In Final project part 1, CNN and VGG models were experimented with for thermal and RGB pictures. For part 2, autoencoders were experimented with.

RGB Autoencoder

Trial1: accuracy: 0.5026 val_accuracy: 0.5045

Model: "functional"

Layer (type)	Output Shape	Param #
input_layer_1 (InputLayer)	(None, 64, 64, 3)	0
conv2d_2 (Conv2D)	(None, 64, 64, 32)	896
max_pooling2d_2 (MaxPooling2D)	(None, 32, 32, 32)	0
dropout_1 (Dropout)	(None, 32, 32, 32)	0
conv2d_3 (Conv2D)	(None, 32, 32, 64)	18,496
max_pooling2d_3 (MaxPooling2D)	(None, 16, 16, 64)	0
flatten_1 (Flatten)	(None, 16384)	0
dense_2 (Dense)	(None, 32)	524,320
dense_3 (Dense)	(None, 16384)	540,672
reshape_1 (Reshape)	(None, 16, 16, 64)	0
conv2d_transpose_2 (Conv2DTranspose)	(None, 32, 32, 32)	18,464
conv2d_transpose_3 (Conv2DTranspose)	(None, 64, 64, 3)	867

Figure 1a: Autoencoder summary

```
ae_model = models.Sequential([
    layers.Input(shape=(x_train.shape[1],)),
    layers.Dense(128, activation='relu'),
    layers.Dense(8, activation='softmax')
])
optimizer = Adam(learning_rate=0.0001)
ae_model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

Figure 1b: Dense classifier: 20 epochs

Training and validation accuracy are both too low so further trials will be done!

Trial 2: accuracy: 0.5961 val_accuracy: 0.6005

Added batch normalization after every convolution layer in autoencoder to improve the performance and stability of training. There was a slight increase in validation and training accuracy

Trial 3: accuracy: 0.6092 val_accuracy: 0.6096

Added more units to convolution layers. First layer: 64 units, second layer: 128 units as can be seen by figure 2. The filter size is still 3x3. The autoencoder was made slightly more complex so that it can represent the feature better. Again, the validation and training accuracy increased slightly.

```
[ ]  input = layers.Input(shape=(64, 64, 3))

     l1 = layers.Conv2D(64, (3, 3), padding='same', activation='relu')(input)
     l1 = layers.BatchNormalization()(l1)
     l2 = layers.MaxPooling2D(2, padding='same')(l1)
     l3 = layers.Dropout(0.3)(l2)
     l4 = layers.Conv2D(128, (3, 3),  padding='same', activation='relu')(l3)
     l4 = layers.BatchNormalization()(l4)
     l5 = layers.MaxPooling2D(2, padding='same')(l4)
     l6 = layers.GlobalAveragePooling2D()(l5)
     encoded = layers.Dense(128, activation='relu')(l6)
     l7 = layers.Dense(16*16*64, activation='relu')(encoded)
     l8 = layers.Reshape((16, 16, 64))(l7)
     l9 = layers.Conv2DTranspose(32, (3, 3), strides=2, activation='relu', padding='same')(l8)
     decoder = layers.Conv2DTranspose(3, (3, 3), strides=2, padding='same', activation='sigmoid')(l9)
```

Figure 2: Trial 2 AE architecture

Trial 4: accuracy: 0.6092 val_accuracy: 0.6176

Trained the autoencoder for more epochs because 25 epochs doesn't seem enough for it to converge. I have also added early stopping so that training stops when there's no signs of improvement. Not much of an improvement so I'm going to make changes to the dense classifier

Trial 5: accuracy: 0.6115 val_accuracy: 0.6256

Added a dropout layer, another dense layer of 64 units, and increased the number of epochs, and added early stopping to the dense classifier. Hopefully, this reduces overfitting

and allows the model to process more with another dense layer. These changes have improved the training and validation accuracy.

Trial 6: accuracy: 0.6359 val_accuracy: 0.6474

Added a lower learning rate (learning_rate=0.0001) optimizer to the classifier. Accuracy was still increasing at 50 epochs therefore I'll try again with more epochs

```
[ ]  # dense classifier for autoencoder
     ae_model = models.Sequential([
         layers.Input(shape=(x_train.shape[1],)),
         layers.Dense(128, activation='relu'),
         layers.Dropout(0.3),
         layers.Dense(64, activation='relu'),
         layers.Dense(8, activation='softmax')
     ])
     optimizer = Adam(learning_rate=0.0001)
     ae_model.compile(optimizer=optimizer, loss='
```

Figure 3: lower learning rate for autoencoder dense classifier

Trial 7: accuracy: 0.6534 val_accuracy: 0.6583

The dense classifier was modified to run with 100 epochs and early_stopping because at 50 epochs accuracy seemed to continue to increase

Trial 8: accuracy: 0.7830 val_accuracy: 0.7918

Increased bottleneck size of the autoencoder to 128 from 32 and used GlobalAveragePooling2D instead of Flatten. Increasing the size allowed the model to learn more rich and complex feature information.

Trial 9: accuracy: 0.8753 val_accuracy: 0.8721

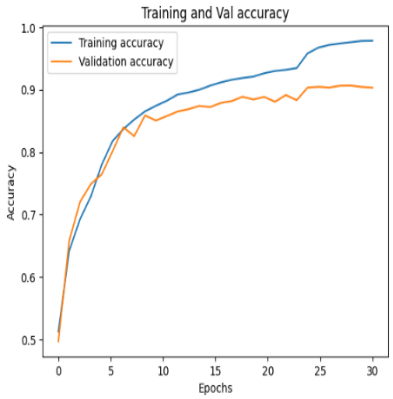
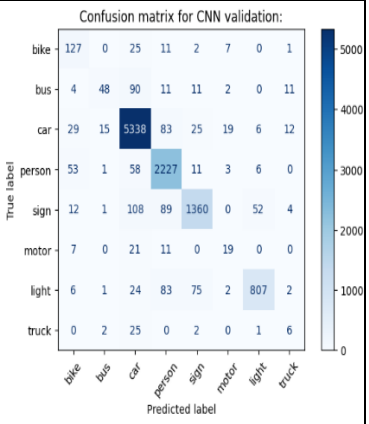
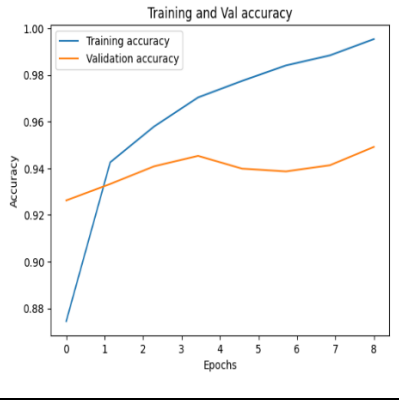
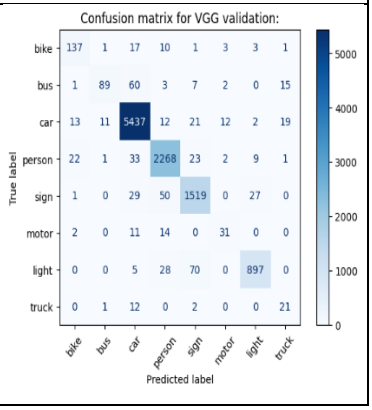
Early stopping was used, and the dataset was normalized before it was mapped so the autoencoder had less loss. Early stopping made sure the model was not overfitting that way it can predict better on new data.

```
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau

callbacks = [
    EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True),
    ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=3, verbose=1)
]
```

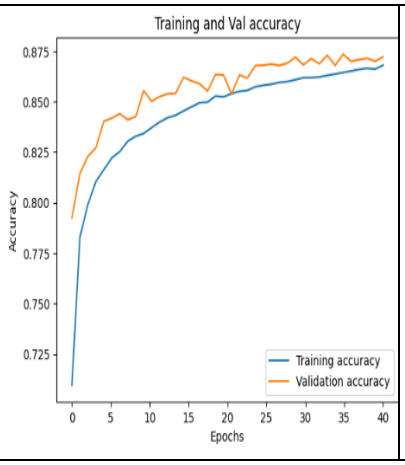
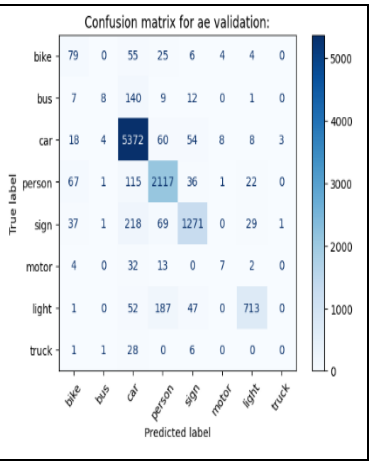
Figure 1c: Early stopping was introduced

Table 1: RGB models

Model	Accuracies	Train vs Val Acc graph	Metric	Confusion Matrix
CNN	**Training: 0.9691** **Validation: 0.9088**		**Acc: 0.9065** **F1: 0.6405** **AUC: 0.9565**	
VGG	Training: 0.9936 Validation: 0.9492		Acc: 0.9560 F1: 0.7802 AUC: 0.9865	

PRASANNA, LAKSITA
4/24/25

FINAL PROJECT PART 2

Auto-encoder	Training: 0.8753 Validation: 0.8721		Acc: 0.8732 F1: 0.5138 AUC: 0.9296	

The CNN model: after the final project part 1 was turned in, all models were introduced to both early stopping and reducing the learning rate on plateau and that greatly helped.

The VGG model: after part 1 the VGG model was improvised by adding a l2 regularizer of 0.001 and the dropout layer was increased to 0.4 from 0.3 units. That way the model was not overfitting too much.

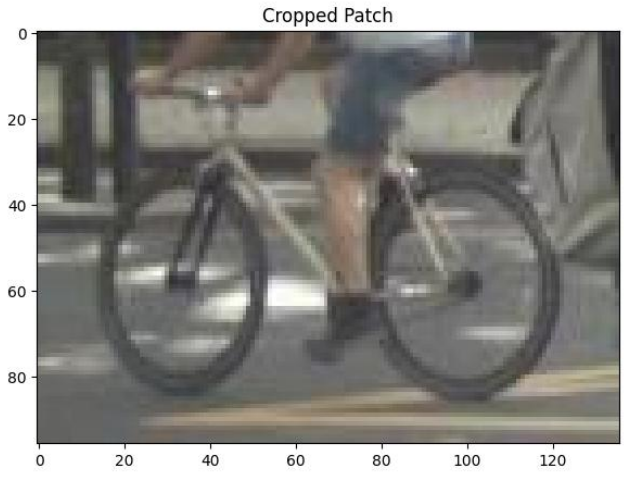


Figure 1d: Some RGB Autoencoder Misclassified examples

Best RGB model:

The CNN model is the best RGB model because the metrics and accuracies are both high. The VGG model is overfitting so it may not be great at classifying new data. If l2 regularizer was added to another dense layer in the VGG model and if it was trained for less epochs, it wouldn't overfit this much.

Figure 1e) Samples classified by RGB model architectures in order of CNN, VGG, and AE

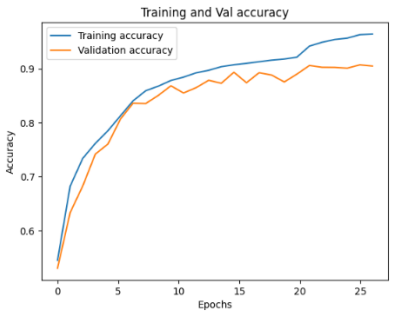
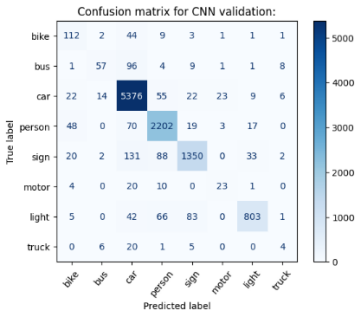
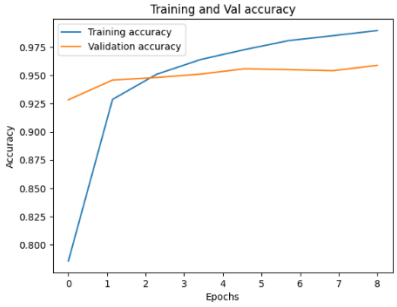
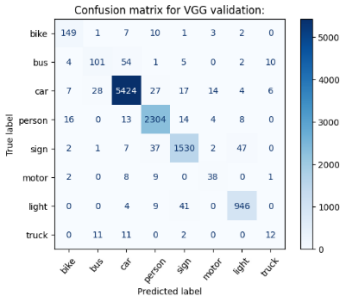
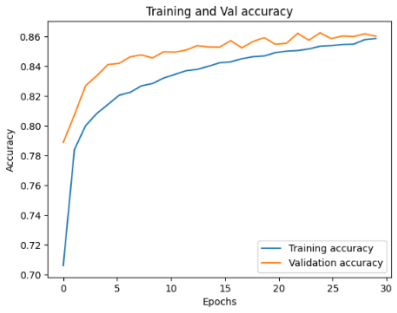
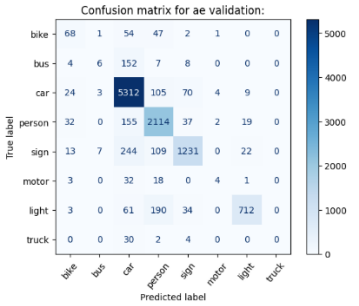
Figure 1e shows how the best model (CNN) in RGB correctly classified the bike as a bike in the first pic. Both VGG and AE incorrectly classify the bike as car and motor, respectively.

IR Autoencoder

All architectural level trials were done with the RGB autoencoder model so same logic was applied to the IR autoencoder and the training and validation accuracies yielded are given below in table 2.

Table 2: IR models

FINAL PROJECT PART 2

Model	Accuracy	Train vs Val Acc graph	Metric	Confusion Matrix
CNN	Training: 0.9633 Validation: 0.9050		Acc: 0.9061 F1: 0.6474 AUC: 0.9614	
VGG	**Training: 0.9894** **Validation: 0.9587**		**Acc: 0.9587** **F1: 0.7904** **AUC: 0.9874**	
Auto-Enco-der	Training: 0.8582 Validation: 0.8600		Acc: 0.8623 F1: 0.5000 AUC: 0.9218	

The CNN model: Early stopping and Reduce Learning rate on Plateau was introduced

The VGG model: for the thermal group I was able to play around with the VGG model more and added l2 regularization to both dense layers, reduced the first dense layer to 128 from 256 and increased the dropout layer units to 0.5 from 0.4

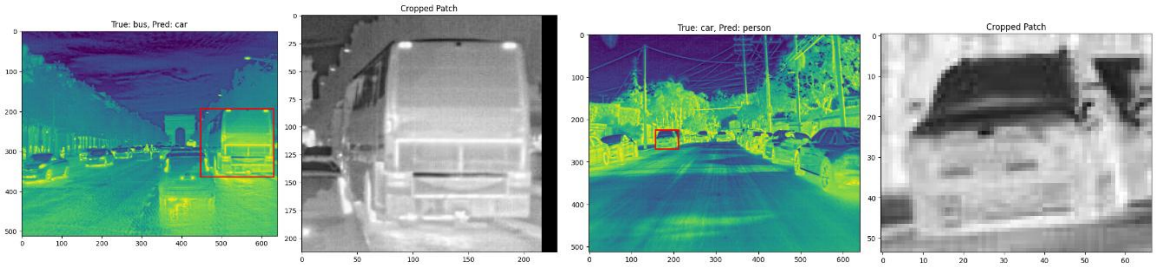
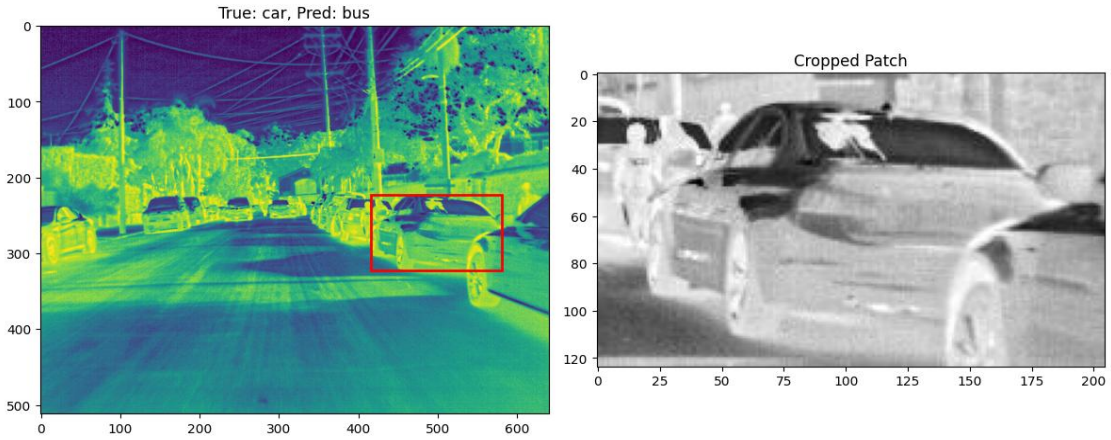
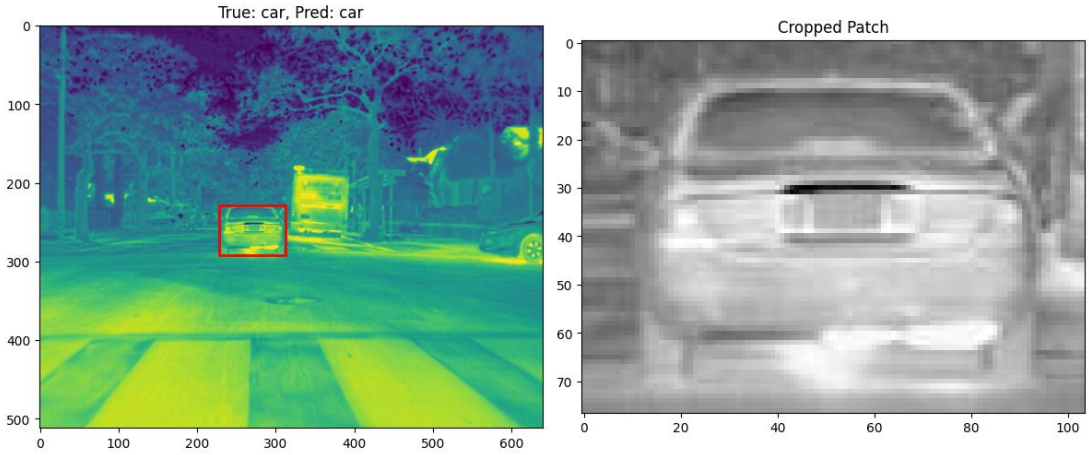


Figure 1f: Some IR Autoencoder misclassified examples

Best Model:

VGG is the best model for thermals because it has the highest training and validation accuracy compared to the other models in table 2. And the metrics for VGG are also higher than the other models in table 2.





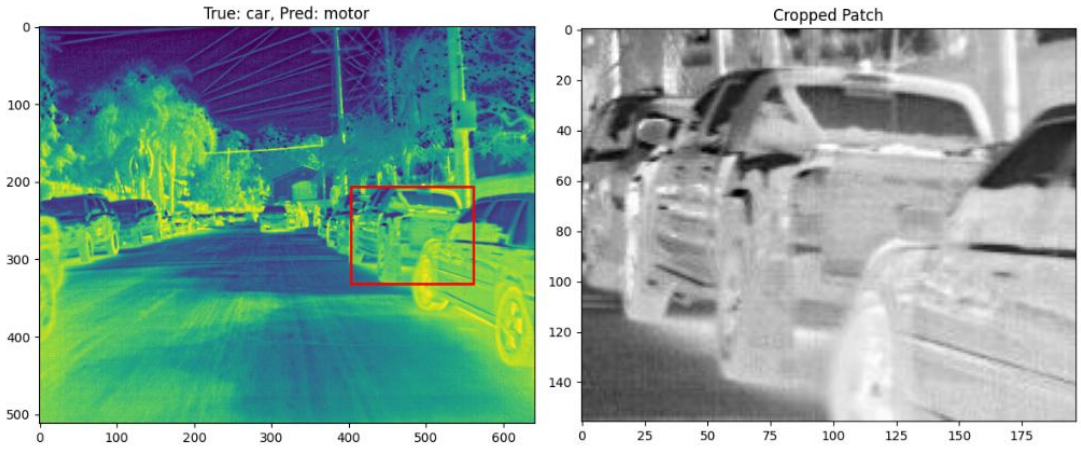


Figure 1g: Samples classified by Thermal model architectures in order of CNN, VGG, and AE

The best model (VGG) of thermal category correctly classifies cars as cars as shown by figure 1g. The other two models misclassify a car as bus or motor.

Using both RGB and InfraRed (IR) picture data, this project examined three deep learning architectures: a customized CNN, a fine-tuned VGG16, and an autoencoder (AE) with a dense classifier. The custom CNN performed best for RGB, particularly when regularization and early_stopping were added. In bright, visually stimulating surroundings, it generalized well. After fine-tuning, the VGG16 model—which had been pretrained in a different domain—performed best on infrared data, thanks to its deep hierarchical feature extraction capabilities. Strong resistance to noise and occlusion was demonstrated by the AE + Dense model in both modalities, which made it useful in more difficult situations.

Every model displayed distinct advantages. Context also affected sensor reliability: IR performed better in low-visibility situations like fog or nighttime, whereas RGB performed better in well-lit, textured surroundings.