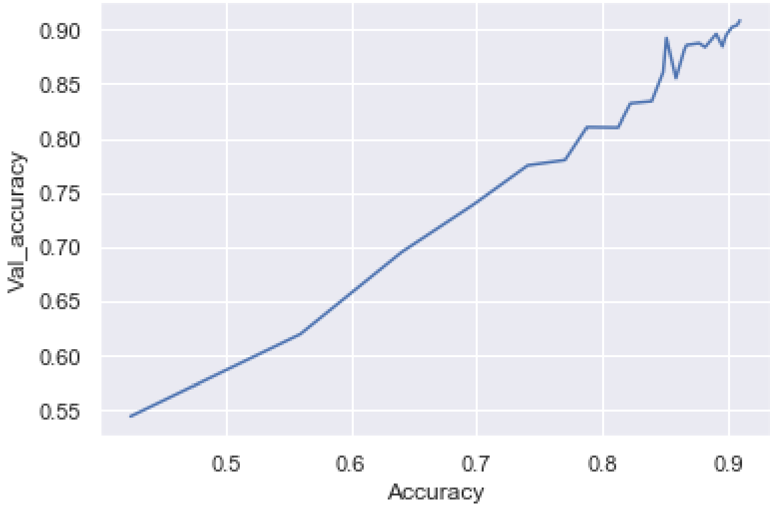
In [35]:

```
bx = sns.lineplot(x="Epochs", y="Time", data = results)
```



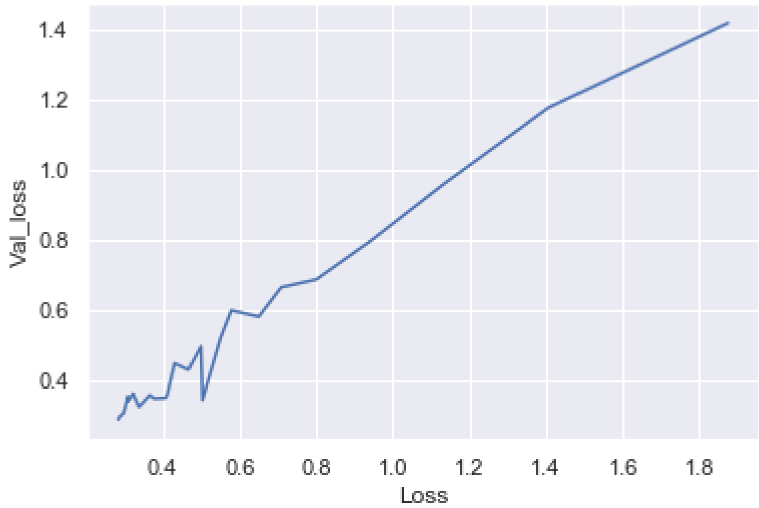
In [36]:

```
bx = sns.lineplot(x="Accuracy", y="Val_accuracy", data = results)
```



In [37]:

```
bx = sns.lineplot(x="Loss", y="Val_loss", data = results)
```



In [ ]: